

UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO - CTC
DEPARTAMENTO DE INFORMATICA E ESTATISTICA
CURSO DE SISTEMAS DE INFORMAÇÃO

Thalisson da Rosa

Trabalho de Conclusão de Curso

**Framework para jogos multiplayer em dispositivos móveis
Android**

Orientador: Roberto Silvino da Cunha

Florianópolis, Novembro de 2014

UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO - CTC
DEPARTAMENTO DE INFORMATICA E ESTATISTICA
CURSO DE SISTEMAS DE INFORMAÇÃO

Thalisson da Rosa

Trabalho de Conclusão de Curso

**Framework para jogos multiplayer em dispositivos móveis
Android**

Trabalho de Conclusão de Curso
apresentado como parte dos requisitos
para a obtenção do grau de Bacharel
em Sistemas de Informação

Florianópolis, Novembro de 2014

Thalisson da Rosa

Framework para jogos multiplayer em dispositivos móveis Android

Trabalho de Conclusão de Curso apresentado como parte dos requisitos para a
obtenção do grau de Bacharel em Sistemas de Informação

Orientador:

Roberto Silvino da Cunha, M Sc.
Universidade Federal de Santa Catarina

Banca examinadora:

Ricardo Pereira e Silva, D Sc.
Universidade Federal de Santa Catarina

Frank Siqueira, D Sc.
Universidade Federal de Santa Catarina

Patrícia Vilain, D Sc.
Universidade Federal de Santa Catarina

Florianópolis, Novembro de 2014

AGRADECIMENTOS

Agradeço a meus pais e minha irmã, por todo o apoio e paciência demonstrados durante todo o processo. Agradeço também a todos os amigos que, com suas palavras de incentivo nas horas mais difíceis forneceram a motivação necessária. Agradeço também ao orientador Roberto e ao professor Ricardo, que nunca desistiram e não me deixaram desistir nos momentos em que tudo parecia perdido.

RESUMO

Nos últimos anos o mercado de jogos vem crescendo de maneira contínua e acentuada, tornando-se uma indústria bilionária equivalente à do cinema. Além disso, o mercado de smartphones, mais especificamente do sistema Android, chega a mais de 1.5 milhão de novos dispositivos ativados diariamente. Juntando esses dois fatores, a indústria de jogos digitais voltada a sistemas móveis foi um passo natural e esperado.

Infelizmente, existe uma grande barreira inicial na curva de aprendizagem do desenvolvimento de jogos. A grande maioria dos frameworks existentes hoje atende somente a parcela com extenso conhecimento, deixando muito a desejar e até mesmo desencorajando quem está iniciando na área. Esse trabalho tem como objetivo o desenvolvimento de um framework simples e fácil de ser utilizado no desenvolvimento de jogos para smartphones Android.

Além de ser possível o desenvolvimento de jogos com apenas um jogador, o framework também dá suporte a partidas entre 2 jogadores remotamente, seja ela em modo cooperativo ou competitivo. Como prova de conceito foram desenvolvidos dois jogos utilizando o framework, para comprovar que sua especificação é válida.

ABSTRACT

In the last years, the gaming market has been growing continuously with a great scale, becoming a billionaire industry that we can compare to movie business. Besides that, the smartphone market, more specifically the ones with the Android system, is getting more than 1.5 million new devices being activated every day. When you put this two facts together, the gaming industry focused on mobile systems is a natural and expected step.

Unfortunately, there is a big barrier in the learning curve of game development. Most of the frameworks and engines available today focus on people that already have a great knowledge in the area, lacking a lot and even discouraging who is still learning. This paper has the objective of developing a simple framework that is easy to use to develop games to Android smart phones.

Besides being possible to develop games for just one player, this framework is capable of supporting games between two remote players, in cooperative or competitive mode. As a proof of concept we've developed two games, to prove that the framework specification is valid.

LISTA DE ILUSTRAÇÕES

Figura 1 - Distribuição de sistemas operacionais móbile	18
Figura 2 - Hagenuk e Snake.....	19
Figura 3 - Real Racing 2 e Infinity Blade	21
Figura 4 - Max Payne e Modern Combat V	21
Figura 5 - Tennis for Two e Pong	23
Figura 6 - Diagrama de casos de uso	34
Figura 7 - Diagrama de atividades	34
Figura 8 - Ciclo de jogo	35
Figura 9 - Classe Renderable.....	36
Figura 10 - Diagrama de Seqüência.....	37
Figura 11 - Classe FrameworkSurfaceView	39
Figura 12 - Métodos para tratamento de toques na tela.....	40
Figura 13 - Métodos para tratamento de eventos de teclado	41
Figura 14 - Classe Conector.....	43
Figura 15 - Diagrama de seqüência da comunicação de rede	44
Figura 16 - Diagrama de classes.....	45
Figura 17 - Classe Game	46
Figura 18 - Classe Server.....	46
Figura 19 - Diagrama de atividade Pedra-Papel-Tesoura	48
Figura 20 - Regras Pedra-Papel-Tesoura	49
Figura 21 - Tela de jogo Pedra-Papel-Tesoura	51
Figura 22 - Tela de jogo BugHunt	54

LISTA DE ABREVIATURAS E SIGLAS

3G - Terceira geração de padrões e tecnologias de telefonia móvel.

PDA - Personal digital assistant

RIM - Research in Motion

NMT - Nordic Mobile Telephone

AMPS - Advanced Mobile Phone System

TDMA - Time Division Multiple Access

CDMA - Code Division Multiple Access

GSM - Global System for Mobile communications

SMS - Short message system

WCDMA - Wideband Code Division Multiple Access

WAP - Wireless Application Protocol

BREW - Binary Runtime Environment for Wireless

J2ME - Java Platform, Micro Edition

PSP - PlayStation Portable

TCP - Transmission Control Protocol

IP - Internet Protocol

FPS - Frames per Second

UDP - User Datagram Protocol

IDE - Integrated Development Environment

SDK - Software Development Kit

TCP/IP - Internet Protocol Suite.

SUMÁRIO

1. INTRODUÇÃO.....	10
1.1 Problema.....	12
1.2 Hipótese.....	12
1.3 Objetivos.....	13
1.3.1 Objetivo Geral.....	13
1.3.2 Objetivos Específicos.....	13
1.4 Organização do Trabalho.....	13
2. HISTÓRICO.....	15
2.1 Celulares.....	15
2.2 Jogos Eletrônicos em Celulares.....	18
3. REVISÃO.....	22
3.1 Jogos Eletrônicos.....	22
3.2 Jogos Multiplayer.....	24
3.2.1 Jogos multiplayer e dispositivos móveis.....	24
3.2.2 Interação em jogos multiplayer.....	25
3.2.3 Sistemas distribuídos em jogos multiplayer.....	26
3.3 Frameworks.....	27
3.3.1 Desenvolvimento de frameworks.....	29
3.3.2 Metodologias de desenvolvimento de frameworks.....	29
3.4 Frameworks Existentes.....	31
4. PROJETO.....	33
4.1 Framework para jogos multiplayer Android.....	33
4.1.1 Ciclo de jogo.....	35
4.1.1.1 Renderização.....	36
4.1.1.2 Atualização.....	37
4.1.2 Controle de eventos e input.....	39
4.1.3 Network.....	42
4.1.4 Arquitetura.....	44
4.2 Framework de um servidor para jogos multiplayer.....	45
5. IMPLEMENTAÇÃO.....	48
5.1 Jogo de exemplo Pedra-Papel-Tesoura.....	48
5.1.1 Cliente.....	49

5.1.2	Servidor	51
5.2	Jogo de exemplo BugHunt	52
5.2.1	Cliente	53
5.2.2	Servidor	55
6.	CONCLUSÃO	57
7.	Referências Bibliográficas.....	59
	ANEXO A - CÓDIGO FONTE.....	61
	ANEXO B - ARTIGO	88

1. INTRODUÇÃO

Mundialmente, o mercado de celulares vem crescendo cada vez mais e já abrange a grande maioria dos habitantes. O Brasil não é uma exceção e, de acordo com a ANATEL (2014), em setembro de 2014 o país chegou a 278,1 milhões de linhas ativas e 136.9 celulares para cada 100 habitantes.

Dentro desse grupo, temos os smartphones, que são aparelhos com capacidade de processamento muito acima de aparelhos convencionais e contam com vários recursos, entre eles acesso a redes 3G e sincronização com serviços online. Resumidamente, seria um celular com funções de PDA (*Personal Digital Assistant*), rodando um sistema operacional que permite personalização.

Até pouco tempo, os smartphones eram mais utilizados por executivos e o mercado era dominado pela RIM (*Research in Motion*) com o Blackberry, mas com o lançamento do iPhone pela Apple e do G1, primeiro aparelho com o sistema Android da Google, gerou-se competição e queda nos preços, o que tornou esse tipo de aparelho mais acessível para o público em geral.

Em paralelo a isso, outro ramo que vem crescendo é dos jogos eletrônicos, que já vem competindo com o tradicional cinema no quesito entretenimento. De pequenas produções no passado, hoje já temos jogos com orçamentos de mais de 100 milhões de dólares, como foi o caso de *Grand Theft Auto IV* (ANDROVICH, 2008), que envolveu mais de 1000 profissionais durante três anos e meio.

Outro fator que explica o crescimento é o fato das empresas estarem investindo em nichos de mercado que até pouco tempo eram deixados de lado, que são os chamados jogadores casuais. Esse tipo de jogador é aquele que não quer ter de passar horas e mais horas decorando muitas informações para poder aproveitar um jogo completamente. O que ele procura são jogos simples, com baixa curva de aprendizado e nível de desafio mas que sejam divertidos e que, se possível, dêem suporte ao jogo em grupo.

Isso ficou claro com a aposta bem sucedida feita pela Nintendo com o lançamento do Wii (BISHOP, 2013) no final de 2006, que focando em um sistema de controle inovador, baseado em movimento, e com jogos acessíveis para toda a família, tornou-se o campeão de vendas da sétima geração de consoles, com um total de mais de 100 milhões de consoles.

Isso ficou ainda mais claro quando as empresas concorrentes, Microsoft e Sony, lançaram o Kinect e o Move, respectivamente, para tentar captar esse usuário casual.

Além disso, um ramo de jogos que faz muito sucesso atualmente é o de jogos online. Esse tipo de jogo é o que proporciona a interação entre jogadores que se encontram em locais físicos diferentes, tornando possível o jogo em grupo. Atualmente, os principais responsáveis desse tipo de jogo são o World of Warcraft (STATISTA, 2014), um jogo de RPG massivo online com mais de 7.4 milhões de usuários no terceiro trimestre de 2014.

Com o mercado de jogos em expansão e o de smartphones também em pleno crescimento, nada mais natural que os jogos também invadissem a

plataforma móvel. Por se tratar de um fenômeno recente, o desenvolvimento não é algo relativamente fácil e faltam ferramentas que auxiliam esse trabalho.

1.1 Problema

O desenvolvimento de jogos não é uma tarefa fácil, exige muito conhecimento específico e experiência. Para facilitar o processo, existem frameworks para auxiliar nesse desenvolvimento, mas nenhum deles tem como foco o desenvolvedor iniciante, aquele que quer aprender fazendo.

Isso acaba gerando uma barreira inicial muito grande que pode acabar afastando o desenvolvedor, pois apesar de ter interesse, a busca por conceitos básicos tem que vir de outras fontes muitas vezes não relacionadas ao desenvolvimento de jogos.

1.2 Hipótese

Smartphones são plataformas ideais para jogos simples, que não exigem extenso conhecimento por parte do desenvolvedor. Muitas vezes o que basta é apenas a idéia certa no momento certo. Tendo isso como base, a hipótese levantada é que um jogo para celular pode ser fácil de ser desenvolvido.

Ao mesmo tempo em que deve ser fácil de ser desenvolvido, essa ferramenta inicial deve ensinar o básico do desenvolvimento de jogos, dando o suporte necessário para que o usuário, mais tarde, possa migrar para ferramentas mais poderosas e complexas.

1.3 Objetivos

1.3.1 Objetivo Geral

Este trabalho tem como objetivo geral efetuar um estudo no cenário do desenvolvimento de jogos multiplayer para dispositivos móveis e implementar uma ferramenta que facilite o desenvolvimento desse tipo de jogos.

1.3.2 Objetivos Específicos

Como objetivos específicos, temos os seguintes pontos:

- Revisão do cenário atual do desenvolvimento de jogos para smartphones.
- Revisão e análise das arquiteturas disponíveis para sistemas distribuídos em dispositivos móveis
- Análise das plataformas móveis disponíveis atualmente
- Análise das soluções disponíveis atualmente para essas plataformas
- Projeto e desenvolvimento de um framework para o desenvolvimento de jogos multiplayer
- Desenvolvimento de dois jogos utilizando o framework como caso de uso

1.4 Organização do Trabalho

Para alcançar os objetivos propostos por esse trabalho, serão seguidas as seguintes etapas:

Etapa 1: Estudo inicial, leitura de material relacionado, levantamento histórico e revisão do estado da arte. Nessa etapa o maior foco é no desenvolvimento de frameworks e jogos em geral

Etapa 2: Definição da estrutura do framework, diagramação e proposta de dois jogos a serem desenvolvidos como prova de conceito.

Etapa 3: Desenvolvimento do framework tanto do cliente quanto do servidor. Também serão desenvolvidos os dois jogos que atuarão como prova de conceito.

Etapa 4: Análise final dos jogos desenvolvidos, avaliando a validade do framework e quão fácil foi desenvolver os jogos em questão. Levantamento de funcionalidades que poderão ser desenvolvidas em trabalhos futuros.

2. HISTÓRICO

2.1 Celulares

O telefone celular é um aparelho de comunicação que utiliza ondas eletromagnéticas que permitem a transmissão de voz e dados em uma área geográfica que é dividida em células (origem do nome Celular). A invenção do aparelho ocorreu em dezembro de 1947, quando engenheiros da Bell Labs propuseram o modelo de células hexagonais com torres nas extremidades, e não no centro, obtendo maior cobertura. Infelizmente na época a tecnologia necessária para o desenvolvimento ainda não existia e foi só no dia 3 de Abril de 1973 que Martin Cooper, um pesquisador da Motorola, efetuou a primeira ligação utilizando um dispositivo móvel.

Em 1979 foi lançada no Japão a primeira rede comercial de celulares, que ficou conhecida como a primeira geração (1G). Nessa primeira etapa foram utilizadas duas tecnologia distintas, o NMT (*Nordic Mobile Telephone*) e o AMPS (*Advanced Mobile Phone System*). Ambas eram modelos analógicos, que possuíam baixa capacidade, suportavam poucos usuários (cada usuário utilizava um canal de frequência) e alto consumo de energia.

Com a massificação do serviço, novas tecnologias foram necessárias e surgiu a segunda geração (2G), com o TDMA (Time Division Multiple Access), CDMA (Code Division Multiple Access) e GSM (Global System for Mobile communications). Nessa geração passou a ser adotado um sinal digital, que possibilitou um ganho na capacidade do sistema, pois permite que o sinal seja

comprimido e mais chamadas ocupem o mesmo canal de frequência. A diferença entre o TDMA e o CDMA se dá no tipo de multiplexação utilizada.

O GSM (*Global System for Mobile communications*) é uma evolução do TDMA que dominou o mercado, atualmente sendo utilizado por mais de 80% dos usuários. (GSMWORLD, 2014).

No GSM, as principais evoluções foram o uso do SIM Card (chip removível) com informações do aparelho, aumento na segurança e cobertura global. Outro ponto interessante em relação à segunda geração é que, além da transmissão de voz, ela tornou possível a transmissão de dados, viabilizando o sistema de SMS (Short Message Service) e o acesso à web, mesmo que de forma extremamente limitada.

Com essa limitação do GSM no quesito dados, em 2001 foi lançado no Japão o padrão WCDMA (Wideband Code Division Multiple Access), que seria considerado como o início da terceira geração (3G). No 3G houve um aumento na segurança disponível, mas um dos maiores avanços foi na velocidade de transmissão de dados. Enquanto no melhor dos casos se conseguia 236.8 kbit/s na 2G, no 3G o valor mínimo para um usuário em movimento (dentro de um veículo) é de 384 kbit/s, devendo chegar a 2mbit/s para um usuário parado.

Na época, toda essa velocidade disponível não era aproveitada, pois os modelos de aparelho celulares existentes tinham poucas funcionalidades além de enviar mensagens e fazer ligações e foi isso que motivou o desenvolvimento dos smartphones.

Conforme diz a tradução do nome, um smartphone é um celular inteligente. Isso se deve ao uso de um sistema operacional bastante robusto em conjunto com boa capacidade de processamento, o que viabiliza o uso de

aplicativos que antes só existiam nos tradicionais desktops. No início, o usuário comum não necessitava desse tipo de aparelho, que encontrou no mercado executivo seu nicho, tendo como principal fabricante a *Research in Motion* (RIM) com o Blackberry. Ainda hoje a RIM continua a lançar seus produtos e em março de 2014, ainda possui 85 milhões de usuários ativos mensalmente e 113 milhões de registrados. (LOMAS, 2014).

Várias empresas lançaram suas soluções nesse meio, como a Nokia com o Symbian OS e a Microsoft com Windows CE, mas nenhuma delas conquistou o grande público. Isso mudou quando, em 2007, a Apple lançou o iPhone, que apesar do valor elevado (US\$500,00 + contrato com a operadora AT&T) vendeu mais de 500 mil unidades no primeiro final de semana. (CNN, 2007).

Baseado nesse enorme sucesso, a Apple continuou a aperfeiçoar seu aparelho (iPhone 3G e iPhone 4) e a roubar da RIM uma boa parcela de mercado e apesar dos esforços dos concorrentes, tudo indicava que ela seria bem sucedida nessa tarefa e reinaria absoluta . E essa previsão se manteve até que a Google entrou no mercado com o SO Android.

Lançado em outubro de 2008, o Android é um sistema operacional aberto, de código livre e rodando sobre o Linux. Uma das principais vantagens é o uso de uma linguagem similar ao Java, o que facilita em muito a portabilidade do código para os mais diversos modelos disponíveis.

Apesar de um lançamento humilde, aos poucos os fabricantes foram comprando a idéia da plataforma e lançando novos aparelhos. Mais importante que isso, os usuários começaram a ser atraídos por essa plataforma que nada

deixava a desejar em relação ao iPhone, permitia uma opção de escolha muito maior e em boa parte dos casos, como valores mais acessíveis.

Hoje, o Android vem crescendo de maneira incontestável, sendo que em agosto de 2010, como pode ser visto na Figura 1, pela primeira vez desde o lançamento, ele foi o campeão de vendas segundo a Nielsen (2010).

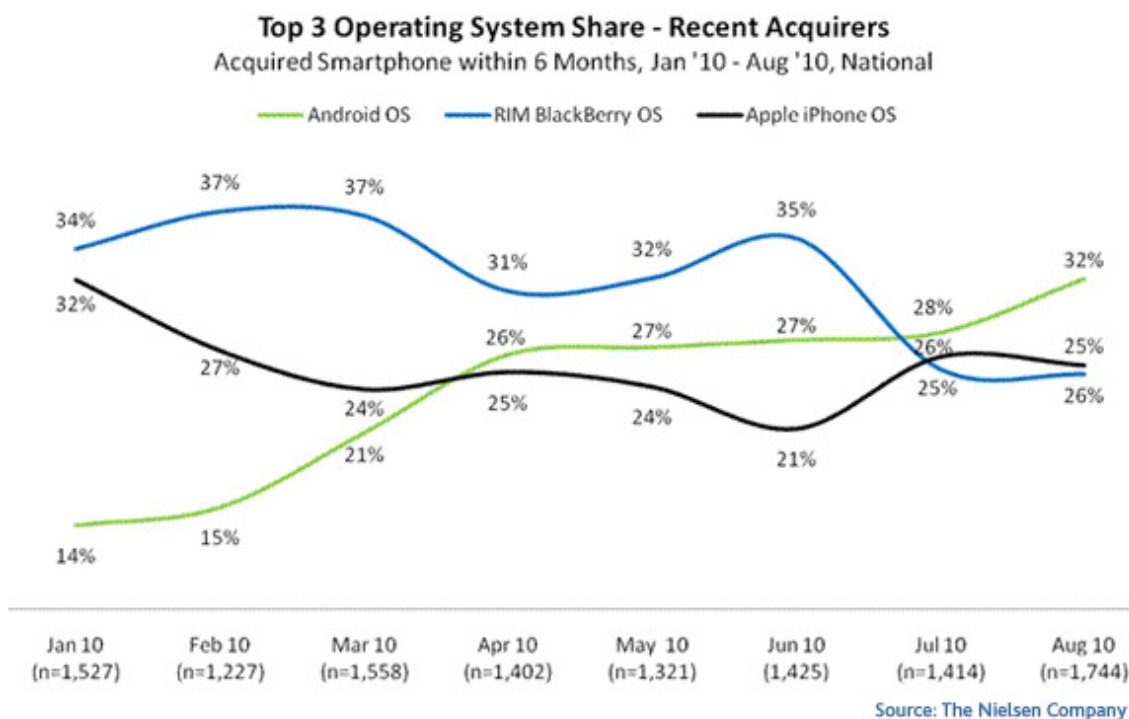


Figura 1 - Distribuição de sistemas operacionais móbile

Além disso, segundo levantamento efetuado pela Gartner (2010), em 2014 o Android será um dos líderes do mercado, ultrapassando o BlackBerry OS e iPhone OS, avançando dos atuais 17,7% de *market share* para previstos 29.6%.

2.2 Jogos Eletrônicos em Celulares

No início os celulares não possuíam capacidade de processamento suficiente para que jogos fossem executados. O primeiro celular que veio com

um jogo pré-instalado foi o Hagenuk MT-2000 (GSMHISTORY, 2014), mas o mesmo não teve a mesma penetração de mercado que seu sucessor.

Três anos mais tarde, em 1997, a Nokia lançou em certos modelos o *Snake* (Figura 2), que fez extremo sucesso e de acordo com fontes da empresa (NOKIA, 2011), é o jogo mais popular em dispositivos móveis do mundo, estando presente em mais de trezentos e cinquenta milhões de dispositivos.

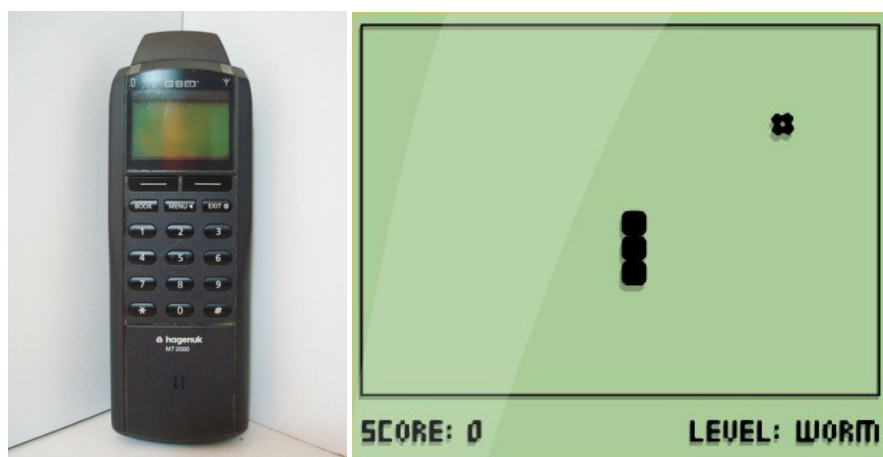


Figura 2 - Hagenuk e Snake

Ainda nessa época, houve algumas tentativas de jogos multiplayer's utilizando a rede WAP mas os mesmos não foram bem sucedidos, talvez devido ao alto custo que a transferência de dados impunha, o que acabou relegando ao WAP apenas o papel de meio de distribuição para os jogos que não vinham pré-instalados no dispositivo.

O cenário de jogos foi evoluindo aos poucos e foi a partir de 2001 que houve um crescimento e mudanças significativas. Primeiro foi o lançamento do BREW (*Binary Runtime Environment for Wireless*) da Qualcomm, que possuía uma API para desenvolvimento de jogos em C++ e era voltado para celulares CDMA. Apesar de ser uma ferramenta poderosa, ela não teve a mesma aceitação de outro lançamento dessa mesma época que foi o J2ME, que é uma

plataforma Java voltada para dispositivos móveis e mais voltada para os aparelhos GSM. O J2ME fez tanto sucesso que ainda hoje é encontrado em vários aparelhos sendo lançados.

O próximo grande passo no mercado de jogos para celulares foi em 2003, com o lançamento de jogos em 3D e com a grande aposta da Nokia com o N-Gage. Apesar de ser um smartphone com ótima capacidade de processamento, o aparelho não vingou no mercado, pois como o mesmo se aproximava mais de um console portátil ele acabou concorrendo com o recém lançado Sony PSP e com o Nintendo Game Boy Advance, que já vinha dominando esse mercado desde 1989.

A partir de 2004, as grandes empresas de jogos como a Electronic Arts, começaram a ver o potencial do mercado e passaram a investir em divisões específicas para dispositivos móveis. No início, esses jogos eram apenas versões mais leves e simplificadas dos produtos principais voltados aos consoles de mesa e PC.

Esse cenário de menor importância do mercado de dispositivos móveis foi modificado em 2007 com o lançamento do primeiro iPhone, da Apple. Com ótimos gráficos (Figura 3), tela sensível ao toque e grande capacidade de processamento, esse aparelho foi muito bem sucedido em dominar o mercado, mas um dos principais diferenciais responsável por atrair os desenvolvedores foi a App Store, a loja virtual da Apple, facilitando muito a distribuição dos jogos e fornecendo um canal direto entre clientes e desenvolvedores.



Figura 3 - Real Racing 2 e Infinity Blade

Na plataforma Android o mercado de jogos teve um início mais lento pois, preocupando-se mais em dar suporte a um grande número de diferentes dispositivos, acabou se deixando de lado a parte de otimização, afetando diretamente o desempenho dos jogos. Com o tempo a diferença entre as duas plataformas diminuiu até chegar aos dias atuais, onde temos jogos como Max Payne e Modern Combat V (Figura 4).



Figura 4 - Max Payne e Modern Combat V

3. REVISÃO

3.1 Jogos Eletrônicos

Segundo (HUIZINGA, 1980) um jogo é “(...) uma atividade ou ocupação voluntária, exercida dentro de certos e determinados limites de tempo e de espaço, segundo regras livremente consentidas, mas absolutamente obrigatórias, dotado de um fim em si mesmo, acompanhado de um sentimento de tensão e de alegria e de uma consciência de ser diferente da 'vida cotidiana'”. Em jogos eletrônicos, uma máquina é a responsável por fazer esse controle de regras e tempo, tornando a experiência mais fluida e fácil.

Tendo isso em mente, um jogador utiliza de uma interface de controle, como um teclado, para enviar suas decisões para a máquina. A máquina processa essa informação e mostra o resultado em uma tela. Por exemplo, em um jogo de xadrez, o comando do usuário seria a movimentação de uma peça. A máquina processaria e validaria o lance e caso fosse válido, mostraria o resultado na tela e ficaria esperando o comando de um segundo jogador.

No início, esses jogos eram simples e resultados de experimentos em universidades. O primeiro jogo que se tem registro é o Tennis for Two (Figura 5), criado no laboratório de pesquisas militares de Brookhaven National Laboratory. Utilizando um osciloscópio, era um jogo simples de tênis que mostrava uma rede e a bola que ia de um lado para o outro. Depois disso vieram outros jogos, como o Spacewar, mas o jogo que fez grande sucesso foi

Pong, lançado pela Atari em 1972 e que vendeu mais de 19000 máquinas (Lipson & Brain).

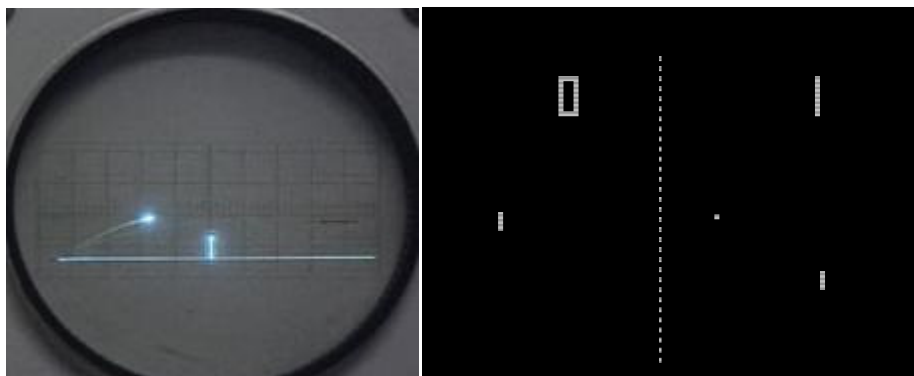


Figura 5 - Tennis for Two e Pong

Com o aumento do poder de processamento das máquinas e o interesse dos consumidores cada vez maior, esse mercado passou a ser levado mais a sério e os jogos passaram a ficar cada vez mais grandiosos, tornando-se super produções. Um dos mais recentes lançamentos, Grand Theft Auto 5, um jogo que permite livre exploração em um grande mapa, custou aproximadamente US\$115 milhões somente no desenvolvimento do jogo e mais US\$150 milhões em marketing (BRUSTEIN, 2013), fazendo dele o jogo mais caro já produzindo, se igualando a super produções de Hollywood.

É importante citar que o jogo pagou facilmente os seus custos e trouxe um grande lucro para a empresa, pois somente nos três primeiros dias após o lançamento, o faturamento foi superior a US\$1 bilhão.(FUTTER, 2013).

Apesar disso, o que torna a indústria de games ainda mais interessante é que todos têm uma chance de prosperar. Não é porque um jogo não conta com gráficos de última geração que o mesmo está fadado ao fracasso. Um jogo pode ser simples, desde que possua outras características que chamem a atenção do público-alvo desejado.

3.2 Jogos Multiplayer

Jogos multiplayer são jogos em que é possível que uma ou mais pessoas joguem juntas, seja de forma cooperativa ou não. Esse tipo de jogo pode ser tanto de forma local, através de redes locais (LAN), ou então utilizando a internet.

Uma das maiores vantagens nesse tipo de jogo é que ao invés de enfrentar inimigos controlados pela inteligência artificial do jogo, que muitas vezes é limitada e repetitiva, o jogador está enfrentando uma pessoa real, o que adiciona uma nova camada de dificuldade.

3.2.1 Jogos multiplayer e dispositivos móveis

Nos dispositivos móveis existem algumas opções para a criação de redes locais, como o infravermelho ou Bluetooth. Apesar de serem alternativas que provêem uma boa velocidade na transmissão de dados, os dispositivos têm que estar relativamente próximos um dos outros e, no caso de infravermelho, devem estar até mesmo apontado um para o outro.

. Uma alternativa que elimina esses problemas é o uso da rede de dados da operadora do celular, seja ela 3G ou GPRS, ou ainda no melhor cenário, o uso de conexão via Wi-Fi. Por serem recursos disponível em todos os celulares atualmente, eles são a forma de comunicação escolhida em todos os jogos atuais. Um dos pontos negativos é que esse tipo de rede sofre mais com a

baixa velocidade na transmissão de dados e com a instabilidade do serviço, fazendo-se necessária a criação de sistemas que mitiguem esses problemas.

3.2.2 Interação em jogos multiplayer

Basicamente são três os modelos de interação entre jogadores em ambiente multiplayer: tempo real, turnos e assíncrona.

Jogos em tempo real são aqueles onde as ações de todos os jogadores acontecem simultaneamente e de maneira contínua. Esse é um modelo mais complexo, pois a cada instante o sistema deve levar em consideração as ações de todos os jogadores, calcular as interações e informar os resultados. Isso exige que o sistema seja tolerante a falhas e tenha poder computacional para processar as jogadas sem causar atrasos. Um dos exemplos desse tipo de jogo são os jogos de tiro em primeira pessoa.

Nos jogos em turnos, cada jogador tem um tempo para fazer sua jogada e enviar a mesma para o servidor, que processa a mesma, atualiza o estado do jogo caso seja necessário e passa a vez para o próximo jogador. Esse modelo é o mesmo utilizado em jogos de tabuleiro e é um dos mais populares em dispositivos móveis, pois é algo rápido e que não exige atenção contínua.

Finalmente temos a interação assíncrona, que é mais utilizada em jogos nas redes sociais, como por exemplo, Farmville. Cada jogador tem seu espaço onde tem plena liberdade e as formas de interação com outros jogadores são

limitadas. Isso elimina a necessidade dos jogadores estarem disponíveis ao mesmo tempo, podendo cada um avançar no seu próprio ritmo.

3.2.3 Sistemas distribuídos em jogos multiplayer

Um sistema distribuído é um conjunto de computadores independentes entre si que se apresenta a seus usuários como um sistema único e coerente (TANENBAUM & VAN STEEN, 2006). Nos sistemas de jogos, pelo menos um desses computadores deve ser o responsável pelo processamento e controle do ciclo do jogo. Duas são opções comumente utilizadas nesse tipo de aplicação: sistemas ponto-a-ponto e cliente/servidor.

Nos sistemas ponto-a-ponto, cada usuário é responsável por manter seu estado de jogo e transmitir o mesmo para todos os outros jogadores, tornando desnecessária a presença de um servidor central. Esse tipo de abordagem é mais bem utilizado quando o número de jogadores é pequeno, para poder garantir a sincronia entre todos os pontos.

Um dos problemas nesse tipo de abordagem é que se um dos pontos da rede for de menor capacidade de processamento ou então estiver enfrentando problemas de rede, todos os envolvidos serão afetados, pois o andamento do jogo é determinado pelo dispositivo mais lento. Levando isso em conta e também o problema de que os dispositivos não são diretamente acessíveis através de um endereço IP, este não é o modelo mais indicado no desenvolvimento de jogos para dispositivos móveis.

Atualmente, o modelo mais utilizado no desenvolvimento de jogos multiplayer para dispositivos móveis é o de cliente/servidor. Nesse modelo, os

clientes (os celulares) enviam as informações para o servidor, que processa esses dados e atualiza o estado do jogo para então enviar o estado atual de volta para os clientes. Uma das vantagens dessa arquitetura é que, ao contrário dos sistemas ponto-a-ponto, o cliente com menor capacidade não interfere no funcionamento do jogo para todos os outros envolvidos.

Para esse trabalho, o sistema adotado será o de cliente/servidor, pois assim é possível ter um controle dos usuários (com possíveis diferentes permissões de acesso) e também não ficamos limitados às capacidades de processamento e memória dos celulares.

3.3 Frameworks

Quando falamos do desenvolvimento de sistemas, problemas semelhantes acabam tendo muitos pontos em comum quando essas soluções são feitas a partir do zero. Uma das maneiras criadas para evitar o retrabalho foi o desenvolvimento do conceito de framework, que segundo (WIRFS-BROCK, R. 1990) é "uma coleção de classes concretas e abstratas e as interfaces entre elas, e é o projeto de um subsistema".

Por exemplo, num domínio de aplicações bancárias, apesar de cada banco ter suas peculiaridades, toda a base de trabalho, como contas, transações e regulamentações do setor são exatamente as mesmas, então toda essa lógica básica é implementada pelo framework. Isso também é conhecido como *coldspot*, pois é uma área inflexível de código que não pode ser modificada posteriormente pelos usuários desse framework. Claro que

somente dessa forma não teríamos como utilizar esse framework, pois tudo já estaria definido. Toda a parte do código que pode ser alterada, ou seja, que é flexível e permite a extensibilidade do framework é chamada de *hotspot*.

Outro ponto de ligação entre o framework e a aplicação sendo desenvolvida tendo ele como base são os métodos *hook*. Esses métodos normalmente são métodos abstratos chamados pelos *coldspots* e que podem ser implementados para se alterar algum comportamento ou se fazer algum tipo de processamento em resposta a alguma ação. Por exemplo, podemos ter um framework que efetua uma chamada de rede como parte de um *coldspot*. Por estar em um coldspot, a maneira que essa chamada é efetuada não pode ser alterada, mas pode existir um método *hook* que indica se a chamada foi concluída com sucesso ou não. Cabe agora ao usuário do framework decidir se ele pretende fazer algo com a informação, então ele pode implementar o método *hook* para mostrar um aviso na tela ou então processar os dados.

Apesar de todas suas vantagens, o desenvolvimento de um framework apresenta alguns pontos que devem ser levados em consideração antes de se iniciar o projeto. Em primeiro lugar, desenvolver uma solução que atenda a um certo domínio e ao mesmo tempo seja facilmente reutilizável e de qualidade exige uma grande quantidade de esforço e tempo. Caso não seja dada a atenção necessária nessa etapa, pode se chegar à conclusão mais tarde que o framework mais complicada do que facilita o desenvolvimento de novos produtos ou que então o esforço despendido na sua construção não justifica um pequeno ganho de tempo nas novas aplicações.

Outro problema comum é que a validação de aplicações construídas a partir de um framework é extremamente complicada, pois pode se tornar muito

difícil distinguir se os bugs encontrados são do próprio framework ou se são da aplicação. Isso só fica ainda mais difícil caso o código do framework seja fechado, pois praticamente impossibilita a depuração de código.

3.3.1 Desenvolvimento de frameworks

Ao se desenvolver um framework, um dos primeiros passos é a análise de domínio das aplicações. Essa é uma das fases mais importantes do processo, pois caso seja mal executada pode fazer com que o framework desenvolvido não atenda aos requisitos. Basicamente é nessa etapa que se avalia qual é o comportamento básico que deve estar no framework, quais são suas funcionalidades, quais serão os *hotspots* e que métodos *hook* serão os responsáveis pela ligação.

Após essa análise, passa-se ao processo de modelagem, onde são definidas as classes, pacotes e interfaces, sempre tendo o cuidado de manter a generalidade e a extensibilidade do framework, para que o mesmo não atenda somente as necessidades atuais.

3.3.2 Metodologias de desenvolvimento de frameworks

Metodologias de desenvolvimento de frameworks basicamente são diferentes caminhos possíveis de se seguir ao se desenvolver um framework. São um conjunto de procedimentos e instruções já testados e comprovados que ajudam na etapa inicial e ao mesmo tempo, não são restritivas em

excesso. De acordo com (SILVA, 2000) são três as principais metodologias: Taligent, projeto dirigido por exemplo e projeto dirigido por *Hotspots*. Ainda de acordo com (SILVA, 2000), estas metodologias não detalham o processo ou técnicas de modelagem, são apenas linhas gerais do processo de desenvolvimento

A primeira (Talgient) é uma metodologia criada pela empresa de mesmo nome onde se propõe que, ao invés de um grande framework, devem ser desenvolvidos vários frameworks menores e mais simples, cada um voltado a um aspecto. A argumentação por trás dessa abordagem é que, dessa forma, podemos utilizar apenas os frameworks estritamente necessários, o que torna mais fácil o uso e aumenta o grau de reusabilidade.

Na metodologia de projeto dirigido por exemplo, proposta por (JOHNSON, R. E, 1993), são utilizados vários softwares já desenvolvidos em um mesmo domínio e todos são comparados para que sejam encontrados os pontos em comum. Ao se encontrar esses pontos, uma solução geral que atenda a todos é proposta e é testada para novos desenvolvimentos. Um dos problemas nessa metodologia é que normalmente são necessárias várias aplicações para conseguir uma solução confiável. Uma das maneiras de se contornar esse problema é a análise de um número menor de aplicações, mas o framework deve ser validado com ao menos mais dois aplicativos diferentes, desenvolvidos a partir do framework gerado.

Finalmente, temos a metodologia de projeto dirigido a *hotspots*, definida por (PREE, W. 1994). Nessa metodologia, busca-se primeiramente encontrar quais são os pontos flexíveis em comum que um domínio necessita e uma

estrutura é pensada em torno desses pontos. Após finalizar a etapa de implementação, o framework é novamente analisado para ver se mantém sua generalidade e, caso necessário, novas interações são feitas e novos *hotspots* são encontrados.

3.4 Frameworks Existentes

Atualmente existem dois frameworks que dão suporte ao desenvolvimento de jogos para Android: AndEngine e Unity.

A AndEngine é um framework desenvolvido por Nicolas Gramlich e que tem como foco os jogos para um único jogador. Ela se encaixa no conceito de jogos simples mas o framework possui um código fonte confuso e não documentado, dificultando muito a vida do desenvolvedor.

Posteriormente essa mesma engine ganhou uma extensão para jogos multiplayer, mas ela é do tipo ponto-a-ponto, que como explicado anteriormente, não será considerada uma alternativa válida. Mais recentemente todo o projeto foi abandonado, tendo sua última atualização em 2012.

A outra opção (Unity) já é famosa para o desenvolvimento de jogos para PC. Infelizmente ela veio para os dispositivos móveis dando suporte somente a jogos em três dimensões e com muitos controles diferentes. Isso significa que o desenvolvedor iniciante, sem ter base nenhuma na área já teria que começar se preocupando com coisas avançadas demais, como sistema de física, modelagem de objetos 3D e sistemas de detecção de colisão complexos (baseados nos modelos).

Uma alternativa que alguns desenvolvedores tentaram fazer foi a de fixar o eixo de rotação Z, dando uma perspectiva forçada. Isso resolveu alguns problemas mas introduziu outros, principalmente no controle de câmera, tornando a experiência do jogo muito ruim ou até mesmo não-jogável.

4. PROJETO

Baseando-se nos levantamentos e considerações feitas anteriormente, nesse capítulo serão mostrados os projetos para implementação de um framework para desenvolvimento de jogos multiplayer na plataforma Android, um pequeno framework de servidor simples, apenas para dar suporte à parte cliente da aplicação e dois jogos que utilizam esses frameworks, sendo um baseado em turnos e o outro em um modelo híbrido entre turnos e tempo real.

Toda a comunicação será feita através da internet, seja ela via 3G/GPRS ou via Wi-Fi. A arquitetura escolhida é a de cliente-servidor através do protocolo TCP, por conta de sua confiabilidade na entrega de pacotes.

4.1 Framework para jogos multiplayer Android

Um jogo é composto de vários subsistemas, cada um responsável por uma parte específica, mas que estão em comunicação constante. Por exemplo, temos sistemas responsáveis pelo controle de cenas, pelo controle do ciclo de jogo, inputs, som, IA, networking, etc.

Como o objetivo desse trabalho é fazer um framework multiplayer que atenda as necessidades mais básicas, não teremos foco em sistemas mais complexos, como de inteligência artificial e focaremos nos seguintes sistemas: ciclo de jogo, controle de eventos, input e network. Com esses sistemas, o usuário será capaz de realizar as atividades descritas na Figura 6.

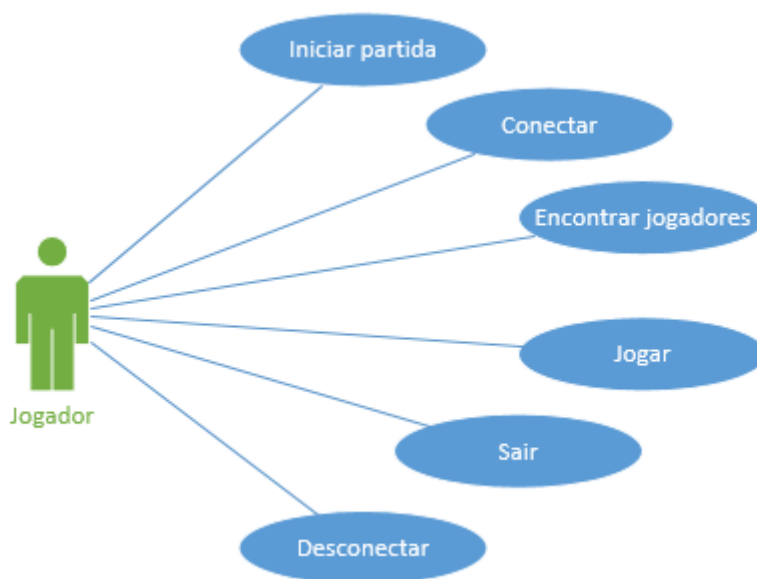


Figura 6 - Diagrama de casos de uso

Com base nos casos de uso vistos na Figura 6, foi determinado o diagrama de atividades visto na Figura 7, que mostra todos os passos desde a conexão inicial até o momento em que o usuário para de jogar.

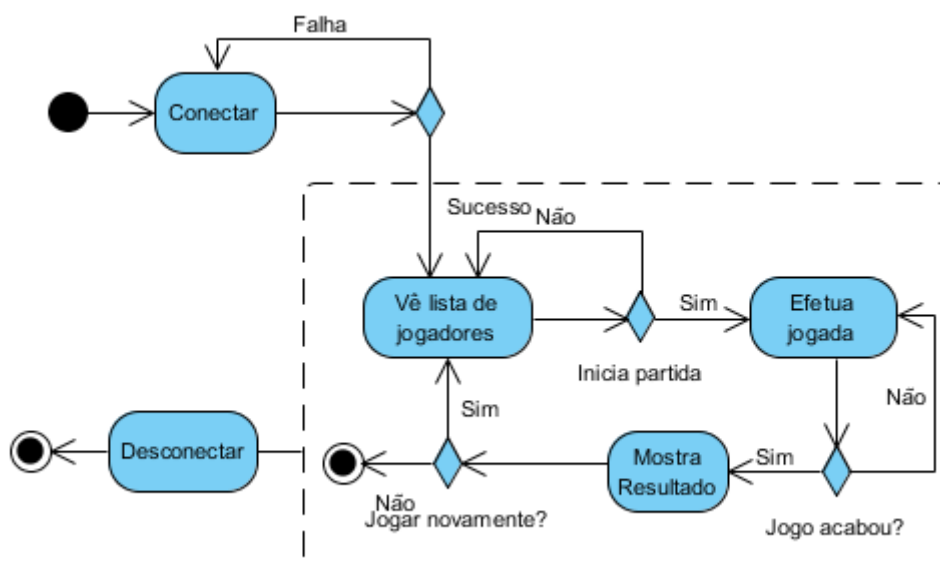


Figura 7 - Diagrama de atividades

É importante notar que em qualquer momento da atividade é possível acontecer o evento de desconectar, seja ela por iniciativa de qualquer um dos jogadores, do servidor ou até mesmo de falhas de rede fora do controle desse framework.

A partir do ponto que é iniciada a partida até o momento que a mesma é encerrada, o jogo entra em loop definido como ciclo de jogo, que será apresentado a seguir.

4.1.1 Ciclo de jogo

O ciclo de jogo é o responsável por duas das funcionalidades mais básicos em um jogo, que são atualizar o estado do jogo e desenhar os elementos na tela. De maneira resumida, baseando-se nas regras definidas ou nos inputs do usuário, um novo estado de jogo é calculado, elementos são movidos de posição e o resultado é desenhado na tela em sua nova posição. Esse ciclo pode ser visto na Figura 8 e é algo que ocorre várias vezes em um segundo ou então como resposta a algum evento, como um toque na tela, por exemplo.

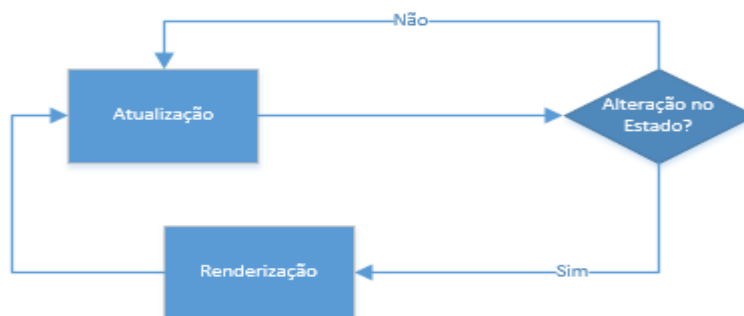


Figura 8 - Ciclo de jogo

Essas duas tarefas, a atualização do estado e o posterior desenho ou, utilizando o termo mais utilizado no meio, renderização dos elementos será mais bem detalhada a seguir.

4.1.1.1 Renderização

A etapa de renderização é a responsável por pegar todos os *assets* que estarão visíveis na tela em um determinado instante e desenhar todos em seus devidos locais. Essa etapa costuma ser uma das mais pesadas em termos de processamento e, conseqüentemente, é a maior responsável pelo consumo de bateria.

Em nosso projeto, teremos uma classe básica chamada *Renderable* (Figura 9), que deverá ser estendida por todos os elementos que serão desenhados na tela, sejam eles um botão, uma animação ou uma imagem estática. Ao herdar a classe *Renderable*, dois métodos deverão ser sobrescritos, um deles responsável pela atualização dos atributos básicos e outro que indica como o objeto deverá ser desenhado.

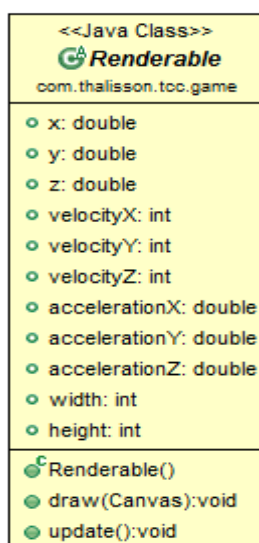


Figura 9 - Classe *Renderable*

Por questão de controle, todos os elementos possíveis de serem desenhados serão referenciados na classe *SurfaceRenderer*, que ao ter seu método de desenho invocado pela Thread que controla o ciclo de jogo (*FrameworkThread*), passará por todos os elementos invocando os seus métodos de desenho, como poderá ser visto na Figura 10. Efetivamente, o desenho de cada elemento na tela será feito pela classe *Canvas*, do próprio Android.

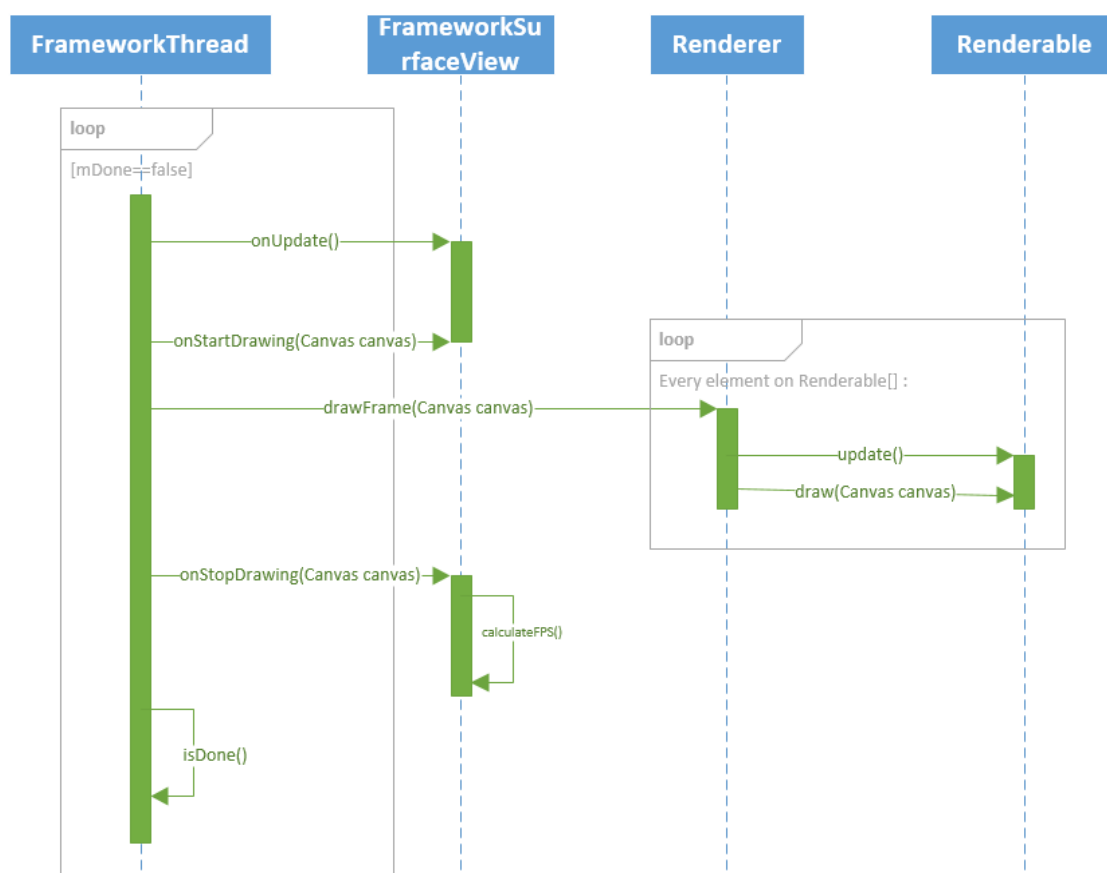


Figura 10 - Diagrama de Seqüência

4.1.1.2 Atualização

Na etapa de atualização é onde, respondendo a algum input do usuário ou mensagem recebida do servidor, os elementos afetados têm sua nova posição ou estado calculado. Por exemplo, em um jogo do estilo de tiro em

primeira pessoa, ao apertar o botão de tiro, é feito uma atualização que calculará se o tiro acertou algo e em caso positivo, o dano causado.

Esses ciclos de atualização, dependendo do estilo de jogo, podem ocorrer diversas vezes em um segundo, pra dar a impressão de fluidez necessária. Como cada atualização gera uma renderização, obtemos daí a informação de frames por segundo (FPS). Em um jogo que exige movimentação contínua, um valor mínimo de frames é necessário, por exemplo, 30 FPS. Isso quer dizer que em um segundo, ocorreram 30 atualizações e renderizações. Normalmente com valores menores que esse, o jogador passa a notar que algo não está correto. Esse problema já não acontece em jogos que não precisam dessa fluidez, como por exemplo, jogos do estilo *point and click*, onde um ciclo de atualização e renderização só é efetuado quando recebe algum input válido do usuário.

Em nosso framework, a atualização dos elementos será disparada pelo Thread que controla o ciclo de jogo e será executado pela classe *FrameworkSurfaceView*. Essa classe é uma das classes básicas do jogo e deverá ser estendida em cada jogo, pois ela é a responsável pelas tarefas mais básicas, como a criação dos elementos visuais, o gerenciamento de eventos (toques na tela, por exemplo) e a atualização do estado. O gerenciamento de eventos é uma importante etapa em um jogo e será abordado com mais detalhes no decorrer desse trabalho. Para maior clareza, os métodos relacionados ao controle de eventos não serão listados na imagem a seguir, pois tornaria a visualização confusa no momento atual.

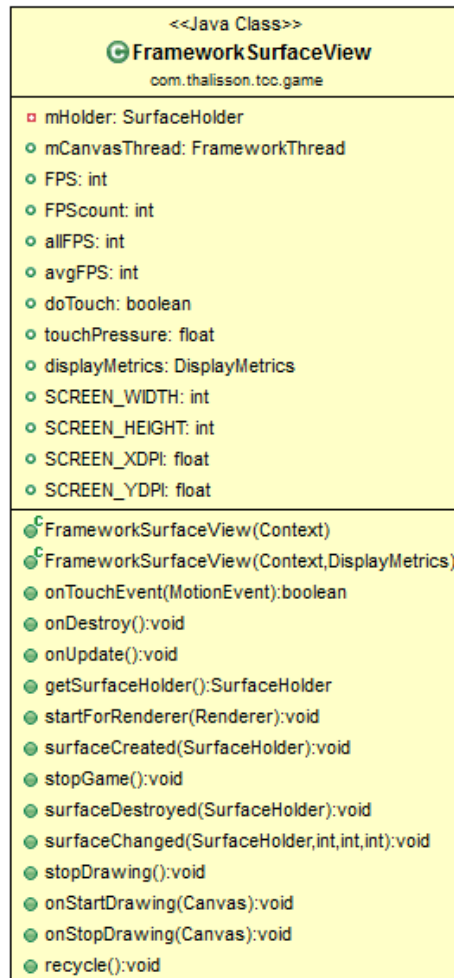


Figura 11 - Classe FrameworkSurfaceView

4.1.2 Controle de eventos e input

O controle de eventos e inputs é um sistema extremamente importante em um jogo. É através dos inputs, sejam eles botões ou toques na tela, que o estado do jogo evolui e são disparados os eventos que atualizam o estado. Na plataforma Android, todos os inputs do usuário podem ser interceptados pela classe responsável pela tela (*View*) atual e sofrerem algum tratamento ou apenas encaminhar para que o sistema se encarregue.

Como os eventos de input são muito particulares a cada cena, a classe responsável por interceptar os eventos é a classe *FrameworkSurfaceView*. Nela, para que todos os eventos (toques e botões) sejam tratados, precisamos implementar dois métodos do sistema: *onTouchEvent()* e *onKeyDown()*.

O primeiro (*onTouchEvent*) é o responsável por todos os eventos de toque e tem como parâmetro de entrada um *MotionEvent* (classe do sistema Android). Esse *MotionEvent* é o que diz as coordenadas do toque, a pressão utilizada (estimativa baseada na área de toque do dedo) e se o evento foi de um toque onde o dedo continua na tela, se é um movimento de arrastar ou se é quando o dedo sai da tela. Baseado nisso, nosso método *onTouchEvent* irá saber qual dos métodos que deverá invocar: *onTouchDown()*, *onTouchMove()* ou *onTouchUp()*, que deverão ser sobrepostos pelo usuário do framework, caso deseje efetuar operações especiais em cada um deles.

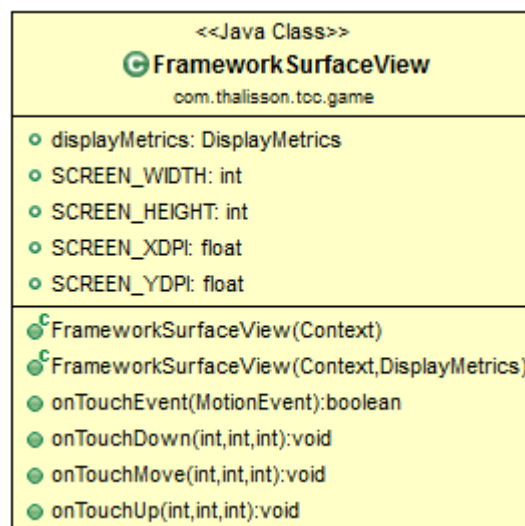


Figura 12 - Métodos para tratamento de toques na tela

O caso do *onKeyDown()* funcionará de maneira similar, determinando qual das teclas foi pressionada e disparará o evento correspondente. Como a grande maioria dos celulares com sistema Android não possui teclado físico, os eventos de teclas que serão tratados serão os de botões direcionais, do botão

home, botão de voltar, botão de busca e botão de menu. Nesses casos, teremos os métodos que podem ser vistos na Figura 13 a seguir. Novamente foram omitidos os métodos não relacionados à tarefa atual.

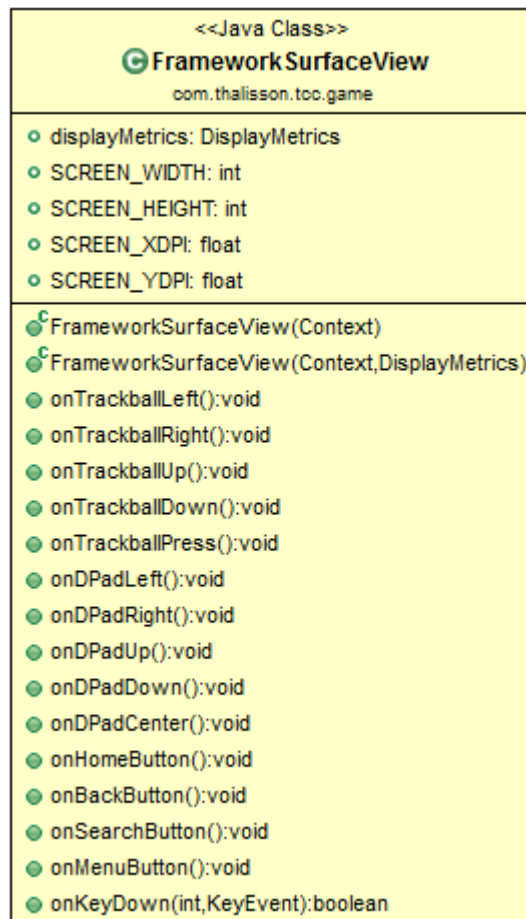


Figura 13 - Métodos para tratamento de eventos de teclado

Além do tratamento de inputs, temos alguns eventos que podem ser disparados como, por exemplo, uma mudança no tamanho da tela quando o dispositivo muda sua orientação de retrato para paisagem. Essa alteração pode ser capturada através do método de sistema `surfaceChanged()`, que em nosso framework irá disparar o método `onWindowResize()`, que deverá ser subscrito caso algum tratamento especial seja necessário.

Outros eventos possíveis já foram citados em pontos anteriores desse trabalho, em um diagrama de sequência (Figura 10) que mostrava o processo de ciclo de jogo. Esses métodos são o `onStartDrawing()` e o `onStopDrawing()`,

que são chamados, respectivamente, no início e no fim do processo de desenho da tela. Um dos possíveis usos para esses métodos é a criação de um *benchmark*, que avalia possíveis gargalos no processo de desenho.

4.1.3 Network

Para a realização da comunicação entre os clientes (smartphones) e o servidor, será necessário estabelecer uma conexão capaz de receber e enviar mensagens, seguindo um protocolo de mensagem estabelecido.

Como opções de protocolos de comunicação temos o TCP (*Transmission Control Protocol*) e o UDP (*User Datagram Protocol*), implementados pelo próprio sistema Android. O protocolo UDP não será utilizado, pois o mesmo não é confiável o suficiente, não garantindo a entrega de mensagens. Isso, em conjunto com os problemas inerentes às redes de comunicação móveis (instabilidade, por exemplo), faria com que fosse necessária a criação de novos mecanismos para lidar com esses problemas, complicando, e muito, a situação atual.

Tendo essas informações em mãos, a conexão será feita utilizando o protocolo TCP implementado na classe *Socket*, do próprio sistema Android. Todas as atividades referentes à comunicação e troca de mensagens serão concentradas na classe *Connector*, que deverá ser sobrescrita pelo usuário do framework. Os atributos e métodos da classe *Connector* podem ser vistos na Figura 144.

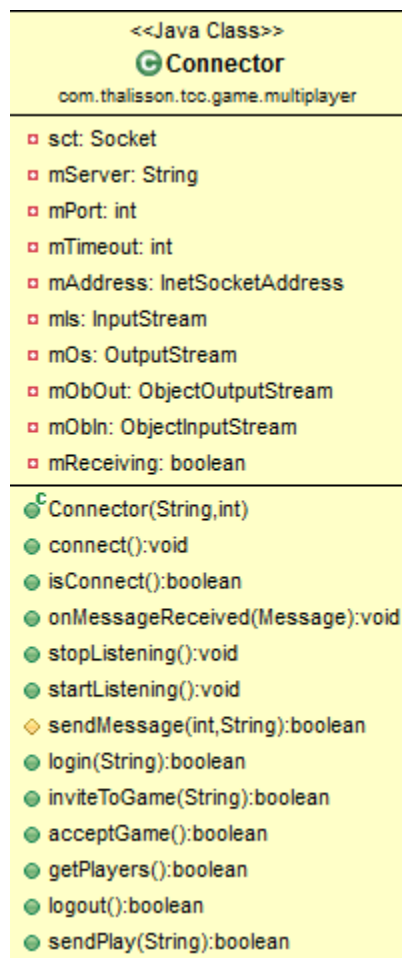


Figura 14 - Classe Conector

Para o envio de mensagens utilizando o método `sendMessage()`, será criada uma instância da classe `Message`, que além da mensagem a ser enviada, irá conter alguns atributos do tipo inteiro que identificarão o tipo de mensagem e o id do cliente. Os tipos de mensagem servem para diferenciar mensagens de login e mensagens de jogadas, por exemplo, e o id do cliente será utilizado pelo servidor para poder direcionar a mensagem para o responsável adequado.

Com relação ao recebimento de mensagens do servidor, a classe `Connector` e suas subclasses deverão ter seu método `startListening()` invocados para que a mesma comece a monitorar o `stream` de entrada de dados. Assim que uma nova mensagem for recebida, ele será transformado em

um objeto da classe *Message* e será passado como parâmetro para o método *onMessageReceived*, que deverá ser implementado pelo usuário. Todo esse processo de enviar e receber mensagens poderá ser melhor visto no diagrama da Figura 15. Importante notar que, além do método mais genérico *sendMessage()*, temos vários métodos prontos para as tarefas mais básicas, que são os seguintes: *login()*, *inviteToGame()*, *acceptGame()*, *getPlayers()*, *sendPlay()* e *logout()*. Todos esses métodos já tem seu tipo definido em constantes e caso tenham respostas, a mesma será tratada de forma básica pelo método *onMessageReceived()*.

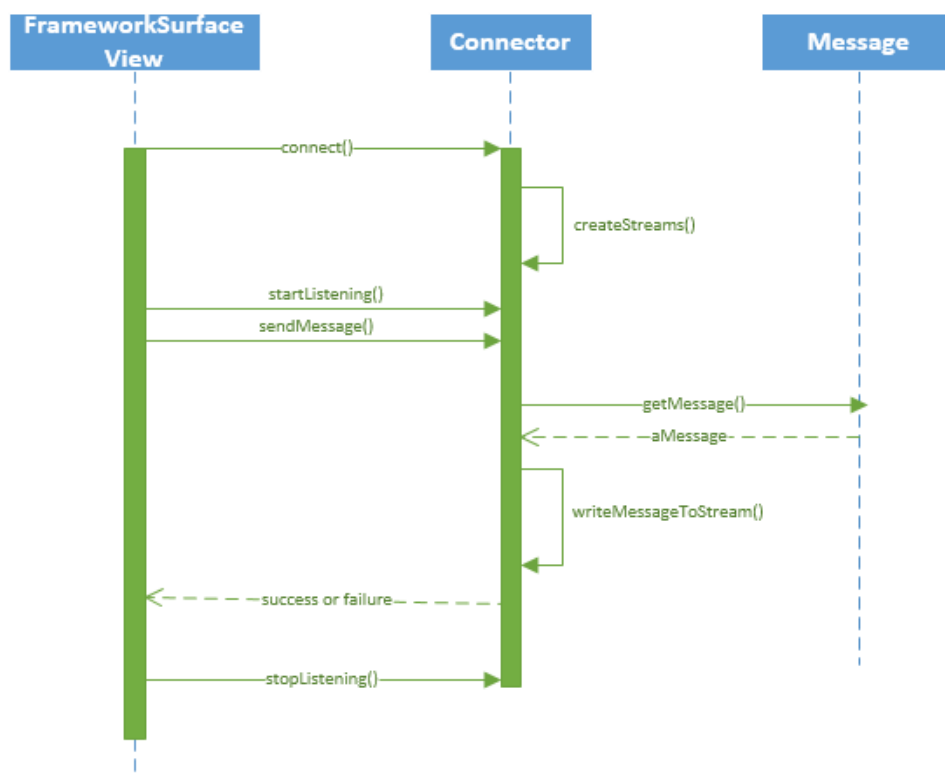


Figura 15 - Diagrama de sequência da comunicação de rede

4.1.4 Arquitetura

Levando em consideração todos os pontos levantados anteriormente, chegou-se à arquitetura no diagrama de classes vista na Figura 16.

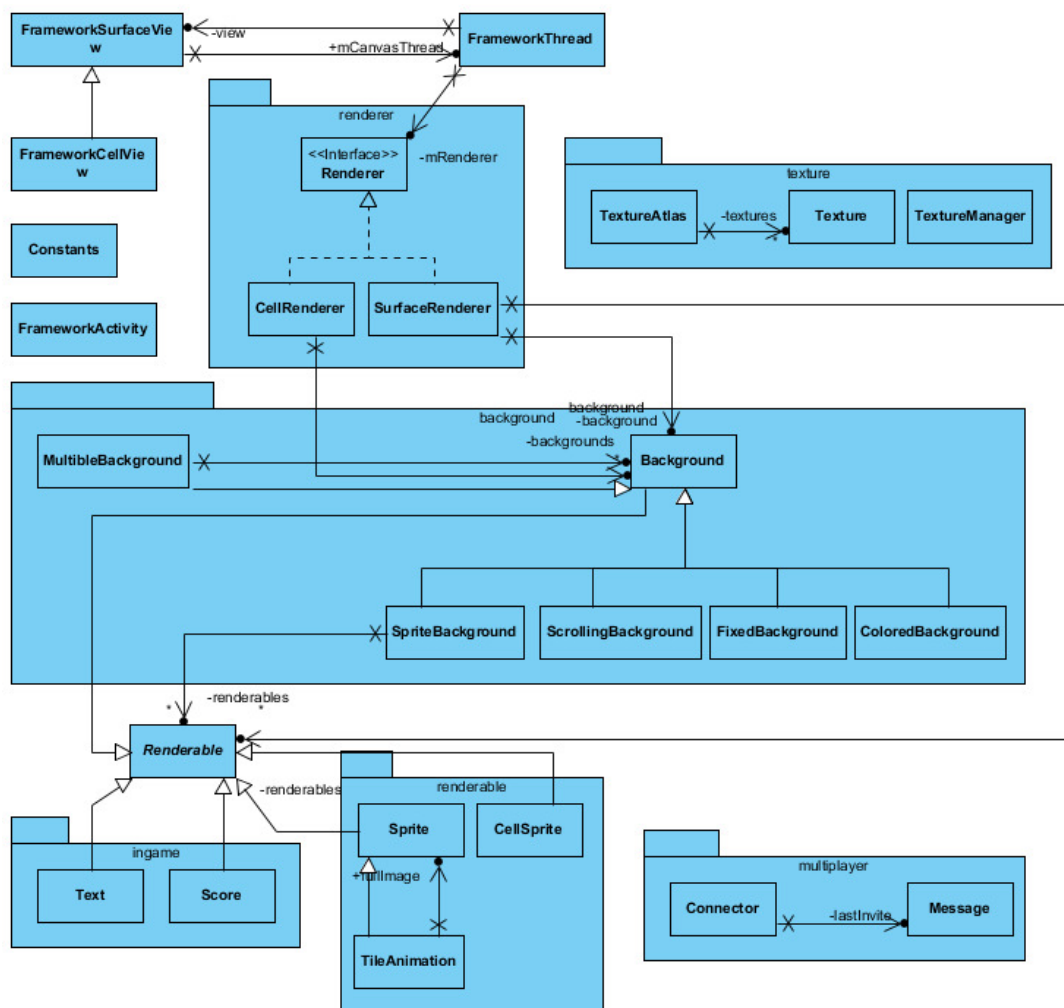


Figura 16 - Diagrama de classes

4.2 Framework de um servidor para jogos multiplayer

Como o objetivo do trabalho é focar na parte cliente, o framework para o servidor deverá ser o mais simples possível, realizando as tarefas básicas de maneira satisfatória, mas sem muitos extras. Somente para relembrar, essas tarefas básicas são a de gerenciar a entrada de novos jogadores, gerenciar uma lista de jogadores, iniciar uma partida, processar as jogadas e informar o resultado desse processamento.

Do ponto de vista do usuário do framework, a principal classe é a Game, que pode ser vista na Figura 17. Ela contém todos os métodos necessários e que devem ser sobrescritos para controlar o fluxo do jogo.

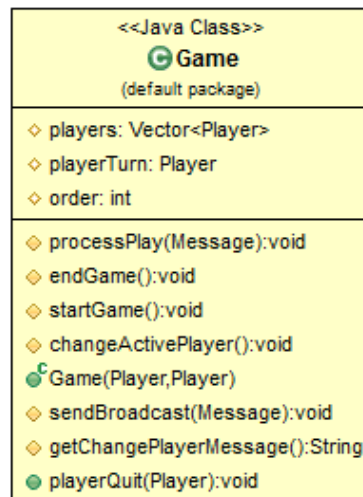


Figura 17 - Classe Game

Basicamente, ao iniciar o servidor uma porta será reservada para o socket de conexão. A cada nova conexão de usuário nessa porta, o servidor validará o usuário e irá criar um objeto da classe Player, que receberá as mensagens vindas do cliente e repassará para o responsável por tratar essa mensagem.

Finalmente, outra classe importante é a classe Servidor. Como dito anteriormente, ela é a responsável por aceitar novos jogadores, gerenciar a lista de jogadores ativos e iniciar novas partidas. Sua estrutura pode ser vista na Figura 18.



Figura 18 - Classe Server

Importante notar que a classe Message utilizada no servidor deve ser exatamente a mesma utilizada no cliente, assim sendo, caso uma delas seja estendida e novos comportamentos adicionados, a outra também deve conter esses comportamentos. O mesmo se aplica a classe de constantes.

5. IMPLEMENTAÇÃO

Para a implementação do código, foi utilizada a IDE (Integrated Development Environment) Eclipse em conjunto com a SDK (Software Development Kit) de desenvolvimento do Android em sua versão 2.3.3. Todo o código criado estará disponível em anexo, em mídia física do tipo CD e em repositório virtual acessível publicamente pelo endereço <https://github.com/thalissonrosa>.

5.1 Jogo de exemplo Pedra-Papel-Tesoura.

O jogo de pedra-papel-tesoura é um simples jogo, conhecido pela maioria das pessoas e que serve perfeitamente para ilustrar um caso de um jogo simples, entre dois jogadores e utilizando o mecanismo de turnos. O funcionamento dele seguirá o fluxo descrito no diagrama de atividade da Figura 19.

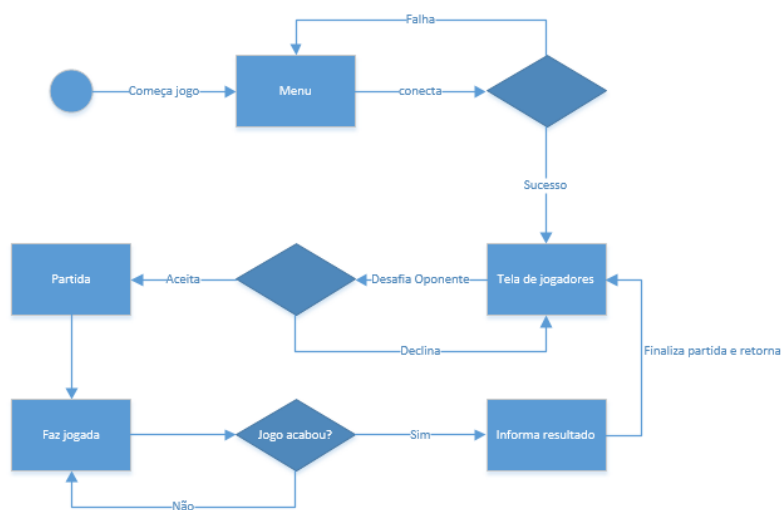


Figura 19 - Diagrama de atividade Pedra-Papel-Tesoura

Basicamente, um usuário conecta ao jogo e em caso de sucesso, requisita a lista de jogadores ao servidor. Com essa lista em mãos, escolhe um adversário e envia um convite de partida. Caso esse convite seja aceito, ambos vão para a tela de jogo, onde escolhem uma das opções (pedra, papel ou tesoura) e enviam essa informação para o servidor, que calcula o vencedor baseado nas regras da Figura 20 e informa o resultado a ambos os jogadores.



Figura 20 - Regras Pedra-Papel-Tesoura

5.1.1 Cliente

Para as tarefas mais básicas como tela inicial de login e lista de jogadores, serão utilizadas as classes do próprio Android, como a Activity. Nessas etapas iniciais, o framework será utilizado para gerenciar a conexão e fazer a troca de mensagens. Para isso, a classe Connector deve ser estendida e o método `onMessageReceived(Message messageReceived)` deverá ser sobrescrito.

Nesse método serão definidos os comportamentos esperados ao receber alguma mensagem do servidor, principalmente as referentes ao convite de partida aceito e a de jogada recebida. As outras mensagens têm comportamentos que pouco mudam de um jogo para outro e o próprio framework se encarregará delas.

Ao receber uma mensagem de convite de partida aceita, um jogo deve ser iniciado e para isso a classe `FrameworkSurfaceView` deve ser estendida pois, lembrando, ela é a responsável pelo controle de inputs. Essa nova classe deverá desenhar na tela os elementos correspondentes a cada uma das escolhas possíveis. Nessa classe, o método que irá saber quando ocorreu o clique e qual foi a opção escolhida será o `onTouchDown()`. Nele será criada uma `String` com a mensagem e o método `sendPlay()` será invocado passando essa `String` como parâmetro.

Após esse passo, o cliente irá esperar pela resposta do servidor que informará quem foi o vencedor da partida. Tendo essa resposta, o resultado é informado na tela e irá retornar para a tela de seleção de adversário, repetindo o processo para um novo jogo.

Com relação à tela de jogo, a mesma pode ser vista na Figura 21. Ali temos todos os elementos, como o nome do nosso oponente e as escolhas possíveis: pedra, papel ou tesoura. Após obter um resultado essa tela se altera levemente, mostrando na esquerda sua escolha, na direita a escolha do adversário e um texto informando se você ganhou ou perdeu.

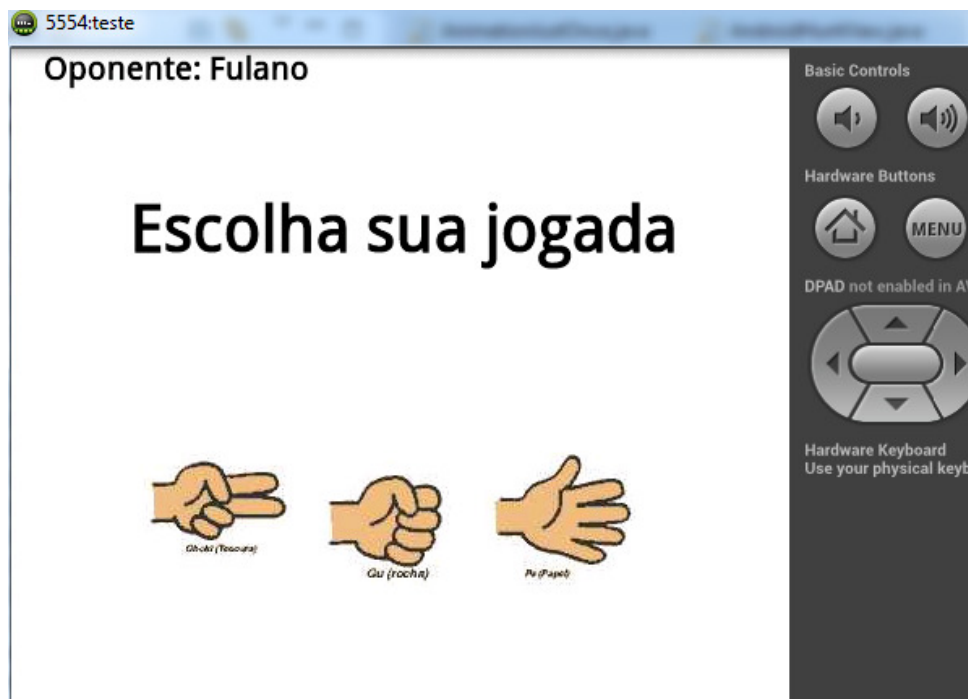


Figura 21 - Tela de jogo Pedra-Papel-Tesoura

5.1.2 Servidor

Para a criação do servidor do jogo, uma das principais classes que deve ser estendida é a classe `Game()`. Nela, nosso principal enfoque será no método `processPlay()`, que é o responsável por receber a opção escolhida de cada jogador, comparar e definir o resultado.

Como sempre é uma partida entre dois jogadores, será criada uma estrutura do tipo array com duas posições, que armazenará as escolhas. Toda vez que o método `processPlay` for invocado, a informação é armazenada e é verificado se o array está completo. Em caso negativo, irá esperar mais informações, mas em caso positivo significa que já é possível definir um vencedor.

Após enviar o resultado, o servidor irá encerrar a partida e irá reinserir os jogadores na lista de jogadores, já que os mesmos são removidos no início do jogo.

Por se tratar de um jogo simples, a maioria das funções utilizadas já estão implementadas, não sendo necessário que mais classes sejam estendidas ou até mesmo criadas. Em caso de um jogo mais complexo, teríamos que ter uma nova classe com mais constantes e pelo menos uma nova classe como mais funcionalidades para a Message.

5.2 Jogo de exemplo BugHunt

BugHunt é um jogo criado especificamente para o teste deste framework. Ele é um jogo para duas pessoas, onde o objetivo é eliminar o maior número de insetos possível em um tempo determinado. Ao iniciar o jogo, insetos irão aparecer em lugares aleatórios na tela, sempre em pares, e o jogador que clicar primeiro no inseto, ganha um ponto. Esse é um jogo de tempo real, e para isso alguns mecanismos tem que ser adotados para garantir um jogo mais justo.

Considerando as diferentes velocidades de rede, para evitar uma vantagem injusta para um dos jogadores, o tempo de clique será controlado no cliente e não no servidor. Ao desenhar na tela um dos insetos, além da posição X e Y, ele também irá armazenar o horário em que foi desenhado (em milissegundos). Ao ser clicado por um jogador, é calculada a diferença entre o horário clicado e o horário armazenado e essa informação será enviada ao servidor.

No servidor, os tempos dos dois jogadores será comparado e um ponto será atribuído ao vencedor. Um dos contrapontos é que o placar não será atualizado em tempo real e sim somente no final da partida, deixando um certo clima de incertezas. Outra decisão de design foi a de que, para tornar o jogo mais rápido, o intervalo de tempo em que os insetos aparecem vai diminuindo conforme o jogo avança. Por exemplo, se a partida for de 1 minuto e no início os insetos aparecem a cada 5 segundos, na metade da partida esse valor pode cair para 3 segundos e nos últimos 10 segundos cair ainda mais, para 1.5 segundos.

5.2.1 Cliente

Para o cliente do BugHunt, mais uma vez teremos que estender a classe `FrameworkSurfaceView`, com a diferença que desta vez o método `onUpdate` deve ser sobrescrito. Por se tratar de um jogo de tempo real, esse método será chamado várias vezes por segundo e será o responsável por atualizar o estado do jogo. Também deveremos estender a classe `Connector`, para tratar as mensagens recebidas do servidor.

Sempre que uma nova mensagem for recebida no método `onMessageReceived()` e ela for uma mensagem de jogada, essa informação será processada e armazenada em um cache de ações. Essas ações podem ser de inclusão de novos insetos na tela ou de remoção de insetos clicados pelo adversário.

No método `onUpdate()`, será verificado se existe algo no cachê de ações. Caso sejam ações de inclusão, um novo item de desenho será criado, adicionado à lista de elementos e será desenhado na tela. Em caso de

remoção, o elemento a ser removido é localizado na lista, uma animação de explosão é desenhada na tela e esse elemento é removido. Outra forma de remoção de elementos da tela é quando o usuário clicar no inseto. O processo é o mesmo descrito anteriormente mas com a diferença que essa informação não irá vir do método `onUpdate()`, e sim do método `onTouchDown()`, que irá verificar se o clique foi em algum elemento e irá chamar a ação correspondente.

Para criar a tela de jogo () foi utilizado um *ColoredBackground* na cor azul e um *SpriteBackground* contendo as nuvens, onde cada uma tem uma velocidade diferente, e o sol em posição estática. Como a classe *SurfaceRenderer* só aceita um background, foi utilizada a classe *MultipleBackground* para combinar ambos.

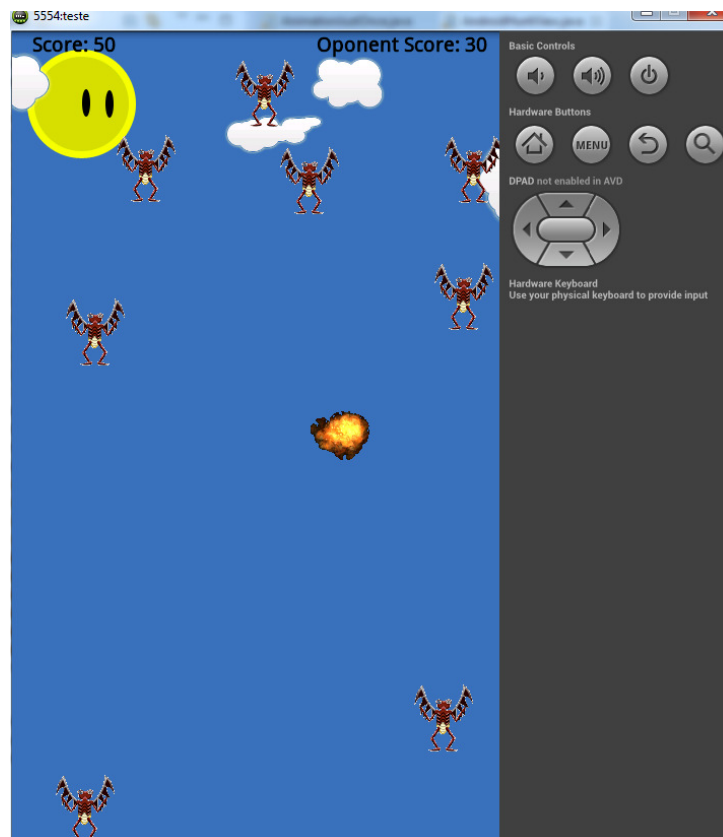


Figura 22 - Tela de jogo BugHunt

É sobre esse background que são desenhados os elementos referentes aos insetos e onde é desenhada a animação da explosão. Essa animação é feita com 9 frames que ficam se alternando rapidamente.

O processo de gerar novos insetos e inserir na tela se repete até que o tempo de jogo se encerre ou um limite de insetos seja alcançado, o que vier primeiro. A princípio o tempo de jogo é de um minuto e o número limite de insetos é de vinte, mas esses parâmetros podem ser alterados se durante a etapa de testes forem determinados melhores valores para manter o balanço e grau de dificuldade coerente. Ao finalizar a partida, é mostrado na tela o placar final e um vencedor será declarado, sendo que após isso o jogo retorna para a tela de seleção de adversários e o processo se repete.

5.2.2 Servidor

Em relação ao servidor, o controle básico de conexões (aceitar novos jogadores e informar lista de conectados, por exemplo) é igual ao relatado anteriormente no jogo Pedra-Papel-Tesoura. As maiores diferenças estarão no fato de que ao estender a classe Game, uma maior atenção foi dedicada ao método startGame(), pois esse método é o responsável pelo ciclo de jogo.

Ao ser invocado, esse método inicia um loop onde são escolhidas as posições X e Y para dois insetos e, utilizando uma extensão da classe Message, essa informação será enviada para os jogadores utilizando o método sendBroadcast(). Para controle de elementos ativos, uma classe Bug será criada, que conterà um ID de ordem, o estado do inseto (vivo ou morto) e os dois tempos dos jogadores. Neste ponto os tempos estarão zerados mas serão

preenchidos mais tarde. Essa instância da classe será armazenada em uma HashTable, tendo como chave o seu ID, tornando mais fácil sua posterior recuperação.

Toda vez que uma nova mensagem é recebida, ela conterá informações de insetos que foram destruídos pelo jogador em questão juntamente com o ID do inseto. Esse inseto será localizado na lista e terá o tempo do jogador atualizado. Além disso, essa informação é enviada para o outro jogador, para caso do inseto ainda estar na tela, que ele seja destruído. Esse processo se repetirá até o final da partida em questão, quando o tempo ou o limiar de insetos seja atingido.

Para calcular o placar final, a lista de insetos será percorrida e pra cada inseto será invocado o método `whoWins()` da classe `Bug`, que indica quem fez o ponto. Caso os dois jogadores tenham um tempo registrado, o menor tempo vence. Caso contrário, vence o jogador que tiver registrado um tempo. Mais uma vez, o resultado é enviado para ambos os jogadores e a sessão de jogo é encerrada.

6. CONCLUSÃO

O objetivo do trabalho era o desenvolvimento de um framework para jogos multiplayer na plataforma Android. Com base nas análises efetuadas, o modelo escolhido para tratar a conexão de rede foi o de cliente-servidor, o que gerou a necessidade do desenvolvimento de um módulo a parte que ficará hospedado em servidor remoto.

O framework resultante do estudo efetuado foca em retirar do usuário a preocupação com aspectos mais técnicos, como o controle da comunicação entre os jogadores e como é desenhada uma imagem ou animação na tela. Dessa forma, a maior preocupação do usuário do framework é com a lógica do seu jogo, com as regras que determinam como o jogo acontece. Por questão de otimização de bateria do dispositivo e diminuição no tráfego de rede gerado, essas regras de jogo ficam no lado do servidor. Isso também facilita a atualização do jogo ou até mesmo a correção de problemas não encontrados durante a etapa de testes.

Conforme pode ser visto nos dois jogos gerados utilizando o framework, os objetivos determinados no início do projeto foram atingidos, pois o framework é de fácil utilização e poucas são as classes que precisaram ser importadas no desenvolvimento dos jogos de prova.

Como trabalhos futuros, fica a necessidade da implementação de outra opção no controle de conexão, mais adaptada ao cenário de jogos de tempo real. O protocolo escolhido para a implementação atual (TCP/IP) não é o mais adequado para esse tipo de jogos, sendo necessário o uso do protocolo UDP. Outra possível trabalho é a melhoria do sistema de animações, permitindo mais

frames, aumentando a fluidez e complexidade da mesma. Finalmente, uma última melhoria seria a criação de um editor de mapas para ser utilizado em jogos do tipo plataforma ou *sidescrolling*.

7. Referências Bibliográficas

- ANATEL. Linhas ativas. 2014. Disponível em: <<http://www.teleco.com.br/ncel.asp>>. Acesso em: 09 Nov. 2014.
- ANDROVICH, Mark. GTA IV: Most Expensive game ever developed?. Games Industry. 2008. Disponível em: <<http://www.gamesindustry.biz/articles/gta-iv-most-expensive-game-ever-developed>>. Acesso em: 09 Nov. 2014.
- BISHOP, Todd. Xbox 360 vs. Wii vs. PS3: Who won the console wars?. 2013. Disponível em: <<http://www.geekwire.com/2013/xbox-360-wii-ps3-won-console-generation>>. Acesso em: 07 Nov. 2014.
- BRUSTEIN, Joshua. Grand Theft Auto V is the Most Expensive Game Ever - and It's Almost Obsolete. Business Week. 2013. Disponível em: <<http://www.businessweek.com/articles/2013-09-18/grand-theft-auto-v-is-the-most-expensive-game-ever-and-it-s-almost-obsolete>>. Acesso em: Nov. 2013.
- CNN. iPhone sales said to hit half-million. 2007. Disponível em: <http://money.cnn.com/2007/07/02/technology/iphone_sales/>. Acesso em: 15 de Ago. de 2014
- FUTTER, Mike. Grand Theft Auto V Hits \$1 Billion in Three Days. Game Informer. 2013. Disponível em: <<http://www.gameinformer.com/b/news/archive/2013/09/20/grand-theft-auto-v-hits-1-billion-in-three-days.aspx>>. Acesso em: Nov. 2013.
- GARTNER. Gartner says Android to Become No. 2 Worldwide Mobile Operating System in 2010 and Challenge Symbian for No. 1 Position in 2014. 2010. Disponível em: <<http://www.gartner.com/it/page.jsp?id=1434613>>. Acesso em: Out. 2013.
- GSMHISTORY. Vintage Mobiles. 2014. Disponível em: <http://www.gsmhistory.com/vintage-mobiles/#hagenuk_mt2000_1994>. Acesso em: Nov. 2014.
- GSMWORLD. Market Data Summary. Disponível em: <http://www.gsmworld.com/newsroom/market-data/market_data_summary.htm>. Acesso em: 08 de Nov. de 2014
- HUIZINGA, Johan. Homo ludens; o jogo como elemento da cultura. 2 ed. São Paulo: perspectiva, 1980.

- JOHNSON, R. E. How to design frameworks. 1993. Disponível em: <http://www.dsc.ufcg.edu.br/~jacques/cursos/map/recursos/how-design-frame.pdf>. Acesso em Nov. 2014.
- LIPSON, Ashley S. & BRAIN, Robert D. Computer and Video Game Law: Cases, Statutes, Forms, Problems & Materials. Carolina Academic Press, 2009.
- LOMAS, Natasha. BBM Now At 85M Monthly Active Users, 113M Registered Users, 500,000 Channels. Techcrunch. 2014. Disponível em: <<http://techcrunch.com/2014/03/28/bbm-usage/>>. Acesso em: 15 de Ago. de 2014.
- NIELSEN. Android most popular operating system in U.S. among recent smartphone buyers. 2010. Disponível em: <<http://www.nielsen.com/us/en/insights/news/2010/android-most-popular-operating-system-in-u-s-among-recent-smartphone-buyers.html>>. Acesso em: Out. 2013.
- NOKIA. 7 Nokia world records that will blow your mind!. 2011. Disponível em: <<http://lumiaconversations.microsoft.com/2011/02/15/7-nokia-world-records-that-will-blow-your-mind/>>. Acesso em: Nov. 2014
- PREE. W. Design Patterns for Object-Oriented Software Development. Addison-Wesley, Reading, Mass. 1994.
- SILVA, R. P. Suporte ao desenvolvimento e uso de frameworks e componentes. 262f. 2000. Tese (Doutorado em Ciência da Computação) – Universidade Federal do Rio Grande do Sul, Porto Alegre.
- STATISTA. Número de jogadores. 2014. Disponível em: <<http://www.statista.com/statistics/276601/number-of-world-of-warcraft-subscribers-by-quarter>>. Acesso em: 07 Nov. 2014.
- TANENBAUM A.;VAN STEEN M. Distributed Systems: Principles and Paradigms. Pearson Prentice Hall, 2007.
- WIRFS-BROCK, R.; JOHNSON, R. E. Surveying current research in object-oriented design. Communications of the ACM, New York, 1990.

ANEXO A - CÓDIGO FONTE

Cliente

Constants

```
package com.thalisson.tcc.game;

public class Constants {

    public static final String LOGTAG = "TCC_FRAMEWORK";

    public static final int BYTE_BUFFER_SIZE = 256;

    public static final int TYPE_LOGIN = 1;
    public static final int TYPE_INVITE = 2;
    public static final int TYPE_ACCEPT = 3;
    public static final int TYPE_GETPLAYERS = 4;
    public static final int TYPE_LOGOUT = 5;
    public static final int TYPE_PLAY_SEND = 6;
    public static final int TYPE_PLAY_RECEIVED = 7;
    public static final int TYPE_GAME_START = 11;

    public static final String JOINREQUEST = "<JOIN>";
    public static final String JOINACCEPTED = "<JOINACCEPTED>";
    public static final String LEAVESOCKET = "<LEAVEGoup>";

}
```

FrameworkActivity

```
package com.thalisson.tcc.game;

import android.app.Activity;
import android.media.AudioManager;
import android.os.Bundle;
import android.util.DisplayMetrics;
import android.util.Log;
import android.view.Window;
import android.view.WindowManager;

/**
 * Atividade basica, podendo setar o controlador de volume e fullscreen.
 * @author Thalisson
 */
public class FrameworkActivity extends Activity{

    private DisplayMetrics dm;

    public boolean isFullscreen = true;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        dm = new DisplayMetrics();
        getWindowManager().getDefaultDisplay().getMetrics(dm);

        Log.d(Constants.LOGTAG, "Atividade criada: (" + dm.widthPixels + "x" + dm.heightPixels + ")");

        this.setVolumeControlStream(AudioManager.STREAM_MUSIC);

        super.onCreate(savedInstanceState);
    }

    /**
     * Alternar a atividade entre fullscreen ou nao
     * @param fullscreen true se deve se tornar fullscreen
     */
    public void setFullscreen(boolean fullscreen){
        if(fullscreen){

            this.getWindow().addFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN);

            this.getWindow().clearFlags(WindowManager.LayoutParams.FLAG_FORCE_NOT_FULLSCREEN);
        }
    }
}
```

```

        this.getWindow().requestFeature(Window.FEATURE_NO_TITLE);
    }else{

this.getWindow().addFlags(WindowManager.LayoutParams.FLAG_FORCE_NOT_FULLSCREEN)
;

this.getWindow().clearFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN);

this.getWindow().requestFeature(Window.FEATURE_CUSTOM_TITLE);
    }
    isFullscreen = fullscreen;
}

/**
 * Metodo para obter o DisplayMetrics da tela utilizada
 * @return Retorna uma instancia de DisplayMetrics.
 */
public DisplayMetrics getDisplayMetrics(){
    return dm;
}

@Override
protected void onDestroy() {
    try {
        finalize();
    } catch (Throwable e) {}
    super.onDestroy();
}
}

```

FrameworkCellView

```

package com.thalisson.tcc.game;

import java.util.Map;

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.Bitmap.Config;
import android.graphics.Canvas;
import android.util.DisplayMetrics;

import com.thalisson.tcc.game.texture.Texture;

/**
 * FrameworkCellView eh utilizado em jogos do tipo sideScroller.
 */

public class FrameworkCellView extends FrameworkSurfaceView{

    /**
     * @author Thalisson
     */

    private Map<Character,Texture> interpreter = null;
    private String[] fullCells = null;
    private int cellWidth;
    private int cellHeight;
    private Bitmap map;

    public boolean allowStart = false;

    public FrameworkCellView(Context context) {
        super(context);
    }

    /**
     * @param context Contexto da aplicacao
     * @param dm Tela especifica por DisplayMetrics. Usar getDisplayMetrics(); da
FrameworkActivity
     */
    public FrameworkCellView(Context context, DisplayMetrics dm) {
        super(context,dm);
    }

    /**
     * Metodo que deve ser chamado antes de iniciar a thread renderizadora

```

```

        * @param interpreter Mapa que contem caracteres e texturas. Os caracteres
        serao substituidos pelas texturas
        * correspondentes no metodo fullCells[].
        * @param fullCells Uma String[] que contem todos os caracteres que serao
        substituidos pelas texturas especificas
        * @param cellWidth Largura da imagem
        * @param cellHeight Altura da imagem
        */
        public void set(Map<Character,Texture> interpreter, String[] fullCells, int
        cellWidth, int cellHeight){
            this.interpreter = interpreter;
            this.fullCells = fullCells;
            this.cellWidth = cellWidth;
            this.cellHeight = cellHeight;
            allowStart = true;
            map = Bitmap.createBitmap(fullCells[0].length()*cellWidth,
            fullCells.length*cellHeight, Config.ARGB_8888);
            init();
        }

        private void init(){
            Canvas c = new Canvas(map);
            if(allowStart){
                int dx = 0;
                int dy = 0;

                for(int i = 0; i < fullCells.length; i++){
                    dx=0;
                    for(int j = 0; j < fullCells[i].length(); j++){

                        Bitmap toDraw =
                        interpreter.get(fullCells[i].charAt(j)).tex;
                        c.drawBitmap(toDraw, dx, dy, null);

                        dx+=cellWidth;
                    }
                    dy+=cellHeight;
                }
            }

            public Texture getMap(){
                return new Texture(map);
            }
        }
    }
}

```

FrameworkSurfaceView

```

package com.thalisson.tcc.game;

import android.content.Context;
import android.graphics.Canvas;
import android.util.DisplayMetrics;
import android.view.KeyEvent;
import android.view.MotionEvent;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

import com.thalisson.tcc.game.renderer.Renderer;

/**
 * FrameworSurfaceView eh a View basica para um jogo.
 * Uma nova classe deve ser criada e estender FrameworkSurfaceView.
 */

public class FrameworkSurfaceView extends SurfaceView implements SurfaceHolder.Callback{

    /**
     * @author Thalisson
     */

    private SurfaceHolder mHolder;
    public FrameworkThread mCanvasThread;
    public int FPS,FPScount,allFPS,avgFPS = 0;

    //Touchscreen
}

```



```

public boolean doTouch = false;
public float touchPressure = 0;

//Display
public DisplayMetrics displayMetrics = null;
public int SCREEN_WIDTH = 0;
public int SCREEN_HEIGHT = 0;
public float SCREEN_XDPI = 0;
public float SCREEN_YDPI = 0;

/**
 * @param context Contexto da aplicacao
 */
public FrameworkSurfaceView(Context context) {
    super(context);

    setFocusable(true);
    mHolder = getHolder();
    mHolder.addCallback(this);
    mHolder.setType(SurfaceHolder.SURFACE_TYPE_NORMAL);

    displayMetrics = new DisplayMetrics();
}

/**
 * @param context Contexto da aplicacao
 * @param dm Tela especifica por DisplayMetrics. Usar getDisplayMetrics(); da
FrameworkActivity
 */
public FrameworkSurfaceView(Context context, DisplayMetrics dm) {
    super(context);

    setFocusable(true);
    mHolder = getHolder();
    mHolder.addCallback(this);
    mHolder.setType(SurfaceHolder.SURFACE_TYPE_NORMAL);

    this.displayMetrics = dm;
    SCREEN_HEIGHT = displayMetrics.heightPixels;
    SCREEN_WIDTH = displayMetrics.widthPixels;
    SCREEN_XDPI = displayMetrics.xdpi;
    SCREEN_YDPI = displayMetrics.ydpi;
}

@Override
public boolean onTouchEvent(MotionEvent event) {
    int x = (int) event.getX();
    int y = (int) event.getY();
    int pressure = (int) (event.getPressure()*100);

    if(event.getAction() == MotionEvent.ACTION_DOWN){
        doTouch = true;
        onTouchDown(x,y,pressure);
        return true;
    }else if(event.getAction() == MotionEvent.ACTION_MOVE){
        onTouchMove(x,y,pressure);
    }else if(event.getAction() == MotionEvent.ACTION_UP){
        doTouch = false;
        onTouchUp(x,y,pressure);
        return super.onTouchEvent(event);
    }
    return super.onTouchEvent(event);
}

/**
 * onTouchDown Chamado quando ha um toque na tela.
 * @param touchX Coordenada X
 * @param touchY Coordenada Y
 * @param pressure tamanho do toque
 */
public void onTouchDown(int touchX, int touchY, int pressure){}

```

```

/**
 *
 * onTouchMove Chamado quando o usuario move o dedo pela tela.
 * @param touchX Coordenada X
 * @param touchY Coordenada Y
 * @param pressure tamanho do toque
 */
public void onTouchMove(int touchX, int touchY, int pressure){}

/**
 * onTouchUp get Chamado quando o usuario retira o dedo da tela.
 * @param @param touchX Coordenada X
 * @param touchY Coordenada Y
 * @param pressure tamanho do toque
 */
public void onTouchUp(int touchX, int touchY, int pressure){}

@Override
public boolean onTrackballEvent(MotionEvent event) {

    return super.onTrackballEvent(event);
}

/**
 * onTrackballLeft chamado quando o usuario move o trackball para a esquerda.
 */
public void onTrackballLeft(){}
/**
 * onTrackballRight chamado quando o usuario move o trackball para a direita
 */
public void onTrackballRight(){}
/**
 * onTrackballUp chamado quando o usuario move o trackball para cima
 */
public void onTrackballUp(){}
/**
 * onTrackballDown chamado quando o usuario move o trackball para baixo
 */
public void onTrackballDown(){}
/**
 * onTrackballPress chamado quando o usuario pressiona o trackball
 */
public void onTrackballPress(){}

/**
 * onDpadLeft chamado quando o usuario aperta o botao para a esquerda no D-Pad
 */
public void onDpadLeft(){}

/**
 * onDpadRightchamado quando o usuario aperta o botao para a direita no D-Pad
 */
public void onDpadRight(){}

/**
 * onDpadUp chamado quando o usuario aperta o botao para cima no D-Pad
 */
public void onDpadUp(){}

/**
 * onDpadDown chamado quando o usuario aperta o botao para a baixo no D-Pad
 */
public void onDpadDown(){}

/**
 * onDpadCenter chamado quando o usuario aperta o botao central do D-Pad
 */
public void onDpadCenter(){}

/**
 * onHomeButton chamado quando o usuario aperta o botao home
 */
public void onHomeButton(){}

/**
 * onBackButton chamado quando o usuario aperta o botao de voltar
 */

```

```

public void onBackPressed(){}

/**
 * onSearchButton chamado quando o usuario aperta o botao de busca
 */
public void onSearchButton(){}

/**
 * onMenuButton chamado quando o usuario a parte o botao de menu
 */
public void onMenuButton(){}

@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {

    switch (keyCode){
    case KeyEvent.KEYCODE_DPAD_UP:
        onDPadUp();
        onTrackballUp();
        break;
    case KeyEvent.KEYCODE_DPAD_DOWN:
        onDPadDown();
        onTrackballDown();
        break;
    case KeyEvent.KEYCODE_DPAD_LEFT:
        onDPadLeft();
        onTrackballLeft();
        break;
    case KeyEvent.KEYCODE_DPAD_RIGHT:
        onDPadRight();
        onTrackballRight();
        break;
    case KeyEvent.KEYCODE_DPAD_CENTER:
        onDPadCenter();
        onTrackballPress();
        break;
    case KeyEvent.KEYCODE_HOME:
        onHomeButton();
        break;
    case KeyEvent.KEYCODE_BACK:
        onBackPressed();
        break;
    case KeyEvent.KEYCODE_SEARCH:
        onSearchButton();
        break;
    case KeyEvent.KEYCODE_MENU:
        onMenuButton();
        break;
    }
    return super.onKeyDown(keyCode, event);
}

public void onDestroy(){

}

/**
 * Local onde eh feito o update no estado do jogo.
 * Aqui voce calcula e move tudo que for necessario.
 */
public void onUpdate(){}

/**
 * Retorna o SurfaceHolder onde o Renderer desenha
 * @return SurfaceHolder
 */
public SurfaceHolder getSurfaceHolder() {
    return mHolder;
}

/**
 * Defina o Renderer especificado pelo usuario e inicia a thead renderizadora.

```

```

        * @param renderer Objeto que estende a classe basica Renderer. Ex: Sprite,
TileAnimation.
        */
        public void startForRenderer(Renderer renderer) {

                mCanvasThread = new FrameworkThread(this, renderer);
                mCanvasThread.start();

        }

        @Override
        public void surfaceCreated(SurfaceHolder holder) {
                if(mCanvasThread != null){
                        mCanvasThread.surfaceCreated();
                }
        }

        public void stopGame(){
                mCanvasThread.requestExitAndWait();
        }

        @Override
        public void surfaceDestroyed(SurfaceHolder holder) {
                onDestroy();
                mCanvasThread.requestExitAndWait();
                mCanvasThread.surfaceDestroyed();
                mHolder.removeCallback(this);
        }

        @Override
        public void surfaceChanged(SurfaceHolder holder, int format, int w, int h) {
                onWindowResize(w, h);
        }

        /**
        * Chamado caso ocorra rotacao da tela
        * @param width Novo comprimento da tela
        * @param height Nova altura da tela
        */
        public void onWindowResize(int width, int height){}

        @Override
        public void onWindowFocusChanged(boolean hasFocus) {
                super.onWindowFocusChanged(hasFocus);
        }

        public void setSecondRunnable(Runnable r) {
                mCanvasThread.setSecondRunnable(r);
        }

        public void clearSecondRunnable() {
                mCanvasThread.clearSecondRunnable();
        }

        @Override
        protected void onDetachedFromWindow() {
                super.onDetachedFromWindow();
                mCanvasThread.requestExitAndWait();
        }

        public void stopDrawing() {
                mCanvasThread.requestExitAndWait();
        }

        /**
        * Chamado quando a View inicia a desenhar
        * Esse metodo deve ser sobrescrevido quando se quiser fazer alguma tarefa
antes do draw.
        */
        public void onStartDrawing(Canvas canvas){}

        /**
        * Chamado quando a View finaliza o desenho
        * Esse metodo deve ser sobrescrevido quando se quiser fazer alguma tarefa
depois do draw.

```

```

    */
    public void onStopDrawing(Canvas canvas){}

    /**
     * Override nesse metodo e recicle todas as texturas e sprites nele. Economia
de memoria.
     */
    public void recycle(){}
}

```

FrameworkThread

```

package com.thalisson.tcc.game;

import android.graphics.Canvas;
import android.util.Log;
import android.view.SurfaceHolder;

import com.thalisson.tcc.game.renderer.Renderer;

public class FrameworkThread extends Thread {

    /**
     * @author Thalisson
     */

    private boolean mDone;

    private Renderer mRenderer;
    private Runnable mEvent;
    private SurfaceHolder mSurfaceHolder = null;
    private FrameworkSurfaceView view;
    private Canvas canvas;

    public boolean hasSurface = false;

    /**
     *
     * @param view A View na qual sera desenhada.
     * @param renderer O Renderer responsavel por desenhar (draw).
     */
    public FrameworkThread(FrameworkSurfaceView view, Renderer renderer) {
        super();
        mDone = false;

        mRenderer = renderer;
        this.view = view;
        mSurfaceHolder = view.getHolder();
        setName("CandroidThread");
    }

    @Override
    public void run() {

        /*
         * Atividade principal da thread. Fica em loop ate que seja
solicitada sua parada
         */
        while (!mDone) {

            synchronized (this) {
                if(!hasSurface) {
                    while (!hasSurface) {
                        try {
                            wait();
                        } catch (InterruptedException
e) {}
                    }
                }

                if (mDone) {
                    break;
                }
            }

            if (mEvent != null) {

```

```

        mEvent.run();
    }

    synchronized(mSurfaceHolder){

        canvas = mSurfaceHolder.lockCanvas();

        if (canvas != null) {
            view.onUpdate();
            view.onStartDrawing(canvas);
            mRenderer.drawFrame(canvas);
            view.onStopDrawing(canvas);

mSurfaceHolder.unlockCanvasAndPost(canvas);

                }else{
                    Log.e(Constants.LOGTAG, "Canvas are
null");
                }
            }
        }

    public void requestExitAndWait() {
        synchronized(this) {
            mDone = true;
            notify();
        }
        try {
            join();
        } catch (InterruptedException ex) {
            Thread.currentThread().interrupt();
        }
    }

    public void setSecondRunnable(Runnable r) {
        synchronized(this) {
            mEvent = r;
        }
    }

    public void clearSecondRunnable() {
        synchronized(this) {
            mEvent = null;
        }
    }

    /**
     * Chamado quando a surface eh criada
     */
    public void surfaceCreated() {
        synchronized(this) {
            hasSurface = true;
            notify();
        }
    }

    /**
     * Chamado quando a Surface eh destruida
     */
    public void surfaceDestroyed() {
        synchronized(this) {
            mDone = false;
            hasSurface = false;
            notify();
        }
    }
}

```

Renderable

```
package com.thalisson.tcc.game;
```

```

import android.graphics.Canvas;

/**
 * Classe base de tudo que eh possevel ser desenhado por um renderer
 */

public abstract class Renderable {

    /**
     * @author Thalisson
     */

    // Posicao
    public double x;
    public double y;
    public double z;

    // Velocidade.
    public int velocityX;
    public int velocityY;
    public int velocityZ;

    // Aceleracao
    public double accelerationX = 1.0;
    public double accelerationY = 1.0;
    public double accelerationZ = 1.0;

    // Tamanho
    public int width;
    public int height;

    public void draw(Canvas canvas){}

    public void update(){
        x+=velocityX;
        y+=velocityY;
        z+=velocityZ;

        velocityX*=accelerationX;
        velocityY*=accelerationY;
        velocityZ*=accelerationZ;
    }
}

```

Background

```

package com.thalisson.tcc.game.background;

import android.graphics.Canvas;

import com.thalisson.tcc.game.Renderable;

/**
 * Classe base pra qualquer background
 */
public class Background extends Renderable{

    /**
     * @author Thalisson
     */

    public Background() {

    }

    /**
     * Chamado pelo Renderer e contem tudo que deve ser desenhado (drawn)
     */
    @Override
    public void draw(Canvas canvas) {

        super.draw(canvas);
    }

    /**
     * Chamado pelo Renderer e contem tudo que deve ser atualizado (update).
     */
    public void update(){}
}

```

```

}

ColoredBackground
package com.thalisson.tcc.game.background;

import android.graphics.Canvas;

public class ColoredBackground extends Background{

    /**
     * @author Thalisson
     */

    private int color;

    /**
     * @param color A cor que sera o background
     */
    public ColoredBackground(int color) {
        this.color = color;
    }

    /**
     * Muda a cor do background
     * @param color
     */
    public void changeColor(int color){
        this.color = color;
    }

    @Override
    public void draw(Canvas canvas) {
        canvas.drawColor(color);
        super.draw(canvas);
    }
}

```

```

FixedBackground
package com.thalisson.tcc.game.background;

import android.graphics.Bitmap;
import android.graphics.Canvas;

import com.thalisson.tcc.game.texture.Texture;

public class FixedBackground extends Background{

    /**
     * @author Thalisson
     */

    private Bitmap mBackground;

    public FixedBackground(Texture tex) {
        mBackground = tex.tex;
        height = tex.tex.getHeight();
        width = tex.tex.getWidth();
    }

    @Override
    public void draw(Canvas canvas) {
        canvas.drawBitmap(mBackground, (int)x, (int)y, null);
        super.draw(canvas);
    }

    public void setXY(int x, int y){
        this.x = x;
        this.y = y;
    }

}

```

```

MultipleBackground
package com.thalisson.tcc.game.background;

import android.graphics.Canvas;

```



```

public class MultibleBackground extends Background{

    /**
     * @author Thalisson
     */

    private Background[] backgrounds;

    public MultibleBackground(Background[] bgs) {
        backgrounds = bgs;
    }

    @Override
    public void draw(Canvas canvas) {
        for(Background bg : backgrounds){
            bg.draw(canvas);
        }
    }

    @Override
    public void update() {
        for(Background bg : backgrounds){
            bg.update();
        }
    }
}

ScrollingBackground
package com.thalisson.tcc.game.background;

import android.graphics.Bitmap;
import android.graphics.Canvas;

import com.thalisson.tcc.game.texture.Texture;

public class ScrollingBackground extends Background{

    /**
     * @author Thalisson
     */

    private Bitmap mBackground;
    private double offsetX;
    private double offsetY;

    private double autooffsetX;
    private double autooffsetY;

    public ScrollingBackground(Texture tex) {
        mBackground = tex.tex;
        height = tex.tex.getHeight();
        width = tex.tex.getWidth();
    }

    @Override
    public void draw(Canvas canvas) {
        canvas.drawBitmap(mBackground, (int)x, (int)y, null);
        super.draw(canvas);
    }

    @Override
    public void update() {
        x+=offsetX;
        y+=offsetY;

        x+=autooffsetX;
        y+=autooffsetY;

        offsetX=0;
        offsetY=0;
    }

    public void setOffset(double offsetX, double offsetY){
        this.offsetX = offsetX;
        this.offsetY = offsetY;
    }
}

```

```

        public void setSpeed(double autooffsetX, double autooffsetY){
            this.autooffsetX = autooffsetX;
            this.autooffsetY = autooffsetY;
        }
    }

Connector
package com.thalisson.tcc.game.multiplayer;

import java.io.IOException;
import java.io.InputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.OutputStream;
import java.net.InetSocketAddress;
import java.net.Socket;

import com.thalisson.tcc.game.Constants;

public class Connector {
    private Socket sct;
    private String mServer= null;
    private int mPort = 0;
    private int mTimeout = 5000;
    private InetSocketAddress mAddress;
    private InputStream mIs;
    private OutputStream mOs;
    private ObjectOutputStream mObOut;
    private ObjectInputStream mObIn;
    private boolean mReceiving;
    private String userName;
    private Message lastInvite;

    public Connector(String server, int port){
        mServer = server;
        mPort = port;
        mAddress = new InetSocketAddress(mServer, mPort);
    }

    public void connect() throws IOException{
        sct = new Socket();
        sct.connect(mAddress, mTimeout);
        mIs = sct.getInputStream();
        mOs = sct.getOutputStream();
        mObOut = new ObjectOutputStream(mOs);
        mObIn = new ObjectInputStream(mIs);
    }

    public boolean isConnect(){
        return sct.isConnected();
    }

    /**
     * onMessageReceived chamado quando o cliente recebe mensagem do servidor
     */
    public void onMessageReceived(Message msgReceived) {
    }

    public void stopListening() {
        mReceiving = false;
    }

    public void startListening() {
        mReceiving = true;
        Thread receivingThread = new Thread() {
            public void run() {
                while (mReceiving) {
                    int bytesReceived;
                    try {
                        bytesReceived = mObIn.available();
                        if (bytesReceived > 0) {
                            Message msg = (Message)
                                mObIn.readObject();
                            onMessageReceived(msg);
                        }
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
            }
        };
        receivingThread.start();
    }
}

```

```

        }
        try {
            Thread.sleep(50);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    } catch (IOException e) {
        e.printStackTrace();
    } catch (ClassNotFoundException e) {
        // TODO Auto-generated catch
        e.printStackTrace();
    }
}
};
receivingThread.start();
}

protected boolean sendMessage(int type, String msgToSend, String from, String to) {
    try {
        Message m = new Message(type,msgToSend, from, to);
        mObOut.writeObject(m);
        mObOut.flush();
        return true;
    } catch (IOException e) {
        // TODO Auto-generated catch block
        return false;
    }
}

public boolean login (String username){
    this.userName = username;
    return this.sendMessage(Constants.TYPE_LOGIN, username, username, null);
}

public boolean inviteToGame (String user){
    return this.sendMessage(Constants.TYPE_INVITE, null, userName, user);
}

public boolean acceptGame(){
    return this.sendMessage(Constants.TYPE_ACCEPT, null, lastInvite.getTo(),
lastInvite.getFrom());
}

public boolean getPlayers(){
    return this.sendMessage(Constants.TYPE_GETPLAYERS, null, null, null);
}

public boolean logout(){
    return this.sendMessage(Constants.TYPE_LOGOUT, null, null, null);
}

public boolean sendPlay(String play){
    return this.sendMessage(Constants.TYPE_PLAY_SEND, play, userName, null);
}
}
Message
package com.thalisson.tcc.game.multiplayer;

public class Message {

    private static final long serialVersionUID = 1L;
    private int type;
    private String message;
    private String to;
    private String from;

    public Message(int type, String message, String to, String from){
        this.type = type;
        this.message = message;
        this.to = to;
        this.from = from;
    }

    public int getType() {
        return type;
    }
}

```

```

    }

    public void setType(int type) {
        this.type = type;
    }

    public String getMessage() {
        return message;
    }

    public String getFrom() {
        return from;
    }

    public void setFrom(String from) {
        this.from = from;
    }

    public String getTo() {
        return to;
    }

    public void setTo(String to) {
        this.to = to;
    }
}

```

CellSprite

```

package com.thalisson.tcc.game.renderable;

import java.util.Map;

import android.graphics.Bitmap;
import android.graphics.Bitmap.Config;
import android.graphics.Canvas;

import com.thalisson.tcc.game.Renderable;
import com.thalisson.tcc.game.texture.Texture;

public class CellSprite extends Renderable{

    /**
     * @author Thalisson
     */

    private Map<Character, Texture>interpretador;
    private int cellWidth,cellHeight;
    private String[] fullCells;
    private Bitmap sprite;

    public CellSprite(int x, int y, Map<Character,Texture> interpreter, String[]
fullCells, int cellWidth, int cellHeight) {
        super();

        this.x=x;
        this.y=y;
        this.cellWidth = cellWidth;
        this.cellHeight = cellHeight;
        this.interpretador = interpreter;
        this.fullCells = fullCells;

        sprite = Bitmap.createBitmap(fullCells[0].length()*cellWidth,
fullCells.length*cellHeight, Config.ARGB_8888);

        init();
    }

    private void init(){
        Canvas c = new Canvas(sprite);
        int dx = (int)x;
        int dy = (int)y;

        for(int i = 0; i < fullCells.length; i++){
            dx=(int)x;
            for(int j = 0; j < fullCells[i].length(); j++){

```

```

        Bitmap toDraw =
interpretador.get(fullCells[i].charAt(j)).tex;
        c.drawBitmap(toDraw, dx, dy, null);

        dx+=cellWidth;
    }
    dy+=cellHeight;
}

@Override
public void draw(Canvas canvas) {
    canvas.drawBitmap(sprite, (float)y, (float)x, null);
}
}

```

Sprite

```

package com.thalisson.tcc.game.renderable;

import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Matrix;

import com.thalisson.tcc.game.Renderable;
import com.thalisson.tcc.game.texture.Texture;

public class Sprite extends Renderable{

    /**
     * @author Thalisson
     */

    public Bitmap sprite;

    private Matrix mMatrix = new Matrix();
    public boolean isVisible = true;

    public Sprite(Texture tex, int x, int y) {
        this.x = x;
        this.y = y;

        sprite = tex.tex;
        this.height = sprite.getHeight();
        this.width = sprite.getWidth();
    }

    @Override
    public void update() {
        super.update();
    }

    public void setVisibility(boolean visible){
        isVisible=visible;
    }

    @Override
    public void draw(Canvas canvas){
        if(isVisible)
            canvas.drawBitmap(sprite, (int)x, (int)y, null);
    }

    public void recycle(){
        sprite.recycle();
        sprite = null;
    }

    public void rotate(int degrees){
        mMatrix.postRotate(degrees);
        sprite = Bitmap.createBitmap(sprite, (int)x, (int)y, height, width,
mMatrix, false);
    }

    public void rotate(int px, int py, int degrees){
        mMatrix.postRotate(degrees,px,py);
    }
}

```

```

        sprite = Bitmap.createBitmap(sprite, (int)x, (int)y, height, width,
mMatrix, false);
    }

    public void scale(int sx, int sy){
        mMatrix.postScale(sx, sy);
        sprite = Bitmap.createBitmap(sprite, (int)x, (int)y, height, width,
mMatrix, false);
    }

    public void scale(int sx, int sy, int px, int py){
        mMatrix.postScale(sx, sy, px, py);
        sprite = Bitmap.createBitmap(sprite, (int)x, (int)y, height, width,
mMatrix, false);
    }

    public boolean pointOnSprite(int x, int y){
        return (x > this.x && y > this.y && x < this.x+this.width
&& y < this.y+this.height);
    }
}

```

TileAnimation

```

package com.thalisson.tcc.game.renderable;

import java.util.ArrayList;
import android.graphics.Bitmap;
import com.thalisson.tcc.game.texture.Texture;

public class TileAnimation extends Sprite{

    /**
     * @author Thalisson
     */

    public Sprite fullImage;//imagem fonte

    private ArrayList<Bitmap> imagelist = new ArrayList<Bitmap>();//lista das
imagens

    private int currentFrameCounter =0;//numero do frame atual
    private int frameskipdelay; // tempo de intervalo entre cada frame da animacao

    //indica como cortar a imagem referente a animacao
    private int tileColumns;
    private int tileRows;

    //true se a animacao estiver acontecendo
    public boolean isStarted = true;

    private long pasteTime = 0;
    private long lastTime;

    //true se a animacao se repete ao infinito.
    private boolean repeatAnimation;

    public TileAnimation(Texture tex, int x, int y, int tileColumns, int tileRows,
int frameskipdelay, boolean repeat) {
        super(tex, x, y);

        this.frameskipdelay = frameskipdelay;
        this.tileRows = tileRows;
        this.tileColumns = tileColumns;
        this.repeatAnimation = repeat;
        fullImage = new Sprite(tex, 0, 0);
        cutImage();
        this.sprite = imagelist.get(0);
        lastTime = System.currentTimeMillis();
    }
}

```

```

    }

    public TileAnimation(Texture[] textures, int x, int y, int frameskipdelay,
boolean repeat) {
        super(textures[0], x, y);

        this.frameskipdelay = frameskipdelay;
        fullImage = null;
        for(int i = 0; i < textures.length; i++){
            imagelist.add(new Sprite(textures[i], x, y).sprite);
        }
        this.repeatAnimation = repeat;
        this.sprite = imagelist.get(0);
        lastTime = System.currentTimeMillis();
    }

private void cutImage(){
    int cx = 0;
    int cy = 0;
    int stepX = fullImage.width/tileColumns;
    int stepY = fullImage.height/tileRows;
    for(int i = 1; i <= tileRows; i++){
        for(int j = 0; j < tileColumns; j++){
            if(cx+stepX <= fullImage.width){

imagelist.add(Bitmap.createBitmap(fullImage.sprite, cx, cy, stepX, stepY));
                cx+=stepX;
            }
            cx=0;
            if(cy+stepY <= fullImage.height){
                cy+=stepY;
            }
        }
    }

@Override
public void recycle() {
    for(Bitmap b : imagelist){
        b.recycle();
    }
    imagelist = null;
    if(fullImage != null)
        fullImage.recycle();
    super.recycle();
}

public void update(){
    super.update();
    if(isStarted){
        long deltaTime = System.currentTimeMillis() - lastTime;
        lastTime = System.currentTimeMillis();
        pasteTime+=deltaTime;
        if(pasteTime >= frameskipdelay){
            pasteTime = 0;
            nextFrame();
        }
    }
}

public void nextFrame(){
    if(currentFrameCounter < imagelist.size()-1){
        currentFrameCounter++;
    }else{
        if (repeatAnimation)
            currentFrameCounter=0;
        else
            this.stop();
    }
    sprite = imagelist.get(currentFrameCounter);
}

public void start(){
    isStarted=true;
}

```

```

    }

    public void stop(){
        isStarted=false;
    }
}

CellRenderer
package com.thalisson.tcc.game.renderer;

import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Color;

import com.thalisson.tcc.game.background.Background;
import com.thalisson.tcc.game.background.ColoredBackground;
import com.thalisson.tcc.game.texture.Texture;

public class CellRenderer implements Renderer {

    /**
     * @author Thalisson
     */

    private Bitmap cells;
    private Background background;
    public float x;
    public float y;
    private float scrollX;
    private float scrollY;

    public int mapHeight;
    public int mapWidth;

    public CellRenderer() {
        background = new ColoredBackground(Color.BLACK);
    }

    public void setMap(Texture map){
        cells = map.tex;
        mapHeight = cells.getHeight();
        mapWidth = cells.getWidth();
    }

    public void drawFrame(Canvas canvas) {
        update();
        background.update();
        background.draw(canvas);
        canvas.drawBitmap(cells,x,y,null);
    }

    public void setBackground(Background bg){
        background = bg;
    }

    public void setMapOffset(float ox, float oy){
        scrollX = ox;
        scrollY = oy;
    }

    public void setMapPosition(float px, float py){
        x = px;
        y = py;
    }

    private void update(){
        x+=scrollX;
        y+=scrollY;
        scrollX = 0;
        scrollY = 0;
    }
}

```

Renderer


```

package com.thalisson.tcc.game.renderer;

import android.graphics.Canvas;

/** Interface de renderizacao generica */
public interface Renderer {

    /**
     * @author Thalisson
     */

    int fps = 0;

    /**
     * Desenha o frame atual.
     * @param canvas O canvas onde sera desenhado o frame.
     */
    public void drawFrame(Canvas canvas);

}

```

SurfaceRenderer

```

package com.thalisson.tcc.game.renderer;

import java.util.ArrayList;

import android.graphics.Canvas;
import android.graphics.Color;

import com.thalisson.tcc.game.Renderable;
import com.thalisson.tcc.game.background.Background;
import com.thalisson.tcc.game.background.ColoredBackground;

public class SurfaceRenderer implements Renderer {

    private ArrayList<Renderable> renderables = new ArrayList<Renderable>();
    private Background background = new ColoredBackground(Color.green(100));

    public void setRenderables(ArrayList<Renderable> renderables) {
        this.renderables = renderables;
    }

    public void addRenderable(Renderable r) {
        synchronized(renderables) {
            renderables.add(r);
        }
    }

    public void removeRenderable(int location){
        synchronized(renderables) {
            renderables.remove(location);
        }
    }

    public void removeRenderable(Renderable r){
        synchronized(renderables) {
            renderables.remove(r);
        }
    }

    public void drawFrame(Canvas canvas) {

        canvas.drawColor(Color.BLACK);
        background.draw(canvas);
        background.update();
        synchronized(renderables) {
            for (Renderable r : renderables) {
                r.update();
                r.draw(canvas);
            }
        }
    }

}

```

```

        public void setBackground(Background bg){
            background = bg;
        }
    }

Texture
package com.thalisson.tcc.game.texture;

import android.graphics.Bitmap;
import android.graphics.Bitmap.Config;

public class Texture {

    /**
     * @author Thalisson
     */

    public Bitmap tex = null;
    public String path;
    public boolean isLoading = false;

    public Texture(String path) {
        this.path = path;
    }

    public Texture(Bitmap bmp){
        tex = bmp;
    }

    public void recycle(){
        tex.recycle();
        tex = null;
    }

    public Texture[] cut(int tileColumns, int tileRows){
        Texture[] textures = new Texture[tileColumns*tileRows];
        int cx = 0;
        int cy = 0;
        int counter = 0;
        int stepX = tex.getWidth()/tileColumns;
        int stepY = tex.getHeight()/tileRows;
        for(int i = 1; i <= tileRows; i++){
            for(int j = 0; j < tileColumns; j++){
                if(cx+stepX <= tex.getWidth()){
                    textures[counter] = new
Texture(Bitmap.createBitmap(tex, cx, cy, stepX, stepY));
                    cx+=stepX;
                    counter++;
                }
            }
            cx=0;
            if(cy+stepY <= tex.getHeight()){
                cy+=stepY;
            }
            textures[textures.length-1] = new
Texture(Bitmap.createBitmap(textures[textures.length-
1].tex.getWidth(),textures[textures.length-1].tex.getHeight(),Config.RGB_565));
        }
        return textures;
    }
}

TextureAtlas
package com.thalisson.tcc.game.texture;

import java.util.ArrayList;

public class TextureAtlas {

    /**
     * @author Thalisson
     */

    private ArrayList<Texture> textures;
    public boolean loadedQueue = false;

```

```

public TextureAtlas() {
    textures = new ArrayList<Texture>();
}

public void addTexture(Texture tex){
    textures.add(tex);
}

public ArrayList<Texture> getQueue() {
    return textures;
}
}

```

TextureManager

```

package com.thalisson.tcc.game.texture;

import java.io.IOException;
import java.io.InputStream;

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.util.Log;

import com.thalisson.tcc.game.Constants;

public class TextureManager {

    /**
     * Carrega uma imagem de um caminho e converte para bitmap.
     * @param context O contexto da aplicacao
     * @param atlas O TextureAtlas que armazena as texturas.
     */

    public static void load(Context context, TextureAtlas atlas){
        BitmapFactory.Options DEFAULT_BITMAP_OPTIONS = new
        BitmapFactory.Options();
        DEFAULT_BITMAP_OPTIONS.inPreferredConfig = Bitmap.Config.ARGB_8888;

        Bitmap bitmap = null;
        InputStream is = null;
        long allTime = System.currentTimeMillis();
        for(Texture tex : atlas.getQueue()){
            try {
                is =
                context.getResources().getAssets().open(tex.path);

                bitmap = BitmapFactory.decodeStream(is, null,
                DEFAULT_BITMAP_OPTIONS);
                tex.isLoaded = true;

            } catch (IOException e) {
                Log.e(Constants.LOGTAG, "Nao foi possivel
                localizar/carregar a textura: " + tex.path);
                throw new NullPointerException("Nao foi possivel
                localizar a textura: " + tex.path);
            } finally {
                try {

                    is.close();
                } catch (IOException e) {
                    Log.d(Constants.LOGTAG, "Erro ao fechar
                    BitmapInputStream");
                }
            }

            tex.tex = bitmap;
        }
        Log.d(Constants.LOGTAG, "Carregado(s) " + atlas.getQueue().size() + "
        Bitmaps em "+(System.currentTimeMillis()-allTime));
    }
}

```

Servidor

```

public class Constants {
    public static final int TYPE_HELLO = 1;
    public static final int TYPE_INVITE_SEND = 2;
    public static final int TYPE_INVITE_RESPONSE = 3;
    public static final int TYPE_PLAY = 4;
    public static final int TYPE_FINAL = 5;
    public static final int TYPE_BYE = 6;
    public static final int TYPE_YOUR_TURN = 7;
    public static final int TYPE_HELLO_RESPONSE = 8;
    public static final int TYPE_GET_PLAYERLIST = 9;
    public static final int TYPE_PLAYERLIST_RESPONSE = 10;
    public static final int TYPE_GAME_START = 11;
}

import java.util.Vector;

public class Game {

    protected Vector<Player> players;
    protected Player playerTurn;
    protected int order;

    protected void processPlay(Message m){
        //Processa a jogada enviada, atualiza o estado do jogo e envia o
        resultado para os jogadores (TYPE_NEW_STATE). Após isso, troca o jogador da vez e avisa
        o mesmo que seu turno.
        sendBroadcast(m);
    }

    protected void endGame(){
    }

    protected void startGame(){
    }

    protected void changeActivePlayer(){
        order++;
        playerTurn = players.elementAt(order);
        playerTurn.sendMessageToDevice(new Message(Constants.TYPE_YOUR_TURN,
        getChangePlayerMessage(), playerTurn.getPlayerName(), "SERVER"));
    }

    public Game(Player p1, Player p2){
        order = 0;
        players = new Vector<>();
        players.add(p1);
        players.add(p2);
        this.sendBroadcast(new Message(Constants.TYPE_GAME_START, null, null,
        null));
        this.startGame();
    }

    protected void sendBroadcast(Message m){
        Player p;
        for (int i=0;i<players.size();i++){
            p = players.elementAt(i);
            p.sendMessageToDevice(m);
        }
    }

    protected String getChangePlayerMessage(){
        return "It's your turn to play!";
    }

    public void playerQuit(Player player) {

```

```

    }
}

public class Main {
    public static void main(String [ ] args)
    {
        Server s = new Server(7788);
        s.startServer();
    }
}

import java.io.Serializable;

public class Message implements Serializable{
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    private int type;
    private String message;
    private String to;
    private String from;

    public Message(int type, String message, String to, String from){
        this.type = type;
        this.message = message;
        this.to = to;
        this.from = from;
    }

    public int getType() {
        return type;
    }

    public void setType(int type) {
        this.type = type;
    }

    public String getMessage() {
        return message;
    }

    public String getFrom() {
        return from;
    }

    public void setFrom(String from) {
        this.from = from;
    }

    public String getTo() {
        return to;
    }

    public void setTo(String to) {
        this.to = to;
    }
}

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;

public class Player extends Thread {

    private String playerName;
    private ObjectInputStream dis;
    private ObjectOutputStream dos;
    private Game mGame;
    private Socket mSocket;
    private Server mServer;
}

```

```

public Player(Socket s, Server base) {
    mSocket = s;
    mServer = base;
}

@Override
public void run() {
    Message m;
    try {
        dis = new ObjectInputStream(mSocket.getInputStream());
        dos = new ObjectOutputStream(mSocket.getOutputStream());
        while ((m = (Message) dis.readObject()) != null) {
            this.processMessage(m);
        }
    } catch (IOException e) {
        System.out.println("EXCEPTION");
        if (mGame != null)
            mGame.playerQuit(this);
        mServer.removePlayer(this);
    } catch (ClassNotFoundException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

private void processMessage(Message m) {
    switch (m.getType()) {
        case Constants.TYPE_HELLO:
            playerName = m.getFrom();
            System.out.println("Player name:" + playerName);
            mServer.insertPlayerInPlayerList(this);
            this.sendMessageToDevice(new Message(
                Constants.TYPE_INVITE_RESPONSE, "Logado
com sucesso",
                playerName, "SERVER"));
            break;
        case Constants.TYPE_INVITE_SEND:
            mServer.invitePlayer(m);
            break;
        case Constants.TYPE_INVITE_RESPONSE:
            mServer.createGameSession(m);
        case Constants.TYPE_PLAY:
            mGame.processPlay(m);
            break;
        case Constants.TYPE_BYE:
            if (mGame == null) {
                mServer.removePlayer(this);
            } else {
                mGame.playerQuit(this);
            }
            break;
        case Constants.TYPE_FINAL:
            break;
        case Constants.TYPE_GET_PLAYERLIST:
            mServer.sendPlayerListToPlayer(this);

        default:
            break;
    }
}

public void sendMessageToDevice(Message m) {
    try {
        dos.writeObject(m);
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

public String getPlayerName() {
    return playerName;
}

protected void endPlayer() {

```

```

        try {
            if (dis != null)
                dis.close();
            if (dos != null)
                dos.close();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Enumeration;
import java.util.Hashtable;

public class Server {
    private int port;
    private Hashtable<String, Player> playerList = new Hashtable<>();
    private ServerSocket ss;
    private Socket mSocket;

    public Server(int portNumber) {
        port = portNumber;
    }

    public void startServer() {
        try {
            ss = new ServerSocket(port);
            System.out.println("Server iniciado na porta:" + port);
            while (true) {
                mSocket = ss.accept();
                System.out.println("Player conectado");
                Player p = new Player(mSocket, this);
                insertPlayerInPlayerList(p);
                p.start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void insertPlayerInPlayerList(Player p) {
        playerList.put(p.getPlayerName(), p);
    }

    public void invitePlayer(Message m) {
        Player p = this.getPlayerByName(m.getTo());
        if (p != null) {
            p.sendMessageToDevice(m);
        }
    }

    public void createGameSession(Message m) {
        // Metodo que cria e inicializa a sessao de jogo.
        Player p1 = this.getPlayerByName(m.getTo());
        Player p2 = this.getPlayerByName(m.getFrom());
        Game g = new Game(p1, p2);
    }

    private Player getPlayerByName(String name) {
        Player p;
        synchronized (playerList) {
            for (Enumeration e = playerList.elements();
                e.hasMoreElements();) {
                p = (Player) e.nextElement();
                if (p.getPlayerName().equals(name)) {
                    return p;
                }
            }
        }
    }
}

```

```

        }
        return null;
    }
}

public void sendPlayerListToPlayer(Player player) {
    Player p;
    StringBuffer sb = new StringBuffer();
    synchronized (playerList) {
        for (Enumeration e = playerList.elements();
e.hasMoreElements();) {
            p = (Player) e.nextElement();
            sb.append(p.getPlayerName());
            sb.append("|");
        }
        player.sendMessageToDevice(new Message(
Constants.TYPE_PLAYERLIST_RESPONSE,
sb.toString(), player
.getPlayerName(),
"SERVER"));
    }
}

public void removePlayer(Player player) {
    System.out.println("Removing player:" + player.getPlayerName());
    if (playerList.containsKey(player.getPlayerName()))
        playerList.remove(player.getPlayerName());
}
}

```


ANEXO B - ARTIGO

Framework para jogos multiplayer em dispositivos móveis Android

Thalisson da Rosa

Departamento de Informática e Estatística
Universidade Federal de Santa Catarina (UFSC) – Florianópolis, SC – Brazil

talico@inf.ufsc.br

Abstract. *There is a big barrier in the learning curve of game development. Most of the frameworks and engines available today focus on people that already have a great knowledge in the area, lacking a lot and even discouraging who is still learning. This paper has the objective of developing a simple framework that is easy to use to develop games to Android smart phones.*

Resumo. *Existe uma grande barreira inicial na curva de aprendizagem do desenvolvimento de jogos. A grande maioria dos frameworks existentes hoje atende somente a parcela com extenso conhecimento, deixando muito a desejar e até mesmo desencorajando quem está iniciando na área. Esse trabalho tem como objetivo o desenvolvimento de um framework simples e fácil de ser utilizado no desenvolvimento de jogos para smartphones Android.*

1. Introdução

Mundialmente, o mercado de celulares vem crescendo cada vez mais e já abrange a grande maioria dos habitantes. O Brasil não é uma exceção e, de acordo com a ANATEL (2014), em setembro de 2014 o país chegou a 278,1 milhões de linhas ativas e 136.9 celulares para cada 100 habitantes.

Dentro desse grupo, temos os smartphones, que são aparelhos com capacidade de processamento muito acima de aparelhos convencionais e contam com vários recursos, entre eles acesso a redes 3G e sincronização com serviços online. Resumidamente, seria um celular com funções de PDA (*Personal Digital Assistant*), rodando um sistema operacional que permite personalização.

Em paralelo a isso, outro ramo que vem crescendo é dos jogos eletrônicos, que já vem competindo com o tradicional cinema no quesito entretenimento. De pequenas produções no passado, hoje já temos jogos com orçamentos de mais de 100 milhões de dólares, como foi o caso de Grand Theft Auto IV (ANDROVICH, 2008), que envolveu mais de 1000 profissionais durante três anos e meio.

Com o mercado de jogos em expansão e o de smartphones também em pleno crescimento, nada mais natural que os jogos também invadissem a plataforma móvel. Por se tratar de um fenômeno recente, o desenvolvimento não é algo relativamente fácil e faltam ferramentas que auxiliam esse trabalho.

O desenvolvimento de jogos não é uma tarefa fácil, exige muito conhecimento específico e experiência. Para facilitar o processo, existem frameworks para auxiliar nesse desenvolvimento, mas nenhum deles tem como foco o desenvolvedor iniciante, aquele que quer aprender fazendo.

Isso acaba gerando uma barreira inicial muito grande que pode acabar afastando o desenvolvedor, pois apesar de ter interesse, a busca por conceitos básicos tem que vir de outras fontes muitas vezes não relacionadas ao desenvolvimento de jogos.

2. Frameworks

Quando falamos do desenvolvimento de sistemas, problemas semelhantes acabam tendo muitos pontos em comum quando essas soluções são feitas a partir do zero. Uma das maneiras criadas para evitar o retrabalho foi o desenvolvimento do conceito de framework, que segundo (WIRFS-BROCK, R. 1990) é "uma coleção de classes concretas e abstratas e as interfaces entre elas, e é o projeto de um subsistema".

Por exemplo, num domínio de aplicações bancárias, apesar de cada banco ter suas peculiaridades, toda a base de trabalho, como contas, transações e regulamentações do setor são exatamente as mesmas, então toda essa lógica básica é implementada pelo framework. Isso também é conhecido como *coldspot*, pois é uma área inflexível de código que não pode ser modificada posteriormente pelos usuários desse framework. Claro que somente dessa forma não teríamos como utilizar esse framework, pois tudo já estaria definido. Toda a parte do código que pode ser alterada, ou seja, que é flexível e permite a extensibilidade do framework é chamada de *hotspot*.

Outro ponto de ligação entre o framework e a aplicação sendo desenvolvida tendo ele como base são os métodos *hook*. Esses métodos normalmente são métodos abstratos chamados pelos *coldspots* e que podem ser implementados para se alterar algum comportamento ou se fazer algum tipo de processamento em resposta a alguma ação. Por exemplo, podemos ter um framework que efetua uma chamada de rede como parte de um *coldspot*. Por estar em um *coldspot*, a maneira que essa chamada é efetuada não pode ser alterada, mas pode existir um método *hook* que indica se a chamada foi concluída com sucesso ou não. Cabe agora ao usuário do framework decidir se ele pretende fazer algo com a informação, então ele pode implementar o método *hook* para mostrar um aviso na tela ou então processar os dados.

Apesar de todas suas vantagens, o desenvolvimento de um framework apresenta alguns pontos que devem ser levados em consideração antes de se iniciar o projeto. Em primeiro lugar, desenvolver uma solução que atenda a um certo domínio e ao mesmo tempo seja facilmente reutilizável e de qualidade exige uma grande quantidade de esforço e tempo. Caso não seja dada a atenção necessária nessa etapa, pode se chegar à conclusão mais tarde que o framework mais complica do que facilita o desenvolvimento de novos produtos ou que então o esforço despendido na sua construção não justifica um pequeno ganho de tempo nas novas aplicações.

Outro problema comum é que a validação de aplicações construídas a partir de um framework é extremamente complicada, pois pode se tornar muito difícil distinguir se os bugs encontrados são do próprio framework ou se são da aplicação. Isso só fica ainda mais difícil caso o código do framework seja fechado, pois praticamente impossibilita a depuração de código.

2.1 Desenvolvimento de frameworks

Ao se desenvolver um framework, um dos primeiros passos é a análise de domínio das aplicações. Essa é uma das fases mais importantes do processo, pois caso seja mal executada pode fazer com que o framework desenvolvido não atenda aos requisitos. Basicamente é nessa etapa que se avalia qual é o comportamento básico que deve estar no framework, quais são suas funcionalidades, quais serão os *hotspots* e que métodos *hook* serão os responsáveis pela ligação.

Após essa análise, passa-se ao processo de modelagem, onde são definidas as classes, pacotes e interfaces, sempre tendo o cuidado de manter a generalidade e a extensibilidade do framework, para que o mesmo não atenda somente as necessidades atuais.

2.2 Metodologias de desenvolvimento de frameworks

Metodologias de desenvolvimento de frameworks basicamente são diferentes caminhos possíveis de se seguir ao se desenvolver um framework. São um conjunto de procedimentos e instruções já testados e comprovados que ajudam na etapa inicial e ao mesmo tempo, não são restritivas em excesso. De acordo com (SILVA, 2000) são três as principais metodologias: Taligent, projeto dirigido por exemplo e projeto dirigido por *Hotspots*. Ainda de acordo com (SILVA, 2000), estas metodologias não detalham o processo ou técnicas de modelagem, são apenas linhas gerais do processo de desenvolvimento

A primeira (Talgient) é uma metodologia criada pela empresa de mesmo nome onde se propõe que, ao invés de um grande framework, devem ser desenvolvidos vários frameworks menores e mais simples, cada um voltado a um aspecto. A argumentação por trás dessa abordagem é que, dessa forma, podemos utilizar apenas os frameworks estritamente necessários, o que torna mais fácil o uso e aumenta o grau de reusabilidade.

Na metodologia de projeto dirigido por exemplo, proposta por (JOHNSON, R. E, 1993), são utilizados vários softwares já desenvolvidos em um mesmo domínio e todos são comparados para que sejam encontrados os pontos em comum. Ao se encontrar esses pontos, uma solução geral que atenda a todos é proposta e é testada para novos desenvolvimentos. Um dos problemas nessa metodologia é que normalmente são necessárias várias aplicações para conseguir uma solução confiável. Uma das maneiras de se contornar esse problema é a análise de um número menor de aplicações, mas o framework deve ser validado com ao menos mais dois aplicativos diferentes, desenvolvidos a partir do framework gerado.

Finalmente, temos a metodologia de projeto dirigido a *hotspots*, definida por (PREE, W. 1994). Nessa metodologia, busca-se primeiramente encontrar quais são os

pontos flexíveis em comum que um domínio necessita e uma estrutura é pensada em torno desses pontos. Após finalizar a etapa de implementação, o framework é novamente analisado para ver se mantém sua generalidade e, caso necessário, novas interações são feitas e novos *hotspots* são encontrados.

3. Projeto

Baseando-se nos levantamentos e considerações feitas anteriormente, nesse capítulo serão mostrados os projetos para implementação de um framework para desenvolvimento de jogos multiplayer na plataforma Android, um pequeno framework de servidor simples, apenas para dar suporte à parte cliente da aplicação e dois jogos que utilizam esses frameworks, sendo um baseado em turnos e o outro em um modelo híbrido entre turnos e tempo real.

Toda a comunicação será feita através da internet, seja ela via 3G/GPRS ou via Wi-Fi. A arquitetura escolhida é a de cliente-servidor através do protocolo TCP, por conta de sua confiabilidade na entrega de pacotes.

3.1 Framework para jogos multiplayer Android

Um jogo é composto de vários subsistemas, cada um responsável por uma parte específica, mas que estão em comunicação constante. Por exemplo, temos sistemas responsáveis pelo controle de cenas, pelo controle do ciclo de jogo, inputs, som, IA, networking, etc.

Como o objetivo desse trabalho é fazer um framework multiplayer que atenda as necessidades mais básicas, não teremos foco em sistemas mais complexos, como de inteligência artificial e focaremos nos seguintes sistemas: ciclo de jogo, controle de eventos, input e network. Com esses sistemas, o usuário será capaz de realizar as atividades descritas na Figura 6.

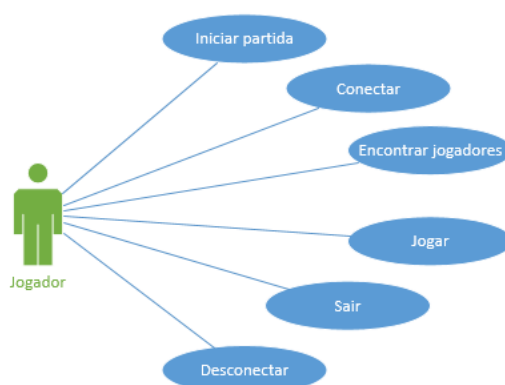


Figura 1 - Diagrama de casos de uso

Com base nos casos de uso vistos na Figura 6, foi determinado o diagrama de atividades visto na Figura 7, que mostra todos os passos desde a conexão inicial até o momento em que o usuário para de jogar.

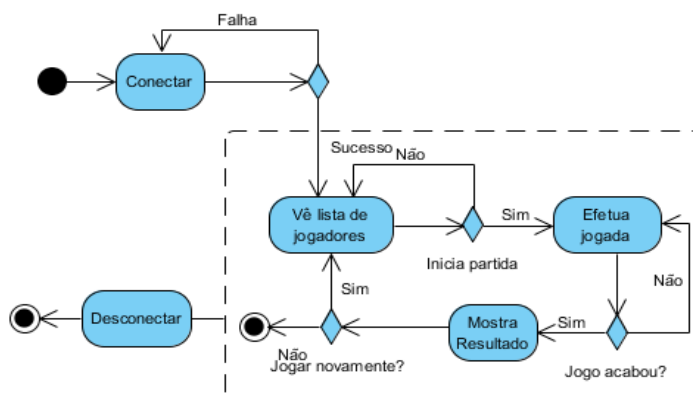


Figura 2 - Diagrama de atividades

É importante notar que em qualquer momento da atividade é possível acontecer o evento de desconectar, seja ela por iniciativa de qualquer um dos jogadores, do servidor ou até mesmo de falhas de rede fora do controle desse framework.

A partir do ponto que é iniciada a partida até o momento que a mesma é encerrada, o jogo entra em loop definido como ciclo de jogo. O ciclo de jogo é o responsável por duas das funcionalidades mais básicas em um jogo, que são atualizar o estado do jogo e desenhar os elementos na tela. De maneira resumida, baseando-se nas regras definidas ou nos inputs do usuário, um novo estado de jogo é calculado, elementos são movidos de posição e o resultado é desenhado na tela em sua nova posição.

3.2 Framework de um servidor para jogos multiplayer

Como o objetivo do trabalho é focar na parte cliente, o framework para o servidor deverá ser o mais simples possível, realizando as tarefas básicas de maneira satisfatória, mas sem muitos extras. Somente para lembrar, essas tarefas básicas são a de gerenciar a entrada de novos jogadores, gerenciar uma lista de jogadores, iniciar uma partida, processar as jogadas e informar o resultado desse processamento.

Do ponto de vista do usuário do framework, a principal classe é a Game, que pode ser vista na Figura 17. Ela contém todos os métodos necessários e que devem ser sobrescritos para controlar o fluxo do jogo.



Figura 23 - Classe Game

Basicamente, ao iniciar o servidor uma porta será reservada para o socket de conexão. A cada nova conexão de usuário nessa porta, o servidor validará o usuário e irá criar um objeto da classe Player, que receberá as mensagens vindas do cliente e repassará para o responsável por tratar essa mensagem.

4. Implementação

Para a implementação do código, foi utilizada a IDE (Integrated Development Environment) Eclipse em conjunto com a SDK (Software Development Kit) de desenvolvimento do Android em sua versão 2.3.3. Todo o código criado estará disponível em anexo, em mídia física do tipo CD e em repositório virtual acessível publicamente pelo endereço <https://github.com/thalissonrosa>.

4.1 Jogo de exemplo Pedra-Papel-Tesoura

O jogo de pedra-papel-tesoura é um simples jogo, conhecido pela maioria das pessoas e que serve perfeitamente para ilustrar um caso de um jogo simples, entre dois jogadores e utilizando o mecanismo de turnos. O funcionamento dele seguirá o fluxo descrito no diagrama de atividade da Figura 19.

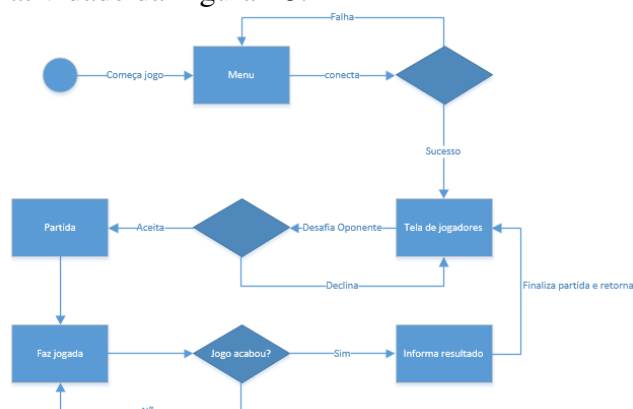


Figura 24 - Diagrama de atividade Pedra-Papel-Tesoura

Basicamente, um usuário conecta ao jogo e em caso de sucesso, requisita a lista de jogadores ao servidor. Com essa lista em mãos, escolhe um adversário e envia um convite de partida. Caso esse convite seja aceito, ambos vão para a tela de jogo, onde escolhem uma das opções (pedra, papel ou tesoura) e enviam essa informação para o servidor, que calcula o vencedor baseado nas regras da Figura 20 e informa o resultado a ambos os jogadores.



Figura 25 - Regras Pedra-Papel-Tesoura

4.1.1 Cliente

Para as tarefas mais básicas como tela inicial de login e lista de jogadores, serão utilizadas as classes do próprio Android, como a Activity. Nessas etapas iniciais, o framework será utilizado para gerenciar a conexão e fazer a troca de mensagens. Para isso, a classe Connector deve ser estendida e o método `onMessageReceived(Message messageReceived)` deverá ser sobrescrito.

Nesse método serão definidos os comportamentos esperados ao receber alguma mensagem do servidor, principalmente as referentes ao convite de partida aceito e a de jogada recebida. As outras mensagens têm comportamentos que pouco mudam de um jogo para outro e o próprio framework se encarregará delas.

Ao receber uma mensagem de convite de partida aceita, um jogo deve ser iniciado e para isso a classe `FrameworkSurfaceView` deve ser estendida pois, lembrando, ela é a responsável pelo controle de inputs. Essa nova classe deverá desenhar na tela os elementos correspondentes a cada uma das escolhas possíveis. Nessa classe, o método que irá saber quando ocorreu o clique e qual foi a opção escolhida será o `onTouchDown()`. Nele será criada uma String com a mensagem e o método `sendPlay()` será invocado passando essa String como parâmetro.

Após esse passo, o cliente irá esperar pela resposta do servidor que informará quem foi o vencedor da partida. Tendo essa resposta, o resultado é informado na tela e irá retornar para a tela de seleção de adversário, repetindo o processo para um novo jogo.

Com relação à tela de jogo, a mesma pode ser vista na Figura 21. Ali temos todos os elementos, como o nome do nosso oponente e as escolhas possíveis: pedra, papel ou tesoura. Após obter um resultado essa tela se altera levemente, mostrando na esquerda sua escolha, na direita a escolha do adversário e um texto informando se você ganhou ou perdeu.

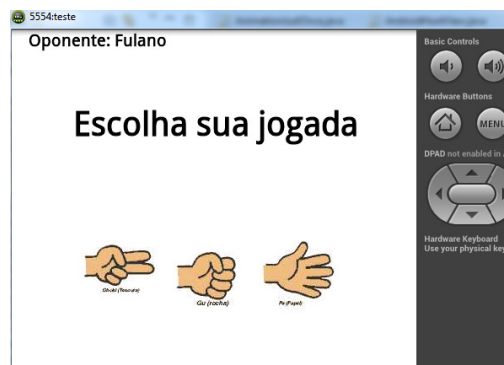


Figura 7 - Tela de jogo Pedra-Papel-Tesoura

4.1.2 Servidor

Para a criação do servidor do jogo, uma das principais classes que deve ser estendida é a classe `Game()`. Nela, nosso principal enfoque será no método `processPlay()`, que é o responsável por receber a opção escolhida de cada jogador, comparar e definir o resultado.

Como sempre é uma partida entre dois jogadores, será criada uma estrutura do tipo `array` com duas posições, que armazenará as escolhas. Toda vez que o método `processPlay` for invocado, a informação é armazenada e é verificado se o `array` está completo. Em caso negativo, irá esperar mais informações, mas em caso positivo significa que já é possível definir um vencedor.

Após enviar o resultado, o servidor irá encerrar a partida e irá reinserir os jogadores na lista de jogadores, já que os mesmos são removidos no início do jogo.

Por se tratar de um jogo simples, a maioria das funções utilizadas já estão implementadas, não sendo necessário que mais classes sejam estendidas ou até mesmo criadas. Em caso de um jogo mais complexo, teríamos que ter uma nova classe com mais constantes e pelo menos uma nova classe como mais funcionalidades para a `Message`.

5. Conclusão

O framework resultante do estudo efetuado foca em retirar do usuário a preocupação com aspectos mais técnicos, como o controle da comunicação entre os jogadores e como é desenhada uma imagem ou animação na tela. Dessa forma, a maior preocupação do usuário do framework é com a lógica do seu jogo, com as regras que determinam como o jogo acontece. Por questão de otimização de bateria do dispositivo e diminuição no tráfego de rede gerado, essas regras de jogo ficam no lado do servidor. Isso também facilita a atualização do jogo ou até mesmo a correção de problemas não encontrados durante a etapa de testes.

Conforme pode ser visto nos dois jogos gerados utilizando o framework, os objetivos determinados no início do projeto foram atingidos, pois o framework é de fácil utilização e poucas são as classes que precisaram ser importadas no desenvolvimento dos jogos de prova.

Como trabalhos futuros, fica a necessidade da implementação de outra opção no controle de conexão, mais adaptada ao cenário de jogos de tempo real. O protocolo escolhido para a implementação atual (TCP/IP) não é o mais adequado para esse tipo de jogos, sendo necessário o uso do protocolo UDP. Outra possível trabalho é a melhoria do sistema de animações, permitindo mais frames, aumentando a fluidez e complexidade da mesma. Finalmente, uma última melhoria seria a criação de um editor de mapas para ser utilizado em jogos do tipo plataforma ou *sidescrolling*.

6. Referências

ANATEL. Linhas ativas. 2014. Disponível em: <<http://www.teleco.com.br/ncel.asp>>. Acesso em: 09 Nov. 2014.

ANDROVICH, Mark. GTA IV: Most Expensive game ever developed?. Games Industry. 2008. Disponível em:

JOHNSON, R. E. How to design frameworks. 1993. Disponível em: <http://www.dsc.ufcg.edu.br/~jacques/cursos/map/recursos/how-design-frame.pdf>. Acesso em Nov. 2014.

PREE, W. Design Patterns for Object-Oriented Software Development. Addison-Wesley, Reading, Mass. 1994.

SILVA, R. P. Suporte ao desenvolvimento e uso de frameworks e componentes. 262f. 2000. Tese (Doutorado em Ciência da Computação) – Universidade Federal do Rio Grande do Sul, Porto Alegre.

WIRFS-BROCK, R.; JOHNSON, R. E. Surveying current research in object-oriented design. Communications of the ACM, New York, 1990.