TÉCNICO LISBOA

First-Person Shooter for Tablets - FpsTab

Oleksandr Bodashko

Thesis to obtain the Master of Science Degree in:

Information Systems and Computer Engineering

Examination Committee

Chairperson: Prof. José Carlos Martins Delgado
Supervisors: Prof. Paulo Jorge Pires Ferreira
Prof. Luís Manuel Antunes Veiga
Member of the Committee: Prof. Alfredo Manuel dos Santos Ferreira Júnior

October 2013

Agradecimentos

Quero agradecer a várias pessoas pela ajuda e disponibilidade ao longo do tempo em que elaborei esta dissertação. Assim, começo por agradecer ao meu orientador, Professor Paulo Ferreira, pelo apoio, incentivo e conselhos dados por ele em todas as fases do desenvolvimento deste trabalho.

Agradeço aos meus amigos que me ajudaram e disponibilizaram o equipamento para os testes. Também agradeço a todos os voluntários que testaram o sistema e deram conselhos acerca do seu melhoramento.

Gostaria ainda de agradecer o suporte financeiro da Fundação para a Ciência e Tecnologia (FCT), através da bolsa de investigação.

Resumo

Atualmente, os dispositivos móveis desempenham um papel importante no nosso dia a dia. O seu elevado potencial torna-os não só em boas ferramentas de trabalho mas também em consolas de jogos. Neste âmbito, os jogos multijogador em redes *ad-hoc* sem fios (onde não existe uma infra-estrutura bem definida nem topologia fixa) colocam vários desafios.

Neste trabalho endereçamos os seguintes objetivos: i) minimizar a quantidade de informação trocada entre os jogadores, poupando a largura da banda, ii) aumentar a escalabilidade do jogo e iii) minimizar o consumo de energia e utilização de recursos do dispositivo. Obviamente, a solução desenvolvida não deve prejudicar a jogabilidade.

Assim, foi desenhado e desenvolvido um sistema de *middleware*, chamado FpsTab, que oferece um conjunto de funcionalidades para os jogos antes mencionados (multijogador em redes *ad-hoc*) do tipo *First-Person Shooter*. O FpsTab oferece uma solução inovadora que permite: i) minimizar os recursos usados pelos dispositivos móveis (i.e. informação enviada/recebida na rede) contribuindo assim para o aumento de escalabilidade do sistema, e ii) garante a consistência dos dados do jogo em cada dispositivo.

O FpsTab foi testado através do porte do jogo *Quake 3 Arena*. Os resultados obtidos no âmbito do consumo de energia e utilização de recursos do dispositivo, poupando a largura da banda sem prejudicar a jogabilidade, são promissores.

Keywords: Dispositivos móveis , Gestão de recursos , Gestão de Interesse , Consistência , Rede *ad-hoc* , Jogo multijogador

Abstract

Nowadays, mobile devices play an important role in our daily lives. Their high performance makes them not only in good working tools but also in gaming consoles. In this context, multiplayer games on *ad-hoc* networks (where does not exist any well defined structure or fixed topology) places multiple challenges.

In this paper we address the following objectives: i) minimize the amount of information exchanged between players for bandwidth saving, ii) increase game scalability and iii) minimize energy consumption and resource utilization of the device. Obviously, developed solutions should not harm the gameplay.

Thus, we designed and developed a middleware software called FpsTab, that offers a set of features for the aforementioned games (multiplayer games on *ad-hoc* networks) for the First-person Shooter genre. The FpsTab offers an innovative solution that allows: i) minimizes the resources used by mobile devices (e.g. data sent/received on the network) thereby contributing to increase of system scalability, and ii) ensures game's data consistency on each device.

FptTab was tested by porting of the game Quake 3 Arena. Obtained results in the scope of energy consumption and resource utilization of the device, that saves bandwidth without affecting the gameplay, are promising.

Keywords: Mobile devices , Resource management , Interest management , Consistency , *ad-hoc* network , Multiplayer game

Índice

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

ADT *Android Developer Tools*

AoI Área de Interesse

API *Application Programming Interface*

FOV Campo de visão

FPS *First-Person Shooter*

FpsTab *First-Person Shooter for Tablets*

JNI *Java Native Interface*

P2P Ponto-a-Ponto

VFC *Vector-Field Consistency*

Capítulo 1

Introdução

Hoje em dia já não imaginamos a vida sem o computador pessoal. Este revolucionou o nosso mundo e tornou-se numa indispensável ferramenta de trabalho, comunicação, armazenamento de dados, entretenimento, etc. No entanto, com o avanço da tecnologia o computador teve a tendencia de se tornar cada vez mais pequeno, até que agora cabe na palma da mão. Surgiram então os dispositivos móveis, tais como *smartphones* e *tablet PCs* [3] que, para algumas tarefas, conseguem substituir os PCs convencionais. Estes dispositivos são pequenos, leves, têm um ecrã de toque e são operados com dedo ou caneta. O seu *hardware* torna-se cada vez mais potente, alguns já possuem processadores com dois ou quatro núcleos e 3D de alto desempenho montados num só *chip* (por exemplo *nVidia Tegra* [28]).

Os dispositivos móveis têm uma vasta gama de aplicação, ligados a *Internet* podem aceder a todos os recursos que a Web 2.0 [31] oferece, como por exemplo, aceder ao correio eletrónico e disco virtual, aceder a redes sociais, comunicar via áudio/vídeo, escrever nos *blogs*, etc. Oferecem a função de livros digitais (*e-book*) e até são usados no ensino como as fontes de informação, substituindo assim os cadernos e livros convencionais(*M-learning* [20]).

O grande desempenho dos GPUs que os *tablet PCs* e os *smartphones* possuem transforma-os em consola de jogos de 2D e 3D com gráficos de alta qualidade. A maioria deles já suporta OpenGL 2.0, Direct3D Mobile e *shaders* programáveis o que permite jogar os jogos com alta qualidade de texturas e sombras, e desfrutar do *gameplay* mais realista [29].

Entre vários géneros de jogos que existem para os dispositivos móveis os jogos de FPS - *First-Person Shooter* têm vindo a ganhar cada vez mais popularidade, nomeadamente os jogos FPS multijogador. Isto deve-se ao facto que o acesso a *Internet* é cada vez mais fácil e mais barato. Estes jogos proporcionam o maior desafio aos jogadores do que os jogos *singleplayer*, visto que, permitem competir com jogadores reais e não somente com os *bots* guiados por Inteligência Artificial.

Num jogo *on-line*, para que este proporcione um bom desempenho é necessário ter em conta dois fatores: consistência e escalabilidade. É crucial manter o estado do jogo consistente em tempo real, ou seja, cada jogador possui uma cópia local do estado do jogo global, onde se encontra a informação

sobre todos os jogadores. Essa manutenção tem um custo, quanto mais jogadores houver no jogo - mais informação é trocada entre eles e maior a largura de banda utilizada. Existem varias técnicas para minimizar a quantidade de mensagens trocadas na rede, tais como Gestão de Interesse [8, 9, 39] baseada em regiões, auras e campo de visão, e também *Dead Reckoning* [15, 33].

A Gestão de Interesse consiste em disseminar as atualizações do estado do jogo apenas para os jogadores que estão interessados nelas. Ou seja, cada jogador tem uma área de interesse associada. Geralmente esta é a área à sua volta, que também se chama aura. Assim, só recebe as atualizações dos jogadores e entidades que se encontram dentro dessa área.

A técnica de *Dead Reckoning* calcula a posição duma entidade baseado na sua velocidade e direção anterior; assim, minimiza o número de atualizações necessárias para a entidade em causa.

Outro fator importante que permite suportar vários jogadores ao mesmo tempo é a escalabilidade. Para que o jogo seja escalável é necessário escolher uma das arquiteturas que mais se adapta ao jogo. As arquiteturas mais usadas em jogos para dispositivos móveis são [2, 23, 30]:

- **Cliente-Servidor** onde um dispositivo, chamado servidor, contem toda a informação do jogo e esta informação é disseminada aos jogadores (clientes).
- **Ponto-a-ponto** onde não existe a definição do nó central, cada nó realiza tanto funções de servidor quanto de cliente.
- **Rede de Servidores** onde existe mais do que um servidor para suportar maior número de ligações.

Os dispositivos móveis são mais flexíveis do que os PCs convencionais, pois usam o meio de comunicação sem fio e permitem aceder à *Internet* em qualquer lugar. Existem várias tecnologias que as permite comunicar entre si. As mais populares são Bluetooth [5, 26, 32], Wi-Fi [17, 25, 37] e 4G [4, 19, 22]. Estas tecnologias suportam redes *ad-hoc* e são interessantes para a tese. Cada uma tem as suas vantagem e desvantagens em relação às outras.

No entanto, os dispositivos móveis possuem um ponto fraco que é o consumo energético. Mesmo que se tornem cada vez mais económicos, usando baterias de carga maior, ecrãs OLED e CPUs de múltiplos núcleos, quanto mais recursos do dispositivo móvel usamos mais bateria se gasta. Os gastos de energia podem ser minimizados ao minimizar e otimizar o uso de recursos dispendiosos. Portanto, importante ter em conta o consumo dos recursos no desenvolvimento de jogos para os dispositivos móveis.

1.1 Objetivos

O primeiro objetivo desta tese é desenvolver um jogo no âmbito da rede *ad-hoc*, para que seja possível jogar sem necessidade de recursos ou infraestruturas adicionais.

O segundo objetivo é minimizar a quantidade de informação trocada entre os jogadores usando as

técnicas de Gestão de Interesse. Para esse efeito, o FpsTab define as Áreas de Interesse (AoI), tais como o campo de visão e a aura, que reduzam significativamente a quantidade de informação necessária para processar o jogo. A solução FpsTab efetua uma filtragem de informação propagada na rede, de modo que, o estado do jogo seja consistente e a jogabilidade não seja prejudicada.

Aumentar a escalabilidade do jogo, de maneira que seja possível manter grande número de jogadores em simultâneo, é o outro objetivo da tese. A escalabilidade não depende apenas dos recursos do servidor do jogo mas também da largura da banda, portanto, o aumento da largura da banda possibilita o aumento da escalabilidade.

Reduzir os gastos energéticos do dispositivo móvel também é um dos desafios da tese. A redução de uso de recursos do dispositivo como CPU, RAM e componente Wi-Fi permitem aumentar a durabilidade da bateria.

O ultimo objetivo é desenvolver a nossa solução na forma duma *middleware*. Podendo assim, criar uma solução mais genérica e reutilizável.

1.2 Desafios

Encontrar um jogo do género FPS com o código aberto para os dispositivos móveis é um dos desafios da tese. Embora existem muitos jogos FPS para os dispositivos móveis com Android OS até agora não existe nenhum que seja *opensource*. Para ultrapassar esse problema temos duas soluções: desenvolver um jogo FPS do raiz, usando uma *engine* 3D ou adaptar um jogo feito para PC ao Android OS. Foi escolhida a segunda solução por seguintes vantagens: i) existe muita oferta de jogos *opensource* do género FPS para os PCs, ii) o conjunto de ferramentas Android NDK permite nos o porte de código escrito em C/C++ para os dispositivos com arquitetura ARM e sistema operativo Android. Contudo, é preciso ter em atenção que o jogo tem de ser suportado pelo *hardware* dos dispositivos móveis, que é mais limitado em relação ao *hardware* dos PCs. O jogo escolhido foi o *Quake 3 Arena*[1], visto que, tem a arquitetura Cliente-Servidor, está implementado em *C* e possui um cliente que permite o porte para sistema operativo Android.

Outro desafio está ligado à natureza dos jogos FPS. Como são jogos de acção é importante que o jogador seja ciente de todos os acontecimentos à sua volta. Não podemos considerar como importantes apenas as entidades que se encontram dentro do campo de visão do jogador. Frequentemente, a posição do inimigo é revelada pelos efeitos sonoros que este imite quando está fora do campo de visão do jogador. Assim sendo, nos jogos FPS é muito importante que o jogador saiba o que se passa nas suas costas e o FpsTab tem isso em consideração.

Por fim, os jogos FPS oferecem mundos virtuais de várias dimensões, desde pequenos com grande oclusão até aos gigantes com grande alcance de visão. Por isso, o FpsTab foi desenvolvido para lidar com qualquer tipo de mundo virtual.

1.3 Contribuições

As principais contribuições desta tese são as seguintes:

- Implementação da biblioteca FpsTab para os jogos FPS que, através das técnicas de Gestão de Interesse, permite: i) minimizar a quantidade de informação disseminada na rede; ii)aumentar a escalabilidade do jogo e assegurar a consistência; iii) reduzir o uso de recursos do dispositivo móvel e aumentar a durabilidade da bateria.
- Melhoramento do jogo *Quake 3 Arena* de forma a acrescentar novas funcionalidades necessárias para o FpsTab.
- Melhoramento do cliente que efetua o porte do jogo para os sistema operativo Android, de forma a acrescentar mais funcionalidades para a interação com o jogo. Foi acrescentado um *joystick* virtual para o controlo do avatar.

1.4 Organização do Documento

Este documento está organizado da seguinte forma. No capítulo 2 descreve-se o Estado da Arte onde se foca nas soluções existentes para resolver os problemas de escalabilidade, consistência e Gestão de Interesse; no capítulo 3 apresentamos a arquitetura do sistema FpsTab; os detalhes sobre a implementação do sistema apresentam-se no capítulo 4; a avaliação e os resultados obtidos encontram-se no capítulo 5 e, por fim, no capítulo 6 apresentamos a conclusão com algumas ideias para trabalho futuro.

[1] http://quake.wikia.com

Capítulo 2

Estado da Arte

Este capítulo começa com a descrição dos jogos para os dispositivos móveis, sobretudo os jogos do género FPS multijogador. Na secção 2.2 vamos falar da comunicação entre os dispositivos móveis, nomeadamente sobre as redes *ad-hoc*, problemas de latência e protocolos de comunicação. A secção 2.3 descreve as arquiteturas mais usadas nos jogos multijogador. De seguida, na secção 2.4 descrevemos o contexto da consistência. Secção 2.5 descreve as técnicas usadas para lidar com a Gestão de Interesse. A secção 2.6 apresenta alguns sistemas existentes que são relevantes para esta tese. Por fim, na secção 2.7 analismos o consumo energético dos dispositivos móveis.

2.1 Jogos móveis

Os jogos para os dispositivos móveis tem vindo a evoluir ao longo dos últimos anos graças aos avanços tecnológicos dos componentes dos dispositivos. Hoje em dia existem jogos de todos os géneros, desde mais simples em 2D até aos mais complexos e de altos gráficos em 3D. Os jogos tornam-se cada vez mais realísticos devido às texturas de alta resolução, sombras dinâmicas, *shaders* programáveis, física realista e os efeitos de *post-processing* que os GPUs dos dispositivos móveis suportam.

O acesso à *Internet* também teve impacto na evolução dos jogos móveis. Com a possibilidade de jogar em modo multijogador *on-line* surgiu um novo género no mudo móvel - *Massive Multiplayer On-line Game* (MMOG) [9]. Os jogos desse género são capazes de suportar centenas de milhares de jogadores em simultâneo.

Entre todos os géneros de videojogos o género *First-Person Shooter* (FPS) tem vindo a ganhar mais popularidade nos dispositivos móveis, nomeadamente, os jogos FPS com modo multijogador.

Os videojogos do género FPS são baseadas em batalhas com armas de fogo, estas podem ser reais ou fantásticas. A principal característica desse género é que o jogador consegue ver o mundo virtual em primeira pessoa, ou seja, através dos olhos do personagem principal, chamado avatar. Geralmente, o jogador apenas vê os braços do avatar e a sua arma, o que é essencial para perceber que esta a jogar

Figura 1: Exemplo dum jogo do género FPS para dispositivo móvel.

em primeira pessoa (Figura 1).

O avatar é controlado pelo jogador através dos periférico como *joystick* ou rato e teclado. No caso dos dispositivos móveis o controlo pode ser feito através do *joystick* virtual que aparece no ecrã (Figura 1) ou através do *hardware* embutido no dispositivo, como por exemplo, acelerómetro ou giroscópio. Cada avatar tem o seu estado associado que se carateriza por posição, pontuação, armas, munições, nível de saúde, armadura, super-poderes, etc. O estado do avatar é alterado através da interação com outros avatares e entidades ou através das próprias ações, como por exemplo, apanhar armas ou *power-ups*.

Muitos jogos FPS possuem o modo de multijogador, onde vários jogadores competem num mundo virtual através da rede. Existem vários modos de competição em modo *on-line*, sendo os mais conhecidos são:

- *Deathmatch* onde os avatares competem uns contra outros com o objetivo de eliminar mais inimigos do que os avatares rivais.
- *Team Deathmatch* onde os avatares estão divididos em duas ou mais equipas. O objetivo de cada equipa é eliminar a equipa adversária ou abater o maior número de avatares rivais.
- *Capture The Flag* onde os avatares têm de apanhar um objeto, geralmente a bandeira, da base da equipa adversária e traze-lo para a sua base.

O jogo decorre num servidor dedicado ao qual os jogadores se conectam. No entanto, um dos jogadores pode ser servidor e cliente ao mesmo tempo. Os avatares ou equipas recebem pontos pelas ações executadas em vários modos. A competição acaba quando o avatar ou a equipa atinge um determinado número de pontuação ou quando acaba o tempo definido pelo servidor do jogo.

Como os jogos FPS estão muito aproximados da realidade, o jogador tem de se basear na perceção visual e auditiva, pois ambas são importantes. Encontrar os inimigos ou outras entidades, que estão fora do limite visual, na maioria das vezes só é possível através da audição. Os eventos sonoros que as entidades emitem, como disparar a arma, apanhar itens, correr ou saltar conseguem revelar a sua

localização relativa no mudo virtual.

Para que os jogadores consigam ver o mundo virtual de forma igual é necessário que o estado do jogo esteja sincronizado entre eles. Para resolver este problema pode ser usada a técnica de replicação que mantém em cada cliente uma cópia do estado global do jogo. A jogabilidade também pode ser afetada por características da rede, nomeadamente a largura de banda e a latência da comunicação.

2.2 Comunicação

A comunicação entre os dispositivos eletrónicos pode ser feita através do cabo ou fibra ótica, que são designados por meio de comunicação com fio, ou através da propagação de ondas de radio - chamado o meio sem fio. O meio sem fio é o mais conveniente para os dispositivos móveis devido à sua flexibilidade e mobilidade, pois os dispositivos não precisam permanecer numa localização fixa. Entre vários protocolos de comunicação existentes os mais usados para os jogos *on-line* são: Bluetooth, Wi-Fi, GPRS, 3G e 4G. Cada um deles tem as suas vantagens e desvantagens dependendo do tipo do jogo e o contexto onde são usados.

2.2.1 Redes *ad-hoc*

O termo *ad-hoc* é geralmente compreendido como algo que é criado ou usado para um problema específico ou instantâneo. Geralmente, numa rede *ad-hoc* não existe uma infraestrutura fixa, como um ponto de acesso, nem controlo centralizado. Cada nó opera na distribuição de modo ponto-a-ponto, atua como um encaminhador independente e gera dados independentes. Os nós comunicam sem conexão predeterminada, podem aparecer em qualquer momento e têm liberdade de movimento. Por isso, a topologia duma rede *ad-hoc* pode mudar duma forma constante e imprevisível. Alguns dos dispositivos da rede podem fazer parte dessa rede apenas durante o tempo da sessão de comunicação ou, no caso de dispositivos móveis, enquanto estão a uma certa proximidade do restante da rede [40].

As redes *ad-hoc* surgiram para os fins militares, mas hoje são usadas para diferentes objetivos, tais como, operações de busca e salvamento para a polícia e os bombeiros, áudio/vídeo conferência, extensão da cobertura das redes infraestruturadas, etc.

Estas redes herdam os problemas tradicionais de comunicações sem fio e redes sem fio, tais como, a perda do sinal causado por obstáculos físicos, interferências causadas por sinais exteriores, etc. A estes problemas ainda acrescentam-se as complexidades e constrangimentos de *design* que são específicos nas redes *ad-hoc* e que colocam muitos desafios [14]. Em termos de transmissão existe a perda de pacotes devido à ausência de gestão de recursos partilhados e frequentes quebras de ligação quando a topologia muda. Existe dificuldade de assegurar a Qualidade de Serviço (QoS) em tempo real e ainda falhas na segurança de comunicação. O consumo de energia também é um dos problemas das redes *ad-hoc*, porque as baterias trazidas por cada nó móvel têm limite. Isto transforma-se num grande problema das redes móveis *ad-hoc* porque, como cada nó age como um sistema de extremidade e num

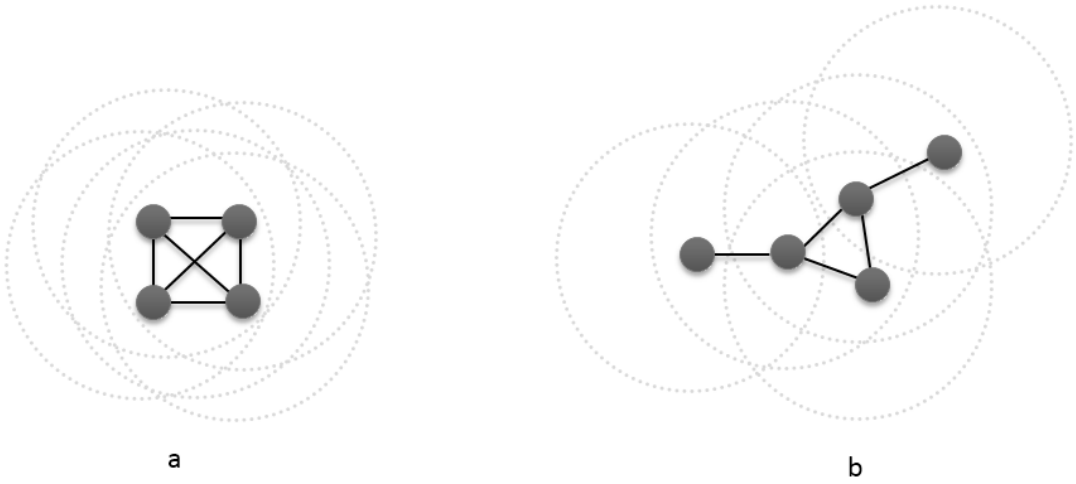

Figura 2: Dois tipos de redes *ad-hoc*: a) comunicação direta e b) múltiplos saltos.

encaminhador ao mesmo tempo, é necessária energia adicional para enviar pacotes de outros nós [18].

As redes *ad-hoc* podem ser subdivididas em duas categorias: redes de comunicação direta e redes de múltiplos saltos (Figura 2). Na primeira, os nós da rede só comunicam com outros nós que estejam dentro do seu raio de cobertura. Já nas redes de múltiplos saltos, os nós móveis comportam-se como as pontes, permitindo que os nós comuniquem mesmo que a distância entre a origem e o destino seja maior do que o raio de cobertura. Consequentemente, as redes *ad-hoc* de múltiplos saltos são mais complexas do que as de comunicação direta [18].

Uma das desvantagens das redes *ad-hoc*, principalmente as de múltiplos saltos, é a grande complexidade dos nós que constituem a rede. Cada nó, além de possuir mecanismos de controlo de acesso ao meio e mecanismos para evitar o problema dos terminais expostos e escondidos, deve atuar como encaminhador. A complexidade do encaminhamento aumenta numa rede *ad-hoc*, pois a topologia da rede é dinâmica.

2.2.2 Latência

Todos os meios de comunicação têm um problema em comum, chamado latência. Por latência [30] designa-se o tempo que a mensagem demora ir dum nó ao outro. Esta varia muito de rede para a rede, depende do seu tipo (com fio ou sem fio) e seu congestionamento. A latência da rede é afetada por *jitters* e perdas de pacotes.

Os *jitters* podem ser definidos como a medida de variação do atraso entre os pacotes sucessivos de dados. Ou seja, a variação de atraso elevada produz uma recepção não regular dos pacotes por parte dos nós. Por sua vez, os pacotes podem não chegar ao seu destino devido à saturação da rede, perda do sinal, falha da rede ou *hardware*.

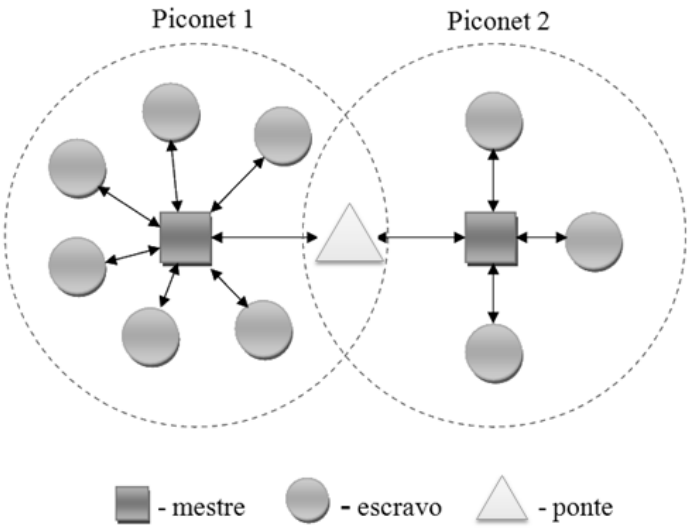

Figura 3: Exemplo duma *scatternet* formada por duas *piconets*.

A latência da rede tem grande impacto nos jogos *on-line* e pode prejudicar a jogabilidade. O jogador cujos pacotes sofrem grande latência está em desvantagem em relação aos outros jogadores. Pois, o estado do jogo desatualizado, que resulta do atraso da disseminação de informação, distorce a perceção da realidade no jogo. O valor de latência máxima para os jogos do género FPS não deve exceder os 75 ms, mas é tolerado até aos 100 ms [10].

No entanto, a latência pode ser compensada de alguma forma usando as técnicas como *Dead Reckoning*, que estima a posição das entidades no mundo, ou *Image Warping* que permite gerar uma nova imagem a partir da imagem anterior.

Nas secções seguintes vai ser falado sobre as tecnologias *ad-hoc* mais utilizadas, como por exemplo Bluetooth, Wi-Fi e 4G.

2.2.3 Bluetooth

Este protocolo de comunicação [5, 26, 32] possui a arquitetura Cliente-Servidor [5, 26]. Um dispositivo mestre pode suportar no máximo sete escravos ligados ao mesmo tempo e esse grupo da rede chama-se *piconet*. Quando o número de dispositivos excede o limite, um dos escravos passa ser o mestre e é criada uma *piconet* nova. As duas *piconets* são ligadas por uma ponte e esse novo conjunto designa-se por *scatternet* (Figura 3). O dispositivo que desempenha o papel da ponte pode ser o mestre e o escravo ao mesmo tempo. Por exemplo, a ponte pode ser o mestre para uma *piconet* e o escravo para outra, ser escravo para ambas, mas nunca um mestre para as duas *piconets*. De referir também, que uma ponte liga no máximo duas *piconets*.

O valor da latência do protocolo depende dos seguintes fatores [1]:

- O intervalo que a mensagem demora ir do escravo ao mestre ou vice-versa é de 625 μs.
- Acumulação de erro da precisão do relógio no mestre e no escravo mais os *jitters* que podem ocorrer em ambos é igual a 7,4 μs (pior caso).

Somando estes fatores verifica-se que o valor total é aproximadamente 0,64 ms o que se enquadra nos requisitos da latência máxima para os jogos FPS.

O protocolo Bluetooth foi desenvolvido para comunicações de curto alcance e proporciona baixo consumo de energia. O alcance depende da classe do adaptador do dispositivo e divide-se em três [32]:

- Classe 1: Alcance até 100 metros.
- Classe 2: Alcance até 10 metros.
- Classe 3: Alcance até 1 metro.

A Classe 2 é usada na maioria de dispositivos móveis. O seu consumo energético é de 50 mW em modo inativo e 94 mW em modo transmissão/recepção. O alcance da cobertura é relativamente pequeno, mas o consumo energético é muito reduzido [5].

A velocidade da transmissão de dados varia consoante a versão do protocolo. As versões mais usadas nos *tablets* e *smartphones* são 2.0 e 3.0. A versão 2.0 permite atingir uma taxa de transmissão acerca de 2.1 Mbps e a versão 3.0 atinge aos 24 Mbps [32]. No entanto, a capacidade total de comunicação de uma *piconet* é cerca de 1Mbps.

Este protocolo protege os dados duma forma segura usando os métodos de *Link Key* que usa uma chave assimétrica, PIN/*Legacy Pairing* - usa um PIN escolhido pelo utilizador para a cifra e *Secure Simple Pairing* que aumenta a segurança adicionando a criptografia ECDH com a chave publica [32].

2.2.4 Wi-Fi

A tecnologia Wi-Fi [17, 25, 37] é muito popular nos dias de hoje e é utilizada na construção de redes *ad-hoc* sem fio. Ganhou a sua popularidade por ser um meio de ligação à *Internet* com grande flexibilidade e baixo custo. É comum encontrar os pontos de acesso (*hotspots*) que permitem aceder à *Internet* ou rede local nos vários lugares, tais como, super-mercados, hospitais, universidades, aeroportos, etc. O equipamento Wi-Fi é incorporado não só nos dispositivos móveis, mas também nos PCs, TVs, *set-top boxes*, consolas de videojogos, máquinas fotográficas e até nos cartões de memória.

A Wi-Fi é baseada no padrão IEEE 802.11 que se divide em 802.11a, 802.11b, 802.11g e 802.11n [17, 37]. Estes são os padrões principais e são utilizadas na comunicação dos dispositivos móveis.

802.11a. Ratificado pelo IEEE (*Institute of Electrical and Electronics Engineers, Inc.*) em 1999. Utiliza *Orthogonal Frequency Division Multiplexing* (OFDM) o que proporciona ao sinal grande resistência à interferência. A sua capacidade máxima de transmissão é de 54 Mbps, com utilização real até 27 Mbps. A cobertura pode chegar até aos 35 metros em espaços fechados e pode atingir aos 120 metros

Módulo Wi-Fi	Modo de transmissão		
	Inativo (mW)	Recepção (mW)	Transmissão (mW)
SX-SDPAG	20	365	904
SN8200	10	363	924
RS9110-N-11-02	3,63	491	660

Tabela 1: Consumo energético dos módulos Wi-Fi em diferentes modos.

em lugares abertos [37, 44]. No entanto, o alcance da transmissão pode sofrer a influência de uma série de fatores, tais como objetos que causam interferência ou impedem a propagação da transmissão a partir do ponto em que estão localizados.

O 802.11a opera na banda ISM (*Industrial, Scientific and Medical*) entre 5,745 e 5,805 GHz e em uma porção da banda UNII entre 5,150 e 5,320 GHz. Isto torna-o incompatível com o 802.11b/g, e uma frequência mais alta implica um alcance menor, comparado com o 802.11b/g, utilizando a mesma potência.

802.11b. Utiliza um tipo de modulação chamado *Direct Sequence Spread Spectrum* (DSSS) para diminuição de interferência e ocupa uma porção da banda ISM entre 2,400 e 2,495 GHz. Possui uma capacidade máxima de transmissão de dados de 11 Mbps, com uma utilização real de cerca de 5 Mbps.

A área de cobertura de uma transmissão 802.11b pode chegar, teoricamente, a 140 metros em ambientes abertos e pode atingir aos 38 metros em lugares fechados [37, 44], tais como escritórios e residências.

802.11g. Provavelmente o padrão de rede sem fio mais utilizado atualmente. O 802.11g utiliza o mesmo espaço de frequência ISM que o 802.11b, mas o esquema de modulação utilizado é o OFDM. Tem a capacidade de transmissão máxima de 54 Mbps (com uma utilização real aproximada de 22 Mbps) e pode diminuir para 11 Mbps DSSS ou ainda menos para a garantia da compatibilidade com o 802.11b. O alcance do 802.11 ronda os 38 metros nos locais fechados e cerca de 140 metros em lugares abertos [37, 44].

802.11n. Tem como principal característica o uso dum esquema chamado *Multiple-Input Multiple-Output* (MIMO), capaz de aumentar consideravelmente as taxas de transferência de dados por meio da combinação de várias vias de transmissão. Isto permite usar dois, três ou quatro emissores e receptores para o funcionamento da rede.

Em relação à sua frequência, o padrão 802.11n pode trabalhar com as faixas de 2,4 GHz e 5 GHz, o que o torna compatível com os padrões anteriores, inclusive com o 802.11a. Sua técnica de transmissão padrão é o OFDM, mas com determinadas alterações devido ao uso do esquema MIMO. Por isso, muitas vezes é chamado de MIMO-OFDM. A sua área de cobertura pode chegar aos 250 metros em ambientes abertos e pode atingir aos 75 metros em lugares fechados [37, 44].

O padrão 802.11n é capaz de atingir taxas de até 600 Mbps, com uma taxa de utilização real de 300 Mbps.

Em comparação com o Bluetooth o Wi-Fi consome mais energia, pois o seu alcance e a taxa de transmissão de dados são maiores. A Tabela 1 mostra o consumo energético dos módulos Wi-Fi que são usados nos dispositivos móveis. Estes são: SX-SDPAG [42],SN8200 [41] e RS9110-N-11-02 [38].

Após o calculo da média dos valores da Tabela 1 podemos concluir que, em modo inativo o consumo é de 11 mW, em modo de recepção ronda os 406 mW e em modo de transmissão consome cerca de 829 mW.

A tecnologia Wi-Fi é considerada segura, pois protege o acesso à sua rede e protege os dados que são transmitidos nela. Existem diferentes mecanismos de segurança para esse fim, sendo os principais são: WEP, WPA e WPA2 [27]. WEP consiste num mecanismo de autenticação que funciona de forma fechada ou aberta através do uso de chaves simétricas e utiliza o algoritmo de criptografia RC4. WPA é mais seguro que o WEP por se basear num protocolo chamado *Temporal Key Integrity Protocol* (TKIP) onde a chave é trocada periodicamente. Por fim, WPA2 é um mecanismo que oferece alto grau de segurança; entretanto, tem como deficiência a alta taxa de processamento, o que pode prejudicar o desempenho do equipamento em que opera. Por essa razão o WPA2 não é aconselhado para ser usado nos dispositivos móveis.

Em termos de arquitetura, a Wi-Fi opera em 4 modos: modo infraestrutura, modo cliente, modo monitor e modo *ad-hoc*. No último modo é criada uma rede multiponto-para-multiponto e cada dispositivo comunica diretamente com os vizinhos. Os nós devem estar ao alcance para que se comuniquem e devem estar de acordo quanto ao nome da rede e o canal utilizado [17].

A latência nas redes Wi-Fi depende do padrão que está a ser usado. Para o 802.11a o tempo que a mensagem demora ir dum nó ao outro é cerca de 34 μs, 50 μs para o 802.11b e 28 μs para o 802.11g [25]. No entanto, a latência pode aumentar devido aos *jitters* e estado da rede, mas em geral os valores mais do que se enquadram nos requisitos de jogos FPS.

2.2.5 4G

A 4G [4, 19, 22] é uma tecnologia inovadora no mundo móvel. Esta é a sucessora direta da 3G, mas em relação à ultima traz avanços significativos principalmente na velocidade de conexão e no carregamento de dados. Além disso, a 4G é uma rede móvel *ad-hoc*, o que não acontecia com a 3G. O principal objetivo de 4G é unir os serviços da rede celular sem fio e serviços de *Internet* numa só tecnologia que possibilita uma ligação à rede em qualquer lugar, a qualquer hora e de qualquer forma de conectividade [22]. No entanto, este serviço não é gratuito.

Os pontos fortes do 4G são: alta velocidade de transmissão de dados, grande capacidade da rede, os serviços multimédia baseados no protocolo IP e pouca latência.

A largura da banda nas redes 4G ultrapassa os 100 Mbps com a latência de 40 ms no máximo [4] o que torna esta tecnologia num forte concorrente ao Wi-Fi. O consumo energético médio de 4G é de 1080 mW, mas este aumenta à medida que a taxa de transmissão aumenta e pode chegar aos 3300

	Bluetooth	Wi-Fi	4G
Largura da banda (Mbps)	1	22 - 300	100
Alcance (m)	10	38 - 250	operadora móvel
Latência (ms)	0,64	0,028 - 0,05	40
Consumo energético médio (mW)	94	406 - 829	1080 - 300
Modo *ah-hoc*	sim	sim	sim
Segurança	sim	sim	sim
Gratuito	sim	sim	não

Tabela 2: Comparação das tecnologias: Bluetooth, Wi-Fi e 4G.

mW quando a taxa atinge o valor máximo [19].

2.2.6 Comparação das tecnologias de comunicação

Cada uma das tecnologias mencionada tem as suas vantagens e desvantagem. Para compara-las vamos usar como critérios os requisitos necessários para a tese. Os requisitos são:

- Largura da banda no mínimo 1 Mbps.
- Alcance pelo menos 10 metros.
- A latência da rede não deve ultrapassar os 75 ms.
- O consumo energético tem de ser o menor possível para aumentar a durabilidade do dispositivo móvel.
- Funcionar em modo *ad-hoc*.
- Possuir mecanismos de segurança.
- Acesso ao meio tem de ser gratuito.

A Tabela 2 mostra a comparação das três tecnologias de comunicação em função dos requisitos especificados para esta tese.

A tecnologia Wi-Fi, em comparação com as outras, tem a melhor relação entre consumo energético e largura da banda. A Wi-Fi satisfaz todos os requisitos do FpsTab, a pesar de não ter o menos consumo energético, e deve permitir uma boa jogabilidade para os jogos FPS.

O Bluetooth é a tecnologia que gasta menos energia do que os concorrentes ao transmitir ou receber os dados, mas é mais adequado para a comunicação ponto-a-ponto a curtas distâncias. Também, a largura da banda duma *piconet* é no máximo 1 Mbps, ou seja, 125 Kbp por jogador. Assim sendo, para um número de jogadores que ultrapassa 8 a largura da banda pode não ser suficiente para permitir

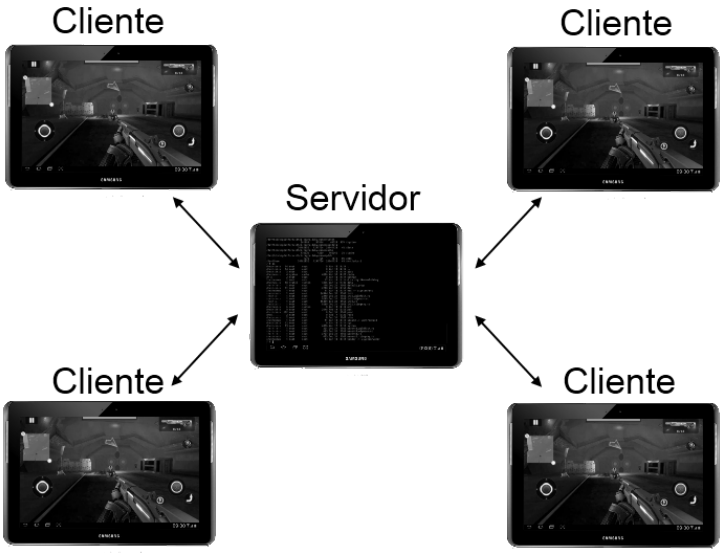

Figura 4: Exemplo duma arquitetura Cliente-Servidor.

boa jogabilidade, pois a comunicação com o servidor vai ser através da ponte. Outra desvantagem é o alcance (10 metros nó máximo) que ainda pode ser reduzido devido aos obstáculos.

Por ultimo, a 4G também foi rejeitada, visto que, tem consumo de energia e latência elevados, e também, o acesso ao meio não é gratuito.

2.3 Arquiteturas

2.3.1 Cliente - Servidor

Nesta arquitetura o processamento da informação é dividido em dois módulos distintos. O primeiro módulo, chamado servidor, é responsável pela manutenção da informação e o segundo, designado por cliente, é responsável pela obtenção dos dados. O servidor é responsável por todas as comunicações entre os clientes. Ou seja, os clientes não comunicam diretamente entre si (Figura 4). Geralmente, nesta arquitetura o cliente faz um pedido ao servidor que o processa e envia a resposta ao cliente.

No entanto, existem duas abordagens dessa arquitetura onde a lógica do jogo esta centrada no servidor ou no cliente.

A **abordagem centrada no servidor** [23] processa toda a lógica do jogo no servidor, os clientes apenas enviam os *inputs* ao servidor e limitam-se a apresentar o conteúdo recebido [16]. Existem vários métodos para apresentar este conteúdo, um deles é o padrão MPEG4 [7]. Este padrão tem características interessantes, tais como a compressão de dados enviados permitem poupara a largura da banda, separação do conteúdo em camadas possibilita ao cliente receber a informação que só lhe faz respeito e permite suportar o jogo nos dispositivos com diferente *hardware*.

Um ponto forte da abordagem centrada no servidor é que os jogadores não conseguem usar *cheats*,

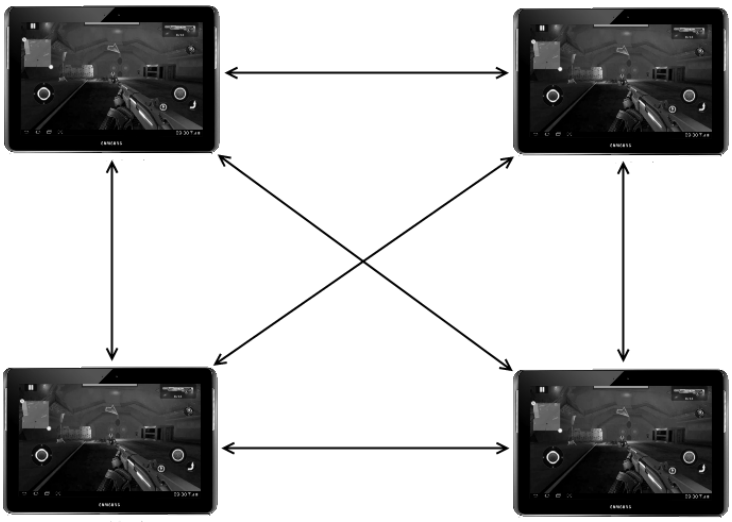

Figura 5: Exemplo duma arquitetura Ponto-a-Ponto.

que lhes dão uma vantagem ilegal sobre os outros jogadores, pois não têm acesso direto ao estado global do jogo.

A sincronização entre os jogadores não é necessária, visto que, todos vêm o jogo no mesmo estado, mas a jogabilidade depende fortemente da latência da rede. Portanto, dois jogadores podem ver o jogo em estados diferentes se a latência dum for maior que a latência de outro. De ponto de vista do consumo energético, esta abordagem é boa para os jogadores (clientes), pois não necessitam de usar muitos recursos para processar o jogo [30]. Quando os clientes estão conetados ao servidor, numa rede sem fios, os atrasos nessa rede podem variar consideravelmente e podem ocorrer os chamados *jitters*. Assim, a informação que é emitida pelo servidor pode chegar aos clientes em alturas diferentes o que causa uma inconsistência entre eles. O servidor tem que emitir os dados com muita frequência para que essa inconsistência diminua, mas isso provoca o aumento da utilização da largura da banda e como resultado torna a arquitetura menos escalável.

A **abordagem centrada no cliente** [23], também se chama *fat client*, tem a lógica do jogo centrada nos clientes e o servidor apenas reenvia as mensagens vindas dos clientes. Esta abordagem minimiza a quantidade de informação difundida na rede e retira a carga do servidor. No entanto, tem desvantagens em relacção à abordagem anterior. A primeira é a dificuldade de desenvolver um jogo que suporta vários dispositivos com diferentes capacidades de memória, resoluções de ecrã diferentes e meios de comunicação diferentes. A segunda, possibilita usar *cheats*, ou seja, alterar o estado do jogo para obter a vantagem sobre outros jogadores. Por fim, o tempo que a mensagem demora a passar dum cliente para outro aumenta para o dobro em comparação com a primeira abordagem, pois a informação passa pelo servidor. Isso pode causar problemas com a consistência do jogo se a latência dos jogadores for muito diferente.

Em termos de escalabilidade a arquitetura Cliente-Servidor é escalável. Mesmo que o número de jogadores suportados por um único servidor é limitado, basta adicionar mais recursos, ou seja, mais servidores, para que esta se torna escalável.

2.3.2 Ponto-a-Ponto

A arquitetura Ponto-a-Ponto (*Peer-to-Peer* ou P2P) [30] é uma arquitetura onde as funções são descentralizadas na rede, portanto, não existe um servidor central, cada nó é servidor e cliente ao mesmo tempo. Nesta arquitetura os nós ligam-se uns aos outros e trocam as mensagens entre si. A função do servidor é distribuída entre os nós o que possibilita manter o estado do jogo consistente nos casos de um dos nós falhar ou sair do sistema (Figura 5).

Existem três arquiteturas P2P tradicionais para as redes móveis: centralizada, descentralizada e híbrida. Na arquitetura centralizada existe um nó que é um servidor dedicado ou é um servidor de indexação. Este nó tem como objetivo controlar toda a rede e manter um índice dos nós conectados e os seus recursos. A arquitetura descentralizada não possui o ponto de controlo, cada nó da rede tem as funcionalidades iguais. Toda a informação de controlo da rede e as funções de comunicação estão distribuídas entre os nós. Por fim, a arquitetura híbrida une o melhor das duas arquiteturas descritas em cima, combinando a eficiência e flexibilidade de ambas [2].

A latência na rede diminui porque a comunicação passa a ser direta, mas com o aumento de número de jogadores a quantidade de mensagens também aumenta. Este aumento é exponencial numa progressão n(n-1), onde n é o número de jogadores, o que quer dizer que, um jogador pode ficar sem largura da banda.

Existem soluções académicas que permitem usar a arquitetura P2P nos jogos MMOG e suportar grande quantidade de jogadores ao mesmo tempo. No entanto, a quantidade de jogadores suportada por esta arquitetura é limitada. Torna-se impraticável a comunicação entre os nós se a sua quantidade for muito elevada. Portanto, a arquitetura P2P não é escalável [24].

Como a lógica do jogo situa-se nos terminais dos jogadores, estes podem facilmente usar *cheats* para obter alguma vantagem ilegal. Para ultrapassar esse problema é necessário usar algum mecanismo de prevenção de batota, como por exemplo o *PunkBuster* [2].

Por fim, a consistência do estado do jogo também não é assegurada e é necessário usar algum mecanismo de sincronização entre os nós.

2.3.3 Rede de Servidores

Em geral esta arquitetura é igual a Cliente - Servidor com a diferença de haver mais do que um servidor para fornecer os clientes. Os servidores comunicam entre eles e cada um tem ligações com os clientes. Isto permite suportar mais ligações e não sobrecarregar um único servidor. Cada servidor pode conter apenas uma parte do mundo virtual e o avatar simplesmente migra dum servidor para outro. Isto levanta alguns problemas da consistência entre os servidores. Esta abordagem é mais adequada aos jogos MMOG do que FPS, pois nos jogos MMOG o número de jogadores situa-se nos milhares.

[2] http://www.punkbuster.com

2.4 Consistência

Um dos grandes desafios dos jogos multijogador é manter o estado do jogo consistente, de maneira que todos os jogadores consigam ver o mundo virtual de forma igual. Para tal, os dispositivos dos jogadores têm de ser atualizados frequentemente com as informações importantes que provem dos outros jogadores. Como já foi referido na secção 2.3, quanto mais informação sobre o estado dos jogadores é disseminada na rede, mais largura da banda e energia se gasta. Contudo, não é necessário que o jogador receba essa informação na íntegra para que possa jogar sem perder a jogabilidade.

Existe um mecanismo chamado replicação [35], que se baseia em guardar uma copia do estado do jogo no dispositivo do jogador. Isto permite poupar o tempo de acesso ao servidor (ou outro dono de informação), visto que o acesso local é mais rápido. O cliente acede somente à sua réplica local que necessita de ser atualizada com a meta-informação dos outros jogadores. Caso a réplica fica desatualizada durante muito tempo o jogador não sabe o que acontece à sua volta. Para manter as réplicas consistentes existem mecanismos de replicação. Estes decidem com que frequência as réplicas devem ser atualizadas e resolvem os conflitos que surgem quando vários clientes alteram o estado global ao mesmo tempo.

As réplicas podem ser geridas através de duas abordagens: replicação pessimista e replicação otimista [35].

Na replicação pessimista o acesso à réplica é bloqueado quando um dos clientes está a alterar o seu conteúdo e só é permitido quando a réplica fica atualizada. Por exemplo, quando um cliente atualizou a sua réplica, outros clientes são notificados que existe atualização e são proibidos de aceder às suas réplicas locais enquanto estas estão a ser atualizadas. Com esta abordagem as réplicas estão sempre atualizadas em todos os clientes, mas a sincronização de vários clientes remotos demora muito tempo.

A replicação otimista não bloqueia o acesso à réplica enquanto está a ser atualizada. Além disso, autoriza a cada cliente atualizar a sua réplica de forma independente. A atualização das outras réplicas pode não ser feita de imediato e é feita em *background*. Os conflitos são resolvidos quando surgem. Esta abordagem é mais flexível e é escalável em comparação com abordagem pessimista. Contudo, não existe a certeza de que a réplica do cliente contém a informação mais recente.

2.5 Gestão de Interesse

As atualizações que os clientes recebem normalmente são genéricas e contêm muita informação desnecessária. A Gestão de Interesse (*Interest Management*) [8, 39] tem como objetivo filtrar a informação de maneira que, os clientes recebem só o que é relevante para eles. Por exemplo, se um jogador virtual, chamado avatar, fica muito longe de outro no mundo virtual e, além disso, eles não se conseguem ver um ao outro devido os obstáculos. Por isso, não há necessidade de trocar informação entre eles, pois ações dum não alteram o estado do outro. Esta técnica permite poupar a largura da banda e diminuir o

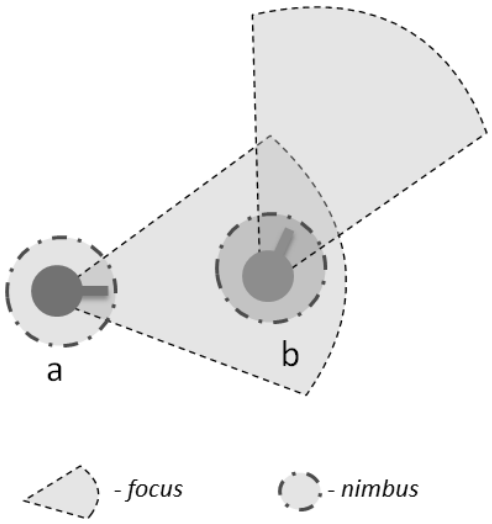

Figura 6: Exemplo de intercessão do *focus* do jogador *a* e *nimbus* do jogador *b*.

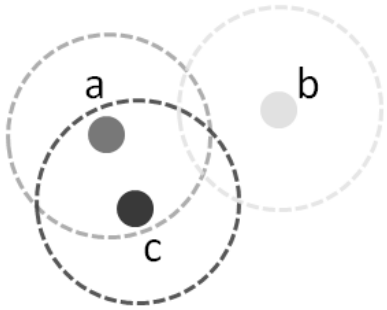

Figura 7: Exemplo de Gestão de Interesse baseada em auras.

número de atualizações das réplicas.

A filtragem de informação é feita através de divisão do mundo virtual em Áreas de Interesse (AoI - *Area of Interest*). O avatar só recebe a informação por completo da AoI na qual está interessado. Do resto do mundo limita-se a receber apenas mensagens mais importantes que são, por exemplo, estado da vida e pontuações. No entanto, o jogador pode não receber informação nenhuma, dependendo da lógica do jogo. A AoI pode ser dividida em *focus* e *nimbus*. O primeiro representa a perceção do observador, ou seja, o campo de visão do avatar e o *nimbus* representa a área dentro da qual o avatar pode ser visto. Ou seja, a interação entre os avatares ocorre quando o *focus* dum interseta o *nimbus* de outro [39] (Figura 6).

2.5.1 Gestão de Interesse baseada em auras

Nesta abordagem a AoI é o circulo desenhado à volta do avatar e designa-se por aura. O raio da aura normalmente corresponde a distância máxima do limite sensorial do avatar, como por exemplo, visão e

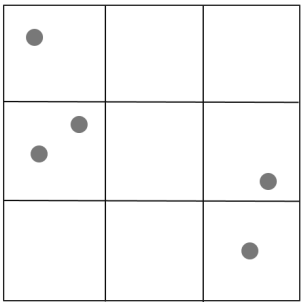

Figura 8: Exemplo de Gestão de Interesse baseada em regiões.

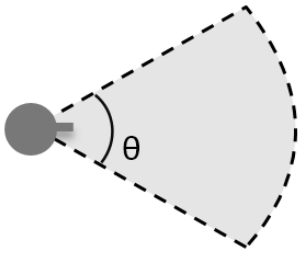

Figura 9: Exemplo de Gestão de Interesse baseada em campo de visão, onde θ representa o ângulo de abertura.

audição. Baseia-se na distância Euclidiana entre os dois pontos e não tem grande peso computacional [9]. A interação entre os avatares ocorre quando um deles entra na aura do outro. Como podemos observar na Figura 7, a interação apenas acontece entre o avatar *a* e *c*.

2.5.2 A Gestão de Interesse baseada em regiões

A gestão de interesse baseada em regiões divide o mundo virtual em segmentos (Figura 8). Assim, o avatar só recebe as atualizações da região na qual está interessado, geralmente da região onde ele se encontra. O mundo pode ser dividido de varias formas: em retângulos, hexágonos ou triângulos.

A divisão em hexágonos tem a vantagem sobre a divisão em retângulos porque a forma hexagonal aproxima-se melhor ao circulo da AoI (aura) do avatar. A vantagem da divisão triangular é que consegue excluir os obstáculos quando particiona o mundo, o que reduz a AoI mas pode aumentar o peso computacional [9].

É preciso ter em conta que quando mais particionado for o mundo mais precisa vais ser a AoI, mas a quantidade de informação disseminada também vai aumentar.

2.5.3 Gestão de Interesse baseada em campo de visão

Esta abordagem defina uma AoI chamada campo de visão (FOV - *Field of View*). Esta AoI é semelhante ao campo de visão humana e representa a perceção visual do avatar [8, 9]. O ângulo de abertura do campo de visão normalmente corresponde ao FOV definido no jogo, que tipicamente é de 90° (Figura 9).

A Gestão de Interesse baseada em campo de visão reduz significativamente a AoI do jogador. Além disso, consegue filtrar as entidades que ficam fora do campo de visão, ou seja, as entidades que se encontram nas costas do avatar. Esta abordagem não tem grande peso computacional e é adequada para os jogos com muita oclusão, como por exemplos os jogos FPS.

2.5.4 *Dead Reckoning*

A técnica *Dead Reckoning* [15, 33] permite reduzir a quantidade de mensagens trocadas na rede. Esta abordagem consegue calcular uma nova posição duma entidade baseando-se na sua posição, velocidade, aceleração e orientação anteriores. Isto permite não só mascarar a latência, mas também reduzir o número de comunicações com o servidor.

A predição nem sempre é certa e frequentemente a entidade, que para o nosso jogador encontra-se numa determinada posição, na realidade encontra-se noutra. Esta discrepância pode não ser muito significativa mas é notável e pode transtornar a jogabilidade.

Esta técnica serve perfeitamente para calcular a posição das entidades com uma trajetória bem definida e que não muda bruscamente. No entanto, também está a ser usada nos jogos FPS onde existe muito mais movimento imprevisível. Por exemplo, o jogo *Quake 3 Arena* usa esta técnica para predizer a posição dos projeteis e avatares antes que a réplica do jogador seja atualizada com a informação do servidor.

2.6 Soluções existentes

Nesta secção vão ser descritos alguns sistemas académicos para assegurar a consistência do jogo e minimizar o fluxo de informação na rede.

2.6.1 ANGEL

O sistema ANGEL [21] é um *middleware* de gestão de jogos multijogador para dispositivos móveis. Baseada na arquitetura P2P híbrida mantém todos os dispositivos sincronizados assegurando assim a consistência do jogo.

Nesta arquitetura todos os nós trocam entre si pequenos pacotes que contêm informações sobre a carga do nó, frequência de perda de pacotes e ainda servem para calcular a latência entre os nós. Na base desses cálculos é escolhido um nó mestre, que têm a menor latência e menor perda de pacotes. Os outros nós enviam periodicamente a informação do seu estado ao nó mestre. Por sua vez o nó mestre constrói uma árvore de encaminhamento e escolhe o melhor caminho para passar informação aos nós terminais.

A perda de pacotes no ANGEL é bastante baixa e ronda os 5%; no entanto, existe algum aumento de

tráfego na rede em relação a P2P híbrida comum e chega aos 8%. Uma das principais desvantagens deste sistema é a ausência de Gestão de Interesse.

2.6.2 MORAP

Este sistema foi desenvolvido para os jogos MMOG e consegue suportar grande número de jogadores. MORAP [34] basea-se na arquitetura P2P descentralizada e cada nó cria dinamicamente uma ligação com o seu nó vizinho. Assim, um jogador comunica diretamente com outro sem precisar de nó coordenador. Ou seja, para poupar largura da banda, os jogadores comunicam entre sim quando os seus avatares se encontram na mesma AoI.

Esta abordagem usa divisão hexagonal para fragmentar o mundo virtual. Cada hexágono designa-se por célula e pode conter três tipos de nós: mestre, escravo e inicial. O nó mestre e nó escravo são posicionados na mesma célula e tem de haver no mínimo um nó mestre por célula. O nó inicial é um nó virtual e serve como identificador da célula. Quando um nó entra na célula passa a ser o mestre, caso o mestre ainda não existe, ou escravo se o mestre já existe.

O nó mestre tem o papel crucial na Gestão de Interesse. Este comunica com os nós mestres das células vizinhas e notifica os nós escravos que se encontram dentro da mesma AoI. Por sua vez, os nós escravos apenas comunicam com o nó mestre da sua célula e enviam a informação quando o seu estado se altera. Os nós têm a liberdade de se movimentar no mundo virtual e podem trocar o seu papel.

A AoI é baseada em auras e pode cobrir mais do que uma célula mas é sempre mais pequena do que a área que o nó mestre consegue cobrir.

MORAP é um sistema escalável, tolerante a falhas e minimiza a quantidade de informação na rede graças a sua arquitetura. No entanto, a sua técnica de Gestão de Interesse não é muito eficiente.

2.6.3 Arquitetura Cliente-Servidor adaptável

Esta abordagem [23] é baseada na arquitetura Cliente-Servidor e em tempo real separa a lógica do jogo entre o servidor e cliente, baseando-se no contexto e latência da rede. Nos casos de grande latência a maior parte da lógica do jogo é executada nos clientes. Quando existe a largura da banda suficiente a lógica do jogo é executada no servidor, os clientes apenas limitam-se a receber a imagem do mundo virtual e enviam os *inputs* ao servidor.

Em casos de pouca latência na rede a maior parte da lógica migra para o servidor e o cliente limita-se a apresentar o conteúdo vindo do servidor. A consistência é assegurada e uma pequena discrepância de velocidade ou posição das entidades é tolerada.

Para que o jogo seja consistente usam-se as técnicas de Gestão de Interesse baseadas em regiões e aura. Quando um avatar se situa longe do outro e o seu movimento é lento basta receber apenas

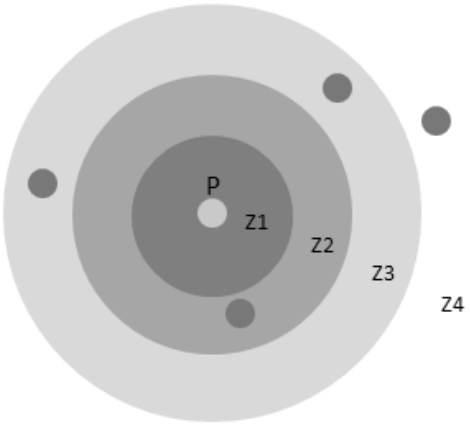

Figura 10: Exemplo do VFC com quatro níveis de consistência. Onde P é o *pivot* e Z1-Z4 são as zonas de consistência. A intensidade da cor representa o nível de consistência.

as atualizações mais importantes do estado do jogo. Por outro lado, quando muitos avatares se encontram na mesma AoI, passam a receber toda a informação do estado dos outros para assegurar a consistência.

Esta abordagem também usa a técnica de *Dead Reckoning* no lado do cliente para calcular a nova posição dos avatares baseado na velocidade, direção e posição anterior.

2.6.4 Vector-Field Consistency (VFC)

VFC [36] é um mecanismo que permite reduzir a quantidade de informação disseminada na rede e manter o estado do jogo consistente sem prejudicar a jogabilidade. A Gestão de Interesse do VFC é baseada em auras, mas existe mais do que uma aura para cada avatar. O número de auras corresponde aos níveis de consistência. Assim, o nível de consistência mais forte se situa dentro de aura mais pequena do avatar. A medida que a distancia entre os níveis e o avatar aumenta a consistência diminui. A Figura 10 ilustra os níveis de consistência, onde Z1 é a zona com nível de consistência mais forte e Z4 é a zona com nível de consistência mais fraca. O VFC dinamicamente fortalece ou enfraquece a consistência, o que permite reduzir a quantidade de informação recebida pelo avatar.

À volta do *pivot*, que pode ser o avatar ou outra entidade, são criadas as zonas de consistência. Estas zonas são as circunferências concêntricas e sua quantidade é especificada pelo programador do jogo. Dentro de cada zona as entidades pertencem ao mesmo nível de consistência. Assim, os avatares ou entidades mais distantes vão ter menos importância para o *pivot*, ou seja, o *pivot* pode não se quer receber a sua informação.

O VFC baseia-se na abordagem de consistência otimista o que significa que as réplicas locais de cada avatar podem divergir um pouco do estado do jogo global. Este sistema tem um mecanismo que consegue gerir as divergências de cada réplica. Os níveis de consistência consistem em vectores tridimensionais que definem os limites de divergência das réplicas locais no tempo, sequência e valor.

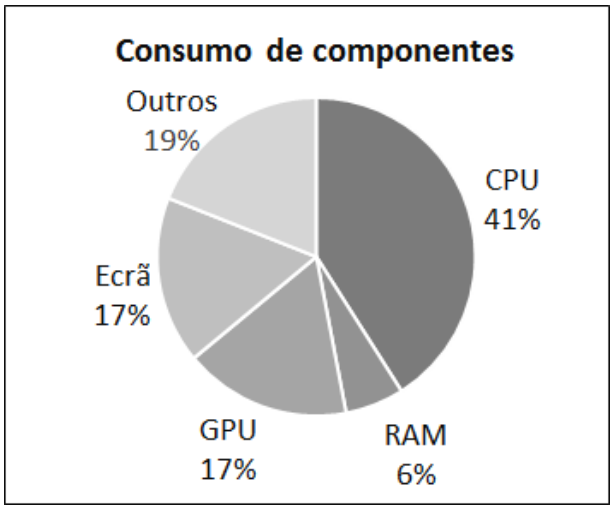

Figura 11: Consumo energético de componentes durante o jogo.

- **Tempo** - define o tempo máximo que cada réplica pode permanecer sem atualização. Este tempo, medido em segundos, começa a contar após a ultima atualização da réplica.
- **Sequência** - define o número máximo de atualizações que a réplica pode ignorar até ser atualizada novamente.
- **Valor** - define a diferença máxima relativa entre o conteúdo das réplicas. Ou seja consulta quanto o estado do jogo atual é diferente da última versão propagada e é medida em percentagem.

Quando uma destas variáveis é excedida a réplica necessita duma atualização. No entanto, se atribuir um valor infinito a qualquer um dos parâmetros, o seu efeito é considerado nulo.

O VFC possui de duas generalizações *multi-pivot* e *multi-zones*. A primeira consiste na utilização de múltiplos *pivots* na mesma vista, e a segunda permite que diferentes conjuntos de entidades com requisitos de consistência diferentes possam ter associadas zonas diferentes.

O VFC oferece garantias de consistência nas redes com pouca largura da banda, é flexível e diminui a utilização da rede.

2.7 Consumo energético

A durabilidade da bateria do dispositivo móvel depende muito da forma como este é utilizado. Quanto mais recursos do dispositivo móvel usamos, por exemplo, CPU, GPU, GPS, Wi-Fi, mais bateria se gasta.

Devido à alta qualidade de computação gráfica processada por CPU/GPU e elevados requisitos da resolução, os videojogos são as aplicações que mais energia consomem. Durante o jogo com pouca interação do utilizador, o consumo total do dispositivo móvel varia entre 1140 mW e 1750 mW. Nos jogos que envolvem muita interação, como toques frequentes no ecrã e uso de sensores como acelerómetro

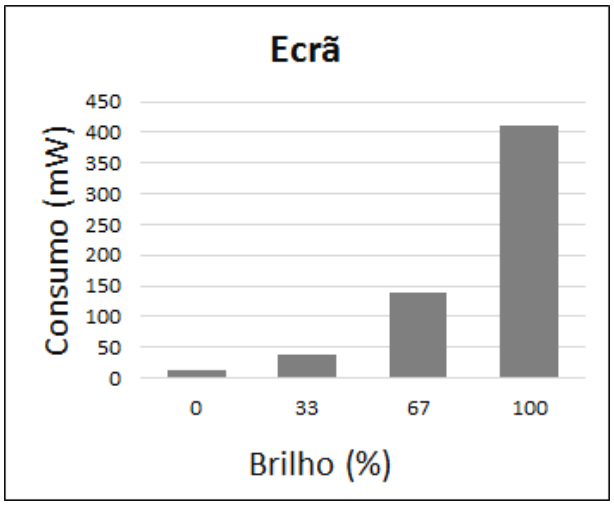

Figura 12: Consumo do ecrã em função do seu brilho.

ou giroscópio, os consumos aumentam até aos 1640 - 2220 mW. Para perceber a grandeza desses valores, a navegação na *Internet* consome apenas 600 - 1500 mW [13].

O CPU consome mais energia em comparação com outros componentes. Devido aos cálculos de processamento do jogo chega consumir até 41% da bateria, o que é superior ao consumo do GPU (Figura 11).

Outro componente que consome muita energia nos *tablets* e *smartphones* é o ecrã. Este tem vindo a evoluir ao longo do tempo e cada vez se torna mais económico. Por exemplo, o ecrã com a tecnologia AMOLED consome em média menos 60% da energia do que os ecrãs LCD/TFT. No entanto, o consumo do ecrã depende muito do seu brilho e quanto mais brilhante for - mais energia gasta. Como podemos observar a Figura 12 o consumo varia entre 12 mW, para o ecrã sem brilho, e 410 mW para o ecrã com brilho no máximo [11, 43]. No entanto, durante o videojogo este consome apenas 17% da bateria do dispositivo (Figura 11).

Como foi referido na secção 2.2 os componentes de comunicação, tais como Wi-Fi e Bluetooth, também consomem energia, principalmente no envio e recepção de dados. A taxa de transmissão também tem impacto no consumo. Por exemplo, no caso de Wi-Fi o aumento da taxa de transmissão de dados por um fator de 10 causa o aumento de consumo energético por um fator de 1,8 [6]. Durante uma hora de funcionamento do Wi-Fi gasta-se mais de 18% da bateria ao transmitir e receber os dados [12].

2.8 Conclusão

Neste capítulo falámos sobre os jogos móveis, nomeadamente, sobre os jogos do género FPS.

Falámos também sobre as redes *ad-hoc* e sobre os desafios que estes colocam para os jogos *on-line* multijogador. Descrevemos e comparámos os protocolos de comunicação sem fio usados por dispositivos móveis, tais como, Wi-Fi, Bluetooth e 4G.

Apresentámos várias maneiras para lidar com a escalabilidade e consistência dos jogos multijogador. Descrevemos as arquiteturas mais usadas nos jogos multijogador, como Cliente-Servidor e P2P. Referimos sobre as técnicas, chamadas Gestão de Interesse, que permitam reduzir a quantidade de informação disseminada na rede, poupando a largura da bada.

Mencionámos algumas soluções existentes, como ANGEL, MORAP, Arquitetura Cliente-Servidor adaptável e VFC, que lidam com replicação e consistência dos jogos móveis. Descrevemos as suas vantagens e desvantagens.

Por fim, analisámos o consumo energético dos dispositivos móveis durante o seu funcionamento, e sobre tudo, durante os jogos.

Capítulo 3

Arquitetura

Este capítulo começa por descrever os fatores que influenciaram o desenho da arquitetura do nosso sistema, a escolha do jogo e tecnologia de comunicação. De seguida, descrevemos em detalhe a arquitetura do FpsTab e seu funcionamento.

3.1 Introdução

Ao longo do desenvolvimento do sistema FpsTab vários fatores tiveram influência na sua arquitetura. Como está implementado para os jogos *on-line* multijogador é crucial que seja possível jogar em qualquer lugar e sem necessidade de recursos externos, ou seja, em modo *ad-hoc*.

Tivemos em consideração os requisitos dos jogos do género FPS, onde existe muito movimento e a perceção da realidade depende tanto de visão como de audição. Para isso, é necessária frequente troca de grande quantidade de informação entre os jogadores, pois o estado global do jogo tem de ser consistente. Estes jogos também são muito afetados por latência da rede que pode prejudicar a jogabilidade.

A escalabilidade da arquitetura Cliente-Servidor não depende apenas das capacidades do servidor, mas também da largura da banda do canal de comunicação. Como foi referido no capítulo 2 o aumento de número de jogadores causa grande aumento de informação disseminada na rede. Este aumento pode causar problemas de jogabilidade se a largura da banda for pouca. No entanto, existem as técnicas, como Gestão de Interesse, que conseguem reduzir significativamente o tráfego na rede.

Considerámos também o consumo energético do dispositivo móvel. Como explicámos no capítulo 2 a redução de uso dos recursos do dispositivo, como CPU, GPU, Wi-Fi pode aumentar a durabilidade da bateria.

A ideia fundamental do nosso sistema é baseada no uso das técnicas de Gestão de Interesse. Definimos as áreas de interesse como campo de visão, que cobre a área da perceção visual do jogador, e aura, que define a perceção auditiva. Contudo, adotamos a ideia do sistema VFC [36] e definimos

vários níveis de consistência por cada AoI. Isso permite que o FpsTab reduza a consistência de forma gradual, consoante a posição da entidade em relação ao jogador. Assim, conseguimos obter um melhor *trade-off* entre filtragem de informação e jogabilidade.

3.2 A escolha do jogo

Existem várias centenas de jogos de vários géneros para o sistema operativo Android. No entanto, existem poucos jogos *opensource* para esta plataforma. Apesar de haver vários motores de jogos em 3D, que permitem desenvolvimento em *C++/Java*, por enquanto, não existe nenhum jogo FPS com código aberto.

Em alternativa, é possível usar um jogo FPS feito para PC, visto que, existem muitos jogos *opensource* desse género para o sistema operativo Windows. Contudo, portar um jogo feito para PC pode ser complicado. Primeiro é necessário bibliotecas adicionais, modificar o jogo para que este corra em OpenGL e criar um cliente adicional. Segundo, o hardware dos dispositivos móveis é mais limitado em comparação com os PCs, o que pode causar uma grande alteração nos gráficos do jogo. E por último, a maioria usa rato e teclado para controlar o avatar, os periféricos que não existem no dispositivo móvel e têm de ser substituídos por um *joystick* virtual no ecrã.

O jogo escolhido para esta tese é o *Quake 3 Arena* por seguintes vantagens:

- Existe um cliente *open source* que permite correr o jogo no sistema operativo Android.
- É *opensource* e é implementado em *C*, logo pode ser portado para o Android via Android NDK.
- O jogo é feito em OpenGL e os gráficos são suportados pela maioria dos *tablets* e *smartphones*.
- Existe modo de multijogador que permite usar *bots*.
- Possui a arquitetura Cliente-Servidor com lógica do jogo centrada no servidor.
- Mapas de varias dimensões, com grande oclusão ou mais abertas.
- Possibilidade de acrescentar mais mapas feitos pela comunidade.

O *Quake 3 Arena*, sendo um jogo antigo mas ainda muito popular, não necessita de grandes alterações para funcionar nos dispositivos móveis. O cliente que permite correr o jogo em Android é feito em *Java* (Android SDK). No entanto, este necessita de ser melhorado, visto que, fornece poucas funções para interação com o jogo.

3.3 A escolha da tecnologia de comunicação

Com base nos valores obtidos da Tabela 2 do capítulo 2, a tecnologia mais adequada para o FpsTab é Wi-Fi. Mesmo que o seu consumo energético não seja o menor de todos os concorrentes, é a tecnologia

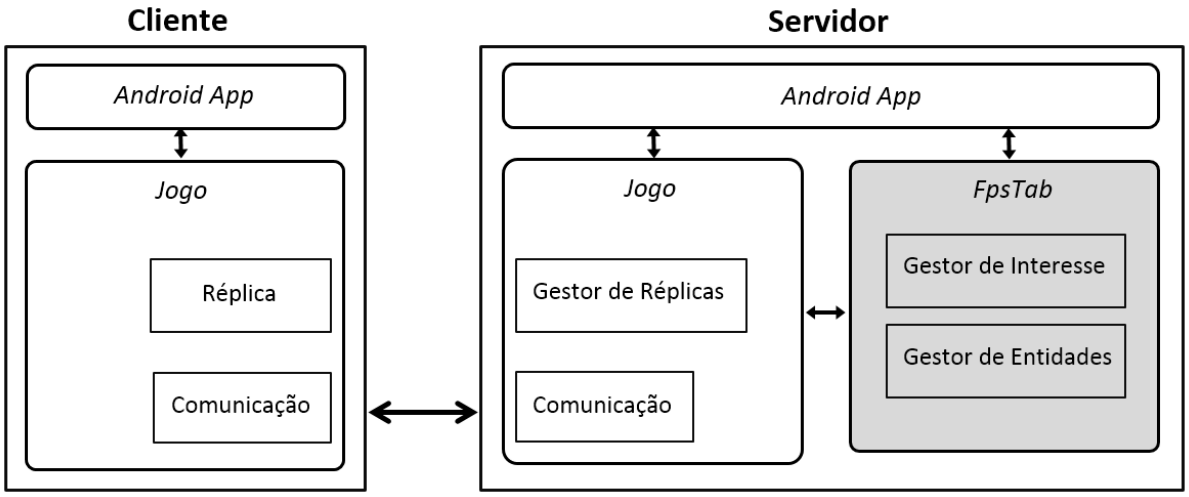

Figura 13: Arquitetura global do sistema.

que mais se enquadra nos requisitos da tese. O alcance, latência da rede, acesso ao meio gratuito e a largura da banda são os pontos fortes da tecnologia Wi-Fi. Além disso, através do Wi-Fi é possível aceder à *Internet* e jogar em modo *on-line*, o que possibilita que os jogadores nos dispositivos móveis combatam contra os jogadores nos PCs, pois o jogo *Quake 3 Arena* foi desenvolvido para o PC.

3.4 Arquitetura global do sistema

A arquitetura global do sistema, como podemos observar na Figura 13, é composta por múltiplas camadas.

A *Android App*, camada que se situa no cliente e no servidor, tem como objetivo executar o jogo *Quake 3 Arena* nos dispositivos móveis com sistema operativo Android. Para tal, comunica diretamente com a camada *Jogo*, onde se encontram os ficheiros específicos do *Quake 3 Arena*. Esta camada também permite trocar entre o jogo original e o jogo com os melhoramentos do FpsTab.

Como já referimos na secção 3.2 o *Quake 3 Arena* tem a lógica do jogo centrada no servidor. Por isso, o servidor é responsável pelo processamento do jogo e pela gestão do estado global, que contém a informação sobre todas as entidades.

O cliente envia os *inputs* do *joystick*, ou seja, as ações que o avatar vai efetuar, ao servidor. Este processa os *inputs* de todos os clientes e atualiza o estado global do jogo. De seguida, o Gestor de Réplicas gera uma réplica com as informações específicas ao cada jogador em cada instante do jogo.

A réplica contém o estado do avatar do jogador (sua posição, velocidade, animação, nível de vida, arma, etc.) e o estado de todas as outras entidades (avatares oponentes, itens, objetos, etc.). O cliente apenas acede à sua réplica e apresenta o jogo no dispositivo, baseando-se nos valores que esta contém.

A função da camada FpsTab consiste em interceptar o Gestor de Réplicas do jogo e filtrar a informação

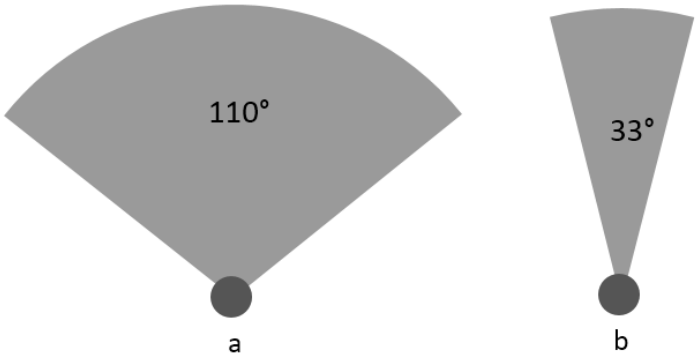

Figura 14: FOV no estado normal (a) e com *zoom* da arma ligado (b).

que consta na réplica antes, que esta seja enviada para o jogador. FpsTab usa a Gestão de Interesse baseada na combinação de aura e campo de visão com diferentes níveis de consistência. Os parâmetros, como o ângulo de abertura do campo de visão e o raio da aura, podem ser configurados através da *Android App*.

A comunicação entre o servidor e os clientes é feita através do módulo de Comunicação existente na camada *Jogo*. Este módulo encripta os dados usando uma chave partilhada, de seguida, comprime-os através da codificação *Huffman* e envia sobre o protocolo UDP/IP.

3.5 Arquitetura do FpsTab

O FpsTab foi desenhado para ser utilizado como um sistema *middleware* cuja função é melhorar a gestão de interesse e gestão de réplicas dum jogo, para minimizar a quantidade e o volume das mensagens trocadas entre o servidor e cliente.

Esse *middleware* contém dois módulos: Gestor de Interesse e Gestor de Entidades. O primeiro decide quais as entidades (avatares, itens, etc.) que são relevantes para o jogador num determinado instante do jogo. O Gestor de Entidades altera o conteúdo da réplica, que posteriormente vai ser enviada ao cliente, de maneira que a consistência do jogo não seja afetada.

3.5.1 Gestor de Interesse

O funcionamento do módulo Gestor de Interesse é baseado nas técnicas de Gestão de Interesse. Definimos duas AoI associadas a cada jogador: campo de visão e aura.

O campo de visão (FOV), semelhante ao campo de visão humana, permite visualizar as entidades que se encontram dentro do limite visual do avatar. As entidades que se encontram fora do FOV, por exemplo, nas costas do avatar, vão ser ignoradas, pois não são visíveis.

O ângulo de visão do avatar pode variar dependendo da situação, por exemplo, quando é usado o *zoom* da arma o FOV é mais reduzido. O ângulo do FOV definido por FpsTab é de 110° no estado normal e

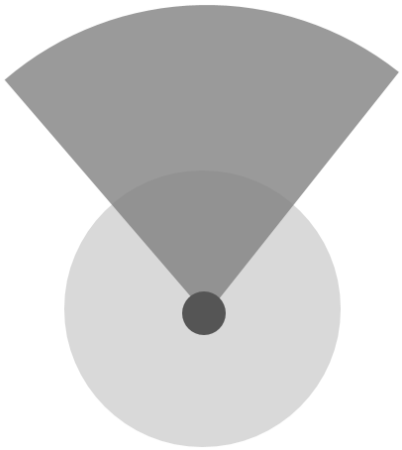

Figura 15: Combinação da aura e do FOV do avatar.

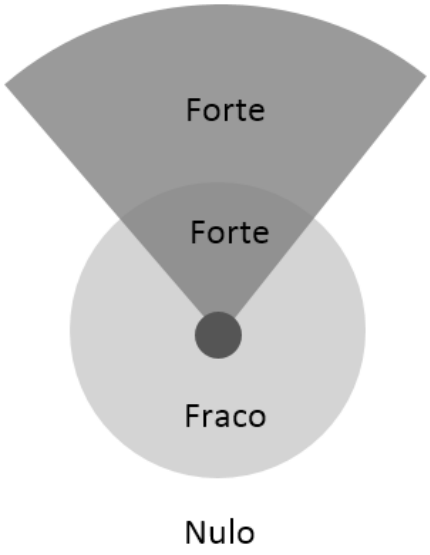

Figura 16: Os níveis de consistência.

é reduzido até aos 33° quando se usa o *zoom* da arma (Figura 14).

Os ângulos são ligeiramente maiores do que os ângulos predefinidos por *Quake 3 Arena*. Por definição o ângulo do FOV no estado normal é de 90° e de 22,5° com *zoom* da arma ligado. Este aumento é feito para evitar os problemas de inconsistências, que podem surgir quando o avatar executa os movimentos rotativo bruscos. Como o tempo que a mensagem demora ir do servidor ao cliente é maior do que o tempo duma rotação brusca, as entidades podem aparecer dentro do FOV passado alguns instantes. Portanto, um ângulo ligeiramente maior do que predefinido permite que o jogador receba o estado das entidades antes de os ver.

Contudo, as entidades que ficam fora do limite visual do avatar nem sempre podem ser ignoradas. Em alguns casos, as entidades invisíveis podem ser ouvidas. Por exemplo, quando um avatar fica nas costas do outro, as suas ações, como disparar ou apanhar itens, só podem ser captadas através da

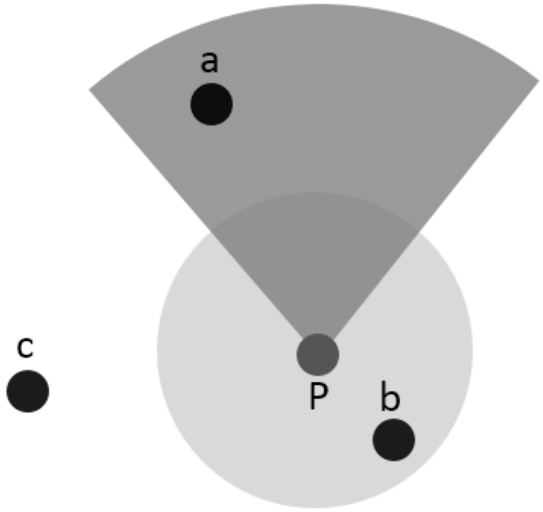

Figura 17: Os níveis de consistência.

audição. Além disso, a perceção auditiva está ligada a natureza de jogos FPS e desempenha um papel fundamental.

Para este efeito foi introduzida uma aura à volta do avatar. Como podemos observar na Figura 15, a aura é representada por uma circunferência concêntrica centrada no avatar, e a sua área corresponde ao limite auditivo do mesmo.

O nível de importância das entidades depende do seu posicionamento e a sua distância em relação ao avatar. Assim sendo, existem três níveis de consistência: Forte, Fraco e Nulo (Figura 16). No nível Forte, são consideradas todas as entidades e o seu estado na íntegra, pois representam o maior interesse para o jogador. O nível de consistência Fraco, que fica fora do FOV do avatar mas dentro do alcance auditivo, só considera os sons emitidos pelas ações dos avatares ou outras entidades. Por último, o nível Nulo ignora todas as entidades, pois estas não afetam o estado do jogador.

Os valores atuais do FOV e da aura foram escolhidos especificamente para o jogo *Quake 3 Arena*, no entanto estes podem ser ajustados consoante as necessidades de qualquer jogo FPS.

3.5.2 Gestor de Entidades

Uma réplica, que é a cópia do estado global do jogo, geralmente contêm a informação sobre todas as entidades. O Gestor de Entidades tem como objetivo reduzir o volume dessa informação de maneira que, o jogador receba apenas o que é necessário.

Para alterar o estado das entidades contidas na réplica, este módulo baseia-se nos valores de consistência fornecidos pelo Gestor de Interesse. Os níveis são: Forte, Fraco e Nulo. A Figura 17 mostra três entidades (*a,b,c*) que se encontram nos diferentes níveis de consistência definidos para o avatar *P*.

A entidade *a*, que se encontra dentro do FOV do avatar *P*, pertence ao nível de consistência Forte.

Portanto, o Gestor de Entidades não altera o estado da entidade *a*, contido na réplica, pelo que o avatar *P* receberá o estado completo dessa entidade.

No entanto, a entidade *b* pertence ao nível de consistência Fraco, pois situa-se dentro da aura do avatar *P*. O Gestor de Entidades altera o estado dessa entidade, de maneira a preservar apenas a informação para reproduzir os eventos sonoros.

Por último, o estado da entidade *c* vai ser retirado da réplica, pois esta fica dentro do nível de consistência Nulo.

3.5.3 Funcionamento do FpsTab

Em geral, o funcionamento do FpsTab difere do jogo original apenas na gestão de réplicas. Quando o servidor do jogo original envia a réplica genérica, o FpsTab altera o conteúdo da réplica de forma a minimizar o seu volume.

A lógica de funcionamento do nosso sistema pode ser separada em várias etapas:

1. O servidor do jogo gera uma réplica para um determinado cliente.
2. Para cada entidade contida nessa réplica o Gestor de Interesse calcula a que níveis de consistência estes pertencem. De seguida, envia estes valores ao Gestor de Entidades.
3. Para cada entidade, o Gestor de Entidades verifica se esta deve ou não continuar na réplica. Caso se verifique que não (nível de consistência Nulo) o seu estado vai ser removido.
4. As restantes entidades, que continuam na réplica, são verificadas:
   (a) Caso a entidade pertencer ao nível Forte, o seu estado não se altera.
   (b) Caso a entidade pertencer ao nível Fraco, o Gestor de Entidades altera o seu estado de maneira a preservar apenas a informação que corresponde aos eventos sonoros.
5. A réplica é enviada ao cliente.

Para explicar a diferença entre o *Quake 3 Arena* original e com FpsTab vamos ver um exemplo em concreto, quando dois avatares encontram-se na mesma sala do mapa. Vamos supor que o avatar *A* vai ao encontro com avatar *B* localizado na sala *S*.

Como o jogo original divide o mundo virtual em salas, e o mesmo se mantém no FpsTab, a interação entre o avatar A e *B* acontece quando estes se encontram dentro da mesma.

O avatar *A* ainda não entrou na sala *S*, por isso, a sua réplica não contém a informação sobre o avatar *B*. O mesmo acontece com o avatar *B*. No entanto, assim que o avatar *A* entra na sala o servidor adiciona o estado do avatar *B* à sua réplica, e o mesmo acontece com avatar *B*. Independentemente da posição do avatar *A* em relação a posição do avatar *B*, o servidor continua a trocar o estado entre eles. Mesmo

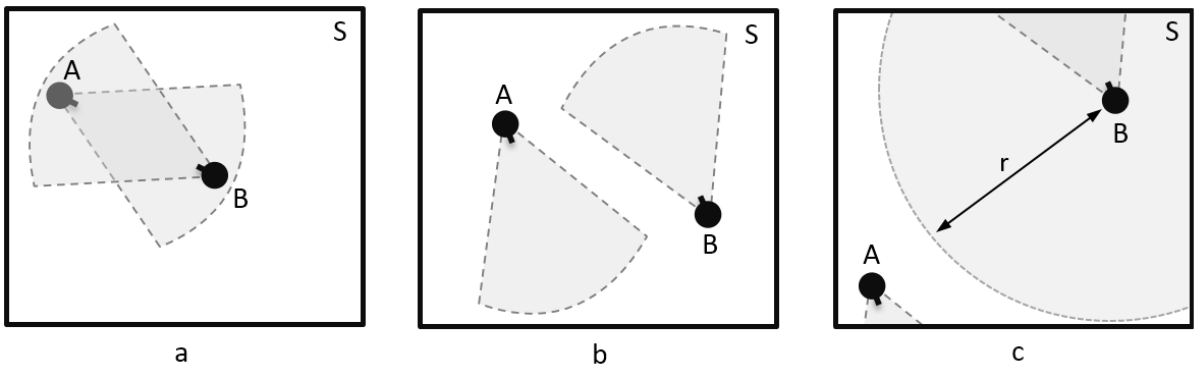

Figura 18: Exemplo da interação entre os avatares *A* e *B* na sala *S*. O *r* representa o raio da aura do avatar *A*.

que o avatar *B* sai por fora do campo de visão do avatar *A*, o último continua a receber o estado do *B*. A interação entre eles acaba quando um dos avatares abandona a sala *S*.

No jogo com FpsTab o procedimento já não é o mesmo. Antes de entrar na sala *S*, tal como no jogo original, a réplica do avatar *A* não contém o estado do avatar *B*, e ao contrario. Quando o avatar *A* entra na sala *S*, a interação entre os dois pode não ocorrer. Pois o Gestor de Interesse é que decide se a réplica do avatar *A* vai conter o estado do avatar *B* e ao contrário.

No caso do avatar *B* se situar dentro do FOV do avatar *A*, o servidor adiciona à réplica do avatar *A* o estado completo do avatar *B*. E o mesmo acontece se o avatar *A* se encontra dentro do FOV do *B* (Figura 18 a)).

Quando o avatar *B* sai do FOV do avatar *A*, mas fica a uma distância próxima do *A*, o avatar *B* continua a receber o estado do *B*. No entanto, esse estado é alterado por Gestor de Entidade de forma a minimizar o seu conteúdo (Figura 18 b)).

Quando o avatar *B* sai do FOV do avatar *A* e a distância entre eles ultrapassa o raio da sua aura, então o servidor deixa de adicionar o estado do *B* à réplica do avatar *A*. Se o avatar *A* estiver nas mesmas condições em relação ao avatar *B*, então a interação entre eles acaba (Figura 18 c)).

Portanto, o FpsTab consegue minimizar a troca de estados entre os avatares mesmo que estes se encontram dentro da mesma sala.

Capítulo 4

Implementação

Este capítulo começa por descrever o ambiente de desenvolvimento. De seguida, descrevemos a implementação das camadas FpsTab, *Jogo* e *Android App*. Por fim, vamos falar sobre o JNI, através do qual é feita a comunicação interna (entre as camadas).

4.1 Ambiente de desenvolvimento

O desenvolvimento do *middleware* FpsTab e a compilação do *Quake 3 Arena* foram feitos no sistema operativo Linux. O FpsTab foi desenvolvido sob forma de biblioteca e implementado em linguagem *C*. O *Quake 3 Arena* também está implementado em *C* e compilado para arquitetura ARM através da ferramenta Android NDK. O cliente que executa o jogo no sistema operativo Android (*Android App*) foi desenvolvido no ADT (*Android Developer Tools*) e escrito em *Java*. A comunicação entre o código implementado em *C* e em *Java* e feita através do JNI (*Java Native Interface*).

4.2 FpsTab - *First-Person Shooter for Tablets*

A camada FpsTab consiste em melhorar a gestão de informação dum jogo FPS para minimizar o conteúdo da réplica sem prejudicar a jogabilidade. É escrita em linguagem *C* para facilitar a interação com o *Quake 3 Arena* e é compilada juntamente com o mesmo.

4.2.1 API do FpsTab

A API é um conjunto de rotinas e padrões estabelecidos por um software para a utilização das suas funcionalidades por outras aplicações. Neste caso, a API do FpsTab fornece as funcionalidades à camada *Jogo* e *Android App*.

As funções que fazem interface entre o *Jogo* e FpsTab:

FPSTAB_Init Inicializa os parâmetros do FpsTab.

FPSTAB_getEntityConsistencyLevel Define o nível de consistência da entidade em relação ao avatar. Recebe a posição do avatar, os ângulos de orientação do jogador (*pitch,yaw, roll*), parâmetro que indique se o avatar tem o *zoom* ligado, a posição da entidade e o *id* da entidade.

A função devolve um inteiro que define o nível de consistência:

- 2 - nível de consistência Forte.
- 1 - nível de consistência Fraco.
- 0 - nível de consistência Nulo.

FPSTAB_ShouldEntityBeRejected Testa se uma entidade tem de ser eliminada da réplica. Caso a função devolver 0 a entidade permanece na réplica, caso devolver 1 - a entidade é rejeitada.

FPSTAB_ManageEntityState Modifica o estado duma entidade contido na réplica caso seja necessário. A função devolve o estado da réplica modificado ou inalterado.

A função que faz interface entre a camada *Android App* e FpsTab:

FPSTAB_SetSettings Recebe os parâmetros de configuração do FpsTab, tais como o ângulo do FOV do avatar no estado normal, ângulo do FOV com *zoom* ativo e o raio da aura.

4.2.2 Estrutura de dados

Como a biblioteca FpsTab está desenvolvida em *C* a sua estrutura de dados é separada em módulos (ficheiros). Cada ficheiro contém funções e variáveis necessárias para seu funcionamento. Para facilitar a integração com a biblioteca, todas as funções estão descritas num único *header file* chamado ***fpstab.h***. De seguida, apresentam-se as funções de cada módulo e como são constituídos.

4.2.2.1 *fpstab.c*

Este ficheiro contêm as seguintes funções:

FPSTAB_Init. A função que inicializa o FpsTab. Esta faz *reset* aos dados temporários do Gestor de Entidades.

Log. Para facilitar o *debug* do cogido em *C* implementamos esta função que envia a sequência de texto para o ADT do Android.

4.2.2.2 *fpstab_im.c*

Este ficheiro contém estrutura do Gestor de Interesse do FpsTab.
FPSTAB_getEntityConsistencyLevel é a principal função deste módulo e para sua funcionalidade usa

outras funções auxiliares. Para determinar o nível de consistência da entidade, a função verifica se esta pertence a alguma das áreas de interesse do avatar. O procedimento é feito por ordem descrito por pseudocódigo:

```
se entidade_pertece_ao_FOV entao
  retorna NIVEL_FORTE
se entidade_pertece_a_aura entao
  retorna NIVEL_FRACO
senao retorna NIVEL_NULO
```

Para determinar o FOV (*EntityInsideFOV*), primeiro é calculado o vector tridimensional entre o avatar em causa e a entidade. De seguida, esse vector é normalizado e calculado o produto interno entre ele e a orientação (normalizada) do avatar. Através do resultado do produto interno, é então determinado se a entidade se encontra dentro do FOV do avatar. Quando o avatar tem o *zoom* ativo, o ângulo do FOV é mais reduzido, mas o procedimento é o mesmo.

A função *EntityInsideAura* calcula se a entidade pertence a aura do avatar. Primeiro é calculada a distância entre o avatar e a entidade através da função *DistanceBetween*. De seguida, verifica-se se a distância é menor ou igual ao raio da aura.

O nível de consistência da cada entidade é enviado ao Gestor de Entidades através da função *addEntityToEM*.

Por fim, o ficheiro contêm a função *FPSTAB_SetSettings* que define os parâmetros do FpsTab.

4.2.2.3 *fpstab_em.c*

Neste ficheiro encontra-se a estrutura do Gestor de Entidades.

O nível de consistência das entidades é armazenado num vetor de inteiros chamado *entityVec*. A posição do vetor corresponde a *id* da entidade e o valor corresponde ao nível de consistência. O vetor tem a capacidade para 1024 entidades e é igual ao valor predefinido por *Quake 3 Arena*.

A função *FPSTAB_ShouldEntityBeRejected* baseia-se nos valores do *entityVec* para determinar se a entidade deve ou não continuar na réplica.

O estado da entidade é gerido por função *FPSTAB_ManageEntityState*. Esta foi implementada para aceitar o estado da entidade, que é uma estrutura do tipo *entityState_t* definido no *Quake 3 Arena*.

Ao modifica o estado da entidade os atributos desnecessários são postos a 0. Assim, quando o servidor calcula a diferença (*delta*) do estado atual com o anterior, estes atributos vão ser ignorador por completo. Dessa maneira conseguimos obter uma redução do tamanho do estado da entidade e, consecutivamente, do volume da réplica.

4.3 Camada Jogo

A camada *Jogo* contém o jogo do género FPS *Quake 3 Arena*. O código fonte do jogo é obtido dum projeto *opensource ioQuake3*[3] que contém o *engine* do *Quake 3 Arena* melhorado e permite o porte para a plataforma ARM.

O *Quake 3 Arena* é compilado especificamente para os processadores com arquitetura ARM em bibliotecas do tipo *Shared Libraries* (*.so*) através da ferramenta Android NDK. São criadas as seguintes bibliotecas:

- *libcgamearm.so* - processamento e apresentação do jogo (Cliente).
- *libqagamearm.so* - processamento do jogo e inteligência artificial (Servidor).
- *libquake3.so* - lógica do jogo com FpsTab (Cliente e Servidor).
- *libquake3_original.so* - lógica do jogo original (Cliente e Servidor).
- *libuiarm.so* - interface do utilizador .

No *Quake 3 Arena* original era impossível o servidor saber se o jogador ativou o *zoom*, porque a lógica do jogo não considera o FOV dos avatares. No entanto, o jogo foi modificado de maneira que o cliente envia um comando especifico ao servidor, sempre que este ativa ou desativa o *zoom* da arma. O servidor, por sua vez, contêm um vetor de inteiros (*clientVec*) que guarda o estado do *zoom* de cada jogador. A posição do vetor corresponde ao *id* do jogador e o valor corresponde ao estado do zoom (0 - desligado e 1 - ligado).

4.3.1 Gestão de réplicas original

O jogo possui a arquitetura Cliente-Servidor com lógica centrada no servidor. O seu módulo, chamado Gestor de Réplicas, recolhe a informação vinda dos outros jogadores e gera uma réplica do estado do jogo para cada jogador.

Podemos dividir a lógica do jogo nas seguintes etapas:

1. O cliente envia os *inputs* e comandos internos do jogo para o servidor.
2. O servidor processa a informação e atualiza o estado global do jogo.
3. O servidor gera uma réplica do estado global do jogo especificamente para cada cliente, que contêm o estado do avatar do cliente e o estado de outras entidades.
4. As réplicas são enviadas aos clientes.
5. O cliente recebe a sua réplica e apresenta o jogo consoante as informações que esta contém.

[3] www.ioquake3.org

Segundo os estudos das réplicas do *Quake 3 Arena*, concluiu-se que estas contêm muita informação desnecessária para o jogador. Isso é causado por fraca Gestão de Interesse do jogo que divide o mundo virtual em regiões, ou seja, o jogador recebe o estado dos jogadores que se encontram dentro da mesma região. No entanto, o estado de algumas entidades, como as armas, itens e *powerups*, o jogador recebe sempre e independentemente se estão na mesma região ou não.

O *Quake 3 Arena* ainda utiliza as técnicas para reduzir a informação propagada na rede. A primeira permite enviar somente as alterações do estado das entidades face ao seu estado anterior (*delta*). A segunda técnica, chamada Predição, calcula a posição duma entidade baseada na sua posição anterior. Contudo, a quantidade de informação propagada ainda pode ser reduzida melhorando a gestão de informação do jogo.

4.3.2 Gestão de réplicas com FpsTab

Como o *Quake 3 Arena* tem a lógica do jogo centrada no servidor, decidimos modificar a réplica durante o seu desenvolvimento em vez de alterar o conteúdo da réplica completada. O FpsTab decide se o estado duma entidade deve ser adicionado à réplica dum cliente quando o servidor a preencha. Assim, conseguimos aumentar o desempenho do sistema, pois evitamos o processamento de dados adicional.

O procedimento da gestão de réplicas tem a seguinte ordem:

1. O cliente envia os *inputs* e comandos internos do jogo para o servidor.
2. O servidor processa a informação e atualiza o estado global do jogo.
3. O servidor começa a criar uma réplica para um determinado cliente.
4. O FpsTab é inicializado (*FPSTAB_Init*).
5. Enquanto o servidor escolhe as entidades para inserir na réplica é chamado o Gestor de Interesse, através da função *FPSTAB_getEntityConsistencyLevel*, que define o nível de consistência da cada entidade.
6. Enquanto o servidor adiciona as entidades à réplica, testa-se sé a entidade permanece na réplica através da função *FPSTAB_ShouldEntityBeRejected*.
7. Para as entidades que continuam na réplica, o Gestor de Entidades procede a alteração do seu estado, caso seja necessário (função *FPSTAB_ManageEntityState*).
8. O servidor envia a réplica ao cliente.
9. O cliente recebe a sua réplica e apresenta o jogo consoante as informações que esta contém.

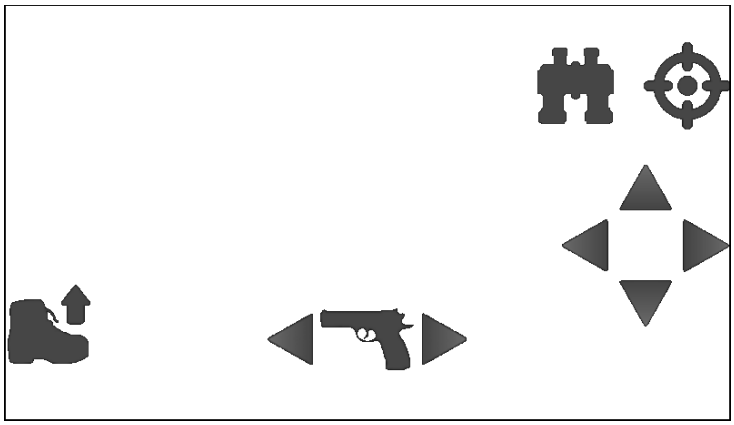

Figura 19: *Layout* do *joystick* virtual.

4.4 Android App

Esta camada tem como objetivo lançar o jogo no dispositivo com sistema operativo Android e é responsável pela interação entre o jogo e o jogador.

A *Android App* está desenvolvida para os *smartphones* e *tablets* com o sistema operativo Android 2.3 ou superior. Como base, para o lançamento do jogo, usamos um projeto *opensource* chamado *Kwaak3*[4].

A camada *Android App* permite configurar os parâmetros do FpsTab, como o ângulo da abertura do FOV no estado normal ou com *zoom* e o raio da aura. Estes parâmetros passam do *Android App* para o *FpsTab API* através de JNI.

Por fim, ainda possibilita a escolha entre o *Quake 3 Arena* original e *Quake 3 Arena* com as funcionalidades do FpsTab. Conforme a escolha é carregada a biblioteca do jogo original (*libquake3_original.so*) ou modificado (*libquake3.so*).

4.4.1 *Joystick* virtual

O *Kwaak3* efetuava apenas o porte do jogo para Android OS com mapeamento dos periféricos, como rato e teclado, o que não chegava para obter a interação completa com o jogo no dispositivo móvel. Portanto, foi adicionado um *joystick* virtual que aparece no ecrã do dispositivo (Figura 19).

O *joystick* foi desenvolvido em ADT, possui uma interface gráfica adequada para o jogo com os botões mais importantes: movimento do avatar, disparar, saltar, *zoom* e alteração da arma. O controlo da mira é feito através do toque e movimento do dedo sobre o ecrã. Também é possível usar dois dedos em simultâneo para melhor interação com o jogo.

O *layout* do *joystick* foi desenhado tendo em conta as capacidades do ecrã tátil e os requisitos dos jogos FPS. Ao segurar o dispositivo móvel nas mãos, só nos restam dois dedos para interagir com o ecrã, logo não é possível controlar a maioria das funções em simultâneo. Assim, atribuímos o controlo

[4] http://code.google.com/p/kwaak3/

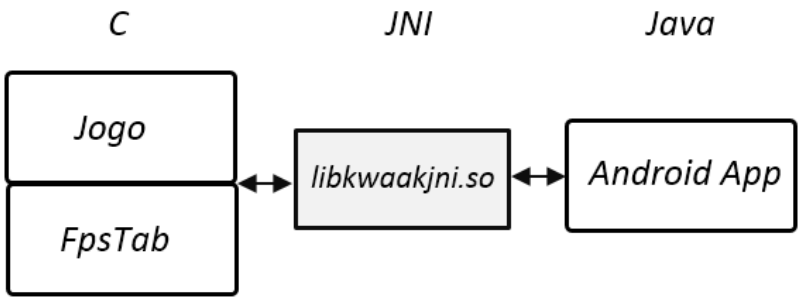

Figura 20: Comunicação entre as camadas através do JNI.

da mira do avatar ao dedo esquerdo e as ações, como o movimento do jogador e disparar, ao dedo direito. Desse modo, damos mais preferência ao apontar a mira e disparar do que movimentar o jogador e disparar, o que geralmente acontece nos jogos para os dispositivos móveis. Como se trata dos jogos FPS, onde eliminar os adversários é o objetivo principal, essa forma de interação é mais adequada.

Para facilitar ainda mais esse objetivo adicionamos a função do *zoom*. Este funciona de forma alternada, ou seja uma vez premido o botão, o *zoom* fica ativo, o que liberta o jogador de estar sempre a carregar no botão para ativar o *zoom*.

A *layout* do *joystick* depende muito da resolução e as capacidades do ecrã. Como a sua implementação se baseia em multitoque, é necessário criar uma *layout* diferente para cada resolução. Atualmente, o *joystick* funciona corretamente apenas nos dispositivos com a resolução do ecrã 800x480 e 1280x800.

4.4.2 JNI - *Java Native Interface*

A interação entre a camada *Android App* (escrita em *Java*) e as camadas do *Jogo* e *FpsTab* (escritas em *C*) é feita através do JNI. Esta comunicação é feita em tempo de execução e funciona por chamadas as funções remotas.

Dessa maneira conseguimos usar as funções das camadas *Jogo* e FpsTab a partir da *Android App* e vice versa. Esta comunicação é feita através de um intermediário *libkwaakjni.so* (Figura 20). Esta biblioteca é compilada através de Android NDK a partir dos ficheiros:

Kwaak3JNI.java API da *libkwaakjni.so* para a *Android App*.

org_kwaak3_KwaakJNI.h API da *libkwaakjni.so* para o *Jogo* e FpsTab. Compilado a partir de *Kwaak3JNI.java*.

kwaakjni.c Contêm as funções para interação entre as camadas e execução do *Quake 3 Arena*.

A comunicação entre as camadas é feita de forma indireta. Ou seja, se a *Android App* pretende usar uma função *F* do FpsTab, terá de chamar a função *F'* da biblioteca *libkwaakjni.so*, que por sua vez chama a função *F*.

De notar que, a execução do jogo também é feita através do JNI. A *Android App* executa a biblioteca *libkwaakjni.so* que carrega, consoante a escolha do utilizador, as restantes bibliotecas do jogo.

Capítulo 5

Avaliação

Nesta secção vamos avaliar o desempenho do FpsTab. Primeiro, falaremos sobre como as medições foram feitas, que infraestruturas foram utilizadas e em que mundos virtuais foram feitos os testes. Continuamos com a avaliação quantitativa, onde apresentamos os resultados obtidos. Concluímos o capítulo com a avaliação qualitativa do FpsTab.

5.1 Jogo original vs FpsTab

A avaliação consiste em comparar o *Quake 3 Arena* modificado por FpsTab com o jogo original. Principalmente, vamos analisar a quantidade de tráfego produzido por ambas soluções. Vão ser comparados os consumos de recursos dos dispositivos móveis, como CPU, RAM e a carga da bateria. Contudo, vamos avaliar a qualidade do FpsTab para perceber se este não afeta a jogabilidade.

O jogo foi corrido nos dispositivos móveis reais e testado por utilizadores reais, o que permite nos obter os resultados próximos da realidade.

5.2 Infraestrutura utilizada

O sistema foi avaliado em quatro dispositivos móveis. A Tabela 3 descreve as suas características principais e o seu papel no jogo (cliente ou servidor). De notar que, o servidor do jogo desempenhava o papel do cliente ao mesmo tempo.

Os dispositivos comunicavam através duma rede *ah-hoc* local. No dispositivo servidor foi criado um ponto de acesso ao qual se ligavam os restantes dispositivos (clientes). O protocolo de comunicação foi o Wi-Fi.

As medidas do desempenho dos dispositivos foram obtidos através da aplicação *System Monitor Lite*[5]. O seu serviço de monitorização em *background* permite recolher os dados sobre a utilização do CPU,

Dispositivo	***Asus Transformer TF-101***	***Samsung Galaxy S***	***Samsung Galaxy S3 Mini***	***LG Optimus 2X***
Papel no jogo	servidor	cliente 1	cliente 2	cliente 3
CPU	1 GHz *dual-core*	1 GHz	1 GHz *dual-core*	1 GHz *dual-core*
GPU	ULP GeForce	PowerVR SGX540	Mali-400 MP	ULP GeForce
RAM (Mb)	1024	512	1024	512
Bateria (Wh)	24,4	5,85	5,7	5,7
Ecrã (polegadas)	10.1	4	4	4
Ecrã (resolução)	1280x752	800x480	800x480	800x480
Versão do Android	4.0.3	4.2.2	4.1.2	4.0.4

Tabela 3: Dispositivos móveis usados nos testes.

RAM, acesso a memória, o trafego recebido/enviado e a utilização da bateria.

5.3 Mapas

O *Quake 3 Arena* possui uma grande quantidade de mundos virtuais (mapas) de várias dimensões. Para efetuar os testes, escolhemos 4 mapas com diferentes tamanhos e oclusão. Os mapas são:

- *Q3DM2* - pequeno e fechado.
- *Q3DM11* - grande e fechado.
- *Q3TOURNEY3* - pequeno e aberto.
- *Q3DM17* - grande e aberto.

Devido a Gestão de Interesse baseada em regiões, o *Quake 3 Arena* divide o mapa em salas. O ponto crítico no jogo é quando todos os jogadores se encontram na mesma sala. Por essa razão, o tamanho da sala é que tem o maior impacto na sobrecarga do servidor e não o tamanho do mapa em si. Em relação aos mapas abertos, o tamanho já faz diferença, pois o servidor trata o mapa como se fosse uma sala.

A Figura 21 demonstra a comparação de quatro mapas em relação ao seu tamanho.

5.4 Metodologia de avaliação

Os testes foram feitos em quatro dispositivos móveis com quatro jogadores de cada vez. O modo de competição foi *Deathmatch* onde todos combatem contra todos. Para que os jogadores se familiarizem

[5] http://cgollner.x10.mx

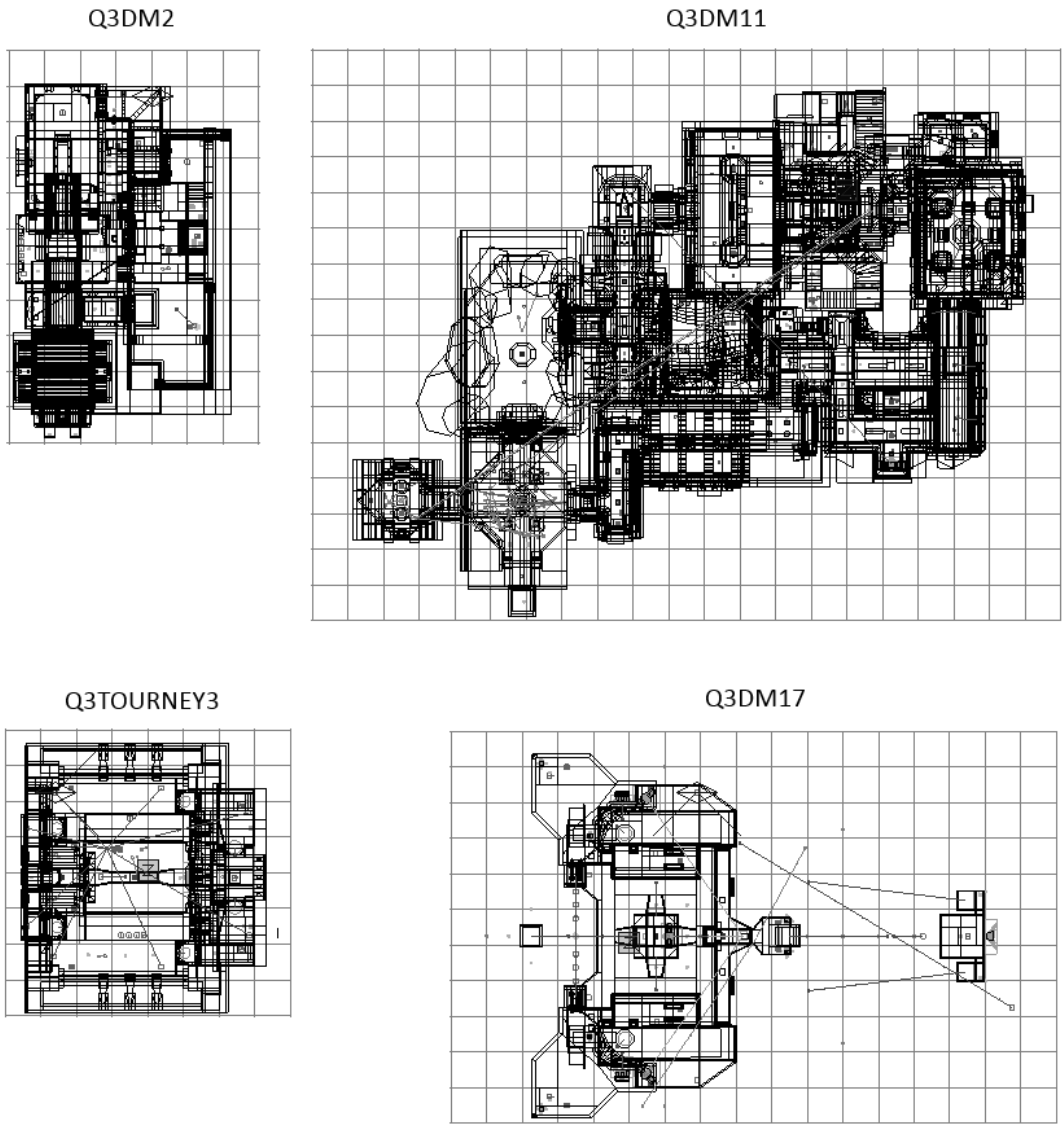

Figura 21: Comparação do tamanho das mapas usadas no teste.

com o jogo e os controlos, foram dados 5 minutos de treino no mapa que não entrou nos testes - *Q3DM1* . Após o treino, começaram com o primeiro mapa dos testes onde o procedimento foi seguinte:

1. Jogo durante 5 minutos. Solução original ou FpsTab escolhidas de forma aleatória.
2. Recolha de dados do desempenhos dos dispositivos.
3. Jogo durante 5 minutos. Caso da primeira vez foi escolhida a solução original, vai ser escolhida a solução com FpsTab e ao contrario.
4. Recolha de dados do desempenhos dos dispositivos.
5. Preenchimento do questionário por parte dos jogadores.

De seguida, o jogo decorreu no segundo mapa e por aí adiante. Para os restantes mapas o procedimento foi sempre o mesmo. A ordem dos mapas também foi aleatória, mas de maneira que os dois

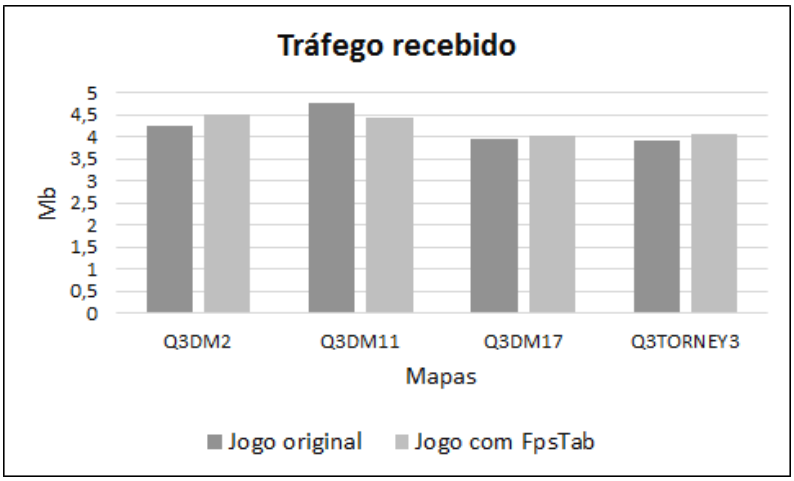

Figura 22: Tráfego recebido por servidor nos diferentes mapas do jogo.

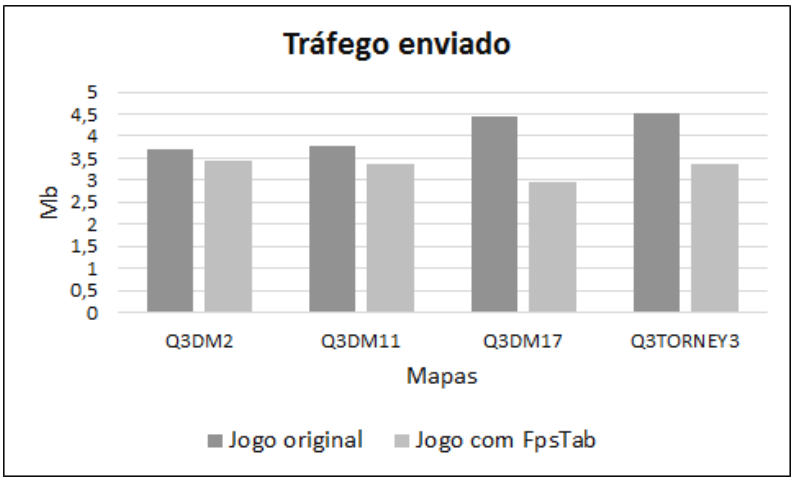

Figura 23: Tráfego enviado por servidor nos diferentes mapas do jogo.

jogos seguidos foram feitos no mesmo mapa. Para obter resultados fiáveis os jogadores não sabiam qual foi a versão de cada jogo.

O tempo do jogo foi de 5 minutos por solução, pois feito as contas, a duração total do teste com 4 mapas foi cerca de 1 hora.

Os resultados obtidos são analisados nas secções 5.5 e 5.6.

5.5 Avaliação quantitativa

Para a avaliação quantitativa realizamos 32 jogos no total, ou seja, 16 para cada solução. Durante o processo foram medidos os consumos de CPU, memória RAM, quantidade de dados transmitidos/recebidos e a utilização da bateria em todos os dispositivos através da aplicação *System Monitor Lite*. O brilho do ecrã em todos os dispositivo foi posto nos 75%.

Esta secção começa com o analise de tráfego produzido. De seguida, fala da utilização do CPU,

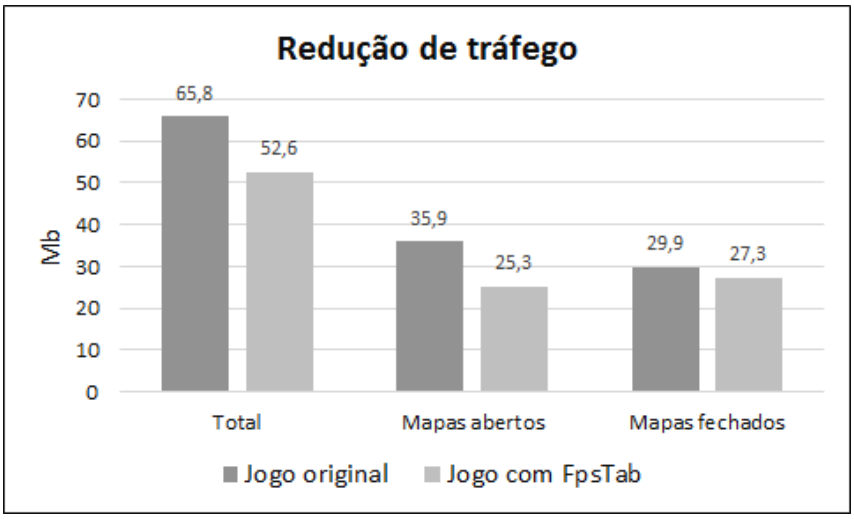

Figura 24: Comparação de tráfego consoante o tipo do mapa.

memoria RAM e o consumo da bateria.

5.5.1 Analise do tráfego

Os gráficos da Figura 22 mostram a média da quantidade de informação recebida pelo servidor do jogo. Como podemos observar, o tráfego recebido nas duas soluções é praticamente igual, pois este é produzido por clientes e não depende da solução em causa. Os dados que o cliente envia ao servidor dependem somente dos acontecimentos no jogo e das circunstância nos quais o jogador se encontra. Ou seja, o aumento de informação enviada corresponde ao aumento de interação no jogo.

No entanto, o tráfego enviado já depende da lógica de gestão de informação do servidor. Como podemos ver no gráfico da Figura 23, o servidor com FpsTab envia menos informação em relação ao servidor com jogo original.

Em 32 jogos o somatório da informação enviada corresponde aos 118,4 Mb, onde 65,8 Mb foi enviado com jogo original e 52,6 Mb com FpsTab. Portanto, a solução FpsTab conseguiu reduzir o tráfego enviado em 20% face ao jogo original.

A maior redução de tráfego (80%) acontece nos mapas abertos, onde o *Quake 3 Arena* praticamente não usa a sua Gestão de Interesse baseada em regiões (Figura 24). Assim, nos mapas abertos os jogadores recebem sempre a informação de todas as entidades que se encontram no mundo virtual, ao contrário do jogo com FpsTab, que consegue reduzir a informação enviada graças às suas técnicas de Gestão de Interesse.

Nos mapas fechados o ganho relativamente ao *Quake 3 Arena* original é menor, pois as técnicas de filtragem de informação do *Quake 3 Arena* já reduzam a quantidade de informação desnecessária. Também, para as salas pequenas, cuja área é menor do que o raio da aura, o FpsTab considera todas as entidades como importantes. Assim sendo, nos mapas pequenos, como *Q3DM2*, o FpsTab aproxima-se do jogo original, e a Figura 24 demostra isso. Já nos mapas grandes, onde o tamanho das

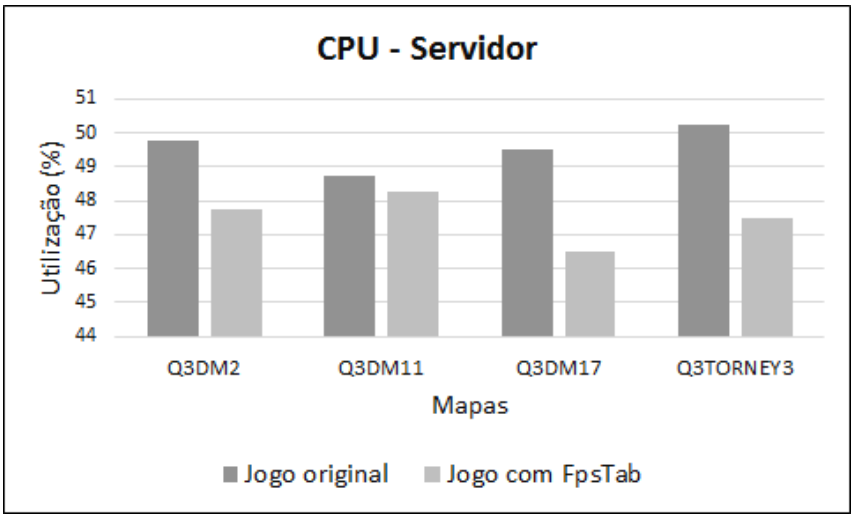

Figura 25: Comparação de utilização do CPU no servidor em relação ao tipo do mapa.

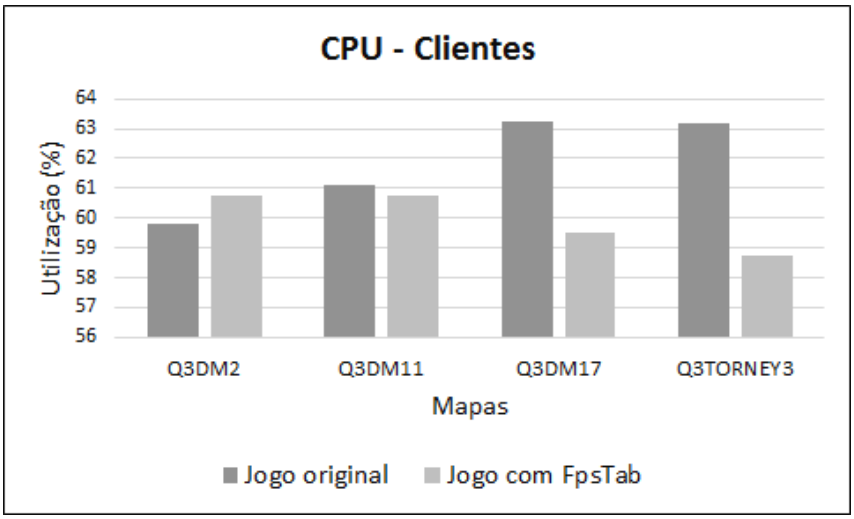

Figura 26: Comparação de utilização do CPU nos clientes em relação ao tipo do mapa.

salas é maior que o raio da aura, o FpsTab produz melhores resultados.

Tendo em conta os resultados obtidos, podemos concluir que, o FpsTab consegue reduzir o tráfego na rede, poupando assim a largura da banda.

5.5.2 CPU

Em termos de consumo de CPU, o jogo com FpsTab reduziu em média o uso do processador do dispositivo móvel em 4,2% no servidor (Figura 25) e em 3% no cliente (Figura 26). Principalmente, a redução no servidor acontece devido a diminuição de quantidade das entidades para processar. No entanto, como o servidor é ao mesmo tempo um cliente, a diminuição das entidades para apresentar no jogo também faz diminuir o uso do CPU.

Tanto no servidor como no cliente, a maior redução do CPU acontece nos mapas abertos como

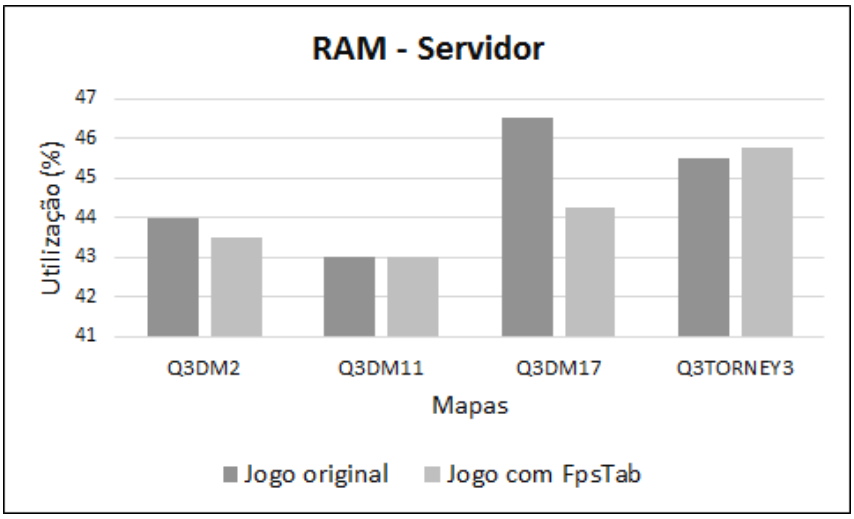

Figura 27: Comparação de utilização da RAM no servidor em relação ao tipo do mapa.

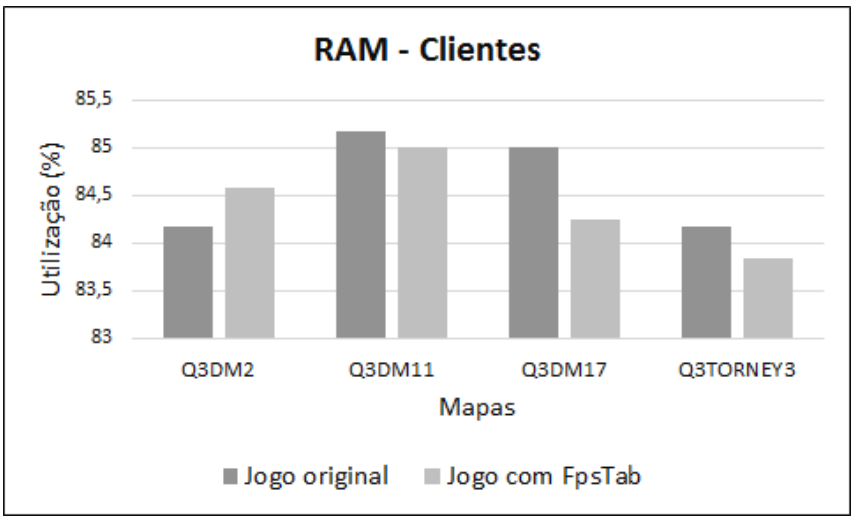

Figura 28: Comparação de utilização da RAM nos clientes em relação ao tipo do mapa.

Q3TOURNEY3 e *Q3DM17*. Nos mapas fechados, o funcionamento do FpsTab aproxima-se da solução original.

5.5.3 RAM

Como mostra o gráfico da Figura 27, existe uma diminuição do consumo de RAM no servidor, que consta 1,4% face ao jogo original e uma menor redução no cliente que consta 0,25% (Figura 28).

Mais uma vez conseguimos obter melhores resultados no mapa grande e aberto *Q3DM17*, onde existe uma vasta quantidade de objetos. A redução da memória RAM nesse mapa chega aos 5% no servidor e 1% no cliente face ao jogo original.

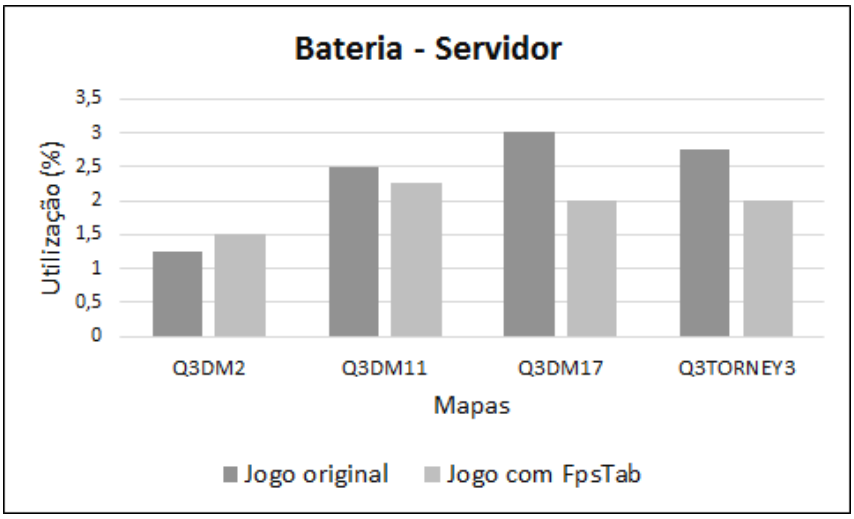

Figura 29: Comparação de utilização da bateria no servidor em relação ao tipo do mapa.

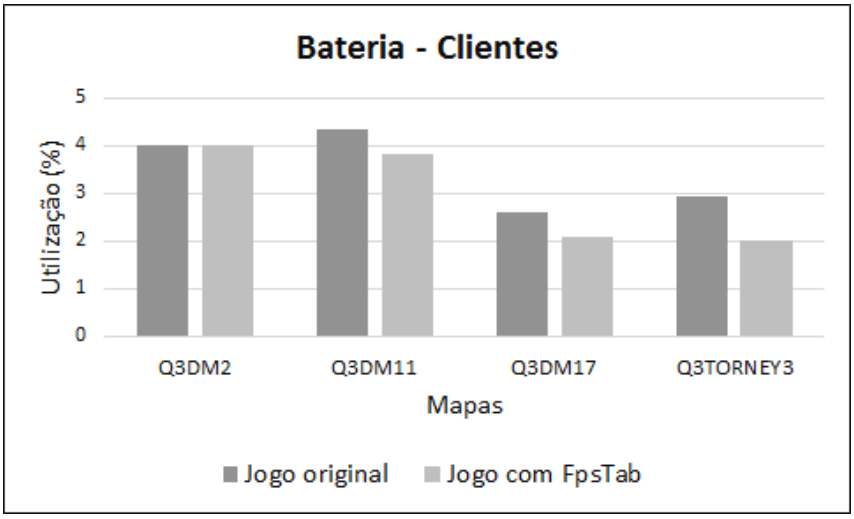

Figura 30: Comparação de utilização da bateria nos clientes em relação ao tipo do mapa.

5.5.4 Bateria

Em relação ao utilização da bateria, o jogo com FpsTab gasta em média menos 18,4% da bateria no servidor e menos 13,9% no cliente (Figuras 29 e 30).

O maior ganho de bateria obtivemos nos mapas abertos, tanto no servidor como no cliente.

Também podemos constatar que, os dispositivos dos clientes gastam mais bateria nos mapas fechados, e o servidor nos mapas abertos, devido a maior quantidade de entidades. Nos mapas pequenos e fechados praticamente não existe a diferença entre o jogo original e com FpsTab.

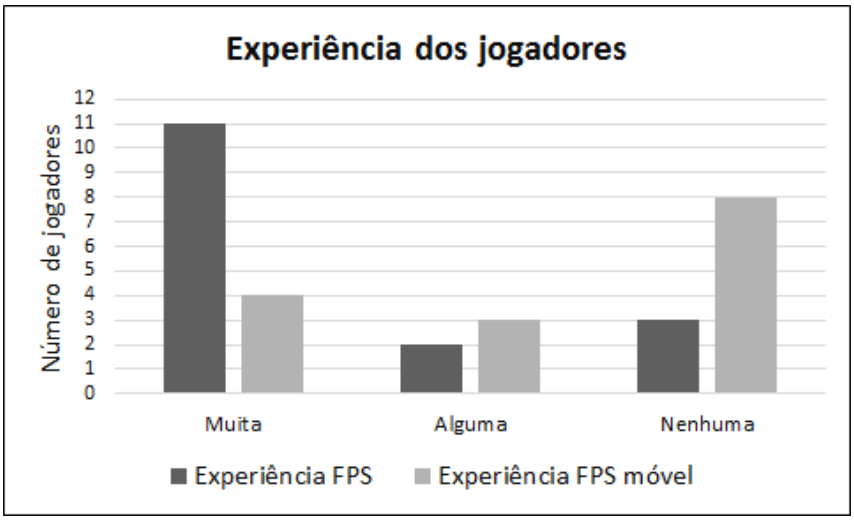

Figura 31: Experiência dos jogadores nos jogos FPS.

5.6 Avaliação qualitativa

Nesta secção pretendemos avaliar se a jogabilidade do *Quake 3 Arena* não foi prejudicada ao usar o FpsTab. Como a jogabilidade é impossível de avaliar objetivamente, esta é medida segundo a opinião subjetiva dos jogadores. Para isso, foi comparado o *Quake 3 Arena* original com o *Quake 3 Arena* melhorado por FpsTab. Após a cada dois jogos, ou seja, após a cada mapa, foi pedido aos jogadores que respondessem a uma das peguntas do questionário (Anexo 1).

No inicio do questionário perguntámos qual é a idade, o sexo e se o jogador tem alguma experiencia com os jogos do género FPS. A pergunta principal era se tinham notado alguma diferença na jogabilidade, ao nível de fluidez e realismo do movimento dos adversários, entre as duas versões do jogo. Caso essa existe, foi pedido para a identificar. Adicionalmente, perguntámos se o jogador usou o *zoom* durante o jogo.

5.6.1 Resultados

Os 32 jogos foram conduzidos com 16 jogadores no total com a idade média de 22 anos. A maioria (81%) foram do sexo masculino. Também, a maioria dos jogadores (56%) teve muita experiência com os jogos FPS, mas apenas 44% teve muita ou alguma experiência com FPS nos dispositivos móveis (Figuara 31).

Dos 16 jogadores apenas 2 (13%) conseguiram notar as diferenças entre as versões, e estes foram os mais experientes. As diferenças foram a favor do jogo original. Um dos jogadores notou que, por vezes, as entidades aparecem com um ligeiro atraso se trocar rapidamente entre o modo normal e *zoom*, e a mesmo tempo efetuar um movimento brusco. O outro jogador, depois de desativar o *zoom* ainda ficou com o FOV reduzido, durante 1 ou 2 segundos. Por isso, notou que o avatar oponente desapareceu antes de sair do ecrã.

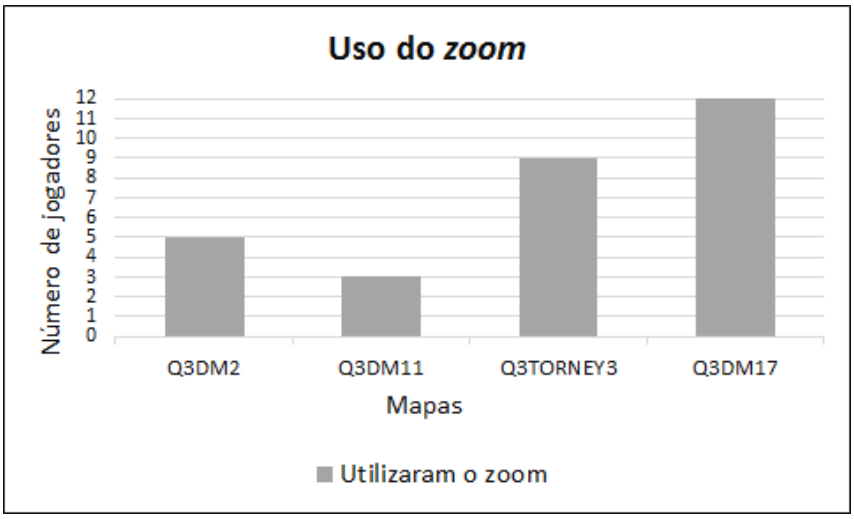

Figura 32: Utilização do *zoom* consoante o mapa.

	***Quake 3 Arena* original**	***Quake 3 Arena* com FpsTab**	**Ganho (face ao jogo original)**
Tráfego enviado por Servidor (Mb)	65,8	52,6	20%
CPU Servidor (%)	49,6	47,5	4,2%
RAM Servidor (%)	44,8	44,1	1,4%
Bateria Servidor (%)	2,4	1,9	18,4%
CPU Cliente (%)	61,8	60	3%
RAM Cliente(%)	84,6	84,4	0,25%
Bateria Cliente (%)	3,5	3	13,9%

Tabela 4: Resumo de resultados da avaliação qualitativa.

Como o gráfico da Figuara 32 demostra, a maioria dos jogadores usou *zoom* nos mapas abertos (*Q3TORNEY3* e *Q3DM17*). Nesse tipo de mapas os jogadores precisam da maior pontaria, visto que, as entidades se encontram a uma distância maior.

Opcionalmente, haviam algumas criticas sobre o *joystick*, principalmente durante o treino. Isto deve-se a inexperiência dos jogadores em jogos desse género nos dispositivos móveis. No entanto, os jogadores conseguiram adaptar-se rapidamente aos controlos.

5.7 Resumo dos resultados

Após a análise de resultados podemos concluir que, os principais objetivos do FpsTab foram alcançados. Obtivemos uma redução de tráfego, de utilização do CPU, RAM e da bateria do dispositivo móvel. Os resultados da avaliação qualitativa estão resumidos na Tabela 4. Segundo os resultados da Avaliação qualitativa, o FpsTab não prejudica a jogabilidade do *Quake 3 Arena*.

Podemos concluir que, o FpsTab comporta-se melhor nos mapas grandes e abertos, pois a redução do

FOV, quando se utiliza o *zoom*, também aumenta a filtragem de entidades desnecessárias. No entanto, os resultados quantitativos podiam ser melhores, caso fosse escolhido um jogo que não usa as técnicas de Gestão de Interesse.

Capítulo 6

Conclusão

Neste trabalho falamos sobre os jogos multi-jogador do género FPS e desafios que estes proporcionam em termos de consistência e escalabilidade. Também falamos sobre as arquiteturas dos jogos multijogador, Gestão de Interesse e replicação. Referimos as redes *ad-hoc*, aos desafios que estes colocam e falamos sobre os principais protocolos de comunicação.

Propusemos o FpsTab, uma biblioteca para melhorar a Gestão de Interesse dos jogos FPS. O nosso sistema usa as técnicas de Gestão de Interesse baseados em campo de visão e aura, que representam os limites sensórias dos avatares. Permitindo assim filtrar a informação desnecessária para o jogador.

Provamos que o FpsTab é eficiente, pois consegue diminuir o uso de recursos do dispositivo móvel, tais como CPU e a memória RAM. Devido ao seu algoritmo, nota-se a diminuição da carga de processamento de informação que beneficia o cliente e o servidor. O *middleware* FpsTab permite reduzir o uso do CPU em 4,2% no servidor e em 3% no cliente. Em termos da utilização da RAM, obtivemos uma redução de 1,4% no servidor e 0,25% no cliente.

A outra vantagem do FpsTab é a redução da largura da banda utilizada para o jogo, pois as suas técnicas de gestão de informação conseguem reduzir o tamanho das réplicas que o servidor dissemina na rede sem prejudicar a jogabilidade. Conseguimos reduzir o tráfego enviado por servidor em 20% face ao jogo original.

O FpsTab também contribui para aumento da durabilidade da bateria do dispositivo móvel. Conseguimos aumentar a vida da bateria em 18,4% no servidor e em 13,9% no cliente.

Podemos concluir que, as técnicas de Gestão de Interesse são muito eficientes na redução e filtragem de informação. A Gestão de Interesse baseada em Auras e em Campo de Visão são especialmente úteis em jogos do género FPS, pois adaptam-se perfeitamente à sua natureza.

6.1 Trabalho futuro

Apesar dos bons resultados demonstrados pelo FpsTab, este ainda pode ser melhorado. Os principais pontos a melhorar são:

- O sistema atual está muito focado no funcionamento do *Quake 3 Arena*. Para que o sistema seja completamente genérico, de maneira que possa ser integrado com qualquer jogo FPS, a biblioteca ainda precisa de ser melhorada nesse aspeto.
- O FpsTab foi testado num jogo com arquitetura Cliente-Servidor e lógica centrada no servidor. Podia-se experimentar a testar o jogo na mesma arquitetura mas com a lógica centrada no cliente. A integração com a arquitetura P2P também é algo que podia ser explorado no futuro.
- Para aumentar ainda mais o desempenho, podem ser acrescentadas mais técnicas de Gestão de Interesse e melhorar ja existentes, como por exemplo, reduzir a distância do FOV quando este esteja tapado por obstáculos.
- Opcionalmente, o *joystick* virtual também pode ser melhorado de maneira a acrescentar mais funcionalidades e adaptar a todas as resoluções de ecrã.

Bibliografia

[1] (2008). Bluetooth master/slave communications and sniff/sniff sub-rating modes. White paper, BARB.

[2] ABIONA, O.O., OLUWARANTI, A.I., ANJALI, T., ONIME, C.E., POPOOLA, E.O., ADEROUNMU, G.A., OLUWATOPE, A.O. & KEHINDE, L.O. (2009). Architectural model for wireless peer-to-peer (wp2p) file sharing for ubiquitous mobile devices. In *2009 IEEE International Conference on Electro/Information Technology, EIT 2009, Windsor, Ontario, Canada, June 7-9, 2009*, 35–39, IEEE.

[3] ATKINSON, P. (2008). A bitter pill to swallow: The rise and fall of the tablet computer. In *Design Issues*, vol. 24, 3–25.

[4] BALAKRISHNAN, M. & SADASIVAN, M. (2007). Mobile interactive game interworking in ims. In *IP Multimedia Subsystem Architecture and Applications, 2007 International Conference*, 1 –5.

[5] BARBEAU, M. & KRANAKIS, E. (2007). *Principles of Ad Hoc Networking*. John Wiley and Sons Ltd, p64-71, 1st edn.

[6] BARRETTA, D. (2011). Tablet pc power consumption analysis. Politecnico de Milano.

[7] BLEIDT, R. & IIS, F. (2005). Understanding mpeg-4: Technologies, advantages, and markets. In *MPEG Industry Forum*.

[8] BOULANGER, J. (2006). *Interest Management for Massively Multiplayer Games*. Canadian theses, McGill University (Canada).

[9] BOULANGER, J.S., KIENZLE, J. & VERBRUGGE, C. (2006). Comparing interest management algorithms for massively multiplayer games. In *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, NetGames '06, ACM, New York, NY, USA.

[10] BRANDT, D.H. (2007). Accelerating online gaming: Defining and defeating lag in online gaming. In *New Technology T-611, Computer Science, Reykjavík University*.

[11] CARROLL, A. & HEISER, G. (2010). An analysis of power consumption in a smartphone. In *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, USENIXATC'10, 21–21, USENIX Association, Berkeley, CA, USA.

[12] CARROLL, A. & HEISER, G. (2010). An analysis of power consumption in a smartphone. In *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, USENIXATC'10, 21–21, USENIX Association, Berkeley, CA, USA.

[13] CHEN, X., CHEN, Y., MA, Z. & FERNANDES, F.C.A. (2013). How is energy consumed in smartphone display applications? In *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications*, HotMobile '13, 3:1–3:6, ACM, New York, NY, USA.

[14] CHLAMTAC, I., CONTI, M. & LIU, J.J.N. (2003). Mobile ad hoc networking: imperatives and challenges. In *Ad Hoc Networks*, 1, 13–64.

[15] DOS SANTOS, M.J.F., VEIGA, L. & FERREIRA, P. (2011). Vfc-reckon for android mobile devices. In *CETC 2011 - Conference on Electronics, Telecommunications and Computers*, ISEL.

[16] EISERT, P. & FECHTELER, P. (2008). Low delay streaming of computer graphics. In *Proceedings of the International Conference on Image Processing, ICIP 2008, October 12-15, 2008, San Diego, California, USA*, 2704–2707, IEEE.

[17] FLICKENGER, R., AICHELE, C., FONDA, C., FORSTER, J., HOWARD, I., KRAG, T. & ZENNARO, M. (2007). *Wireless Networking in the Developing World*. wndw.net, 2nd edn.

[18] FRODIGH, M., JOHANSSON, P. & LARSSON, P. (2000). Wireless ad hoc networking – the art of networking without a network. In *Ericsson Review*, 4.

[19] HUANG, J., QIAN, F., GERBER, A., MAO, Z.M., SEN, S. & SPATSCHECK, O. (2012). A close examination of performance and power characteristics of 4g lte networks. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, MobiSys '12, 225–238, ACM, New York, NY, USA.

[20] JUNIOR, J.B.B. (2012). Do computador ao tablet: Vantagens pedagógicas na utilizaçãoo de dispositivos móveis na educação. In *Revista EducaOnline*, vol. 6.

[21] KANEDAY, Y., MINEMATSUZ, M., SAITOZ, M., AIDAZ, H. & TOKUDAY, H. (2005). Angel: A hierarchical state synchronization middleware for mobile ad-hoc group gaming. In *Proceedings of the 1st workshop on Network and system support for games*.

[22] KAUR, B. (2010). Factors influencing implementation of 4g with mobile ad-hoc networks in m-governance environment. In *IJCA Special Issue on MANETs*, 3, 141–146.

[23] KHAN, A.M., ARSOV, I., PREDA, M., CHABRIDON, S. & BEUGNARD, A. (2010). Adaptable client-server architecture for mobile multiplayer games. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, SIMUTools '10, 11:1–11:7, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium.

[24] KNUTSSON, B., LU, H., XU, W. & HOPKINS, B. (2004). Peer-to-peer support for massively multiplayer games. In *IEEE INFOCOM*.

[25] LACAGE, M., MANSHAEI, M.H. & TURLETTI, T. (2004). Ieee 802.11 rate adaptation: a practical approach. In *Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, MSWiM '04, 126–134, ACM, New York, NY, USA.

[26] LAI, A.S. & BEAUMONT, A. (2010). Mobile bluetooth-based multi-player game development in ubiquitous computing. In *Journal of Computational Information Systems 6:14 (2010) 4617-4625*.

[27] LINHARES, A.G. & DA S. GONÇALVES, P.A. (2009). Uma análise dos mecanismos de segurança de redes ieee 802.11: Wep, wpa, wpa2 e ieee 802.11w. Universidade Federal de Pernambuco (UFPE) - Centro de Informática (CIn).

[28] NVIDIA (2010). Nvidia tegra multi-processor architecture. White paper, Nvidia, Inc.

[29] NVIDIA (2011). Bringing high-end graphics to handheld devices. White paper, Nvidia, Inc.

[30] OPOKU, S.K. (2012). A simultaneous-movement mobile multiplayer game design based on adaptive background partitioning technique. vol. abs/1209.3052.

[31] O'REILLY, T. (2007). What is web 2.0: Design patterns and business models for the next generation of software. Mpra paper, University Library of Munich, Germany.

[32] PADGETTE, J., SCARFONE, K. & CHEN, L. (2012). *Guide to Bluetooth Security Recommendations*. National Institute of Standards and Technology.

[33] PANTEL, L. & WOLF, L.C. (2002). On the suitability of dead reckoning schemes for games. In *Proceedings of the 1st workshop on Network and system support for games*, NetGames '02, 79–84, ACM, New York, NY, USA.

[34] (PEIQUN), Y.A. & T., V.S. (2005). Mopar: a mobile peer-to-peer overlay architecture for interest management of massively multiplayer online games. In *Proceedings of the international workshop on Network and operating systems support for digital audio and video*, 99–104, ACM, New York, NY, USA.

[35] SAITO, Y. & SHAPIRO, M. (2005). Optimistic replication. vol. 37, 42–81, ACM, New York, NY, USA.

[36] SANTOS, N., VEIGA, L. & FERREIRA, P. (2007). Vector-field consistency for ad-hoc gaming. In *ACM/IFIP/Usenix International Middleware Conference (Middleware 2007)*, Lecture Notes in Computer Science, Springer.

[37] SENDRA, S., GARCIA, M., TURRO, C. & LLORET, J. (2010). Ieee 802.11a/b/g/n indoor coverage and performance comparison. In *Proceedings of the 2010 6th International Conference on Wireless and Mobile Communications*, ICWMC '10, 185–190, IEEE Computer Society, Washington, DC, USA.

[38] SIGNALS, R. (2012). Rs9110-n-11-02 802.11bgn wlan module. White paper, Redpine Signals, Inc.

[39] SMED, J., KAUKORANTA, T. & HAKONEN, H. (2002). *A Review on Networking and Multiplayer Computer Games*.

[40] STOJMENOVIC, I. (2007). *Handbook of Wireless Networks and Mobile Computing*. John Wiley and Sons, Inc., 1st edn.

[41] SYCHIP (2012). Sn8200 wi-fi network controller module. White paper, SyChip, LLC.

[42] TECHNOLOGY, S. (2010). Sx-sdpag 802.11a/b/g sdio surface mount module. White paper, Silex Technology, Inc.

[43] THAEY, E.D., FRANSSENS, N. & HELSEN, S. (2012). Smartphone power consumption. In *Materials, Energy and Sustainable Growth*, Antwerp, Belgium.

[44] TJENSVOLD, J.M. (2007). Comparison of the ieee 802.11, 802.15.1, 802.15.4 and 802.15.6 wireless standards.

Anexo 1

Questionário

Idade: ___ Sexo: M ☐ F ☐

Tem experiência com os jogos *FPS*: Muita ☐ Alguma ☐ Nenhuma ☐

Tem experiência com os jogos *FPS* no dispositivo móvel: Muita ☐ Alguma ☐ Nenhuma ☐

Mapa 1

1. Notou alguma diferença na jogabilidade, ao nível de fluidez e realismo do movimento dos adversários, entre as duas versões do jogo?

   Sim ☐ Qual? _____

   _____

   Não ☐

2. Usou *zoom* neste mapa? Sim ☐ Não ☐

Mapa 2

1. Notou alguma diferença na jogabilidade, ao nível de fluidez e realismo do movimento dos adversários, entre as duas versões do jogo?

   Sim ☐ Qual? _____

   _____

   Não ☐

2. Usou *zoom* neste mapa? Sim ☐ Não ☐

Mapa 3

1. Notou alguma diferença na jogabilidade, ao nível de fluidez e realismo do movimento dos adversários, entre as duas versões do jogo?

   Sim ☐ Qual? _____

   _____

   Não ☐

2. Usou *zoom* neste mapa? Sim ☐ Não ☐

Mapa 4

1. Notou alguma diferença na jogabilidade, ao nível de fluidez e realismo do movimento dos adversários, entre as duas versões do jogo?

   Sim ☐ Qual? _____

   _____

   Não ☐

2. Usou *zoom* neste mapa? Sim ☐ Não ☐