Arquiteturas de rede em jogos multi jogador no Unity

Nuno Carapito
Universidade da Beira Interior
Covilhã, Portugal
nfac93@gmail.com

Frutuoso G. M. Silva
Instituto de Telecomunicações
Universidade da Beira Interior, Covilhã, Portugal
fsilva@ubi.pt

RESUMO

O desenvolvimento de jogos digitais tem vindo a aumentar em todo o mundo, sendo estes desenvolvidos por empresas, pessoas individuais ou pessoas organizadas em pequenas equipas. Estes últimos, normalmente, contam com o apoio de motores de jogo, que têm como principal objetivo acelerar o desenvolvimento de jogos.

Nos jogos digitais, a componente multi jogador tem vindo a ganhar uma importância enorme, sendo atualmente raro um jogo sair para o mercado sem ter esta componente.

Este artigo apresenta a comparação entre as arquiteturas *peer-to-peer* e cliente/servidor usadas no desenvolvimento de um jogo multi jogador. Para este teste foi desenvolvido um protótipo usando o *Unity* para verificar qual das duas arquiteturas de rede é melhor para um jogo de estratégia multi jogador. Os testes preliminares realizados com jogadores, indicaram que a arquitetura *peer-to-peer* parece funcionar melhor.

Palavras Chave

Jogos digitais; multi jogador; arquitetura cliente-servidor; arquitetura peer-to-peer; motor de jogo; *Unity*.

INTRODUÇÃO

Nos dias que correm qualquer pessoa pode desenvolver um jogo, visto que existe uma quantidade elevada de tutoriais na Internet, uma documentação bastante completa e uma comunidade que ajuda imenso. No entanto, existe falta de documentação sobre como implementar diferentes arquiteturas de rede para jogos multi jogador *online*, e, entre as diferentes arquiteturas, quais as vantagens de umas comparativamente às outras, bem como saber qual delas se deve implementar para o tipo de jogo desejado. Este trabalho tenta colmatar essa falha de informação, orientada aos jogos de estratégia multi jogador desenvolvidos em *Unity*

Problema e Objetivos

O desenvolvimento de jogos digitais tem vindo a encontrar diversas barreiras ao longo dos anos, o que por vezes impediram o lançamento dos mesmos. Um exemplo disto é o *World Of Warcraft*, que encontrou uma barreira muito grande no lançamento do jogo a nível de arquitetura de rede, uma vez que cada servidor aguentava no máximo cinco mil jogadores ao mesmo tempo, tendo sido criados centenas de servidores diferentes. Com o passar do tempo, com maior capacidade de processamento por parte do *Hardware*, este limite tem sido aumentado e os servidores já chegam a ter cerca de dez mil jogadores e muitos deles interligados entre si, podendo os jogadores de um servidor jogar com jogadores de outro servidor. No entanto, continua a ser usada uma arquitetura cliente/servidor, que traz elevados custos para a empresa, que tem de sustentar toda a infraestrutura. Nos últimos anos tem sido feito um esforço a nível global para arranjar forma de transformar este tipo de arquitetura numa arquitetura *peer-to-peer*, na qual a informação é trocada diretamente entre os clientes, aliviando assim os encargos das empresas com os servidores.

O principal objetivo deste projeto é verificar qual das duas arquiteturas funciona melhor em jogos de estratégia multi jogador. Para isso, foi implementado um pequeno protótipo em *Unity* usando os dois tipos de arquitetura: a cliente/servidor e *peer-to-peer*. Um fator crítico no desenvolvimento deste protótipo foi o facto de não existir muita documentação do *Unity* sobre o desenvolvimento de jogos multi jogador *online*, usando qualquer uma das duas arquiteturas.

JOGOS MULTI JOGADOR

Os jogos multi jogador são jogos nos quais duas ou mais pessoas podem jogar no mesmo ambiente e ao mesmo tempo. Estes jogos permitem uma interação entre jogadores, quer seja em parceria ou em competição, oferecendo uma componente social que não existe nos jogos de um só jogador. Este tipo de jogos pode funcionar em dois modos distintos: o modo local e o modo *online*.

Videojogos'16, Nov 24–25, 2016, Covilhã, Portugal

Frutuoso Silva e Pedro Santos (eds.) ISBN 978-989-20-7148-0.

Em modo *online*, os jogos requerem que os jogadores se liguem a um servidor central (independente da arquitetura de rede usada), de forma a conseguir jogar com outros jogadores, utilizando uma ligação à Internet. Em modo local, os jogadores podem usar *Local Área Network (LAN)/Wireless Local Area Network(WLAN)*, ou até mesmo *Bluetooth*, para criar uma rede local com outros jogadores, não sendo necessária uma ligação à Internet.

O Aparecimento dos Jogos Multi Jogador

Os jogos multi jogador tiveram um aparecimento discreto, uma vez que os primeiros eram simples jogos só para um jogador, mas com características que eram visíveis em comunidades, como por exemplo, listas de melhores pontuações, torneios e *chats*. Isto permitia aos jogadores comparar os seus resultados com outros jogadores, bem como competir para ver quem conseguia obter mais pontos. Este tipo de jogos foram apelidados de "jogos competitivos orientados à comunidade" [1].

Os jogos multi jogador apareceram há mais de quarenta anos. Uns dos primeiros foram o *Empire* de 1973 e o *Spasim* de 1974, suportando oito e trinta e dois jogadores, respetivamente. Antes destes foram desenvolvidos jogos multi jogador, mas com um limite máximo de dois jogadores, como por exemplo o *Tennis for Two* e o *SpaceWar!*. No entanto, os recursos do jogo tinham de ser partilhados por todos os jogadores. Anos mais tarde, começaram a ser lançados jogos com base numa ligação de rede, como por exemplo o *Midi Maze*. Após estes, começaram a aparecer os jogos mais conhecidos, como por exemplo o *Doom*, um *First-Person Shooter*, com dois estilos de jogo, *Deathmatch* e *Arena*. Posteriormente, os mais conhecidos a surgir foram o *Counter-Strike*, o *Halo* e o *Unreal Tournament*.

Hoje em dia, o desenvolvimento de jogos com uma componente multi jogador em tempo real é cada vez mais popular.

Arquiteturas para Jogos Multi Jogador

Com o avançar do tempo e da tecnologia, a quantidade de jogos multi jogador tem vindo a crescer, não só em popularidade mas também em escala. Existem duas grandes arquiteturas de redes de computadores para jogos multi jogador: a arquitetura cliente/servidor e a arquitetura *peer-to-peer*.

Atualmente, na maioria dos jogos multi jogador é usada a arquitetura cliente-servidor. Neste tipo de arquitetura existe um servidor central ao qual todos os clientes (ou seja, todos os jogadores) se ligam para poderem começar a jogar. Todas as ações tomadas pelos jogadores são enviadas para o servidor sob a forma de uma mensagem. O servidor processa de forma sequencial as mensagens recebidas e vai responder aos clientes os efeitos das suas ações, também através de mensagens. A comunicação só ocorre entre o servidor e cliente e nunca entre os vários clientes. Este tipo de arquitetura começou a ser usada não só pela maior facilidade de implementação, mas também pela capacidade de evitar que os clientes façam batota[0], visto ser o servidor a tomar as decisões mais importantes.

Com o aumento da complexidade dos jogos multi jogador *Massively Multiplayer Online Games (MMOGs)*, está a aparecer um problema grave na arquitetura clássica cliente-servidor (principalmente nos *Massively Multiplayer Online Role-Playing Games, MMORPG*), devido ao elevado custo do preço em *hardware* e *data centers*. Nos últimos tempos, tem sido feita uma pesquisa intensa para adaptar a arquitetura *peer-to-peer* aos *MMOGs*, para espalhar a carga pelas máquinas dos jogadores [2]. Neste tipo de arquitetura, cada um dos jogadores funciona tanto como cliente quanto como servidor, existindo assim uma troca de informação entre os jogadores, sem que haja necessidade de um servidor central. No entanto, só a informação mais importante pode ser trocada com os vários jogadores, caso contrário, vai existir uma quantidade de informação bastante grande para ser trocada entre os vários jogadores, o que pode causar alguns problemas. É uma arquitetura bastante difícil de implementar, para além de também ser muito difícil prevenir que alguns jogadores façam batota, visto que os jogadores com experiência em redes podem alterar a informação dos pacotes de rede que enviam aos outros jogadores, sendo assim beneficiados (tal não é possível na arquitetura cliente/servidor pois é o servidor que faz todos os cálculos).

Principais Problemas dos Jogos Multi Jogador

Segundo Badar et al.[1], os principais problemas dos jogos *online* são: latência, perda de pacotes e pouca largura de banda.

A latência e o limite na largura de banda são os dois principais problemas nos jogos *online*. Relativamente à perda de pacotes, era um grave problema até há uns anos atrás. No entanto, já foram criados métodos para evitar que a perda de um pacote afete uma partida entre vários jogadores.

A largura de banda é definida como a quantidade de dados que pode ser transmitida do ponto A para o ponto B, num determinado período de tempo, sendo chamada na gíria por "limite de tráfego". Embora nos países mais desenvolvidos as larguras de banda sejam já bastante elevadas, ainda existem países com larguras de banda bastante reduzidas. Para além disso, existe também um limite bastante reduzido nas redes móveis, sendo do interesse do jogador que a quantidade de dados a serem transmitidos seja o mais baixo possível.

Outro problema é a latência, que é o tempo demorado entre o envio de um pedido (pacote de dados) e a chegada da sua resposta. Os jogos em tempo real necessitam de baixas latências, visto que um atraso no tempo de resposta pode causar degradação na experiência de jogo dos utilizadores. *Michael Powers* [3] afirma que o principal problema de um jogo em tempo real é a latência da rede. Um jogo com elevada latência vai afetar os jogadores, levando a um elevado nível de inconsistência, podendo até dar vantagem injusta para alguns deles, removendo assim a igualdade entre todos.

[0]Jogador altera a informação enviada nos pacotes de rede, ganhando vantagem sobre os outros jogadores.

Segundo Wang et al.[6], os efeitos da latência em jogos *online* podem ser categorizados da seguinte maneira: eficiência da rede, consistência visual, consistência do mundo de jogo e igualdade.

Para resolver o problema de eficiência da rede pode ser usado a técnica de *Content Addressable Network(CAN)*, que é um sistema distribuído *peer-to-peer* e descentralizado que mapeia chaves n-dimensionais em valores. O benefício principal de usar esta técnica advém da menor necessidade de pacotes relativamente a uma transmissão simples. Outra solução é gravar em *buffer* as mensagens do estado do jogo, e, em vez de se enviar várias mensagens individualmente, enviar um conjunto delas, causando menos problemas na rede.

Para resolver o problema da consistência visual existe uma técnica chamada *Dead Reckoning Technique* [6]. Normalmente, num jogo em tempo real, existem bastantes objetos, tais como criaturas, veículos, entre outros. Para que todos os jogadores possam receber uma atualização de todos os objetos é necessário transmitir uma quantidade elevada de informação entre o servidor e os clientes. Isto causa um enorme consumo de banda de rede e aumenta o risco de latência. Esta técnica extrapola a posição exata do objeto ao saber a sua última posição e velocidade. Com esta técnica, os clientes apenas precisam de dizer ao servidor as mudanças de velocidade dos objetos sobre o controlo do utilizador.

Relativamente à consistência do mundo de jogo, existe um método chamado de *Bucket Synchronisation Mechanism*, que tem a capacidade de recolher todas as mensagens de eventos do jogo e guardá-las num *bucket*. Em conformidade com um intervalo específico, todas as mensagens no *bucket* são processadas e é criada uma visão local do estado global. Esta é a melhor forma de sincronização para jogos multi jogador *online*, usando o sistema *peer-to-peer* [6].

Relativamente ao último problema, a igualdade é uma parte essencial para que um jogo seja justo e divertido para os jogadores. Uma das formas de resolver este problema é com o método chamado *Sync-MS*, que promove a igualdade nos jogos ao balancear o tempo de resposta. Visto que clientes de todo o mundo experienciam latências variadas, alguns jogadores podem tomar ações de acordo com a última mensagem, antes de outros jogadores terem sequer recebido essa mesma mensagem. Idealmente, todos os clientes devem receber as mensagens de atualização do servidor ao mesmo tempo, o que é conseguido através do mecanismo *Sync-out*. Este mecanismo coloca as mensagens de atualização em linha de espera e aguarda que estas tenham chegado a todos os clientes. Só após todos os clientes terem recebido a mensagem é que ela é entregue ao jogo, o que permite que os jogadores possam reagir ao mesmo tempo.

Sheldon et al.[4] partilharam na sua pesquisa o impacto da latência nos jogos de estratégia, referindo que não existem efeitos percetíveis quando a latência aumenta até 500 milissegundos. Latências por volta de 800 milissegundos causam degradação da experiência do jogo e transmitem informação errada ao jogador. No entanto, a frequência de atualizações nos jogos de estratégia não precisa de ser assim tão elevada, pelo que se pode "esconder" a latência, sem que o jogador tenha precepção da mesma.

Motores de Jogo

Um motor de jogo (*Game Engine* em Inglês) é um *framework* desenhado para a criação e desenvolvimento de videojogos. É usado principalmente por equipas pequenas para criação de jogos para computadores, consolas e dispositivos móveis. A principal funcionalidade de um motor de jogo é que engloba um motor gráfico para gráficos *2D* e *3D*, um sistema de física, sistema de som, sistema de *script*, animação, sistema de inteligência artificial, *networking*, entre outros. O processo de desenvolvimento de jogos é, assim, acelerado, ao reusar/adaptar os módulos do motor de jogo para vários jogos, ou para lançar o jogo em várias plataformas.

Atualmente existem muitos motores de jogo disponíveis. Eles variam em muitos aspetos, desde a principal linguagem de programação, à linguagem de *scripting*, se é *cross-platform*, se é orientado a *2D/3D*, a plataforma alvo e o tipo de licença.

Segundo a *Statista* [5], os cinco motores de jogo mais utilizados no Reino Unido são: *Unity 3D* (62%), *Unreal Engine* (12%), *Cocos2d* (9%), *CryEngine* 3 (5%), *Marmalade* (5%).

Neste projeto foi usado o *Unity*, cujos detalhes vão ser descritos na próxima subsecção. Relativamente ao *Unreal Engine*, é um motor de jogo desenvolvido pela *Epic Games* que foi usado pela primeira vez em 1998. Desde então tem vindo a ser melhorado, estando atualmente na versão 4. Tem como linguagens principais o C++ e o *UnrealScript*. Já o *cocos2D* foi lançado em 2008 e é *open source*. Este tem várias versões, cujas diferenças entre si são as plataformas alvo e as linguagens de programação (*Python, C++, ObjectiveC, C♯, Javascript*, entre outras). O *CryEngine* é o motor de jogo usado no *Far Cry*, desenvolvido pela *Crytex*. Foi tornado gratuito a 19 de agosto de 2011 e tem como linguagens principais o *C++, Lua e o C♯*. Por fim, o *Marmalade* é um motor de jogo desenvolvido pela *Marmalade Technologies Limited*. Tem como linguagens de programação *C/C++* e suporta um grande leque de plataformas.

O *Unity* foi o motor de jogo usado para o desenvolvimento do projeto. Inicialmente foi pensado usar-se o *Unity* ou o *Unreal Engine*. No entanto, a quando do desenvolvimento do projeto, o Unreal Engine não era gratuíto e para além disso tinha uma curva de aprendizagem bastante elevada. O *Unity* tinha algumas vantagens comparativamente aos concorrentes: não só tinha uma curva de aprendizagem bastante baixa, como também tinha um grande suporte *online* e muitos tutoriais por onde se podia aprender. Adicionalmente, tinha a possibilidade de programar os *scripts* em C♯.

IMPLEMENTAÇÃO DO PROTÓTIPO

O *Warcraft 3* foi um jogo desenvolvido pela *Blizzard Entretaintment* em 2002, que recebeu a sua primeira e única expansão em 2003. É um jogo de estratégia em tempo real (com alguns elementos *Role-Playing Game*) para computador. Foi um jogo muito bem recebido pela comunidade, não só por ser o terceiro jogo da saga *Warcraft*, mas também por trazer uma inovação no modo multi jogador, a adição de criaturas

controladas pelo computador. Além disso, trouxe também um editor de mapas, no qual a própria comunidade pode criar mapas customizados e jogá-los a partir da plataforma *online*.

A partir da customização dos mapas nasceu um dos mais conhecidos mapas do *Warcraft 3*, o *Defense of the Ancients: Allstars (DOTA)*. Este mapa tem sido atualizado ao longo dos anos, tendo a sua última atualização sido lançada em 27 de Março de 2015. Foi este mapa que fez surgir dois dos maiores *Multiplayer Online Battle Arena (MOBA)* da atualidade, o *League of Legends* e mais tarde o *Dota 2*.

Foi com esta premissa que foi pensando desenvolver o protótipo baseado no *Vampirism Fire*, criando assim um *Standalone* com funcionalidades parecidas.

O *Vampirism Fire* é jogado por duas equipas. Uma equipa é composta por um máximo de oito jogadores (daqui em diante chamados de humanos) e a outra é composta por um máximo de dois jogadores (daqui em diante chamados de vampiros).

O objetivo dos humanos é angariar recursos pelo mapa, construir a sua base num dos vários pontos disponíveis, defendê-la com torres, que atacam o inimigo automaticamente, e treinar um herói, de forma a derrotar os vampiros. O objetivo dos vampiros é encontrar as bases dos humanos pelo mapa e destruí-las, ganhando mais poder por cada unidade/edifício que destroem.

Cada humano tem uma unidade principal (o trabalhador), que é o que tem a possibilidade de criar edifícios/reparar edifícios. Cada jogador só tem um trabalhador e não pode produzir mais. Para além desta unidade, existe o recoletor, que serve para apanhar os recursos (árvores neste caso). Tem ainda uma unidade especial, o herói, que durante o desenvolvimento vai ser um tanque. O vampiro apenas tem a sua unidade principal, o herói.

O trabalhador pode construir três tipos de edifícios: a refinaria, que vai servir para construir recoletores e estes irem recolher madeira; a fábrica de guerra, que vai permitir ao jogador construir os seus tanques; e a torre de defesa, que irá atacar os inimigos mal estes se aproximem dela.

Existem dois tipos de recursos neste jogo: a madeira, que serve para construir novos edifícios, melhorar os mesmos e construir unidades; e a população, que vai limitar o treino de unidades. Cada humano só pode possuir no máximo um herói. A população do jogo pode ser aumentada até um máximo de duzentos através da construção de refinarias. Ganham os humanos se matarem os vampiros ou, por outro lado, ganham os vampiros se destruírem todos os trabalhadores.

Funcionalidades Implementadas

Para que este protótipo pudesse ser testado, foi necessário implementar as seguintes funcionalidades:

1. Movimento da câmara - a câmara move-se com os movimentos do rato e quando é usado o botão de scroll, a câmara desce ou sobe, o que permite fazer o zoom in ou zoom out, dependendo da direção em que foi feito o scroll;

2. Interface Gráfica - informa o jogador do que pode ou não fazer, permitindo-o interagir com vários botões. Foram também implementadas vários tipos de cursores, que vai transmitir ao jogador as ações que este pode tomar, um mini mapa e tooltips, que dão informação extra ao jogador;

3. Cor diferente para cada jogador;

4. Barra de vida - informa o jogador de uma forma visual a quantidade de vida das suas unidades;

5. Ações associadas aos botões do rato - selecionar e dar ações às unidades (mover, atacar, apanhar recursos);

6. Construção de edifícios e treino de unidades;

7. *SpawnPoint*;

8. Sons associados às varias ações disponíveis;

9. Condições de vitória e derrota;

10. Inteligência Artificial - torres atacam as unidades inimigas automaticamente, desde que estejam dentro do seu raio visual;

11. *Navigation Meshes* - indicar às unidades de forma inteligente para onde é que elas se podem ou não mover;

12. *Steering Behaviours* - quando se ordena o movimento de um grupo de unidades para um sítio, elas não colidem entre si, ajustando a sua posição final de acordo com a posição final das outras;

13. Sistema de grelha - indica ao jogador o sítio onde pode e não pode construir as suas unidades;

14. Fog of War - jogadores podem apenas ver uma região à volta das suas unidades.

É possível ver uma imagem do protótipo implementado na figura seguinte.

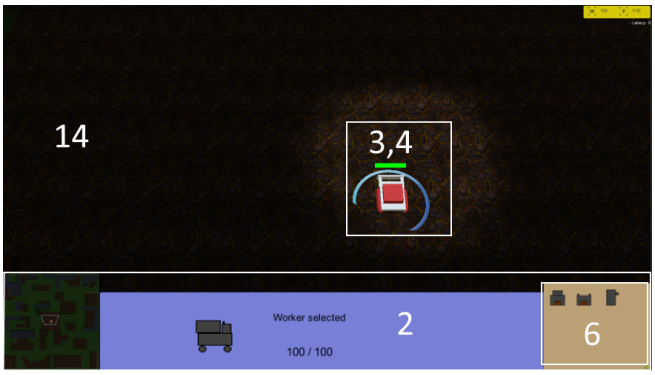

Figura 1: Protótipo implementado

Implementação do Networking

O *Networking*, apesar de ter sido a última componente a ser implementada, foi uma componente fundamental neste projeto, uma vez que os testes recaíram sobre esta componente. Para esta implementação, foi preciso criar um *player prefab*, um objeto em que vai ser instanciado sempre que um cliente se ligar ao jogo. Foi necessário criar uma nova cena com a

finalidade de fazer a ligação entre os jogadores e mandá-los para a cena de jogo.

Esta nova cena, com nome de Lobby, continha apenas um objeto vazio com duas componentes, o *Network Manager* e um *Network Manager HUD*. Foi também preciso adicionar algumas configurações ao *Network Manager*, tais como indicar que objeto seria o *player prefab* e quais seriam os prefabs que iriam ser gerados durante o jogo.

Chegando a esta parte do projeto foi preciso criar as duas versões para fazer os testes com os utilizadores. Foi feita uma cópia do projeto para que fosse possível criar as duas versões. O desenvolvimento deste projeto seguiu a ideologia *peer-to-peer*, ou seja, a autoridade dos objetos estava no lado dos jogadores. Neste caso, o servidor servia apenas para criar a sala de espera do jogo.

Para a implementação da arquitetura cliente-servidor houve necessidade de fazer mais alterações. Inicialmente teve de ser retirada a autoridade dos objetos por parte dos clientes. Esta alteração foi simples, bastou tirar o visto de "Local Player Authority" em todos os prefabs.

Após isso teve de ser feita uma alteração a todas as funções, porque qualquer ação que fosse feita tinha de passar pelo servidor (atacar, mover, recolher recurso). Assim tiveram de ser alteradas as funções que tinham a tag Command, para que toda a ação fosse comandada pelo servidor e depois replicada por todos os jogadores. Visto que as funções já estavam todas feitas, foi uma alteração rápida.

O último aspeto que teve de ser levado em consideração foi a forma como estavam a ser feitas as verificações de a quem pertencia a unidade/edifício. Até aqui eram feitas a partir da função "hasAuthority", o que agora iria retornar sempre *false*, porque quem tinha autoridade era o servidor. A função usada passou a ser isLocalPlayer, para que o jogador apenas pudesse controlar as suas unidades. O instanciamento de unidades passou a ser feito por NetworkServer.Spawn em vez de ser feito por NetworkServer.SpawnWithClientAuthority.

Faltava apenas criar algo para que os jogadores pudessem escolher a sua equipa. Foi usado como base o projeto *Lobby Manager* do *Unity*. As alterações feitas a este LobbyManager ocorreram ao nível da implementação de novas funcionalidades sobre ele, já que ele trata de todas as ligações entre os jogadores.

Este lobby, que apenas fazia com que vários jogadores se juntassem a uma sala de espera antes de começar o jogo, sofreu várias alterações, em que foi implementado as duas equipas, correção das cores de fundo dos painéis, implementação de scrollbar se houver muitos jogadores, apenas o hoster pode começar a partida, e só se todos os jogadores estiverem prontos e os jogadores podem cancelar o seu estado de pronto.

PROTOCOLO DOS TESTES

Foi criado um teste para verificar qual das duas arquiteturas implementadas funciona melhor no protótipo desenvolvido.

Quarenta utilizadores realizaram os testes ao protótipo desenvolvido, tendo avaliado as duas versões, cliente/servidor e *peer-to-peer*. Visto que a latência é um fator que influencia este tipo de testes, estes não foram realizados presencialmente, mas sim com cada uma das pessoas em sua casa, usando o seu computador pessoal. Assim, os jogadores não se ligaram à sala de espera do jogo através de uma ligação local, mas sim através de uma ligação a partir da Internet, ou seja, uma ligação real. Durante os testes, toda a conversação foi realizada mediante uma ligação *Skype* e a partida foi gravada para posterior análise.

Os testes foram aplicados a grupos de quatro pessoas, duas por equipa. Cada teste foi dividido em dois pequenos sub-testes, que serviram para avaliar os dois tipos de arquitetura. No teste da arquitetura cliente/servidor existiu um servidor dedicado para testar o protótipo, enquanto que no teste da arquitetura *peer-to-peer*, como a comunicação é feita entre os jogadores, apenas houve necessidade de um servidor para fazer a sala de espera do jogo.

Foi construído um formulário usando o *Google Forms*, que continha informação importante para os participantes que iriam realizar o teste. No início do formulário fez-se uma introdução do teste aos participantes, enunciando o seguinte:

- Foi explicado o que ia acontecer durante o teste e o motivo de ele estar a ser realizado;
- Foi referido que o teste ia testar o protótipo desenvolvido e não o participante;
- Foi dito que se o participante tivesse dificuldades em usar ou compreender o que era pedido, deveria referi-lo, visto que possivelmente outros participantes iriam sentir a mesma dificuldade;
- Foi dito ao participante que, se quisesse, poderia desistir do teste a qualquer momento e que o teste duraria cerca de trinta minutos;
- Foi enunciado que todos os dados recolhidos seriam usados apenas para a pesquisa, salvaguardando-se o anonimato dos participantes, e que em caso de publicação dos dados, estes seriam compilados com a informação de todos os outros participantes.

De seguida, foi solicitado aos participantes o preenchimento de um pequeno inquérito com alguma informação, tal como o género, a faixa etária, se costuma jogar e, se sim, qual o tipo de jogos que joga, quantas horas por semana e se os jogos que joga são *online* ou não.

Após esta informação foi mostrada uma lista de tarefas que os participantes deveriam completar por ordem sequencial, sem saltar nenhuma. Foram também informados que caso tivessem alguma dificuldade em completar uma das tarefas, deveriam apresentá-las no final do teste. Foi também questionado se tinham alguma dúvida antes de se dar início ao teste.

Assim que o participante acabava todas as tarefas deveria aguardar que todos os participantes o tivessem feito de modo a poder mudar de jogo (para a outra arquitetura), fazendo a mesma lista de tarefas. Após ter completado todas as tarefas usando as duas arquiteturas, deveria passar para a última

página do formulário, onde lhe era perguntado se tinha compreendido todas as tarefas, os objetivos, se conseguiu perceber como interagir com o protótipo e questões para comparar os dois tipos de arquitetura.

Resultados

Dos quarenta participantes, três eram raparigas, duas delas com uma idade superior a 27 anos e uma entre os 18 e os 20 anos. Os restantes participantes eram rapazes, treze entre os 18 e os 20 anos, catorze entre os 21 e os 23, oito entre os 24 e os 26 e dois com idade superior a 27 anos.

Entre os participantes, cinco deles não costumavam jogar no dia a dia. Dos restantes, dezasseis costumavam jogar jogos de estratégia, enquanto que dezanove estavam igualmente divididos entre *MOBA's* e *MMORPG*.

Por semana, três dos participantes jogavam menos de três horas, sete jogavam entre três a seis horas, dez jogavam entre seis e nove horas, cinco jogavam entre nove e doze horas e dez jogavam mais de doze horas. Dos trinta e cinco participantes que jogavam, dois não costumavam jogar jogos *online*.

Relativamente às perguntas sobre o teste, dos quarenta participantes, trinta e nove perceberam o objetivo de protótipo e um não. No entanto, todos eles perceberam como interagir com o protótipo, inclusive os cinco que não costumam jogar. Relativamente à pergunta sobre se tinham percebido as tarefas que foram pedidas, todos os participantes compreenderam à exceção de um, que referiu que não tinha percebido como atacar as unidades adversárias.

De modo geral, os problemas indicados pelos participantes puderam dividir-se em três categorias:

- *Fog of war* - alguns participantes sentiram dificuldades em encontrar as suas unidades, visto que a área de abertura no *fog of war* estava demasiado pequena. Este problema intensificava-se nas plataformas;
- Criação de unidades nas bordas das plataformas - se um participante construísse um edifício na borda de uma plataforma e treinasse uma unidade, existia a possibilidade de ela ir para a parte de baixo da plataforma;
- Falhas no movimento das unidades e no ataque - algumas unidades não respondiam corretamente à ordem de ataque e não evitavam as colisões da melhor maneira.

Um participante não conseguiu jogar o modo cliente/servidor. Embora não exista total certeza sobre qual a origem do problema, o participante estava ligado à rede *MEO-WIFI*. Houve também três participantes que perderam a ligação com o servidor (modo cliente-servidor). Estes três participantes estavam a partilhar a ligação à Internet com um quarto que não perdeu a ligação.

Em relação à pergunta se notaram diferença entre os dois testes efetuados, dezoito participantes responderam que não, enquanto vinte e dois afirmaram que sim. Relativamente às diferenças encontradas, as respostas foram unânimes, os jogadores referiram que sentiram menos lentidão na arquitetura *peer-to-peer* comparativamente à arquitetura cliente/servidor, assim como uma maior fluidez do protótipo (*frame rate* constante) e maior rapidez de resposta das unidades. Já na arquitetura cliente/servidor as unidades demoravam algum tempo a realizar as ordens dadas, quer fossem de movimento ou de ataque.

Relativamente à fluidez do protótipo, trinta e quatro participantes disseram que a arquitetura *peer-to-peer* era fluída, enquanto que os outros seis referiram sentir alguma lentidão. Para além disso, dezassete deles afirmaram que a arquitetura cliente/servidor também era fluída, enquanto que os restantes vinte e dois disseram que era lenta. Um participante não respondeu a esta pergunta pois não conseguiu testar este modo.

Já na última pergunta, a qual questionava se os participantes tinham notado alguma demora por parte das unidades em reagir quando era dada alguma ordem, vinte e seis referiram não terem sentido nenhuma lentidão, enquanto que catorze disseram que sim, relativamente à arquitetura *peer-to-peer*. Já na arquitetura cliente/servidor, dezasseis disseram que não sentiram nenhuma lentidão, enquanto que vinte e dois afirmaram que sim. Um participante não respondeu a esta pergunta pois não conseguiu testar este modo.

PRINCIPAIS CONCLUSÕES E TRABALHO FUTURO

Com base nos resultados obtidos através dos testes efetuados com quarenta participantes, pode-se concluir que a arquitetura *peer-to-peer* no *Unity* funcionou melhor para a maioria dos participantes, em que apenas uma pequena minoria sentiu alguma lentidão neste modo. Esta lentidão poderá não ser causada pela arquitetura de rede, mas pela ligação à Internet do participante ou por causa do participante estar a usar um computador com algumas limitações. Este resultado difere dos resultados obtidos na arquitetura cliente-servidor, onde a maioria dos participantes disseram que este modo era lento.

De forma a tentar obter resultados mais conclusivos relativamente a estas duas arquiteturas, seria necessário realizar os testes em ambiente controlado, ou seja, todos os participantes deveriam usar computadores com especificações iguais e usarem uma ligação à Internet igual, estando todos no mesmo local. Poderia também ser usada uma ligação dedicada de *1gb* por segundo de *upload* no servidor (foi usada uma ligação dedicada de *100mb* por segundo de upload neste teste).

Poderiam também ser feitos testes automáticos, usando as *test tools* do *Unity*, de forma a simular uma partida com vários jogadores e conseguir perceber qual das duas arquiteturas funciona melhor, sem ser necessário ter pessoas a jogar presencialmente. Poderiam também ser feitos testes usando uma arquitetura híbrida, apesar de o *Unity* não fazer nenhuma referência se é possível a sua implementação.

O facto de o *Unity* apenas ter documentação e tutoriais de implementações usando a arquitetura *peer-to-peer* pode indicar que esta seja a melhor arquitetura a ser usada na implementação de jogos multi jogador ou que deverá ser a arquitetura privilegiada pelos programadores.

AGRADECIMENTOS

O autor filiado ao Instituto de Telecomunicações agradece o suporte dado através do programa FCT projeto UID/EEA/50008/2013.

REFERENCES

1. M. Badar and T. Nikhil. 2013. Development Guidelines for Mobile Multiplayer Games. (2013), 26.
2. R. Humphrey, A. Allan, and G Di Fatta. 2012. Using Spacial Locality and Replication to Increase P2P Network Performance in MMO Games. (2012), 6.
3. M. Powers. 2006. Mobile Multiplayer Gaming, Part 1: Real-Time Constraints. (2006).
4. N. Sheldon, Girard E., S. Bord, M. Claypool, and E. Agu. 2003. The Effect of Latency on User Performance in Warcraft III. (2003), 12.
5. Statista. 2014. Leading game engines used by video game developers in the United Kingdom (UK) 2014. (2014). `http://www.statista.com/statistics/321059/game-engines-used-by-video-game-developers-uk/`
6. A. I. Wang, M. Jarrett, and E. Sorteberg. 2009. Experiences from implementing a mobile multiplayer real-time game for wireless networks with high latency. (2009), 15.