Projeto de Mestrado em

Engenharia Informática - Computação Móvel

# *LiNGS – Framework de desenvolvimento de jogos multijogador em rede para dispositivos móveis*

**Valter José Correia Costa**

Leiria, setembro de 2014

Projeto de Mestrado em

Engenharia Informática - Computação Móvel

# *LiNGS – Framework de desenvolvimento de jogos multijogador em rede para dispositivos móveis*

**Valter José Correia Costa**

Projeto de Mestrado realizado sob a orientação da Doutora Catarina Silva, Professora da Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Leiria

Leiria, setembro de 2014

# Agradecimentos

A realização de um projeto desta envergadura é algo física e mentalmente desgastante. Felizmente, não realizei este trabalho sozinho, contei com a ajuda de excelentes pessoas que ao meu lado se mostraram uma mais-valia.

Desde já agradeço à Doutora Catarina Silva pela orientação deste projeto e por todas a contribuições que me levaram à conclusão do projeto com sucesso. As experiências obtidas de um mentor são algo que define o nosso trabalho e que nos mostra como aplicar o nosso conhecimento de forma a superar dificuldades e a atingir os objetivos definidos.

Quero agradecer com um especial carinho à minha família por esta oportunidade e por todo o apoio que me deram durante todo o desenvolvimento do projeto.

Agradeço também a todas as pessoas amigas e conhecidas que direta ou indiretamente influenciaram positivamente a concretização deste projeto.

Por último quero agradecer à Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Leiria, a todos os professores do Departamento de Informática e, em especial, aos professores do Mestrado em Engenharia Informática – Computação Móvel, com especial menção ao Professor Doutor António Pereira e aos coordenadores do mestrado durante a elaboração do projeto: Professor Doutor Nuno Costa e Professor Doutor Carlos Grilo.

*Esta página foi intencionalmente deixada em branco.*

# Nota prévia

Do trabalho efetuado resultou o seguinte artigo para submissão:

- Valter Costa, Catarina Silva, "*LiNGS: Framework for networked multiplayer mobile games*", 37th International Conference on Software Engineering, 2015 (a submeter à *Software Engineering In Practice (SEIP) Track*).

*Esta página foi intencionalmente deixada em branco.*

# Resumo

O crescimento do mercado de dispositivos móveis causou uma grande procura de formas de entretenimento, nomeadamente jogos digitais, já existentes noutras plataformas como consolas e computadores pessoais. A facilidade na forma de desenvolvimento e distribuição para este tipo de dispositivos criou oportunidades para empresas e programadores produzirem jogos para plataformas móveis.

As limitações dos dispositivos móveis obrigaram a certas restrições de conteúdo e de funcionalidades nos jogos desenvolvidos, nomeadamente a opção de multijogador em rede. Várias características dos dispositivos móveis não permitem a correta implementação de jogos multijogador em rede da mesma forma que é feito em plataformas fixas, como por exemplo computadores pessoais. Por outro lado, a evolução das tecnologias de rede e de processamento trouxe a possibilidade de criação de jogos multijogador em rede para os dispositivos móveis, existindo no entanto ainda várias limitações e precauções que devem ser respeitadas no desenvolvimento de jogos para este tipo de dispositivos.

Neste projeto são propostos a especificação e o desenvolvimento de uma *framework* de suporte para a criação de jogos multijogador em rede – LiNGS (*Lightweight Networked Game System*). A framework LiNGS tem como principal objetivo facilitar a criação de jogos multijogador em rede para dispositivos móveis, abstraindo as limitações e os problemas característicos dos dispositivos e dos meios associados e fornecendo funcionalidades que permitam agilizar o desenvolvimento e execução do jogo.

O sistema LiNGS foi desta forma desenhado e desenvolvido para fornecer o suporte e tratar os problemas dos dispositivos móveis no contexto de jogos multijogador em rede, limitando e precavendo os problemas de redes, reduzindo o consumo de bateria e o processamento a cargo dos dispositivos.

Foram realizados vários testes quem comprovam a eficácia e fiabilidade do sistema LiNGS. Para tal foi criado um jogo de teste com a utilização do LiNGS. As várias técnicas utilizadas no sistema permitem a redução de 50% dos pacotes enviados nas redes de comunicação. Estes resultados mostram que o desenvolvimento de jogos multijogador para dispositivos móveis é, atualmente, uma realidade e que os problemas do seu desenvolvimento e funcionamento podem ser limitados ou resolvidos por sistemas como o LiNGS.

**Palavras-chave:** Jogos, Dispositivos Móveis, Multijogador, Redes, *Framework*, LiNGS

*Esta página foi intencionalmente deixada em branco.*

# Abstract

The growth of the mobile devices market generated a huge demand for entertaining applications, mostly digital games, which were previously available on other platforms, such as personal computers and game consoles. This huge demand and the now easy way for developers to create and distribute applications on these devices created opportunities for companies and independent developers to produce games for mobile devices.

However, the constrained power and capabilities of mobile devices restricted the content and functionalities of the games. The characteristics of these devices do not allow for the correct implementation of certain functions, namely the multiplayer over network connections.

With the constant evolution and improvement of mobile devices, the possibility to create games with network multiplayer is now a reality. Nevertheless, there are still issues that developers should take into account when developing this type of games.

In this work we present the design, development and testing of a framework – LiNGS (*Lightweight Networked Game System*). The purpose of this system is to facilitate the development of network multiplayer games, while handling all the restrictions and problems that arise from the mobile devices and their networks.

The design and development of the LiNGS system provides support for networked games, while also providing support to handle the problems of networks, battery consumption and limited processing power.

Several tests, using a custom created game, were carried out and were able to prove the efficiency and reliability of the LiNGS framework. Furthermore, the conducted tests indicate that this system is able to reduce the packet transmission rate by 50% with all the designed components active. These results show that the development of network multiplayer games

is now easier and that the problems that once restricted these type of applications are now handled or even completely resolved by systems that apply similar solutions as the LiNGS system.

**Keywords:** Games, Mobile Devices, Multiplayer, Networks, Framework, LiNGS

*Esta página foi intencionalmente deixada em branco.*

# Lista de Figuras

2.1 Tipos de aplicações móveis mais populares. . . . . . . . . . . . . . . . . . 7
2.2 Camadas de uma aplicação nativa. . . . . . . . . . . . . . . . . . . . . . . . 8
2.3 Camadas do sistema Android e iOS. . . . . . . . . . . . . . . . . . . . . . 10
2.4 Camadas base de aplicação com utilização de *frameworks* especializadas **(a)** e aplicação com utilização de *framework* sistema **(b)**. . . . . . . . . . . . 12
2.5 Camadas numa aplicação desenvolvida com Marmalade. . . . . . . . . . 16

3.1 Máquinas ligadas numa configuração em anel. . . . . . . . . . . . . . . . . 30
3.2 Ligação de rede ponto-a-ponto. . . . . . . . . . . . . . . . . . . . . . . . . 31
3.3 Configuração de rede tipo cliente-servidor. . . . . . . . . . . . . . . . . . . 32
3.4 Representação do funcionamento em rede da versão inicial do Quake. . . . 32
3.5 Representação de estado de um jogador com problemas de rede, visto por outro jogador. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33

4.1 Integração do sistema LiNGS nas camadas de um jogo. . . . . . . . . . . 49
4.2 Arquitetura do LiNGS - Parte incluída com o jogo. . . . . . . . . . . . . 53
4.3 Arquitetura do LiNGS - Parte incluída com o servidor. . . . . . . . . . . 55
4.4 Estabelecimento da ligação entre cliente e servidor. . . . . . . . . . . . . 58
4.5 Sequência de sincronização de dados entre servidor e cliente. . . . . . . . 60
4.6 Processo de envio de mensagens importantes. . . . . . . . . . . . . . . . 62
4.7 Arquitetura do LiNGS - Integração dos componente cliente e servidor. . . . 63

5.1 Estrutura dos elementos que compõem um jogo em Unity. . . . . . . . . . 69
5.2 Unity Editor com as zonas de configuração de elementos de jogo realçadas. 70
5.3 Processo de compilação / execução de uma aplicação na linguagem C#. . . 72
5.4 Solução LiNGS com os projetos que compõem o sistema LiNGS. . . . . . 73

5.5 Arquitetura de funcionamento do sistema LiNGS quando integrado nas *frameworks* .NET e Unity. . . . . . . . . . . . . . . . . . . . . . . . . . . 74
5.6 Diagrama de classes do sistema de atualizações. . . . . . . . . . . . . . . 75
5.7 Diagrama de classes do encapsulamento de mensagens do LiNGS. . . . . . 77
5.8 Diagrama de classes dos identificadores de objetos a sincronizar. . . . . . 78
5.9 Classe com os marcadores para o sistema LiNGS. . . . . . . . . . . . . . 80
5.10 Classe central `LiNGSClient`. . . . . . . . . . . . . . . . . . . . . . . . 81
5.11 Tipos utilizados como parâmetro do construtor da parte cliente do sistema LiNGS. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 82
5.12 Interação do sistema LiNGS com o jogo. . . . . . . . . . . . . . . . . . . 82
5.13 Sistema de referências centralizado - `LiNGSClient`. . . . . . . . . . . . 83
5.14 Diagrama de classes da parte de comunicação do sistema LiNGS - parte cliente. 84
5.15 Diagrama de classes da parte de gestão interna do sistema LiNGS - parte cliente. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 86
5.16 Diagrama de classes do grupo de processamento de dados do sistema LiNGS - parte cliente. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 88
5.17 Diagrama de classes do grupo de simulação no sistema LiNGS - parte cliente. 90
5.18 Diagrama de classes do ponto de entrada da implementação do sistema LiNGS - parte servidor. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 91
5.19 Diagrama de classes da arquitetura da implementação do sistema LiNGS - parte servidor. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 93
5.20 Diagrama de classes da parte de comunicação do sistema LiNGS - parte servidor. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 94
5.21 Diagrama de classes do grupo de gestão da parte de servidor do sistema LiNGS. 95
5.22 Diagrama de classes de processamento de dados do sistema LiNGS - parte servidor. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 97
5.23 Diagrama de classes da parte de sincronização do sistema LiNGS - parte servidor. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 98
5.24 Exemplo de sincronização de objetos baseada na distância do cliente. . . . 99
5.25 Diagrama de classes de persistência de estados do sistema LiNGS - parte servidor. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 100
5.26 Estrutura de diretórios utilizada para persistir as sessões de clientes. . . . . 101
5.27 Integração do sistema LiNGS nas camadas de um jogo. . . . . . . . . . . . 102
5.28 Exemplo de integração do sistema LiNGS num objeto de jogo. . . . . . . . 103

6.1 Representação do jogo Striker utilizado para testar o sistema LiNGS. . . . . 109
6.2 Diagrama de classes da implementação do servidor para o jogo Striker. . . . 110
6.3 Jogo Striker a executar em dois dispositivos móveis. . . . . . . . . . . . . . 113
6.4 Processo de deteção de falha de ligação e restabelecimento da ligação. . . . 114
6.5 Dados recolhidos do sistema LiNGS em funcionamento normal. . . . . . . 115
6.6 Dados relativos ao número de mensagens, recolhidos do sistema LiNGS com o sistema de agregação de mensagens desativado. . . . . . . . . . . . . . . 116
6.7 Dados relativos ao tamanho de mensagens, recolhidos do sistema LiNGS com o sistema de agregação de mensagens desativado. . . . . . . . . . . . 117
6.8 Dados recolhidos do sistema LiNGS sem a utilização de sincronização contextual. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 118
6.9 Comparação dos resultados dos testes realizados. . . . . . . . . . . . . . . 119
6.10 Duração dos vários testes realizados. . . . . . . . . . . . . . . . . . . . . . 121
6.11 Duração dos vários testes realizados. . . . . . . . . . . . . . . . . . . . . . 122

A.1 Diagrama de classes da implementação comum da *framework* LiNGS. . . . 139
A.2 Diagrama de classes da implementação da parte cliente da *framework* LiNGS. 140
A.3 Diagrama de classes da implementação da parte servidor da *framework* LiNGS. 141

*Esta página foi intencionalmente deixada em branco.*

# Lista de Tabelas


2.1 Vantagens do desenvolvimento nativo e desenvolvimento com *frameworks* . 14
2.2 Quadro de comparação entre as *frameworks* analisadas . . . . . . . . . . . 26

3.1 Comparação entre as topologias de funcionamento em rede analisadas . . . 37

4.1 Protocolo de mensagens utilizado pelo LiNGS . . . . . . . . . . . . . . . . 50

5.1 Pontos de acesso do sistema LiNGS - parte cliente. . . . . . . . . . . . . . 104
5.2 Pontos de acesso do sistema LiNGS - parte servidor. . . . . . . . . . . . . 105

6.1 Utilizadores que realizaram testes ao jogo Striker. . . . . . . . . . . . . . . 120

*Esta página foi intencionalmente deixada em branco.*

# Acrónimos e Siglas

| **Acrónimo/Sigla** | **Significado** |
|---|---|
| API | *Application Programming Interface* |
| CIL | *Common Intermediate Language* |
| DLL | *Dynamic-Link library* |
| GUID | *Globally Unique IDentifier* |
| IA | Inteligência Artificial |
| IDE | *Integrated Development Environment* |
| IETF | *Internet Engineering Task Force* |
| IO | *Input / Output* |
| IP | *Internet Protocol* |
| IPX | *Internetwork Packet Exchange* |
| LiNGS | *Lightweight Networked Game System* |
| LTE | *Long-Term Evolution* |
| MIDI | *Musical Instrument Digital Interface* |
| MMOG | *Massively Multiplayer Online Game* |
| NPC | *Non-Player Character* |
| RFC | *Request for Comments* |
| RPC | *Remote Procedure Call* |
| SDK | *Software Development Kit* |
| UDP | *User Datagram Protocol* |
| XML | *eXtensible Markup Language* |

*Esta página foi intencionalmente deixada em branco.*

# Conteúdo


**1 Introdução** **1**
- 1.1 Motivação . . . . . 2
- 1.2 Objetivos . . . . . 2
- 1.3 Metodologia e estrutura do documento . . . . . 3

**2 Criação de jogos para dispositivos móveis** **5**
- 2.1 Introdução . . . . . 5
- 2.2 Desenvolvimento nativo . . . . . 8
- 2.3 Desenvolvimento com *frameworks* . . . . . 11
- 2.4 Comparação entre desenvolvimento nativo e desenvolvimento com *frameworks* 13
- 2.5 *Frameworks mobile* para desenvolvimento de jogos . . . . . 15
  - 2.5.1 Marmalade . . . . . 16
  - 2.5.2 Unreal . . . . . 18
  - 2.5.3 Corona SDK . . . . . 19
  - 2.5.4 Unity3D . . . . . 21
  - 2.5.5 Cocos2d-x . . . . . 22
  - 2.5.6 MonoGame . . . . . 24
  - 2.5.7 Comparação de *frameworks* . . . . . 25
  - 2.5.8 *Frameworks mobile* com suporte a multijogador em rede . . . . . 27
- 2.6 Conclusão . . . . . 27

**3 Tecnologias multijogador em rede** **29**
- 3.1 Introdução . . . . . 29
- 3.2 Modos de funcionamento . . . . . 35
  - 3.2.1 Arquitetura em Anel . . . . . 35
  - 3.2.2 Arquitetura Ponto-a-ponto . . . . . 35
  - 3.2.3 Arquitetura Cliente-servidor . . . . . 36

3.2.4 Comparação dos modos de funcionamento . . . . . . . . . . . . . 37
3.3 Multijogador em rede em dispositivos móveis . . . . . . . . . . . . . . . . 38
3.3.1 Problemas no desenvolvimento de jogos multijogador em rede para dispositivos móveis . . . . . . . . . . . . . . . . . . . . . . . . . 38
3.3.2 Trabalhos e frameworks multijogador para dispositivos móveis . . . 39
3.3.3 Análise de soluções para os problemas de redes em dispositivos móveis 42
3.4 Conclusão . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 44

**4 LiNGS – *Framework* proposta 47**
4.1 Introdução . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 47
4.2 Solução proposta . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 48
4.3 Arquitetura . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 52
4.3.1 Cliente . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 52
4.3.2 Servidor . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 55
4.4 Sequências de acontecimentos . . . . . . . . . . . . . . . . . . . . . . . 57
4.4.1 Estabelecimento da ligação . . . . . . . . . . . . . . . . . . . . . 58
4.4.2 Sincronização de dados . . . . . . . . . . . . . . . . . . . . . . . 59
4.4.3 Fecho da ligação . . . . . . . . . . . . . . . . . . . . . . . . . . . 61
4.4.4 Outros pontos de comunicação . . . . . . . . . . . . . . . . . . . 62
4.5 Integração . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 63
4.6 Conclusão . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 64

**5 Implementação 65**
5.1 Introdução . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 65
5.2 Requisitos e funcionamento base do LiNGS . . . . . . . . . . . . . . . . 65
5.3 Framework base . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 66
5.3.1 Funcionamento Interno do Unity . . . . . . . . . . . . . . . . . . 67
5.4 Implementação do sistema LiNGS . . . . . . . . . . . . . . . . . . . . . 71
5.4.1 Estrutura de implementação . . . . . . . . . . . . . . . . . . . . . 72
5.4.2 Arquitetura interna . . . . . . . . . . . . . . . . . . . . . . . . . 73
5.4.3 Implementação Comum . . . . . . . . . . . . . . . . . . . . . . . 75
5.4.4 Cliente . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 80
5.4.5 Servidor . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 91
5.5 Integração com jogos . . . . . . . . . . . . . . . . . . . . . . . . . . . . 102
5.5.1 Pontos de acesso - API . . . . . . . . . . . . . . . . . . . . . . . 104

5.6 Conclusão . . . . . . . . . . 106

**6 Análise de funcionamento** **107**

6.1 Introdução . . . . . . . . . . 107

6.2 Metodologia de teste . . . . . . . . . . 107

6.2.1 Implementação de jogo para testes . . . . . . . . . . 108

6.3 Resultados e análise . . . . . . . . . . 113

6.3.1 Testes de utilização . . . . . . . . . . 120

6.4 Conclusão . . . . . . . . . . 123

**7 Conclusões e trabalho futuro** **125**

**Bibliografia** **129**

**A Diagramas de classes** **139**

A.1 Diagrama de classes da implementação da parte comum da *framework* LiNGS 139

A.2 Diagrama de classes da implementação da parte cliente da *framework* LiNGS 140

A.3 Diagrama de classes da implementação da parte servidor da *framework* LiNGS 141

# Capítulo 1

# Introdução

A utilização de jogos como forma de entretenimento é algo bastante comum desde a criação do computador. A curiosidade e ingenuidade dos programadores permitiram a criação de jogos mesmo em sistemas que não eram originalmente desenhados para tal. O sucesso desta forma de entretenimento foi de tal forma notório que começaram a ser lançados sistemas concebidos apenas para jogos, tendo os computadores começado a trazer sistemas desenhados para suportar jogos.

A evolução destes sistemas permitiu enriquecer os jogos possibilitando a utilização de conteúdos com melhor qualidade e a criação de novas funcionalidades. A possibilidade da utilização de redes de comunicação provou-se um conceito bastante importante na evolução dos jogos.

Várias plataformas como as consolas de jogos e computadores pessoais permitem a utilização de redes de comunicação para disponibilizar componentes multijogador para jogos. Desde a década de 90 do século XX que redes de comunicação são utilizadas para permitir a funcionalidade de multijogador em consolas de jogos, como na Sega Mega Drive [3].

Atualmente, os jogos multijogador em rede são bastante comuns. Vários jogos são desenvolvidos de forma a funcionarem exclusivamente em multijogador em rede. Este tipo de jogos, especialmente os MMOG (*Massively Multiplayer Online Game*), são bastante populares devido a todas as interações possíveis em tempo real com outros jogadores. O World of Warcraft é um jogo do tipo MMOG e desde o seu lançamento já foi jogado por mais de 100 milhões de utilizadores [4].

O conceito de multijogador em rede em dispositivos móveis, nomeadamente telemóveis, tem vindo a ser introduzido no mercado, no entanto, as limitadas capacidades destes dispositivos nunca permitiram criar experiências semelhantes ao que outros dispositivos permitem. Atualmente as plataformas móveis têm ainda capacidades limitadas, mas a evolução das tecnologias utilizadas possibilita a criação de jogos multijogador em rede.

## 1.1 Motivação

A introdução de jogos multijogador em dispositivos móveis é um conceito pouco explorado pela indústria atual. Os problemas afetos a este tipo de jogos junto com as limitações dos dispositivos tornam o desenvolvimento difícil, o que afasta as empresas da integração de multijogador em rede em jogos para dispositivos móveis.

A motivação principal deste trabalho surge com a fraca adesão ao conceito multijogador em rede demonstrada pelos criadores de jogos para dispositivos móveis, tendo em consideração o enorme mercado deste tipo de dispositivos e o mercado de aplicações.

## 1.2 Objetivos

Pretende-se com este projeto a criação de uma *framework* para suporte a jogos multijogador em rede para dispositivos móveis. Esta *framework* tem como principal tarefa permitir o desenvolvimento de jogos em rede, para isso deve fornecer capacidades de integração em jogos e suportar a sincronização de dados em tempo real. A *framework* deve abstrair os problemas afetos aos dispositivos móveis e às redes de comunicação e deve contribuir de forma a reduzir o impacto nos recursos dos dispositivos, nomeadamente no consumo de bateria.

Com a concretização deste projeto pretende-se desenhar, implementar e testar uma *framework* para jogos multijogador em rede para dispositivos móveis. Este tipo de jogos podem ser bastante complexos e, por esse motivo, não se pretende realizar uma *framework* fechada, mas sim realizar uma contribuição para a comunidade e distribuir a estrutura e implementação da *framework* sobre uma licença *open-source* para que possa ser utilizada, analisada e alterada de acordo com as necessidades de cada projeto.

## 1.3 Metodologia e estrutura do documento

De acordo com os objetivos definidos para o projeto, a primeira fase consiste em realizar uma análise das formas de desenvolvimento para dispositivos móveis. O Capítulo 2 apresenta uma análise das técnicas utilizadas para desenvolvimento, incluindo desenvolvimento de forma nativa e desenvolvimento com auxílio de *frameworks*. São estudadas várias *frameworks* e é realizada uma comparação das suas características e ferramentas.

De seguida, no Capítulo 3, é realizada uma análise às formas de desenvolvimento de jogos em multijogador em rede, descrevendo as técnicas utilizadas para o funcionamento deste tipo de jogos e as adaptações já existentes para dispositivos móveis.

Com base na recolha e análise das informações anteriores, no Capítulo 4 a *framework* é especificada e é apresentada a sua arquitetura. São ainda estabelecidas as várias características e funcionalidades que devem ser cumpridas pela *framework*.

O Capítulo 5 consiste na implementação do sistema. A *framework* base utilizada para realizar a implementação do sistema é analisada de forma detalhada e é apresentada a estrutura de implementação. Vários detalhes da implementação são expostos e documentados de acordo com os requisitos e necessidades do sistema.

A análise do funcionamento do sistema é apresentada no Capítulo 6, onde a implementação realizada é submetida a vários testes. É implementado um jogo de forma a testar o funcionamento do sistema e são ainda realizados vários testes com o objetivo de validar o funcionamento e comportamento do sistema de acordo com os requisitos definidos anteriormente.

No Capítulo 7 é analisado o trabalho desenvolvido e são apresentadas as conclusões. São ainda apresentadas neste capítulo várias propostas de trabalho futuro.

# Capítulo 2

# Criação de jogos para dispositivos móveis

## 2.1 Introdução

Os jogos para dispositivos móveis são um conceito já com alguns anos. Vários dispositivos móveis lançados na década de 90 do século XX traziam jogos pré-carregados. Um dos mais icónicos jogos para dispositivos móveis, Snake, foi lançado em 1997 em dispositivos Nokia [5]. Outros dispositivos móveis criados por outras marcas começaram também a integrar este tipo de jogos que, apesar de serem simples, mostravam ter um grande potencial pela forma cativante que conquistavam os jogadores.

Com o crescimento abrupto do mercado de dispositivos móveis que se verificou principalmente na última década [6], os jogos para este tipo de dispositivos tiveram também uma grande evolução [7], que deu origem ao mercado dos jogos para dispositivos móveis. O mercado de vendas de jogos para dispositivos móveis teve o seu grande início por volta de 2002/2003, com a massificação de telemóveis com ecrã a cores e tecnologia 3G [8]. O ecrã a cores e a possibilidade de acesso à Internet permitiam aos utilizadores descarregar e usufruir de músicas, vídeos e jogos para os seus terminais. Para potenciar este mercado as operadoras de comunicações criaram portais de venda *online* onde os utilizadores podiam facilmente comprar conteúdos. Surge, desta forma, o mercado de venda de jogos para dispositivos móveis.

Apesar do crescimento do mercado de venda de jogos para dispositivos móveis, o seu funcionamento não era perfeito, existiam muitas limitações. Entre as maiores limitações estava o facto de os jogos criados terem de ser aprovados e publicados pelas operadoras móveis. Esta

restrição limitava o número de criadores de jogos, já que existiam custos elevados para criar e publicar um jogo. Adicionalmente, o número de utilizadores finais era também limitado, visto que os jogos e restantes conteúdos podiam não estar disponíveis em todas as operadoras. Este mercado era assim bastante restrito e consequentemente era também pequeno.

Em 2007 é lançado um dispositivo móvel que vem alterar este conceito, o iPhone [9][10]. Este dispositivo apresentava um ecrã a cores de alta resolução e capacidades de comunicação em rede. No ano seguinte, 2008, com a criação da loja App Store [11] qualquer utilizador podia criar e publicar de forma fácil e simples aplicações e jogos para o iPhone. Esta loja baseou-se no conceito da loja Xbox Live Marketplace lançada em 2005 para a plataforma Xbox 360 [12] [13]. A App Store permite que empresas e pessoas individuais tenham acesso aos mecanismos de criação de jogos e aplicações para a plataforma. Com todas estas ferramentas os programadores criam jogos e enviam-nos para publicação. Depois de um processo de aprovação, os jogos ficam disponíveis para os utilizadores da plataforma. Os criadores dos jogos tem o controlo sobre os mesmos, incluindo o preço de venda. Os lucros gerados desta forma são divididos entre o publicador e o criador do jogo. No caso da App Store os lucros obtidos são divididos geralmente em 30% para a loja e os restantes 70% para o criador do jogo.

Outros sistemas seguiram a ideia de lojas virtuais devido às suas vantagens. Ainda em 2008, a Google lança a Google Play Store [14] para o sistema Android [15], permitindo aos utilizadores submeter e descarregar aplicações, jogos e outros conteúdos multimédia. Por sua vez, a Microsoft lançou em 2010 a loja Windows Phone Store [16] para o sistema operativo Windows Phone [17], que veio substituir o sistema Windows Mobile. Estes sistemas operativos partilham o conceito de criação e distribuição de conteúdos. Os programadores têm acesso às ferramentas para criação de aplicações, podendo distribui-las nas respetivas lojas virtuais de cada plataforma, de forma a que essas sejam disponibilizadas aos utilizadores de forma simples e rápida.

Atualmente, a quantidade e variedade de aplicações submetidas para as lojas virtuais é enorme. Ainda assim, o mercado de jogos para dispositivos móveis é um dos mais fortes destas lojas. A Figura 2.1 demonstra que os jogos são o tipo de aplicação mais popular nos dispositivos móveis. Esta figura assinala que 64% dos jogos descarregados são utilizados durante 30 ou mais dias, sendo o tipo de aplicação mais utilizado. Seguem-se outras aplicações como aplicações de meteorologia e redes sociais com 60% e 56%, respetivamente. As apli-

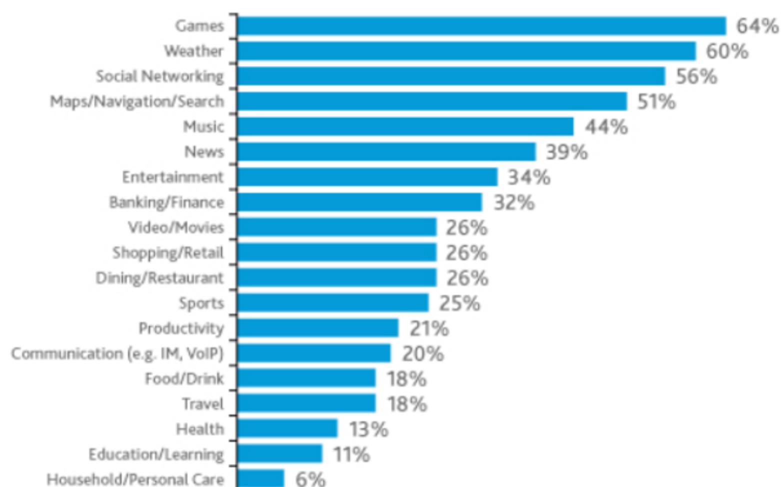


Figura 2.1: Tipos de aplicações móveis mais populares (Adaptado de [1]).

cações menos utilizadas são as de categorias como educação, saúde e tratamento pessoal em que a percentagem de utilização durante os 30 ou mais dias depois do descarregamento das aplicações é de 6%. No contexto atual, com a enorme difusão de dispositivos móveis com um ecrã grande, capacidade de processamento gráfico e acesso à Internet, as lojas virtuais são inundadas com novas aplicações que são descarregadas e utilizadas inúmeras vezes pelos vários utilizadores das plataformas, mas tal como demonstrado, as aplicações com mais uso são os jogos. Estes tiram partido da capacidade de processamento, comunicação e do ecrã dos dispositivos móveis de forma a proporcionar entretenimento aos utilizadores.

Depois da fase inicial, a criação de jogos para dispositivos móveis está agora numa nova etapa. A concorrência é enorme e os criadores de jogos procuram formas de desenvolver jogos rapidamente, com baixo custo e que tenham um público-alvo alargado, i.e. disponibilizar os jogos em múltiplas plataformas, o que implica a sua implementação para cada plataforma. Surge então, mais do que nunca, a necessidade de utilizar mecanismos que facilitem estas condições, tais como *frameworks*.

As *frameworks* fornecem mecanismos que permitem criar aplicações de forma rápida e ágil, podendo ainda suportar a exportação da aplicação para múltiplas plataformas, reduzindo drasticamente o esforço de programação necessário. No entanto, o desenvolvimento nativo, sem auxílio de *frameworks*, ainda é bastante utilizado. Ambos os tipos de desenvolvimento fornecem vantagens na sua utilização. Assim, vamos proceder a uma avaliação comparativa

Figura 2.2: Camadas de uma aplicação nativa.

das características de cada tipo de desenvolvimento, como se apresenta de seguida.

## 2.2 Desenvolvimento nativo

Cada plataforma móvel (Android, iOS, Windows Phone, etc.) tem uma interface e uma camada aplicacional que permitem a construção e integração de novas aplicações. O sistema de integração vulgarmente denominado de SDK (*Software Development Kit*), disponibilizado pelas plataformas permite usufruir das suas capacidade de forma limitada mas segura. O acesso ao sistema operativo é reduzido para evitar problemas de segurança e inconsistências técnicas entre dispositivos. O SDK de cada sistema operativo garante aos utilizadores que as aplicações instaladas não podem alterar o sistema ou obter e alterar dados a que não tenham acesso. Utilizando estes mecanismos é possível criar aplicações que tirem partido de todo o sistema, incluindo outro *software* instalado e *hardware*.

O desenvolvimento nativo consiste, assim, na utilização somente do sistema de integração fornecido pela plataforma para a criação de aplicações.

Em desenvolvimento nativo existem normalmente 3 camadas na execução de uma aplicação, como apresentado na Figura 2.2. Assim, a aplicação não sofre penalizações em relação ao desempenho, visto não existirem camadas intermédias para encapsular ou executar dados, ou mesmo o código da aplicação em si.

Utilizando o sistema de integração (SDK) é possível obter todas as funcionalidades disponíveis na plataforma, podendo no entanto ser necessário implementar mecanismos que não venham incluídos no SDK, o que no caso de jogos digitais é bastante comum. Os jogos digitais utilizam sistemas de simulação de ambientes físicos, comunicação em rede, reprodução de som, etc., que normalmente necessitam de funcionalidades que muitas vezes não estão direta e adequadamente disponíveis nos sistemas operativos. Desta forma, a disponibilização destes sistemas utilizados em jogos implica a sua implementação baseada nas funcionalidades disponíveis no sistema operativo.

No entanto, existem outros problemas, eventualmente mais importantes, no desenvolvimento nativo. Quando uma aplicação é desenvolvida de forma nativa, esta apenas irá executar no sistema para qual foi implementada. Por exemplo, uma aplicação desenvolvida utilizando o Android SDK [18] apenas executa no sistema operativo Android, ou em sistemas que implementem a interface de programação do Android, como por exemplo o BlackBerry 10 [19]. A falta de portabilidade é, assim, um dos maiores problemas do desenvolvimento nativo. Cada sistema operativo tem, geralmente, um SDK diferente. Até sistemas com a mesma base têm mudanças significativas quando se trata da interface disponível para programadores (API - *Application Programming Interface*). Os sistemas Android e iPhone, apesar de baseados no Unix, têm interfaces de programação completamente diferentes.

Existem vários *standards* adotados pelos fabricantes de sistemas operativos de forma a implementar uma interface de desenvolvimento que permita portabilidade nas aplicações desenvolvidas. O POSIX [20] estabelece um conjunto de normas que definem uma interface portátil independente do sistema operativo base. Esta interface contém acesso a funcionalidades base para desenvolvimento de aplicações. A criação e gestão de processos, *multithreading* e acesso a ficheiros são exemplos das funcionalidades estabelecidas pelo POSIX. Outras funcionalidades, como gestão de interfaces, não estão incluídas no POSIX. Portanto, apesar de existirem conjuntos de normas *standard* que definem a criação de sistemas com suporte a portabilidade, existem ainda algumas limitações. Apenas funcionalidades para suporte à lógica da aplicação estão especificadas. A criação e gestão de interfaces gráficas é normalmente realizada de forma nativa ou com recurso a outras ferramentas, como *frameworks*.

Adicionalmente, estes *standards* são integrados em vários sistemas operativos: Windows, MacOS e outros baseados em Unix e Linux. A utilização deste tipo de normas permite aos programadores desenvolver aplicações portáveis com base nativa, conseguindo partilhar

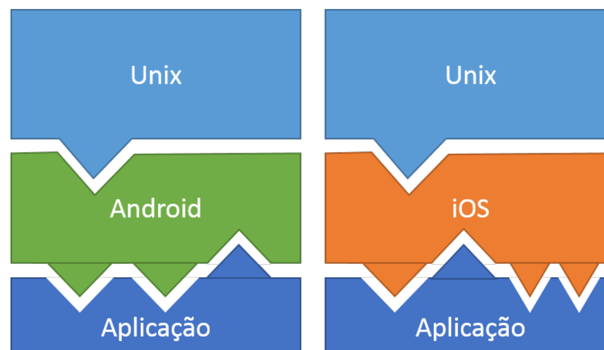


Figura 2.3: Camadas do sistema Android e iOS.

grande parte do código da aplicação entre vários sistemas.

Na Figura 2.3 é mostrada uma analogia das camadas de software desde o núcleo de cada sistema operativo até às aplicações. No caso demonstrado, os dois sistemas operativos Android e iOS têm um núcleo baseado em Unix. No entanto, a diferença a nível de desenvolvimento de aplicações nativas é bastante acentuada. A interface de cada sistema é completamente diferente e existe uma enorme disparidade nas operações e funções que cada aplicação pode fazer dentro do sistema. Até mesmo a linguagem de programação é diferente entre os sistemas [21] [22]. Desta forma, uma aplicação desenvolvida para iOS não "encaixa" na interface disponível no sistema Android e vice-versa.

Assim, podemos concluir que o desenvolvimento nativo é uma forma de desenvolvimento com várias vantagens e desvantagens. A criação de aplicações nativas permite utilizar todas as funcionalidades disponibilizadas pela plataforma alvo utilizando a interface disponível no sistema. Este tipo de aplicações tem ainda vantagens ao nível de desempenho. A inexistência de camadas de *software* intermédias elimina processamentos intermédios. Assim, a aplicação é executada diretamente pelo sistema. Apesar de favoráveis, a forma como se obtêm estas vantagens introduz inconvenientes a outros níveis. A falta de portabilidade é um fator de peso na criação de aplicações. Esta limitação implica a parcial ou mesmo total recriação da aplicação para que possa executar em múltiplas plataformas.

O desenvolvimento nativo implica também a implementação de todos os sistemas lógicos para suporte às funcionalidades da aplicação. Sendo que os sistemas operativos não integram várias funcionalidades que podem ser necessários a aplicações, por exemplo, simulação de

ambientes físicos, os programadores têm de implementar estes sistemas lógicos caso sejam necessários na aplicação.

Estes problemas tornam o desenvolvimento nativo ineficaz quando o objetivo é disponibilizar uma aplicação em múltiplas plataformas. Para minimizar, ou até em vários casos eliminar este problema, foram criadas *frameworks* que auxiliam o desenvolvimento e permitem portabilidade entre sistemas, como é explicado na próxima secção.

## 2.3 Desenvolvimento com *frameworks*

O desenvolvimento com *frameworks* vem trazer vários benefícios aos programadores, principalmente na criação de aplicações multiplataforma. Uma *framework* consiste num conjunto de sistemas que fornecem suporte à implementação de aplicações, contendo funcionalidades implementadas e prontas a utilizar por aplicações sem que os programadores as tenham de reimplementar para cada aplicação. O pacote global de uma *framework* geralmente contém ainda ferramentas para o auxílio na utilização da *framework* e no desenvolvimento de aplicações.

De certa forma, as *frameworks* são acrescentos aos sistemas operativos das plataformas, que utilizam a interface disponibilizada por esses sistemas operativos e fornecem funcionalidades específicas de acordo com o objetivo funcional da *framework*. De forma a obter portabilidade, as *frameworks* são implementadas com base em vários sistemas operativos.

As *frameworks* podem fornecer um conjunto de funcionalidades relacionadas com um tema em específico ou podem consistir num sistema de desenvolvimento completo. Desta forma, é possível dividir as *frameworks* em dois grandes grupos:

- *Frameworks* especializadas
- *Frameworks* de sistema

A diferença entre os dois tipos reflete-se nas funcionalidades da *framework*. Uma *framework* que fornece um conjunto de funcionalidades restritas a um tema é denominada de *framework* especializada. Um exemplo é a *framework* Box2D [23], utilizada na criação de jogos, especializada apenas na simulação física de objetos. Este tipo de *frameworks* cinge-se princi-

palmente ao acrescento de funcionalidades apenas para a aplicação e não para o desenvolvimento, não fornecendo ferramentas para agilizar diretamente o desenvolvimento da aplicação.

Por outro lado, uma *framework* de sistema consiste na simulação de um ambiente de desenvolvimento. Exemplos deste tipo são a .NET Framework [24] ou Marmalade [25]. Estas *frameworks* fornecem ao programador funcionalidades suficientes para a criação de aplicações e fornecem ainda ferramentas de auxílio ao desenvolvimento. Várias *frameworks* deste tipo, como o Unity [26], fornecem até um ambiente de desenvolvimento completo (IDE - *Integrated Development Environment*) onde é possível programar a aplicação sem recorrer a ferramentas fora do ambiente fornecido. Estes pacotes de desenvolvimento são aliciantes para os programadores não apenas por todo o ambiente fornecido, com várias ferramentas de desenvolvimento incluídas com a *framework*, mas também pelo suporte e comunidade envolvidas. O desenvolvimento deste tipo de *frameworks* é regularmente influenciado pela comunidade, sendo a .NET Framework um exemplo. A .NET Framework tem uma comunidade bastante forte e muitas mudanças realizadas na *framework*, inclusive alterações à linha de desenvolvimento, são pedidos que partem da comunidade [27].

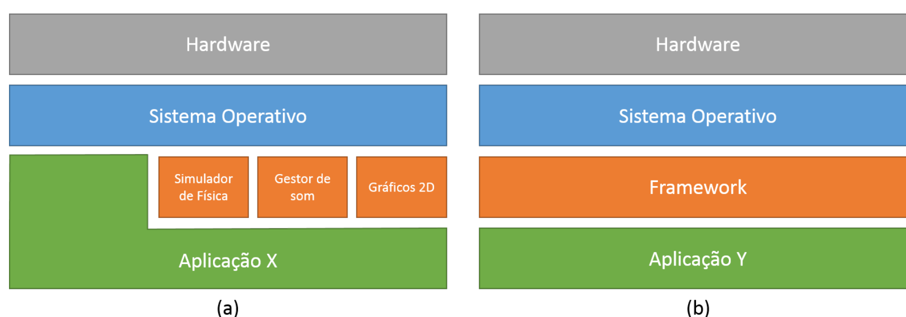


Figura 2.4: Camadas base de aplicação com utilização de *frameworks* especializadas **(a)** e aplicação com utilização de *framework* sistema **(b)**.

Na Figura 2.4 encontra-se uma representação das camadas de uma aplicação que utiliza várias *frameworks* e de outra aplicação que utiliza uma *framework* do tipo sistema. A aplicação X, apresentada à esquerda na figura, utiliza várias *frameworks* especializadas com o intuito de aumentar as funcionalidades fornecidas pelo sistema operativo. Desta forma, são disponibilizadas várias funcionalidades pelas *frameworks*, tais como simulação de ambientes físicos e gestão de som, que são cruciais para a aplicação. No entanto, o núcleo da aplicação é de-

senvolvido de forma nativa, sendo suportado diretamente pelo sistema operativo e tirando partido de todas as funcionalidades do sistema e plataforma.

Uma *framework* de sistema fornece várias funcionalidades específicas, mas fornece também uma camada entre a aplicação e o sistema operativo. Esta camada geralmente abstrai o sistema operativo, o que permite que a mesma aplicação consiga executar em vários sistemas operativos diferentes. Desta forma, a aplicação Y, apresentada à direita na Figura 2.4, não acede diretamente ao sistema operativo. A interface do sistema operativo não é explorada diretamente pela aplicação, sendo a *framework* uma ponte entre a aplicação e o sistema. Desta forma, a aplicação é independente do sistema operativo. Assim, estas *frameworks* têm um grande potencial, pois conseguem facultar as funcionalidades básicas de sistemas operativos de uma forma genérica e simplificada e, ao mesmo tempo, abstrair a ligação entre a aplicação e o sistema operativo, conseguindo, assim, tornar a aplicação portável entre sistemas operativos, eliminando a necessidade de reimplementação da aplicação para múltiplos sistemas.

## 2.4 Comparação entre desenvolvimento nativo e desenvolvimento com *frameworks*

O desenvolvimento nativo e o desenvolvimento com auxílio de *frameworks* são ambos muito utilizados atualmente. Durante a década de 90 do século XX a utilização de *frameworks* era pouco usual. Em geral, cada empresa desenvolvia todos os subsistemas para as suas aplicações. Com o avanço das tecnologias, principalmente a Internet, a partilha de informação cresceu exponencialmente e possibilitou a massificação de *frameworks*. Os utilizadores partilhavam conhecimento entre si de forma a criar uma base com várias funcionalidades úteis para os seus projetos. Por sua vez, a comercialização de *frameworks* tornou-se também num negócio para as empresas.

No entanto, o desenvolvimento nativo ainda é bastante utilizado, principalmente quando a portabilidade não é um requisito essencial da aplicação. As aplicações nativas são otimizadas para a plataforma alvo e conseguem, dessa forma, ter um desempenho superior, sendo ainda que na ausência de camadas intermédias, normalmente oriundas de *frameworks*, evitam adicional processamento de dados. Além da vantagem ao nível de desempenho, o desenvolvimento nativo tem o benefício de disponibilizar acesso a todas as funcionalidades da

plataforma alvo. As *frameworks* limitam funcionalidades de baixo nível para ficarem facilmente portáveis para múltiplas plataformas, o que não acontece no desenvolvimento nativo onde todas as funcionalidades da plataforma estão acessíveis.
Ainda assim, a utilização de desenvolvimento nativo puro tem vindo a diminuir. As vantagens provenientes da utilização de *frameworks* são, principalmente no contexto atual, bastante importantes devido ao número de plataformas alvo e às necessidades empresariais, como o tempo de desenvolvimento. O desenvolvimento com *frameworks* é agora bastante comum na criação de aplicações, incluindo jogos. A possibilidade dos programadores não precisarem de implementar sistemas de suporte às várias funcionalidades da aplicação é bastante vantajoso, quer a nível de custo como de tempo de desenvolvimento. A possibilidade de criar aplicações portáveis é também uma das principais vantagens das *frameworks*.

| **Vantagens** | |
|---|---|
| **Desenvolvimento nativo** | **Desenvolvimento com frameworks** |
| Performance | Portabilidade |
| Acesso a todos os sistemas da plataforma | Funcionalidades prontas a utilizar |
| Controlo total da aplicação | Ambiente de desenvolvimento |

Tabela 2.1: Vantagens do desenvolvimento nativo e desenvolvimento com *frameworks*

As vantagens de cada tipo de desenvolvimento encontram-se resumidas na Tabela 2.1. Para além das vantagens apontadas, o desenvolvimento com auxílio de *frameworks* pode trazer vários benefícios indiretos aos programadores e às empresas que desenvolvem aplicações, nomeadamente o facto de as *frameworks* disponibilizarem várias funcionalidades, que reduz o tempo que seria gasto a implementar essas funcionalidades e que será aproveitado para melhorar o produto ou até mesmo lançar o produto mais cedo. Esta vantagem é ainda mais relevante no desenvolvimento para dispositivos móveis, visto que as plataformas móveis têm uma capacidade de processamento limitada, as *frameworks* são implementadas para realizar as tarefas mais pesadas computacionalmente de forma otimizada. Ou seja, não existe necessidade de conhecimento de sistemas de baixo nível de cada plataforma, visto que a *framework* trata de realizar as otimizações para que a aplicação execute da forma mais eficiente em cada plataforma.

Sendo assim, o custo de desenvolvimento de uma aplicação utilizando *frameworks* é relativamente mais baixo do que desenvolver uma aplicação totalmente nativa para cada plataforma alvo. Mesmo tendo em conta que as *frameworks* têm usualmente um custo associado, este

custo poderá ser contido pelas vantagens que reduziriam os custos de produção e também pelos benefícios da possibilidade de lançar a aplicação em múltiplas plataformas, o que abrange um mercado ampliado de utilizadores.

## 2.5 *Frameworks mobile* para desenvolvimento de jogos

Com o forte crescimento do mercado de jogos para dispositivos móveis e dadas as vantagens do desenvolvimento com *frameworks*, têm vindo a ser criadas *frameworks* especializadas para este tipo de tarefa: criação de jogos para dispositivos móveis. Desta forma, existe atualmente uma vasta seleção de *frameworks* para criação de jogos destinados a dispositivos móveis. As partes principais de um jogo podem ser desenvolvidas com o auxílio de *frameworks*, desde o áudio, desenho gráfico, simulação de física ou até mesmo o sistema de pontuações [28]. No entanto, tal como analisado previamente na Secção 2.3, uma *framework* não se define apenas pela parte do código / execução. As ferramentas e auxílio para o desenvolvimento do jogo, tanto a nível de código como de ferramentas visuais, são cada vez mais importantes para as empresas que desenvolvem jogos. Assim, torna-se imprescindível tomar uma decisão ponderada quando se escolhe uma *framework* para desenvolver um produto.

Tendo em consideração o mercado atual de jogos digitais para dispositivos, apresentamos de seguida as principais características que as empresas e os programadores têm em consideração quando escolhem uma *framework* para desenvolver um jogo para dispositivos móveis:

- **Portabilidade –** A aplicação desenvolvida deverá executar em múltiplas plataformas.
- **Desenvolvimento rápido –** Menos programação e tarefas de desenvolvimento menos demoradas.
- **Desenvolvimento ágil –** O desenvolvimento do produto com a utilização da *framework* deve ser mais fácil do que desenvolver de forma nativa para as várias plataformas

Outras características são também analisadas durante a escolha de *frameworks* para o desenvolvimento de jogos, nomeadamente:

- **Custo** (Método de Licenciamento)
- **Performance** (Suporte para jogos complexos)
- **Capacidades técnicas** (2D, 3D, etc.)

- **Comunidade** (Suporte, conteúdos, etc.)

Repare-se que estas caraterísticas apenas são determinantes em casos específicos e não numa visão global do mercado. Existem projetos que podem necessitar de funcionalidades mais invulgares e, nestes casos, podem ser necessárias outras características para completar o projeto conforme os requisitos definidos.

Em suma, as empresas e os programadores querem que as *frameworks* lhes permitam disponibilizar o jogo em mais plataformas, diminuir o tempo de desenvolvimento e também requerer menos capacidades técnicas para o desenvolvimento. Apesar de nem sempre ser possível atingir todos estes objetivos, por vezes contraditórios, vamos proceder a uma análise de várias *frameworks* disponíveis no mercado e realizar uma comparação das características de cada uma.

### 2.5.1 Marmalade

A Marmalade (Marmalade SDK), previamente conhecida como AirPlay SDK, é uma *framework* desenvolvida pela empresa Marmalade Technologies Limited [25], que tem como principal objetivo fornecer uma camada de abstração no acesso às APIs de cada plataforma.

Figura 2.5: Camadas numa aplicação desenvolvida com Marmalade.

A *framework* Marmalade serve de intermediário entre a aplicação e o sistema operativo, como se apresenta na Figura 2.5. Desta forma, fornece suporte para que a aplicação execute em múltiplos sistemas operativos sem necessidade de ajustes ou recriação da mesma. Esta *framework* executa em múltiplos tipos de dispositivos e sistemas operativos, nomeadamente:

iOS, Android, Windows Phone, Tizen, BlackBerry, MacOS, Windows [29]. Existe inclusive suporte para SmartTVs. A portabilidade é um dos pontos fortes desta framework.

A nível de funcionalidades esta é uma *framework* bastante completa. Todos os subsistemas básicos para a implementação de jogos para dispositivos móveis estão presentes, por exemplo, processamento gráfico, áudio e Internet. Outros sistemas mais avançados como simulação de física não são englobados na *framework*, estando a sua utilização apenas disponível com a utilização de pacotes extra. A utilização de funcionalidades individuais de cada plataforma tem, na sua maioria, de ser acrescentada à *framework*. Sendo que apenas existe acesso a funcionalidades genéricas como a camara fotográfica e acelerómetro.

O desenvolvimento nesta *framework* é feito em código, ou seja, não recorre a um ambiente gráfico para desenvolvimento. O desenvolvimento em Windows é feito no Microsoft Visual Studio [30] e em MacOS o desenvolvimento é feito em XCode [31]. A linguagem de desenvolvimento é C++ e, sendo esta uma linguagem de baixo nível, existe pouco *overhead* na utilização da *framework*. De forma a ser compatível com múltiplos sistemas operativos, é suportada a compilação para as várias plataformas, sendo que a Marmalade permite ainda realizar uma configuração fina das opções individuais de cada plataforma, tais como os ícones da aplicação, definir as versões do sistema operativo compatíveis ou definir requisitos necessários da plataforma, no entanto esta configuração tem de ser feita através de *scripting*, não existindo um editor visual para tal.

Outras ferramentas disponíveis incluem um criador básico de interfaces gráficas e também um emulador para simular a execução do jogo nas várias plataformas suportadas. Estes editores ajudam o desenvolvimento, permitindo acelerar a fase de testes em várias plataformas. Esta *framework* inclui também a Marmalade Quick. Esta é uma versão de mais alto nível da *framework* Marmalade e é destinada à criação de jogos em 2D. Esta versão permite que programadores com menos experiência consigam desenvolver jogos de forma mais simples, sem necessitarem de recorrer a todas as capacidades da Marmalade quando tal não é necessário.

A Marmalade tem um custo associado de $499 por ano por empresa na versão mais simples. Versões profissionais com acesso a suporte dedicado, sem limite de vendas e acesso a portabilidade para todas as plataformas disponíveis podem chegar a $3500 por ano por empresa. Existe ainda uma versão gratuita com limite de vendas e que pode ser utilizada por 3 progra-

madores por empresa que apenas permite a criação de jogos para iOS e Android, sendo que estes têm ainda de reproduzir um símbolo indicando que foram feitos com auxílio de esta *framework*.

A Marmalade fornece ainda uma página com vários jogos expostos de forma a demonstrar as capacidades da *framework* [32].

### 2.5.2 Unreal

O motor de jogo Unreal (mais conhecido como Unreal Engine) [33] e o seu ambiente de desenvolvimento Unreal Development Kit constituem uma novidade no ambiente de desenvolvimento para dispositivos móveis. A tecnologia, atualmente já na versão 4, é bastante reconhecida pelas suas capacidades técnicas, portabilidade entre plataformas e pelo conjunto de ferramentas que a completa. No entanto, e apesar de a versão 3 ter suporte para plataformas móveis, a versão 4 é a primeira totalmente ajustada para estas plataformas [34].

O Unreal Engine é um motor de jogo bastante robusto e a sua tecnologia permite criar integralmente jogos complexos e com muitos detalhes [35]. Com décadas de experiência em jogos digitais, os produtores desta tecnologia conseguem que a mesma suporte uma vasta gama de plataformas e que consiga obter excelentes resultados em performance, detalhe visual e principalmente suporte a qualquer tipo de jogo [36]. Esta *framework* suporta os principais sistemas operativos para computadores pessoais, Linux, MacOS e Windows, consolas de última geração e plataformas móveis de última geração.

O desenvolvimento de um jogo nesta tecnologia abstrai completamente a plataforma alvo. O desenvolvimento é feito em C++, como na framework Marmalade na secção anterior, mas, neste caso, pode também ser feito integralmente num ambiente visual [35]. O Unreal Editor é uma ferramenta contida no *kit* de desenvolvimento e permite implementar a lógica de jogos de forma visual. Esta ferramenta permite ainda testar o jogo diretamente no ambiente de desenvolvimento, alterar variáveis durante a execução e contém ainda um enorme conjunto de funcionalidades e ferramentas extra que permitem acelerar e agilizar o desenvolvimento de jogos [37].

Desta forma, o Unreal permite criar jogos para dispositivos móveis de forma ágil e simples. Não existe a necessidade de uma programação extensa e praticamente todos os sistemas ne-

cessários para jogos estão disponíveis a ser utilizados. No entanto, devido ao forte foco no desenvolvimento de jogos existem limitações no acesso a funcionalidades específicas de cada plataforma. Existem várias extensões disponíveis que amplificam o motor neste sentido e que permitem aceder a funcionalidades particulares [38]. Caso não exista suporte a uma funcionalidade, esta terá de ser implementada pelo programador. O código fonte do Unreal Engine é totalmente disponibilizado a qualquer entidade com a licença de utilização do motor [38], assim é possível modificar e estender esta tecnologia para permitir mais funcionalidades.

O custo de licenciamento do Unreal Engine e de todas as ferramentas auxiliares ao desenvolvimento é de 19€ por mês por programador [39]. Existe ainda uma cláusula que indica que a Epic Games (criadores do Unreal Engine) recebe 5% do lucro bruto de qualquer produto lançado que contenha código do motor. Desta forma, esta tecnologia está disponível com custo razoável tendo em conta as suas capacidades e as ferramentas incluídas, no entanto a divisão de lucro é um ponto negativo.

É possível encontrar uma grande quantidade de jogos para várias plataformas móveis realizados com o Unreal, tal como mostrado na página de exposição da framework [40]. Mesmo sendo esta uma *framework* com forte vertente para jogos de computador e consolas, a sua capacidade técnica e as ferramentas associadas cativam muitos programadores de jogos para dispositivos móveis a criarem os seus jogos utilizando o Unreal.

### 2.5.3 Corona SDK

A Corona SDK é uma *framework* especializada para desenvolvimento de aplicações e jogos em 2D. Esta *framework*, para além de disponibilizar um amplo conjunto de funcionalidades, junta também várias outras *frameworks* para fornecer aos programadores um sistema completo para desenvolvimento de jogos. Desta forma, a *framework* providencia um vasto suporte a funcionalidades únicas para jogos, como simulação de física e sistemas de pontuações. Existe ainda acesso a outros sistemas como redes sociais, mapas, gestão e armazenamento de dados, etc. [41].

Assim, a Corona SDK permite aos programadores criar um jogo completo sem a necessidade de recorrer a outras *frameworks* ou de implementar sistemas base. No entanto, este conjunto de funcionalidades não está disponível de forma simples. O desenvolvimento utilizando esta

*framework* terá de ser feito de forma total em código na linguagem utilizada, Lua [41]. Trata-se de uma linguagem de alto nível, que proporciona um desenvolvimento rápido ao estilo de linguagens de *scripting*, onde a linguagem é compacta e a programação é pouco restrita a regras pré-estabelecidas.

O editor para o desenvolvimento de aplicações é o Sublime Text [42] com várias extensões instaladas para permitir depurar, testar e compilar as aplicações criadas. Este editor apenas permite uma edição em código não contendo qualquer editor visual [43]. Não existe também suporte para testar as aplicações diretamente no editor, sendo necessário compilar e instalar a aplicação num dispositivo para realizar testes. Ainda assim, a Corona permite um desenvolvimento muito mais rápido do que o desenvolvimento nativo. Uma grande quantidade de código é abstraído por várias funções simples [44]. Portanto, o desenvolvimento é totalmente efetuado em código, mas existe uma simplificação que garante um desenvolvimento mais rápido e ágil.

As versões existentes desta *framework* incluem uma versão gratuita, que permite o desenvolvimento e publicação de aplicações. No entanto, esta versão não permite a utilização de compras dentro da aplicação. Esta restrição limita a monetização, ou seja, a criação de lucro, das aplicações criadas, no entanto é uma ótima forma de testar a *framework* ou criar aplicações sem compras. Existem versões pagas, a versão *Basic* tem um custo de $192 anuais e permite distribuir aplicações que tenham lucro até $100.000. Existe ainda a versão *PRO* com acesso prévio a novas funcionalidades da *framework*, integração de análises e diagnóstico dentro da aplicação e ainda mais funcionalidades via extensões. Esta versão permite a criação de aplicações que tenham lucros até $500.000 e tem um custo de $588 anuais. Para além destas, existem duas versões destinadas a empresas com a adição de suporte e funcionalidades. A versão *Enterprise Unlimited* não contém restrições no lucro das aplicações criadas e tem um custo de $2388 anuais. Todas as licenças de cada versão servem apenas para um programador, sendo que os custos associados multiplicam-se pela quantidade de programadores. Esta *framework* conta com um vasto reportório de jogos, grande parte devido à sua facilidade de utilização para desenvolvimento. Uma lista com vários destes jogos está disponível na página *web* da *framework* [45].

### 2.5.4 Unity3D

Lançado em 2005, o Unity foi criado com o propósito de desenvolver jogos para o sistema MacOS [46]. Com o crescimento de novas plataformas móveis, o Unity foi adaptado para funcionar e tirar partido de todas as capacidades que estas plataformas oferecem. Atualmente, na versão 4.3, o Unity não é apenas uma simples *framework* mas sim um motor de jogo completo. Desenhado para criar jogos, principalmente em 3D, o Unity fornece um vasto conjunto de funcionalidades que permite aos programadores a criação de jogos para plataformas móveis de última geração (iOS, Android e variantes e Windows Phone) [47]. Apesar de existir suporte para jogos 2D, apenas na versão 4.3 foram totalmente incluídas funcionalidades para suportar este tipo de jogos sem necessidade de *frameworks* externas [48].

O acesso a funcionalidades nativas das plataformas é garantido pela *framework* apenas em certos casos. Existe suporte nativo para acesso à câmara, mecanismo de vibração e vários sensores. Para acesso a funcionalidades nativas mais específicas existe a necessidade de criar extensões que são facilmente incluídas com o Unity e com as suas ferramentas [26]. Existe ainda a possibilidade de acrescentar código nativo ao jogo que é executado paralelamente ao motor do Unity, sendo possível a comunicação entre o código nativo e o Unity [49]. Esta funcionalidade apenas é suportada com o código nativo incluído no pacote do jogo, sendo que aplicações externas não conseguem comunicar diretamente com o Unity, existindo a necessidade de utilizar técnicas externas ao Unity de forma a permitir essa comunicação, o que, tal como acontece nas outras *frameworks*, é uma desvantagem para os programadores, mas garante aos utilizadores dos sistemas operativos que aplicações não podem aceder a dados de outras aplicações sem permissão para tal.

O desenvolvimento com esta *framework* é efetuado em dois editores. O Unity Editor é o editor onde é possível criar jogos e criar toda a parte visual do jogo. Para programação é utilizado o editor MonoDevelop [50]. Ambos os editores vêm incluídos nos pacotes do Unity.

O Unity Editor, para além da edição visual dos jogos, permite realizar outras operações de forma a simplificar o desenvolvimento de jogos. Este editor suporta a execução direta do jogo, permitindo assim aos programadores testarem o seu jogo durante o desenvolvimento, sem necessidade de o instalar num dispositivo. Ainda durante a execução de um jogo, o editor permite visualizar e alterar o estado de todos os elementos incluídos no jogo, não sendo necessário parar o mesmo para alterar valores. O editor contém ainda outras funcionalidades,

tais como: exploração e customização dos dados utilizados no projeto, edição de definições de importação de materiais (texturas, sons, etc.), várias opções de exportação para as diferentes plataformas, acesso a funcionalidades de depuração, criação de animações e lógica de transições de estados. Assim, este editor engloba um grande conjunto de funcionalidades e suporta ainda mais através de extensões, que são programadas na linguagem C# ou JavaScript e são integradas no editor. Para tal, basta incluir o código fonte da extensão na diretoria do jogo, não sendo necessário realizar compilações e instalações para estender o editor.

A programação de jogos é feita no editor Mono. As linguagens suportadas são: C#, JavaScript e Boo [51]. Existe ainda suporte para que as várias linguagens possam ser utilizadas no mesmo jogo/projeto [52]. Desta forma, o Unity está disponível para um grande conjunto de programadores. Esta é uma excelente forma de promover a criação e partilha de pequenos utilitários para serem utilizados em jogos. Até mesmo dentro de uma equipa de desenvolvimento pode haver programadores que utilizem linguagens diferentes. Como exemplo para a criação da interface pode ser utilizada uma linguagem, para a criação da lógica do jogo pode ser utilizada outra.

Esta *framework* está disponível em 2 versões principais. Existe a versão gratuita e a versão profissional com um custo de $1500 por programador. A versão gratuita tem severas limitações nas funcionalidades disponíveis, nomeadamente a nível do motor de jogo, do editor e ainda limitações a nível de testes. A versão profissional é a versão completa do Unity com todas as funcionalidades disponíveis. São ainda disponibilizados pacotes individuais para plataformas que adicionam várias funcionalidades apenas estão disponíveis nas plataformas específicas. Estes pacotes têm um custo individual de $1500.

O Unity é uma *framework* bastante popular devido à facilidade de entrada para o desenvolvimento e aos tipos de licenciamento disponíveis. Assim, é possível encontrar bastantes jogos criados com esta tecnologia. Uma lista destes jogos está disponível na página de mostra do Unity [53].

### 2.5.5 Cocos2d-x

Esta *framework* é baseada na Cocos2d originalmente desenvolvida para iPhone (Cocos2d - iphone) [54]. O crescimento de outras plataformas levou à criação da versão Cocos2d-x que tem suporte para multiplataforma [55].

A Cocos2d-x, tal como indica o nome, suporta a criação de jogos em 2d, fornecendo um vasto leque de funcionalidades para este tipo de jogos. Desde gestão de cenas, efeitos visuais, partículas, animações baseadas em esqueletos, desenho de fontes e outras funcionalidades visuais, esta *framework* contém tecnologia suficiente para a criação integral de jogos em 2d [56]. Para isso são incluídas outras *frameworks* que fornecem estas e outras funcionalidades como gestão de sons, comunicação em rede e simulação de física que é suportado pela *framework* integrada Chipmunk2D [56]. Todas estas funcionalidades tornam a Cocos2d-x apta a criar jogos com requisitos exequíveis por os sistemas incluídos. Caso uma funcionalidade não seja suportada, é possível acrescentá-la à *framework* de forma simples. Esta *framework* utiliza uma licença *open-source* [56], sendo o seu código disponibilizado gratuitamente e é permitido que este seja alterado. Desta forma, existem várias versões da *framework* alteradas pela comunidade de programadores que suportam outro tipo de aplicações e funcionalidades não suportadas originalmente pela *framework*, como por exemplo a Cocos3d, que tem como principal objetivo acrescentar suporte a jogos 3D [57].

O desenvolvimento com esta *framework* é feito em C++, Lua e JavaScript [56]. A codificação é feita em editores genéricos disponíveis nas várias plataformas. No caso de Windows a programação pode ser realizada no Microsoft Visual Studio e em MacOS a programação pode ser feita no IDE XCode. A compilação para cada plataforma alvo terá de ser feita em IDEs separados devido à inexistência de uma editor único que suporte a exportação para todas as plataformas compatíveis. Ainda assim, é suportado oficialmente um editor visual, Coco Studio [58], que fornece capacidades visuais para criar interfaces, animações e gestão de cenas. Desta forma, uma grande parte visual de um jogo pode ser realizada com auxílio desta ferramenta. Existem ainda editores desenvolvidos por outras empresas que são aconselhados para auxiliar o desenvolvimento de jogos [56], assistindo em várias tarefas, tais como criação de níveis, que pode ser feito no editor Tiled [59], ou conversão e encapsulamento de materiais, no editor TexturePacker [60].

O custo de utilização desta *framework*, tal como referido anteriormente, é nulo. A Cocos2d-x tem uma licença *open-source* e portanto o seu código fonte é disponibilizado gratuitamente. Desta forma, não existe qualquer custo direto para desenvolver com a *framework* e subsequentes ferramentas, tal como o Coco Studio [56][58]. Apesar destas vantagens existem inconvenientes que levam programadores a não escolher esta *framework* para desenvolver jogos, como por exemplo a falta de um editor único com funções de exportação para as vá-

rias plataformas, o que torna o processo de criação dos vários pacotes para distribuição mais demorado e complicado.

Segundo os criadores da Cocos2d-x, a *framework* já fui utilizada para construir milhares de jogos destinados a várias plataformas móveis. Na página *web* da *framework* é disponibilizada uma extensa lista que demonstra as capacidades da *framework* mostrando vários jogos criados nela [61].

### 2.5.6 MonoGame

A *framework* MonoGame foi criada com a intenção de permitir exportar jogos desenvolvidos na framework XNA [62] para múltiplas plataformas. A XNA é uma *framework* criada pela Microsoft para desenvolvimento de jogos para a Xbox 360. Durante o percurso de desenvolvimento desta *framework* foi adicionado suporte para criação de jogos para Windows e Windows Phone. Devido ao sucesso e popularidade da XNA, foram criadas várias *frameworks* que a simulam e que possibilitam a exportação de jogos para mais plataformas. Com a descontinuação do desenvolvimento da XNA, a MonoGame ganhou bastante popularidade devido ao suporte de desenvolvimento com base na XNA e por permitir exportar jogos para as principais plataformas atuais: Windows, MacOS, iOS, Android e Windows Phone [63].

Esta *framework* é apenas a base para criação de jogos. O desenvolvimento é feito na linguagem C# com auxílio da .NET Framework ou da *framework* Mono [64], de acordo com as plataformas alvo. Apesar de ser possível criar jogos apenas com a MonoGame, funcionalidades mais avançadas ou invulgares têm de ser implementadas ou delegadas para uma *framework* especializada. No entanto, a falta dessas funcionalidades mais avançadas não limitou a popularidade desta *framework*, que devido à sua estrutura simples e flexível permite servir de base para jogos dos mais variados tipos e complexidades.

Não existem ferramentas oficiais para desenvolvimento, os editores recomendados para o desenvolvimento são o Microsoft Visual Studio e o MonoDevelop. O desenvolvimento para Windows, Windows Phone e Xbox pode ser realizado no Microsoft Visual Studio, sendo o desenvolvimento para outras plataformas também suportado neste editor com a utilização de extensões [65].

A MonoGame é uma *framework* disponibilizada de acordo com licenças *open-source*, pelo que não existem custos diretos da sua utilização [66], apenas os editores e outras ferramentas externas podem acarretar custos. Esta é uma *framework* criada por uma equipa *indie* e não por uma empresa, existindo assim uma mentalidade pouco virada para o negócio, o que beneficia a partilha de conhecimentos e como o código da *framework* é aberto, leva à existência de muitas modificações criadas pela comunidade, que fornecem suporte e funcionalidades que não existem originalmente na *framework*, como é possível constatar com a quantidade de sub-projetos criados com base no repositório da *framework* no Github [67].

### 2.5.7 Comparação de *frameworks*

Com base na análise das *frameworks* mais utilizadas para a criação de jogos para dispositivos móveis, podemos concluir que o desenvolvimento para estas plataformas está atualmente, em 2014, num período onde existe bastante abundância. Existem muitas *frameworks* disponíveis para criar aplicações e existem muitos programadores que as utilizam. O mercado de jogos para plataformas móveis está bastante forte e existe muita procura. Desta forma, várias empresas estão a criar novas *frameworks*/tecnologias para facilitar e agilizar o processo de desenvolvimento. Até mesmo *frameworks* destinadas a outras plataformas estão agora a ser adaptadas para funcionar com dispositivos móveis e para tirar o melhor partido dessas plataformas.

Cabe aos programadores e empresas escolherem a melhor forma de desenvolverem os seus produtos. Os benefícios em utilizar *frameworks* como auxílio ao desenvolvimento são claros, no entanto a dúvida que se mantém é como escolher a *framework* certa. Tal como previamente referido, existem dois tipos de *frameworks*: especializadas e de sistema, no entanto pode-se considerar ainda um subtipo. Com base nas *frameworks* analisadas podemos observar que certas englobam um ambiente de desenvolvimento e noutras o desenvolvimento é feito totalmente por código num editor de texto.

*Frameworks* como a Unreal, Unity3D e Cocos2D-x incluem ferramentas, como editores visuais, que auxiliam bastante a criação de jogos. Assim, estas *frameworks* são bastante úteis para criar jogos, mesmo que complexos, de forma mais simples, rápida e ágil. No entanto, estas *frameworks* podem não ser ideais para certos projetos, em casos específicos pode ser vantajoso escolher uma *framework* onde a maioria do desenvolvimento é feito em código. Estes casos acontecem quando existem necessidades específicas que são apenas conseguidas

via programação ou quando a equipa de programadores domina uma linguagem/*framework* e a mudança para um ambiente totalmente visual diminuiria a eficiência em vez de a aumentar. Um exemplo típico é a mudança de *frameworks* como o XNA para MonoGame. Apesar de existirem outras alternativas, é melhor manter o desenvolvimento na mesma tecnologia, do que mudar para outras tecnologias/*frameworks* diferentes. O mesmo pode acontecer na criação de sequelas de jogos, onde existe trabalho previamente desenvolvido que pode voltar a ser utilizado no novo jogo. Para estes casos as *frameworks* MonoGame, Corona SDK e Marmalade são escolhas interessantes. Até mesmo *frameworks* como Unreal e Cocos2d-x que disponibilizam o seu código fonte podem ser usadas quase sem utilização de ferramentas visuais.

| ***Framework*** | **Editor** | **Extensível** | **Plataformas suportadas** | **Custo** |
|---|---|---|---|---|
| Marmalade | Externo, Textual | Sim | Android, iOS, WP | - Gratuita limitada<br>- $499/Ano |
| Unreal | Integrado, Visual | Sim | Android, iOS | 19€/Mês +<br>Divisão lucro |
| Corona SDK | Externo, Textual | Sim | Android, iOS, WP | - Gratuita limitada<br>- $192/Ano |
| Unity3D | Integrado, Visual | Sim | Android, iOS, WP | - Gratuita limitada<br>- 1500€/Ano |
| Cocos2D-x | Integrado, Visual | Sim | Android, iOS, WP | Gratuita |
| Monogame | Externo, Textual | Sim | Android, iOS, WP | Gratuita |

Tabela 2.2: Quadro de comparação entre as *frameworks* analisadas

Fica então ao critério dos programadores e empresas escolher a melhor alternativa tendo em conta o projeto a desenvolver, a equipa e as funcionalidades que vão utilizar. De uma forma sucinta, a Tabela 2.2 mostra as principais características das *frameworks* analisadas. Existem *frameworks* para satisfazer os vários tipos de programadores, quer sejam adeptos de codificação ou de programação visual, portanto a adaptação do método de trabalho não é um obstáculo ao desenvolvimento com estas *frameworks*. O preço é um fator bastante importante mas, mais uma vez, as várias *frameworks* analisadas permitem a sua utilização sem obstáculos, sendo que a maioria tem uma licença gratuita, permitindo a sua utilização de forma limitada, mas sem entraves à inicialização. Assim, é possível testar as várias *frameworks* de forma a encontrar a mais adequada para o contexto (tipo de jogo, equipa, requisitos, etc.) do projeto a desenvolver. Um outro ponto bastante interessante é que todas as *frameworks* analisadas permitem a sua extensão pelos programadores, garantido assim que existe acesso

às várias funcionalidades de cada plataforma, caso tal seja necessário.

Assim, não é possível definir uma *framework* como a “melhor”, de uma forma geral. Cada *framework* tem uma forma de trabalho e ferramentas diferentes que se focam para um ou mais tipos de projetos e equipas, permitindo assim a qualquer programador e equipa de desenvolvimento encontrar as *frameworks* mais adequadas para si.

### 2.5.8 *Frameworks mobile* com suporte a multijogador em rede

As várias *frameworks* analisadas na secção anterior fornecem suporte à criação de jogos para dispositivos móveis, contendo funcionalidades adequadas para a criação deste tipo de aplicações. No entanto, uma área pouco explorada por essas *frameworks* é o funcionamento em multijogador em rede neste tipo de jogos. Várias *frameworks* apenas disponibilizam o acesso básico de comunicação em rede suportado pelos sistemas operativos, nomeadamente *sockets* que permitem comunicação genérica entre dispositivos. O Unity e o Unreal são duas *frameworks* que integram o conceito de multijogador em rede e contêm funcionalidades de suporte para o funcionamento deste tipo de modo de jogo. Ainda assim, o funcionamento destas *frameworks* está preparado para funcionar nos melhores casos, ou seja, em redes cabladas e com equipamentos sem limitações energéticas e de processamento. Apesar de suportado, o funcionamento em rede em dispositivos móveis é um caso que não está enquadrado nas funcionalidades atuais dessas *frameworks*.

No próximo capítulo será realizada uma análise do funcionamento em rede de jogos e quais as alterações para o funcionamento destes em dispositivos móveis.

## 2.6 Conclusão

A constante evolução dos dispositivos móveis trouxe este tipo de dispositivos para o foco de desenvolvimento e utilização. A taxa de utilização de dispositivos móveis é atualmente a maior de sempre e, como tal, o desenvolvimento para este tipo de dispositivos é também um foco bastante importante das empresas e programadores do meio. Um grande tipo de aplicações desenvolvidas para estes dispositivos são jogos. Várias empresas começaram a desenvolver jogos com o objetivo de introduzir entretenimento nos dispositivos móveis. Atu-

almente, os jogos são um dos maiores tipos de aplicações utilizadas nestes dispositivos e os programadores procuram formas de agilizar o seu desenvolvimento.

As várias plataformas móveis têm APIs de desenvolvimento que permitem desenvolver aplicações apenas para a plataforma respetiva. Muitos programadores utilizam estas APIs de forma a desenvolver nativamente para cada plataforma, podendo tirar partido de todas as funcionalidades do dispositivo e do sistema operativo. No entanto, a introdução de várias plataformas no mercado criou a necessidade de desenvolver aplicações compatíveis com vários sistemas operativos.

A utilização de *frameworks* permitiu agilizar o desenvolvimento de aplicações, e principalmente jogos, devido às funcionalidades que estas incorporam. Várias *frameworks* introduzem funcionalidades que permitem auxiliar e reduzir o tempo de desenvolvimento de jogos, funcionalidades estas que passam por componentes utilizados na execução do jogo ou até mesmo ferramentas que são utilizadas pelos programadores durante o desenvolvimento de jogos, incluindo a exportação de jogos para várias plataformas.

No entanto, o desenvolvimento de jogo multijogador para dispositivos móveis é ainda um ponto pouco aprofundado nesta área de desenvolvimento. As *frameworks* existentes têm, na maioria dos casos, suporte para jogos multijogador, mas as implementações realizadas não estão adequadas para jogos em dispositivos móveis.

# Capítulo 3

# Tecnologias multijogador em rede

## 3.1 Introdução

Desde o início da criação de jogos digitais a área de multijogador tem sido um aspeto bastante focado. Até mesmo o Tennis For Two, que é considerado o primeiro “videojogo”, funcionando com base em equipamento eletrónico, tem a sua jogabilidade definida pela conceito de multijogador. O Tennis For Two foi criado em 1958 e simula uma partida de ténis entre dois jogadores [68], e apesar deste ser um jogo simples e de apenas ter sido disponibilizado num único sítio, o Brookhaven National Laboratory nos Estados Unidos da América, a novidade e o simples facto de o jogo funcionar em modo multijogador levou ao seu sucesso e reconhecimento posterior. A implementação de multijogador continuou, ao longo dos anos, a ser um foco importante no desenvolvimento de jogos digitais. No entanto, a implementação de multijogador era feita localmente no mesmo sistema. Jogadas por turnos ou divisão do ecrã entre os vários jogadores eram formas de implementação do mecanismo de multijogador. Com o avanço das ligações de redes e subsequente Internet a implementação de funcionalidades multijogador entre dispositivos remotos tornou-se um foco para os programadores de jogos.

Os primeiros jogos multijogador em rede utilizavam sistemas e protocolos primitivos para comunicação. O jogo Spectre, lançado em 1990, utiliza o protocolo AppleTalk [69] para comunicação em rede. Outros jogos, como o Midi Maze (1989) [70], utilizam outras tecnologias para permitir a comunicação entre máquinas. No caso do Midi Maze é utilizado o sistema de MIDI [71] disponível nas máquinas Atari ST para comunicar em rede. O jogo permite até 16 jogadores em simultâneo sendo que as várias máquinas tinham de estar ligadas

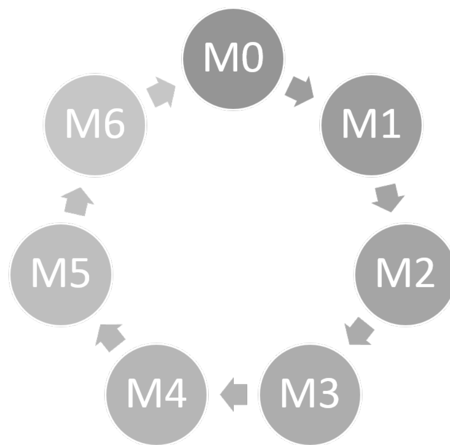


Figura 3.1: Máquinas ligadas numa configuração em anel.

entre si num formato em anel.

Na Figura 3.1 está representado a configuração em anel utilizado no jogo Midi Maze. A ligação das várias máquinas forma um ciclo de comunicação. Este ciclo de comunicação e a tecnologia MIDI (*Musical Instrument Digital Interface*) permitem até 16 jogadores/-máquinas ligadas na rede dado o limite de 16 canais disponíveis nas ligações MIDI. Esta configuração permite a cada máquina enviar dados para outra máquina na rede.

Com o avanço das tecnologias de rede, outras técnicas foram utilizadas para comunicação nos jogos digitais. O jogo Doom (1993) [72] utiliza uma configuração de rede ponto-a-ponto. Este tipo de ligação, tal como representado na Figura 3.2, requer uma ligação entre todos os jogadores ligados no jogo. Todos os jogadores conseguem comunicar diretamente entre si. Desta forma, não existe necessidade de um servidor central para tratar de encaminhamentos. No entanto, a utilização desta configuração relevou-se problemática tendo em conta a implementação realizada. Sendo que existe a necessidade de transmitir N * (N - 1) pacotes por cada ciclo de jogo a possibilidade de congestionamentos de rede aumenta e alguns pacotes podem não chegar ao destino, principalmente quando existem muitos clientes/jogadores ligados. O Doom utiliza os protocolos IPX (*Internetwork Packet Exchange*) e UDP/IP (*User Datagram Protocol/Internet Protocol*) [73] que fornecem comunicação *connection-less* entre dispositivos, assim estes protocolos têm pouco *overhead*, mas, no entanto, não garantem a entrega dos pacotes de forma consistente visto que não fornecem mecanismos de retransmissão e deteção de erros. [74].

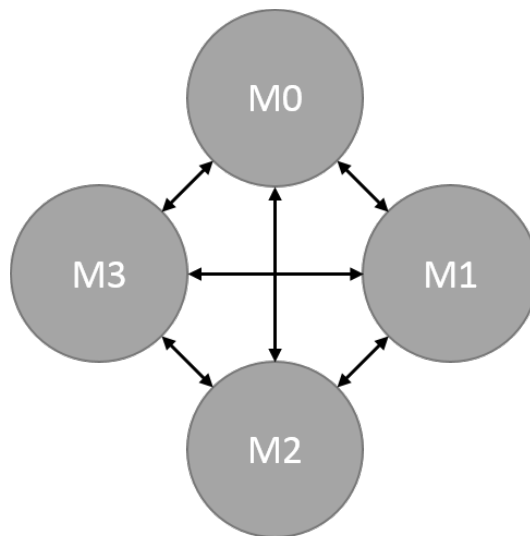


Figura 3.2: Ligação de rede ponto-a-ponto.

Na camada aplicacional, o Doom trata os dados a enviar/receber de uma forma simplista. Cada comando do jogador (movimento, ação, etc.) é colocado numa lista a processar. A cada ciclo do jogo a lista é processada, os comandos são enviados para todos os outros jogadores e o jogo fica à espera para receber os comandos destes. Apenas depois de receber os comandos dos outros jogadores (na ordem correta), o jogo volta a enviar novos comandos do jogador para a rede.

A forma de implementação do multijogador em rede do Doom é bastante simplista, no entanto esta regista alguns problemas. É garantido que todos os jogadores estão completamente sincronizados, no entanto a latência de cada jogador vai ser igual à maior latência existente na rede. Sendo que o jogo fica completamente parado enquanto não receber os pacotes de todos os jogadores, a latência torna-se assim num problema muito relevante. Esta configuração implica também que todos os clientes tenham de calcular o estado de cada jogador. Os dados enviados na rede são os comandos introduzidos por cada jogador, então é necessário os clientes calcularem o resultado destes comandos.

No entanto, mesmo com todos estes possíveis problemas, que apenas se apresentam nos piores casos, o jogo teve grande sucesso. Quando utilizado em ambientes favoráveis a sua execução era correta, e a (excessiva) utilização da componente multijogador até levou à criação de bloqueadores, para evitar que redes públicas ficassem congestionadas [75].

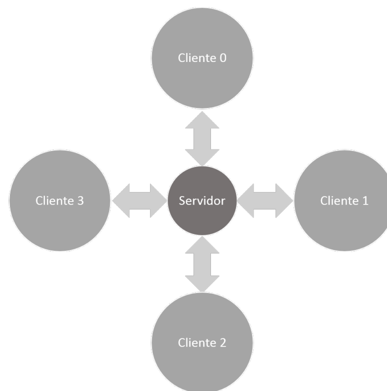


Figura 3.3: Configuração de rede tipo cliente-servidor.

Anos mais tarde, o jogo Quake (1996) [76] apareceu como o primeiro jogo multijogador verdadeiramente construído para ser utilizado via Internet. Com uma arquitetura de rede do tipo cliente-servidor são eliminados vários problemas encontrados noutras arquiteturas.

Tal como representado na Figura 3.3, a configuração de tipo cliente-servidor, elimina a necessidade de um grande número de ligações entre os vários clientes. Todas as comunicações passam pelo servidor, como tal o servidor realiza encaminhamento entre os vários clientes o que reflete apenas uma ligação necessária por cliente. Em contrapartida é necessário um servidor, o que pode implicar um dispositivo/máquina extra.
Tendo como base esta configuração de rede, a implementação inicial do Quake tratava os clientes como simples terminais e toda a lógica do jogo era calculada no servidor.

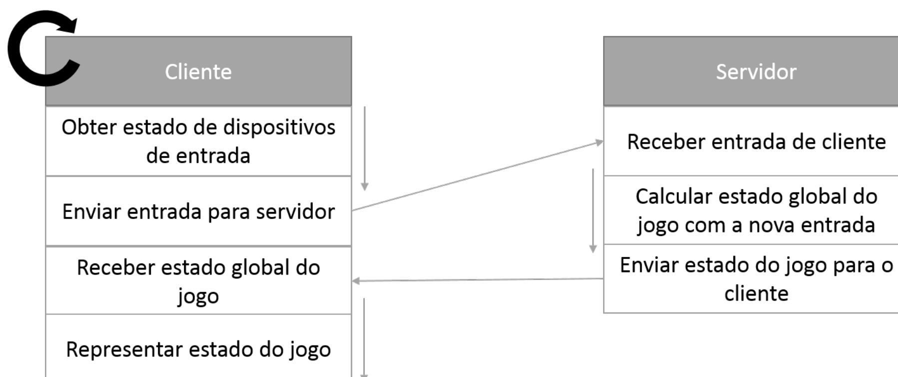


Figura 3.4: Representação do funcionamento em rede da versão inicial do Quake.

A representação simplificada do funcionamento em rede do Quake, Figura 3.4, mostra a particularidade da delegação de todo o processamento de lógica para o servidor. Com base nesta arquitetura é possível idealizar vários conceitos interessantes. Um exemplo é poder ter o mesmo cliente para jogos completamente diferentes. Sendo que o cliente apenas lê entradas e representa o estado do jogo não existe vinculações a lógicas definidas para um certo jogo. A única dificuldade para este conceito funcionar seria a necessidade do cliente obter os materiais (texturas, sons, etc.) necessários para representar o estado do jogo.

No entanto, apesar das possibilidades desta arquitetura e da simplicidade do cliente como terminal, a mesma teve de ser alterada para precaver os problemas de comunicação em rede. A implementação funcionava bem em rede locais, no entanto os problemas de inter-redes expõem as falhas da arquitetura. Se os pacotes de dados não fossem trocados entre o cliente e servidor em tempo útil, o cliente não iria saber o estado do jogo correto e o servidor e restantes jogadores não saberiam a posição correta do jogador afetado pelos problemas de rede. Os clientes e servidor não estariam sincronizados.

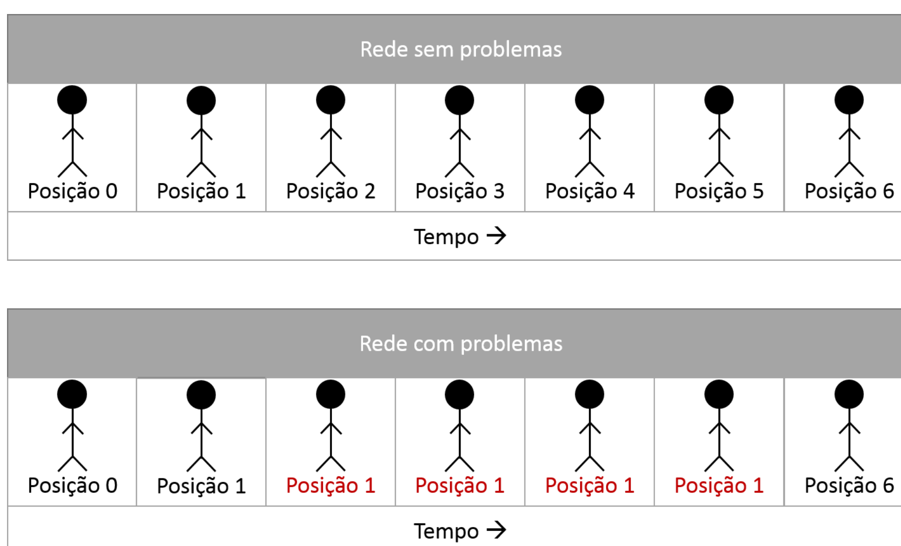


Figura 3.5: Representação de estado de um jogador com problemas de rede, visto por outro jogador.

Quando os pacotes chegassem ao destino poderia ainda ocorrer o problema representado na Figura 3.5. O tempo entre a receção de dois pacotes ou a perca de dados na rede leva a erros lógicos e visuais no jogo. A movimentação de um jogador pode não ser suave, tal como

esperado por outros jogadores.

No entanto, tal como indicado por John Carmack (programador principal do Quake), o principal objetivo não é criar sistemas ideais, mas sim funcionais: *"I am sad to see the elegant client-as-terminal setup go away, but I am practical above idealistic"* [77]. Por vezes é necessário abdicar de certa elegância estrutural de forma a criar sistemas que forneçam funcionalidades de forma eficiente e prontas para serem utilizadas.

Desta forma, a arquitetura foi alterada para minimizar os problemas de rede. A aplicação cliente passou agora a conhecer o funcionamento do jogo e simula localmente a lógica de jogo. O cliente baseia-se nos últimos dados recebidos e nas regras/lógica do jogo e simula acontecimentos futuros. Um exemplo é a simulação de continuação de movimento. Se um objeto se está a movimentar numa certa direção então, caso não exista obstruções, o objeto deve continuar a movimentar-se. Esta técnica é denominada de Dead Reckoning [78]. É claro que este sistema não é perfeito, pois a ação simulada pode não ser a que o outro jogador efetuou. No entanto, a simulação serve apenas para dissimular os pequenos problemas de rede enquanto os dados do servidor não chegarem para confirmar a movimentação real.

Esta mudança trouxe ao Quake a capacidade de ser jogável através da Internet. Ao implementar a simulação local nos clientes, o jogo passou a ser jogável em ligações com 200ms [77] de latência sem perda de qualidade e sem problemas aparentes nos objetos onde a simulação é utilizada.

Atualmente todos os jogos com comunicação em rede utilizam sistemas de simulação de estado nos clientes. Pode mesmo ser dito que os jogos são totalmente simulados nos clientes e que os pacotes de dados recebidos servem apenas para confirmar ou corrigir as simulações efetuadas.
Apenas em jogos por turnos ou jogos com poucos estados não é normalmente utilizado simulação. Nestes jogos os problemas da rede são mascarados por animações nas interfaces ou em situações de atrasos nas comunicações é indicado que o outro jogador ainda esta a realizar a sua jogada, isto em casos de jogos por turnos.

## 3.2 Modos de funcionamento

Tal como analisado na secção anterior, existem várias formas para estabelecer uma rede para jogos multijogador. As principais arquiteturas para tal são: Anel, Ponto-a-Ponto e Cliente-Servidor. Estas arquiteturas fornecem características específicas de acordo com as suas aplicações. Desta forma, será realizada uma análise a cada tipo de arquitetura de forma a enunciar as vantagens e problemas destas.

### 3.2.1 Arquitetura em Anel

A topologia em anel implica uma ligação entre vários dispositivos para que cada dispositivo esteja diretamente ligado ao anterior e ao próximo fazendo um ciclo. Uma representação visual pode ser encontrada na Figura 3.1. Todos os dispositivos conseguem comunicar entre si, sendo que o protocolo utilizado tem de suportar encaminhamento ou replicação dos pacotes/sinais. Assim é possível utilizar este modo de duas formas: com um servidor ou apenas com clientes. Isto é: poderá haver um dispositivo que realiza as operações de um servidor, comunicação com redes exteriores, processamento da lógica do jogo, etc. ou então cada cliente envia para os restantes clientes o seu estado. Desta forma cada cliente realiza o seu processamento da lógica do jogo.

Este tipo de arquitetura apresenta vários problemas. No caso de um dispositivo se desligar da rede existe a necessidade de reorganização da rede para que a comunicação possa continuar. Este processo pode demorar algum tempo o que pode provocar problemas para todos os clientes na rede, sendo que o ciclo de ligação foi abruptamente cortado. Outro problema típico é o excesso de tráfego que poderá passar pelos dispositivos. Em certos dispositivos a quantidade de pacotes de rede a receber e enviar poderá afetar o desempenho e estado do dispositivo em si, exemplo: um telemóvel com bateria poderá perder bastante carga devido a ter a placa de rede sempre ativa. Este problema é ainda ampliado pelo fato de vários pacotes que passam por um dispositivo não terem esse dispositivo como destino. Será necessário receber o pacote, processa-lo para determinar o destino e envia-lo novamente para a rede.

### 3.2.2 Arquitetura Ponto-a-ponto

A configuração de rede ponto-a-ponto estabelece que quaisquer dois dispositivos ligados na rede têm uma conexão entre si. Poderão existir certos casos excecionais em que nem todos

os dispositivos precisam de estar ligados entre si, no entanto os dispositivos principais estão ligados entre si. Desta forma todos os dispositivos partilham informação de forma direta sem a necessidade de encaminhamentos.

Esta topologia é bastante utilizada nos jogos digitais quando o envolvente é ideal. Isto é: em casos onde existe um baixo número de clientes/jogadores e quando não existe um servidor. Neste ambiente esta configuração de rede é empregue.

A principal vantagem desta topologia é a ausência de encaminhamentos. A informação segue exatamente para o destino. No entanto esta vantagem acarreta uma desvantagem bastante relevantes. O número de canais de comunicação é bastante elevado. Mesmo com apenas 4 clientes existem 12 canais de comunicação entre eles. Em dispositivos com limitações de processamento e energéticas este poderá ser um fator impeditivo.

### 3.2.3 Arquitetura Cliente-servidor

A topologia Cliente-servidor é a mais utilizada em jogos digitais, definindo um dispositivo, servidor, que realiza operações de tomada de decisão e de processamento central. Este servidor está ligado a todos os clientes. Assim cada cliente apenas tem uma ligação ativa e o servidor tem N ligações (sendo N o número de clientes).

Nesta configuração os clientes não enviam dados entre si. Estes enviam informações para o servidor e, depois de processada, essa informação é encaminhada para os restantes clientes. Desta forma, os clientes recebem apenas as informações que precisam e no formato adequado. Sendo que existe um servidor central, este pode realizar operações mais pesadas computacionalmente de forma a eliminar estas operações dos clientes e por consequente baixar a carga de tráfego da rede, eliminando a troca fina de dados.

É ainda possível utilizar o servidor para guardar e manter estados e sessões relativas aos clientes. Desta forma se um cliente perder a sua ligação então existe a possibilidade de restaurar a sessão anterior.

No caso de não existir um dispositivo dedicado para realizar as operações de servidor, a aplicação servidor pode estar no mesmo dispositivo de um cliente. Neste caso as funcionalidades disponíveis podem ser limitadas para evitar a carga excessiva do dispositivo.

### 3.2.4 Comparação dos modos de funcionamento

Os três modos de funcionamento analisados fornecem estruturas que suportam jogos multi-jogador em rede, que foram desenhados para redes de computadores imóveis. Visto que os dispositivos móveis têm características que invalidam algumas vantagens de certas topologias, de forma a encontrar estas situações apresenta-se de seguida uma tabela de comparação das várias topologias analisadas.

| **Topologia** | **Número de ligações** | **Encaminhamento** | **Servidor dedicado** | **Desconexão de cliente** |
|---|---|---|---|---|
| Anel | N | Local - Distribuído | Não | Reconfiguração da rede |
| Ponto-a-ponto | N * (N - 1) | Local - Distribuído | Não | Fecho de 1 canal em cada cliente |
| Cliente-servidor | N | Servidor - Centralizado | Sim | Fecho de 1 canal no servidor |

Tabela 3.1: Comparação entre as topologias de funcionamento em rede analisadas

Na Tabela 3.1 estão resumidas as características principais, para o contexto dos dispositivos móveis, das várias topologias analisadas. A configuração em anel tem um número de ligações baixo mas o encaminhamento é feito localmente pelos clientes o que aumenta o processamento necessário. O ponto-a-ponto tem um número de ligações bastante elevado e requer, tal como a configuração em Anel, encaminhamento local, não sendo por isso um bom candidato para jogos multijogador em rede em dispositivos móveis. A topologia cliente-servidor tem um número de ligações baixo, relativo ao número de clientes ligados, e o encaminhamento é feito de forma centralizada por um servidor.

Assim, as configurações anel e cliente-servidor estão aptas para suportar jogos multijogador em rede nos dispositivos móveis de última geração. A configuração cliente-servidor tem a desvantagem de requerer um sistema extra para realizar processamento e encaminhamento de dados entre os vários jogadores, no entanto quando um dos problemas comuns das redes móveis se junta à equação, a perca de ligação, esta configuração tem uma vantagem significativa quando comparada com a configuração em Anel. Na configuração cliente-servidor caso um jogador se desligue da rede basta o servidor fechar o canal de comunicação com este e, caso suportado, guardar a sessão do jogador para permitir continuação posterior, enquanto na configuração em anel é necessário reconfigurar o esquema da rede, o que é mais doloroso para os dispositivos móveis com baixas capacidades de processamento e bateria. Desta

forma a configuração cliente-servidor é a ideal para suportar este tipo de jogos, podendo as tarefas de servidor ser delegadas para o dispositivo móvel com as melhores características (capacidade de processamento, nível de bateria, etc.) no momento.

## 3.3 Multijogador em rede em dispositivos móveis

O alargamento do mercado de jogo para dispositivos móveis levou à procura e criação de sistemas para multijogador neste tipo de dispositivos. Durante longos anos a tecnologia Bluetooth [79] serviu como base para as redes entre dispositivos, fornecendo uma baixa fiabilidade com latências elevadas e pouca largura de banda, que apesar de suportar jogos em rede, não era a tecnologia ideal para tal. No entanto, em 2003, já existiam esforços por parte da Nokia para a criação e aperfeiçoamento deste tipo de jogos para dispositivos móveis, explorando formas de ultrapassar ou reduzir as implicações das limitações das tecnologias presentes [80]. Os jogos criados com recurso a estas tecnologias eram simples e com mecanismos onde a latência não era um fator importante, e.g. jogos de cartas por turnos. Com o avanço das tecnologias, principalmente de rede, o paradigma de criação de jogos para dispositivos móveis alterou-se. As redes móveis 3G, LTE [81] e WIFI começaram a estar suficientemente avançadas e distribuídas para servirem como plataforma base de rede, suportando a criação de jogos multijogador em rede com mecanismos em tempo real. No entanto, e apesar da evolução dos dispositivos móveis, ainda existem vários problemas que limitam a criação de jogos multijogador em rede, estes são apresentados de seguida.

### 3.3.1 Problemas no desenvolvimento de jogos multijogador em rede para dispositivos móveis

Os problemas dos dispositivos móveis no contexto de jogos com multijogador em rede podem ser resumidos aos seguintes pontos: fiabilidade da rede, capacidade de processamento e limitação de energia. Estes pontos tornam o desenvolvimento de jogos multijogador em rede para dispositivos móveis um desafio.

Mesmo redes de última geração (LTE e 802.11N [82]) têm problemas de fiabilidade. A imprevisibilidade do contexto de utilização destas podem levar a problemas como atrasos na entrega de dados, perda de pacotes de dados ou mesmo perda completa de ligação. Devido a estas imprevisibilidades, a implementação da comunicação em rede tem estar preparada para

minimizar estes problemas.

A capacidade de processamento é outra limitação dos dispositivos móveis. De forma a minimizar o consumo energético e o consequente aumento do calor produzido, os processadores incluídos nos dispositivos móveis têm uma capacidade de processamento menor do que outros, em sistemas fixos. Desta forma, o processamento é naturalmente limitado e as aplicações têm a responsabilidade de serem eficientes para evitar desperdícios.

No entanto, a maior limitação é a fonte de energia. Visto que os dispositivos móveis são alimentados por baterias pequenas com uma capacidade de armazenamento de energia notavelmente limitada. De forma a minimizar o gasto de energia, todos os sistemas destes dispositivos são ajustados para serem eficientes e para estarem desligados o maior tempo possível. Desta forma, o processador tem ciclos de pausa maiores e as placas de rede tendem a ter também uma latência superior. Qualquer utilização desnecessária de energia deve ser evitada, prolongando assim a duração de cada carga energética que a bateria armazena.

Existem ainda outros problemas, como a interrupção do jogo devido ao ambiente. Em geral os sistemas operativos para dispositivos móveis apenas permitem uma aplicação em execução em primeiro plano, no entanto, normalmente suportam multitarefa, permitindo assim que os utilizadores mudem entre aplicações de uma forma simples. Assim, se um jogo multijogador em rede sair de primeiro plano de execução é necessário detetar esse evento e proceder de forma a não prejudicar os vários jogadores presentes no jogo, inclusive o próprio jogador. Note-se que a saída de primeiro plano pode não ter sido despoletada pelo jogador, por exemplo, se o jogador receber uma chamada no seu telemóvel o jogo fica em segundo plano e o jogador perde o controlo sobre este.

Verificando-se estes problemas, vários grupos de investigação e desenvolvimento procuram encontrar soluções de forma a reduzir o impacto destes problemas, sendo de seguida apresentado um breve resumo de principais soluções encontradas.

### 3.3.2 Trabalhos e frameworks multijogador para dispositivos móveis

Apesar dos problemas no desenvolvimento de jogos multijogador para dispositivos móveis existe já algum trabalho realizado na criação deste tipo de aplicações. Várias soluções foram empregues para minimizar o problema de latência, de consumo energético e de interrupções

de ligação.

Os problemas da elevada latência e das interrupções nas redes entre dispositivos móveis são dos mais severos. De forma a reduzir os efeitos destes problemas Christian Hiedels et al. [83] decidiram criar o UbiSettlers onde aplicam várias técnicas para resolver os problemas de fiabilidade das redes e de poupança de energia. Segundo a sua análise os problemas de rede nos dispositivos móveis, tais como baixa fiabilidade das ligações, latência e interrupções esporádicas podem ser resolvidos com base em mecânicas implementadas no jogo. No seu exemplo é retratado o caso em que um jogador compra um objeto dentro do jogo, no entanto por falha da rede o pacote com o resultado da compra não chega ao destino dentro de um tempo limite sendo indicado que "piratas" capturaram os bens comprados. Desta forma a fiabilidade é um ator no jogo. É indicado que a rede está incorporada como um gerador de números aleatórios que decide quando várias ações acontecem.

No problema de quebras de ligação momentâneas, Yung-Wei Kao et al. [84] sugere a implementação de sistemas locais para substituir a interação com outros jogadores quando a ligação é fechada. O trabalho realizado incluí a construção de uma *framework* em que a arquitetura desenvolvida suporta a possibilidade de entrada em jogo de objetos controlados localmente caso a ligação à rede seja abruptamente terminada. Desta forma, o jogador não vai poder realizar todas as funcionalidades que tinha com a ligação ativa, mas consegue manter um nível de jogabilidade aceitável para continuar a jogar.

Técnicas de simulação são também aplicadas para minimizar problemas de latência no contexto dos dispositivos móveis. Ahmad Hoirul Basori et al. [85] criaram um jogo onde aplicam a técnica de Dead Reckoning para simulação de posicionamento. Com a utilizam de simulação local conseguem mascarar o problema de latências altas neste tipo de redes. A simulação é ainda utilizada para poupar energia. O envio dos pacotes é temporizado para que a simulação trate de calcular o estado dos jogadores. Quando a simulação se torna inviável então os pacotes são transmitidos para garantir que a representação dos jogadores é fiável.

Uma das principais causas da elevada latência nas redes entre dispositivos móveis é o congestionamento de tráfego na rede, tal como retrata Alf Inge Wang et al. [86] na sua análise do problema. Este congestionamento é prejudicial para os jogos em dispositivos móveis não apenas por aumentar a latência de ligação entre dispositivos mas também por aumentar a carga de processamento nestes dispositivos. O trabalho realizado por Alf Inge Wang et al.

demonstra com detalhe vários problemas nos jogos em rede em dispositivos móveis e como os resolver. É indicado problemas de latência elevada, principalmente com a utilização de tecnologias antigas [86]. Como consequência o jogo é representado de forma diferente em vários dispositivos no mesmo índice temporal. Para resolver estes problemas, Wang et al., aplicam uma divisão de processamento entre o servidor e cada cliente. Em áreas sensíveis, tais como a colisão com outros materiais, o processamento é feito localmente para evitar latência. Em casos não sensíveis o processamento é feito no servidor onde normalmente não existe restrições energéticas e de processamento. É ainda indicado que de forma a evitar o congestionamento e para baixar o *overhead* é utilizado o protocolo UDP para enviar dados.

Ainda com o objetivo de minimizar os problemas de latência, interrupções da rede e principalmente o consumo energético dos jogos em rede para dispositivos móveis, Bhojan Anand et al. criaram dois sistemas. O Arivu [87] é uma camada que observa o seu contexto para evitar gastar energia. Este sistema analisa a área de interesse do jogador e apenas transmite os dados necessários. Assim o jogador não recebe dados que não vai utilizar. O sistema analisa ainda quando será a próxima interação com outros jogadores para evitar enviar dados em momento em que tal não seja necessário. Caso o jogador não esteja perto dos outros jogadores então a sua posição não vai interessar a estes. Portanto a transmissão destes dados não é feita até ao momento em que uma interação entre jogadores possa ocorrer. Nesse momento os dados são enviados com uma frequência consistente até a que os jogadores se separem. O tempo onde não vai existir transmissões é ainda utilizado para desligar os mecanismos de rede, inclusive as placas físicas, de forma a evitar gastar energia.

Um outro sistema criado por Anad et al., denominado de PGTP, consiste na especificação e implementação de um protocolo com objetivo baixar o *overhead* dos protocolos atuais de rede [88]. É indicado que dados de controlo do TCP ocupam cerca de 46% do tráfego nos jogos em rede. O protocolo criado tem um cabeçalho pequeno e exibe características que servem para poupar energia. O cabeçalho apenas contem os dados necessários para a transmissão e controlo de qualidade básico. Para reduzir o consumo energético os dados enviados são agrupados antes do envio para baixar o número de envios. É ainda analisado o estado do jogo para alterar a frequência de envio de pacotes. Se o jogo estiver numa situação crítica então o envio é instantâneo, caso o jogo se encontre numa situação vulgar, por exemplo num menu, então os dados são enviados de forma retardada.

É indicado que a utilização destes sistemas pode levar a poupanças energéticas na ordem os

35% a 40%.

Outras soluções para evitar o gasto de energia são aplicadas nas redes de dispositivos móveis. O trabalho de Hiedels et al. [83] propõem ainda o encaminhamento de pacotes para a Internet realizado apenas por uma ligação num conjunto de N jogadores. Sendo que o seu trabalho retrata um jogo que funciona na Internet é proposto que todos os jogadores que estejam ligados localmente partilhem apenas uma ligação para enviar os dados para os servidores na Internet. A escolha do jogador para enviar os dados para a Internet é baseada no estados dos dispositivos. O jogador com o dispositivo com melhor carga de bateria, sinal e tipo de rede vai servir como ponte para os outros jogadores se ligarem à Internet. Desta forma o dispositivo de outros jogadores vai ser poupado.

Existem assim várias soluções para resolver os problemas afetos à rede em jogos multijogador em dispositivos móveis e que conseguem minimizar os efeitos desses problemas. No entanto as várias soluções propostas ainda refletem problemas e não são aplicadas de forma genérica, tal como analisado na próxima secção.

### 3.3.3 Análise de soluções para os problemas de redes em dispositivos móveis

Existem várias soluções aplicadas em trabalhos realizados. Os principais problemas onde são encontradas mais soluções são: a falta de fiabilidade da rede e o limite de bateria. Tendo em conta estes problemas de rede são aplicadas várias técnicas para limitar as consequências. A simulação local de dados é uma delas. No entanto, esta não é largamente aplicada devido à arquitetura dos jogos onde é aplicada. Não existem sistemas genéricos que incorporem a simulação na criação deste tipo de jogos. As *frameworks* existentes suportam a comunicação em rede nos jogos em dispositivos móveis mas deixam a resolução dos problemas da rede para os programadores finais, que em certos casos não implementam as soluções necessárias para reduzir ou até mesmo eliminar esse problemas.

As técnicas mais utilizadas para precaver os problemas de rede são do tipo *graceful degradation*. Isto é: vários jogos limitam as funcionalidades quando não existe ligação ou quando a ligação é lenta, assim os jogadores conseguem usufruir de funcionalidades limitadas em vez de não poderem utilizar o jogo por completo. Outras técnicas passam por contextualizar

no jogo as falhas de rede. Assim é possível mascarar estas falhas como acontecimentos do jogo.

Na poupança de energia as soluções cingem-se em técnicas contextuais e arquiteturais. Existe uma análise do estado do jogo e por consequente dos jogadores para determinar a melhor altura para enviar dados e como os enviar. Sendo que o tempo onde não existem envios pode ser utilizado para desligar os sistemas de ligações. Os protocolos utilizados são ainda adequados para estes dispositivos, sendo evitado utilizar protocolos com muitos dados de controlo. Assim, existe pouco tráfego com dados extra-jogo o que diminui a necessidade de processamento e congestionamentos na rede. Ainda de forma a baixar o processamento local e consequente consumo de energia o processamento de dados é delegado para o servidor. Caso exista a necessidade de realizar operações pesadas computacionalmente num jogo, o processamento é feito no servidor e os dados processados são enviados para os clientes. Assim, é possível libertar a carga do processador local como por subsequente baixar o consumo energético.

Desta forma, em resumo, estas são as principais soluções aplicadas para resolver e minimizar os problemas de comunicações em rede de jogos multijogador em dispositivos móveis:

- **Soluções para problemas de rede e contextuais**
  - *Dead Reckoning*
  - *Graceful degradation*
  - Utilização de protocolos apropriados (simples e com baixo *overhead*)
  - Enviar apenas os dados necessários
- **Soluções para Limitação de processamento e Restrições energéticas**
  - Dividir processamento entre cliente e servidor
  - Processar apenas dados relevantes
  - Enviar dados para a rede em conjuntos
  - Definir diferentes intervalos de processamento e troca de dados com base no estado do jogo

Tal como é possível visualizar na lista apresentada acima, existem atualmente várias soluções para problemas de redes em jogos multijogador em dispositivos móveis. No entanto,

algumas soluções são limitadas no seu efeito e existem vários problemas sem solução. O restauro de sessão quando um jogador volta a ter ligação de rede no seu dispositivo, depois de uma falha, é um exemplo de um problema que ainda não tem solução. Desta forma é apresentado no capítulo seguinte uma proposta de especificação de uma *framework* com o intuito de criação de jogos em rede para dispositivos móveis.

## 3.4 Conclusão

O desenvolvimento de jogos em multijogador em rede é uma área com uma vasta evolução. Desde o início dos jogos digitais se percebeu as vantagens e as possibilidades dos jogos em multijogador. O natural avanço da tecnologia permitiu não apenas a criação de jogos multijogador, mas também jogos multijogador em redes locais e redes interligadas remotamente. No entanto, este paradigma forçou a mudança da arquitetura interna que era utilizada no jogos de forma a permitir o multijogador e para responder às necessidades especiais que este modo traz. Os problemas inerentes às redes entre dispositivos tornaram-se um inconveniente ao correto funcionamento dos jogos multijogador em rede, assim os programadores foram forçados a criar formas de prevenir e de lidar com estes problemas.

Com o avanço dos dispositivos móveis surge também a oportunidade de inserir o conceito de jogos multijogador em rede nestes dispositivos. As ferramentas de criação de jogos para este tipo de dispositivos suportam as bases necessárias para a criação de jogos multijogador mas carecem suporte de várias necessidades que surgem na utilizam de mecanismos de rede nestes dispositivos. As limitações dos dispositivos, nomeadamente o processamento, a bateria e a fragilidade nas ligações de redes, tornam o desenvolvimento de jogos multijogador em rede um desafio devido aos problemas relacionados. Vários projetos de investigação e desenvolvimento chegaram a conclusões satisfatórias de métodos para a resolução dos problemas de rede nos jogos multijogador em dispositivos móveis. Várias soluções aplicadas em computadores foram adequadas para dispositivos móveis e novas soluções para os problemas únicos destes dispositivos foram criadas.

Ainda assim, as soluções criadas não cobrem todo o espectro de problemas e limitações presentes nas tecnologias utilizadas nos dispositivos móveis. Existe a necessidade de resolver vários problemas ainda existentes que limitam a criação de jogos multijogador em platafor-

mas móveis. Assim, no próximo capítulo é especificada uma *framework* com objetivo de suportar a criação de jogos em rede para dispositivos móveis.

# Capítulo 4

# LiNGS – *Framework* proposta

## 4.1 Introdução

A criação de aplicações multijogador para plataformas móveis é restrita pelos problemas e desafios apresentados no capítulo anterior. Esses problemas estão normalmente relacionados com características dos dispositivos onde se encontram, sendo as baterias destes dispositivos um bom exemplo. Como consequência do pequeno tamanho dos dispositivos móveis, o tamanho das baterias é também reduzido e, como tal, as suas capacidades energéticas são também reduzidas. Ainda no capítulo anterior foram apresentados vários problemas e respetivas soluções atualmente deferidas no desenvolvimento de jogos multijogador em rede para dispositivos móveis. No entanto, existem problemas sem solução e existem também problemas com soluções desadequadas, tendo em consideração o contexto atual de dispositivos móveis. Todos os problemas detetados levam os criadores de jogos a preferir apenas a criação de jogos jogados localmente com apenas um jogador. Assim, existe um fosso na criação de jogos com capacidades de multijogador para dispositivos móveis.

Neste capítulo é proposto um sistema com o objetivo de lidar com os problemas apresentados no capítulo anterior, referentes ao desenvolvimento de jogos multijogador para dispositivos móveis. O sistema tem como principais objetivos gerir a comunicação em rede entre os vários intermediários num jogo, tendo a função de manter o sincronismo dos elementos de jogo nos vários jogadores, servindo de ponte de comunicação e abstraindo os mecanismos necessários para a comunicação em rede. De forma a evitar e resolver os problemas anteriormente enunciados, o sistema terá ainda vários mecanismos que analisam o contexto do jogo, dispositivo e da rede de forma a adequar o seu funcionamento. Este sistema será ainda

incorporado numa *framework* base, utilizada para a construção de jogos, para que possa ser utilizado em conjunto com esta no desenvolvimento de jogos para dispositivos móveis. De seguida é apresentado em detalhe o sistema proposto e a sua especificação.

## 4.2 Solução proposta

De forma a diminuir os efeitos dos problemas de multijogador em dispositivos móveis é proposto criar um sistema de encapsulamento que realiza as operações necessárias para o funcionamento deste tipo de jogos. Ou seja, criar uma camada que abstrai os problemas de redes e de contexto permitindo a comunicação em rede. Este sistema terá como principal objetivo lidar com os seguintes problemas:

- Elevada latência na comunicação (Ligação indisponível durante um curto período de tempo)
- Atraso na entrega de dados (Ligação em pleno funcionamento mas elevado *overhead* ou tráfego)
- Dessincronização dos clientes
- Quebra da ligação
- Desperdício de energia

A solução, denominada de LiNGS (*Lightweight Networked Game System*), abordará estes problemas de forma silenciosa para o programador do jogo posicionando-se entre a *framework* base utilizada para construir o jogo e o jogo em si.

Na Figura 4.1 está demonstrada a forma de integração do sistema LiNGS com a arquitetura geral de um jogo. Este sistema irá estender a *framework* base utilizada para implementar o jogo providenciando funcionalidades de comunicação em rede. Desta forma, os programadores podem continuar a utilizar todas as vantagens da *framework* base e obter ainda mais funcionalidades utilizando o LiNGS, sem que para tal seja necessário realizar alterações na arquitetura interna do jogo.

Figura 4.1: Integração do sistema LiNGS nas camadas de um jogo.

De forma a combater os problemas acima expressos (e em detalhe no capítulo 2) este sistema terá de suportar várias funcionalidades e atributos. De forma a precaver latência de curta duração, o sistema irá realizar simulação de estado com o intuito a permitir que os vários elementos do jogo tenham transições suaves e previsíveis, mesmo com falhas temporárias na ligação. A quebra definitiva de ligação é tratada de outra forma, visto que neste caso um jogador pode perder a ligação durante um tempo maior e imprevisível. Assim, é necessário guardar o estado de cada jogador num servidor. O servidor irá, tal como o jogo em si, estar integrado com o LiNGS para permitir realizar esta operação. Desta forma quando um jogador perde a ligação o seu estado fica guardado no servidor e a sessão pode ser retomada mais tarde, caso esta ainda seja válida.

O gasto de energia e atrasos na ligação são dois problemas que são tratados de forma semelhante. O sistema LiNGS apenas deve transmitir as modificações existentes no jogo e que sejam significantes para cada jogador em vez de transmitir todo o estado atual, reduzindo assim consideravelmente o tráfego necessário para manter os clientes sincronizados e a carga de processamento necessária. Ainda assim, grande parte do tráfego em ligações de rede são dados de controlo, ou dados de formatação. De forma a evitar congestionamentos na rede e evitar processamento desnecessário por parte dos constituintes da rede o sistema irá funcionar de forma simplificada e compacta, como se demonstra de a seguir. O protocolo utilizado pelo LiNGS será suportado pelo protocolo UDP, pela sua simplicidade e baixos recursos utilizados, e será composto da seguinte forma:

| Message – 512 Bytes (Configurable) | | | | |
|---|---|---|---|---|
| 4 Bytes<br>TYPE | 4 Bytes<br>ID | 1 Byte<br>NEEDS_ACK | 4 Bytes<br>DATA_LENGTH | DATA<br>[DATA_LENGTH DATA]{0,N} |

Tabela 4.1: Protocolo de mensagens utilizado pelo LiNGS

A Tabela 4.1 apresenta o protocolo de mensagens utilizado pelo LiNGS. As mensagens são compostas por 5 partes:

- `TYPE` - Tipo da mensagem
  - `Connect` - Utilizado para estabelecer a ligação
  - `Disconnect` - Utilizado para terminar a ligação
  - `Ack` - Indica que emissor recebeu uma mensagem
  - `Data` - Indica que a mensagem contém dados do jogo
  - `Game` - Mensagem enviada pelo jogo, externa à implementação do LiNGS
  - `Event` - Mensagem enviada pelo jogo, externa à implementação do LiNGS
  - `Error` - Mensagem enviada pelo jogo, externa à implementação do LiNGS
  - `ErrorConnect` - Indica um erro a estabelecer a ligação
  - `Heartbeat` - Mensagem para garantir que o cliente se mantém ligado
- `ID` - Identificador da mensagem
- `NEEDS_ACK` - Indica se a mensagem requer um aviso de receção
- `DATA_LENGTH` - Tamanho da parte `DATA` contida na mensagem
- `DATA` - Dados do jogo ou dados auxiliares ao LiNGS

A parte `DATA` da mensagem é de tamanho variável e pode ter um tamanho até 499 Bytes. Esta limitação tem o intuito de evitar pacotes UDP com mais de 576 Bytes e assim garantir que os pacotes não vão ser fragmentados pelos equipamento da rede onde são transmitidos, de acordo com o RFC1122 da IETF (*Internet Engineering Task Force*) [89]. A fragmentação de pacotes pode significar a necessidade de ordenação dos fragmentos e de reconstrução do pacote, sendo assim é imposto o limite de 512 Bytes, por omissão, para cada mensagem do

protocolo utilizado no LiNGS de forma a evitar tais operações.

De reafirmar ainda que o LiNGS é um sistema genérico destinado a ser utilizado por vários tipos de jogos com várias necessidades e a executar em contextos diferentes (capacidades do dispositivo, tipo de rede, etc.), assim, torna-se necessário a existência de uma capacidade de configuração fácil dos vários parâmetros deste sistema. O tamanho máximo das mensagens é um parâmetro que deve ser possível configurar na execução do LiNGS.

A formatação dos dados inseridos na `DATA` é um ponto de foco na transmissão de dados. Como este é um sistema genérico e não um sistema apropriado para apenas um único jogo ou plataforma torna-se necessário formatar os dados para que possa ser feito o encaminhamento para os vários elementos de jogo. Assim, é necessário determinar em cada pacote enviado qual é o objeto e quais os atributos deste a que se destinam os dados enviados. A formatação deste tipo de informação deverá ser feita com a utilização mínima de informação extra para que a quantidade de dados transmitidos seja diminuta.

Ainda para remediar o problema do gasto de energia deve ser implementada uma deteção do estado do jogo para que a frequência no envio de dados seja adequada às necessidades deste. Desta forma a transmissão de dados pode ser mais lenta em situações paradas ou mais rápida quando o jogo tem muitas interações.

Assim, seguem-se as principais funcionalidades do LiNGS de forma a combater os problemas afetos às redes em dispositivos móveis:

- Transmissão de dados de forma adequada ao contexto
  - Pouco *overhead* (tamanho dos pacotes pequeno e número de transmissões de rede baixo)
- Gestão de energia
  - Agrupamento de dados a enviar
  - Utilização do estado do jogo e estados do dispositivo para determinar quando enviar dados e quais os dados a enviar (estado da bateria, força do sinal de rede, estado do jogo, etc.)

- Delegação de processamento para o servidor

- Minimização de problemas de latência
  - Sistema de simulação de estado
  - Sincronização de dados
- Qualidade de serviço
  - Diferenciação de tráfego
- Limitar problemas de perda de ligação
  - Possibilidade de religar ao servidor mantendo a sessão

A integração de todas estas soluções num único sistema é demonstrada na próxima secção onde é representada a arquitetura do sistema.

## 4.3 Arquitetura

Visto que se pretende que os jogos desenvolvidos com LiNGS sejam executados plataformas móveis com limitações tanto em processamento como energéticas é necessário que este sistema tenha um efeito reduzido no agravamento destes atributos. Assim, o LiNGS é um sistema que deve ser compacto e deve utilizar uma baixa quantidade do processamento disponível. A arquitetura do LiNGS foi desenhada de forma a cumprir estes requisitos. Desta forma, é necessário separar o sistema em duas partes: cliente e servidor. Esta separação determina que parte do sistema estará a atuar junto do cliente (ou neste caso no jogo) na plataforma móvel e a outra parte estará a atuar no servidor, que em casos extraordinários poderá estar no mesma dispositivo que o cliente. É de seguida descrita a arquitetura da parte cliente do sistema LiNGS.

### 4.3.1 Cliente

O cliente é normalmente executado nos dispositivos dos utilizadores e é utilizado em conjunto com a restante implementação do jogo para representar o estado atual de um jogo e recolher dados de entrada do utilizador de forma a que estes sejam enviados para o servidor

com o objetivo de serem tratados.

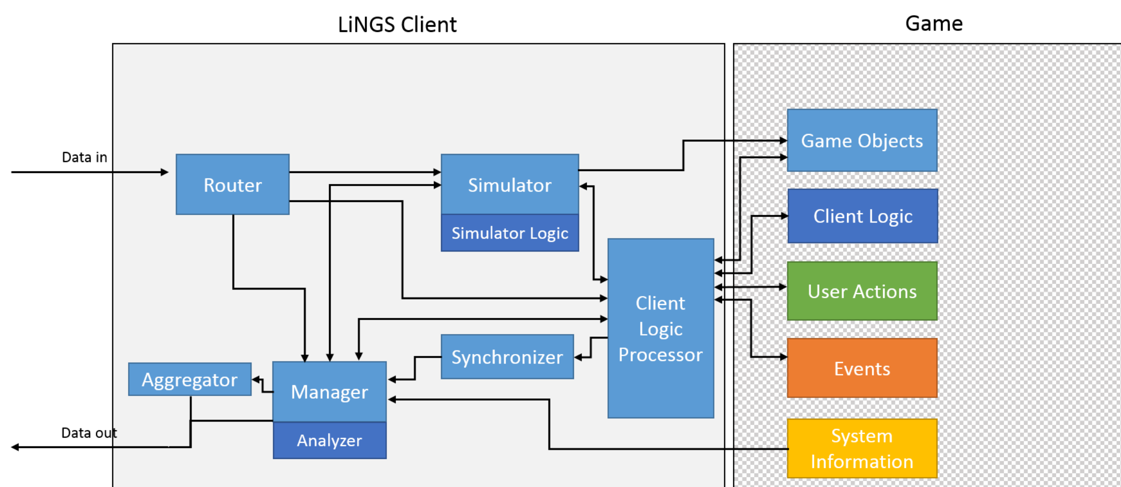


Figura 4.2: Arquitetura do LiNGS - Parte incluída com o jogo.

Na Figura 4.2 está representada a arquitetura geral do LiNGS na parte cliente. O sistema é bastante compacto, tendo apenas os subsistemas estritamente necessários para garantir as funcionalidades da solução apresentada na secção 4.2. Na Figura 4.2 estão representados os vários componentes do sistema e também a interação de alto nível entre estes. É ainda representado a interação dos vários componentes com o resto do jogo. Os vários componentes do LiNGS são:

- `Router`
- `Simulator`
  - `Simulator Logic`
- `Client Logic Processor`
- `Synchronizer`
- `Manager`
  - `Analyzer`
- `Aggregator`

O componente `Router` é o ponto de entrada de dados da rede e garante que os dados recebidos são encaminhados para os componentes certos. Dados de alteração de estado do jogo são enviados para o `Simulator` ou para o `Client Logic Processor`, de acordo com o seu tipo. Por sua vez os dados de controlo de rede são redirecionados para o `Manager`.

Os dados recebidos pelo `Simulator` são agregados aos dados previamente recebidos para serem utilizados de forma a simular o estado dos elementos de jogo em momentos posteriores. Associado ao `Simulator` está o `Simulator Logic`. Este componente é composto por vários mecanismos que contêm a lógica de simulação de vários tipos de estados, tais como: simulação de movimentação e simulação de transações e animações. Assim, o `Simulator` é um dos componentes mais importantes do sistema LiNGS. Este componente recebe e transforma os dados provindos do servidor para serem utilizados no vários objetos presentes no jogo. Os dados recebidos, de acordo com o seu tipo, podem ser inseridos diretamente nos objetos de jogo ou então podem ser utilizados para simular o seu estado futuro. Um exemplo clássico deste tipo de simulação é a previsão da posição de um objeto tendo a informação das suas posições anteriores, sendo possível simular a próxima posição do objeto de forma aproximada. Esta simulação garante que os objetos de jogos vão ter valores reais mesmo quando existe perda momentânea de dados de rede.

A comunicação com os objetos de jogos é feita diretamente pelo `Simulator` e também pelo componente `Client Logic Processor`. O componente `Client Logic Processor` é a ponte entre o sistema LiNGS e a restante implementação do jogo. Este permite à implementação do jogo conhecer quais os objetos que devem estar presentes no jogo, permite a troca de mensagens entre o sistema cliente e servidor e notifica ainda a implementação do jogo de eventos que ocorreram no sistema de rede, LiNGS.

O componente `Synchronizer` permite sincronizar objetos de jogo criados localmente no cliente para o servidor. Este componente permite portanto sincronização *full-duplex* dos objetos de jogo, ou seja para além dos clientes receberem o estado dos objetos criados no servidor é também possível por este componente que o estado de objetos de cada cliente sejam enviados para o servidor. Assim, é permitido a troca fácil de informação entre o cliente e servidor e vice-versa.

De forma a controlar todo o sistema existe o componente `Manager`. Este recebe dados de controlo da rede e com o auxílio do `Analyzer` procede a uma observação constante de todo

o sistema para garantir que o mesmo está estável. Este garante que as comunicações em rede estão a ser feitas com os intervalos certos e que as respostas estão a chegar ao destino. Assim é capaz de informar os outros componentes de atrasos ou falhas na ligação. Este componente admite ainda toda a informação a ser enviada pelo jogo e de acordo com a prioridade desta encaminha-a para a rede ou para o `Aggregator`.

Por último o `Aggregator` é o subsistema que acolhe dados com prioridades menos altas e apenas os envia quando existir uma quantidade excessiva destes ou quando o tempo de envio dos mesmos expirar. Assim os dados são enviados o mais em conjunto possível baixando o número de interações de rede necessárias para o funcionamento do jogo.

Assim, a composição arquitetural do LiNGS é formada pelos componentes acima descritos e garante as várias funcionalidades da solução, em conjunto com a parte de servidor que é demonstrada de seguida.

### 4.3.2 Servidor

A parte Servidor do sistema LiNGS completa as funcionalidades do Cliente mantendo a sincronização dos dados de jogo pelos vários clientes, fornecendo ainda mecanismos para suporte à lógica e controlo do jogo.

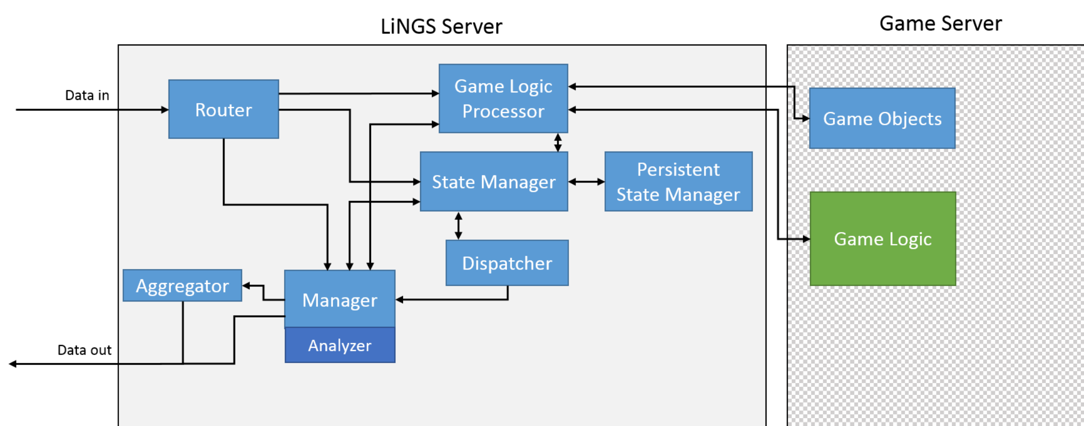


Figura 4.3: Arquitetura do LiNGS - Parte incluída com o servidor.

A parte de servidor do LiNGS é responsável por completar todo o sistema de multijogador,

esta realiza as tarefas de processamento principal de lógica do jogo, garante o estado imediato de todos os jogadores, encaminhando os dados necessários e, permite ainda guardar o estado de jogadores quando existe falhas de ligação.

Esta parte do sistema é composta por 7 componentes como demonstrado na Figura 4.3. A estrutura interna de funcionamento, tal como a parte cliente deste sistema, é bastante simples e direta com o objetivo de ter uma carga de processamento pequena e consumo de memória reduzido de forma a evitar atrasos no processamento do jogo e sobrecarga na plataforma onde está alojado.

Analisando os vários componentes da Figura 4.3 é possível encontrar semelhanças com a parte de cliente deste sistema. Existem vários componentes nas duas partes com objetivos semelhantes. O componente `Router` encaminha todos os dados recebidos da rede para os componentes respetivos. Por sua vez os componentes `Manager` e `Aggregator` seguem a mesma responsabilidade que na parte Cliente do LiNGS. O `Manager` obtém toda a informação do sistema e recorrendo ao componente auxiliar, `Analyzer`, mantêm uma noção do funcionamento total do sistema. Portanto, este componente observa o funcionamento do sistema, focando a parte de rede, e adota medidas para limitar problemas encontrados. Ainda realizado por este componente é o tratamento de dados a enviar para a rede. O componente recebe os dados e de acordo com a prioridade destes, são enviados para a rede ou para o componente `Aggregator`. Tal como na parte cliente do sistema, o componente `Aggregator`, tem o objetivo de juntar os dados de forma a envia-los em conjunto para a rede evitando realizar muitas transações pequenas. Desta forma, o hardware de rede pode ficar mais tempo adormecido para conservar energia.

Estes componentes são então utilizados de forma semelhante nas duas parte do sistema mantendo a eficácia e coerência operacional de forma transversal entre os vários pontos. É necessário, no entanto, outros componentes para completar o lado servidor deste sistema. O componente `Game Logic Processor` recebe dados de lógica de jogo, nomeadamente eventos e transações e reencaminha-os para o processador de lógica do jogo em si ou para o gestor de estados do LiNGS. De uma forma abstrata o `Game Logic Processor` fornece uma ponte entre o processador de lógica do jogo e o `State Manager`, o componente que serve para gerir o estado dos vários elementos de jogo.

O `State Manager` recebe dados dos clientes relativos a mudanças de estado de elementos

de jogo e aplica essas mudanças ao estado atual de cada elemento para que os processadores de lógica tenham acesso a essa informação e consigam introduzir consequências às ações realizadas. O `State Manager` analisa também mudanças de estado nos elementos de jogo presentes no servidor e seleciona os dados necessários para enviar para os clientes, caso estes os requeiram. Assim, é ainda mantido o estado de cada cliente de forma a saber quais os dados que são necessários enviar, enviando apenas as diferenças de estados para os clientes, evitando enviar todo o estado atual do jogo. Para garantir a sessão de jogadores que fiquem sem ligação ao servidor é necessário guardar os estados dos elementos respetivos a esse jogador de forma persistente. Para tal existe o componente `Persistent State Manager` que tem o objetivo de manter sessões de jogadores de forma persistente no sistema. Estas sessões têm características que as tornam inválidas e desta forma têm de ser geridas por este componente para garantir a sua usabilidade.

Por último o `Dispatcher` é o componente que decide quais os dados que devem ser enviados para cada cliente ou neste caso jogador. Este é um dos componentes mais relevantes na implementação do LiNGS sendo que o seu trabalho, juntamente com o `State Manager`, tem o objetivo de manter os vários clientes sincronizados enviando o mínimo de dados necessários para tal. Este componente verifica quais os dados processados pelo `State Manager` têm alterações e procede enviando para os clientes respetivos as informações adequadas para recriar o estado dos elementos de jogo que sofreram alterações.

A arquitetura do LiNGS é assim composta pelos vários componente acima descritos de forma a permitir a comunicação em rede entre as várias partes do jogo, nomeadamente clientes e servidor. Depois de analisado as duas partes do sistema LiNGS e os seus componentes integrantes segue-se uma especificação das comunicações entre estas partes.

## 4.4 Sequências de acontecimentos

A sequência de comunicação entre os vários elementos do sistema LiNGS é fulcral para o correto funcionamento do sistema. Assim, torna-se necessário especificar a interação das várias partes do sistema de forma concreta. Existem momentos de comunicação em que são impostas regras de forma garantir que o sistema LiNGS se encontra em correta sincronia.

A especificação destas sequências garante ainda que qualquer implementação do LiNGS consegue comunicar entre si ostentando um comportamento idêntico.

### 4.4.1 Estabelecimento da ligação

O estabelecimento da ligação entre o cliente e o servidor tem sequência representada na seguinte figura.

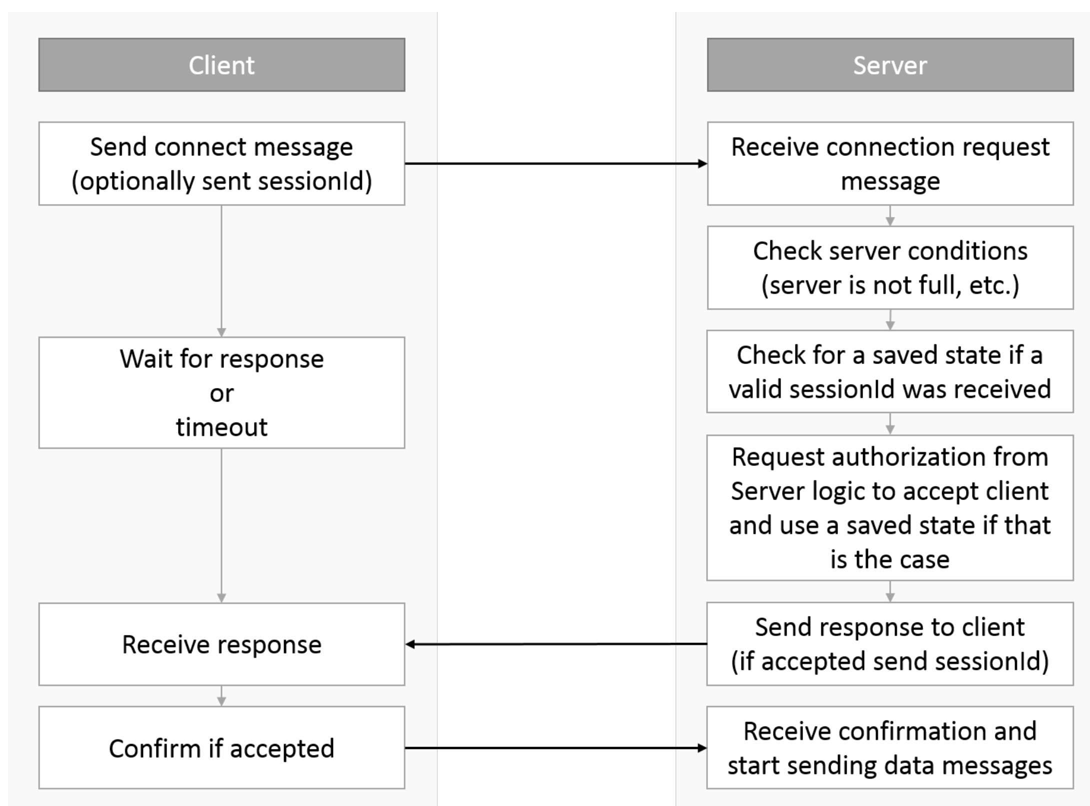


Figura 4.4: Estabelecimento da ligação entre cliente e servidor.

A sequência de acontecimentos da Figura 4.4 representa o estabelecer da ligação entre um cliente e o servidor. Numa primeira fase o cliente envia a requisição para se ligar ao servidor com a possibilidade de envio do identificador da sessão. Este identificador é o valor que identifica uma sessão anterior e é utilizado para reestabelecer uma ligação perdida ao servidor. Depois de enviar a mensagem de ligação com o tipo correto – `Connect` – o cliente fica à espera de uma resposta do servidor durante um período de tempo configurável na implementação do sistema. Caso o cliente não receba uma resposta do servidor dentro desse

período então a tentativa de ligação é cancelada.

Na fase seguinte da ligação o servidor recebe a mensagem do cliente e realiza as validações necessárias de forma a aceitar ou recusar o pedido de ligação feito pelo cliente. É verificado se o número de clientes ligados ao servidor já atingiu o limite e, caso o cliente tiver enviado o identificador de sessão, é verificado também se existe uma sessão do cliente previamente guardada.

Depois das verificações do sistema LiNGS cabe ao jogo decidir se aceita ou não o cliente. As informações recolhidas pelo sistema (identificador de sessão, existência de uma sessão guardada, etc.) são encaminhadas para a implementação do jogo e é indicado por este se o cliente deve ser aceite e caso existe uma sessão guardada, se esta deverá ser utilizada. De acordo com a resposta da implementação do jogo é enviado uma mensagem para o cliente a indicar a resposta ao pedido de ligação e em caso de sucesso é enviado o identificador de sessão atual e é ainda indicado se foi utilizada uma sessão previamente guardada.

O cliente deve ainda confirmar ao servidor a receção desta resposta de forma a finalizar o processo de estabelecimento da ligação. Quando a confirmação chega ao servidor então o processo de ligação está concluído e o LiNGS começa a realizar a sincronização dos dados de forma a suportar a lógica do jogo.

### 4.4.2 Sincronização de dados

O processo de sincronização de dados no sistema LiNGS não tem uma sequência rígida de funcionamento. Este tipo de troca de informação deve ser feito o mais rapidamente possível e deve ocupar poucos recursos da rede e dos dispositivos tal a quantidade e regularidade de mensagens normalmente trocadas nos jogos digitais. Assim, apesar de não existir uma sequência complexa de troca de informação, existem várias regras que devem ser cumpridas de forma a aumentar a eficiência do sistema LiNGS.

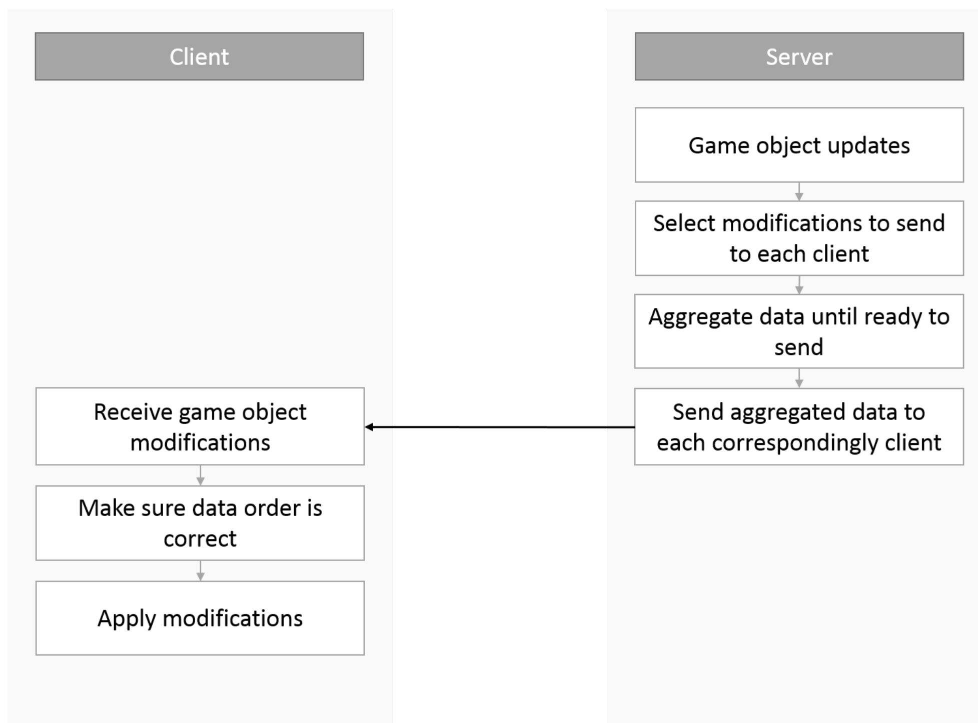


Figura 4.5: Sequência de sincronização de dados entre servidor e cliente.

Na Figura 4.5 é mostrado o processo de sincronização de dados. Quando existe uma modificação a qualquer objeto de jogo, ou quando um objeto é criado ou destruído, é necessário sincronizar esse acontecimento pelos vários elementos da rede. O servidor deteta estes acontecimentos e compõe mensagens personalizadas para enviar a cada cliente de forma útil e eficiente, i.e. o servidor envia apenas as modificações e não todos os dados e agrupa as mensagens para reduzir o número de comunicações. Quando o cliente recebe tais mensagens pode existir necessidade de processar os dados da mensagem com uma ordem específica.

Quando existe a criação de objetos a mensagem enviada pelo servidor pode conter a informação da criação de cada objeto e os valores associados aos campos desse objeto. Este agrupamento de informação é feito, tal como indicado previamente, de forma a reduzir o número de comunicações realizadas entre os elementos da rede. Sendo assim, é necessário garantir que a criação de objetos é realizada antes de processar os dados relativos a objetos, visto que de outra forma o objeto a que se destinam os dados pode ainda não ter sido criado.

Este é apenas um caso particular na sincronização dos dados, mas o seu cumprimento garante

que os recursos da rede são utilizados de forma eficiente. A restante ordem de transmissão de dados relativamente a sincronização é feita de forma normal sem outros casos especiais. Quando um objeto é alterado as modificações ocorridas são sincronizadas pelos vários elementos na rede. O servidor tem a responsabilidade de apenas enviar as modificações dos objetos e não toda a informação do mesmo.

### 4.4.3 Fecho da ligação

Nos sistemas onde existe o estabelecimento de ligação ou sessão existe também o término destas. Nos sistemas de rede o término da ligação pode ocorrer normalmente por via de 3 ações:

- O cliente termina a ligação
- O servidor termina a ligação
- A ligação é fechada por problemas de comunicação

Quando a ligação é terminada pelo cliente é enviada um mensagem do tipo `Disconnect` para o servidor a informar o fecho da ligação. O cliente deixa então de iniciar e receber comunicações do servidor. Quando o servidor recebe a mensagem do cliente deve realizar as ações necessárias para informar a implementação do jogo do fecho da ligação com o cliente.

O processo do fecho da ligação por parte do servidor é procedido de forma inversa. O servidor envia para o cliente uma mensagem com o tipo `Disconnect` contendo uma explicação do fecho da ligação. Quando a mensagem é recebida pelo cliente a implementação do jogo é informada da perda de ligação e o cliente deixa de realizar e aceitar comunicações do servidor.

A última causa para o fecho da ligação entre cliente e servidor é um dos principais problemas das redes sem fios. A rede utilizada para comunicação pode ficar instável e deixar de conseguir suportar a comunicação entre os vários intervenientes, levando a quebras de ligação abruptas. Assim, cabe ao servidor e também ao cliente garantir que a comunicação entre o respetivo está a ser realizada com sucesso. Estes sistemas devem analisar a comunicação realizada entre si e averiguar a latência de ligação e garantir que a outra parte se mantém ativa a acessível.

O sistema de análise do estado das ligações torna-se essencial para um sistema que utilize redes com baixa fiabilidade. Este sistema verifica o estado da ligação e permite ainda que o fecho da ligação de forma voluntária pelo cliente ou servidor não tenha de ser confirmado. Quando uma parte do sistema LiNGS termina a ligação mesmo que a mensagem a indicar tal acontecimento não chegue ao destino a ligação é encerrada nas duas partes, visto que deixa de ser possível realizar comunicação a então a ligação é então terminada.

### 4.4.4 Outros pontos de comunicação

O sistema LiNGS deve permitir a troca de mensagens personalizadas entre o servidor e os vários clientes. Estas mensagens são marcadas como importantes e devem ser confirmadas pelo recetor, como ilustrado na seguinte imagem:

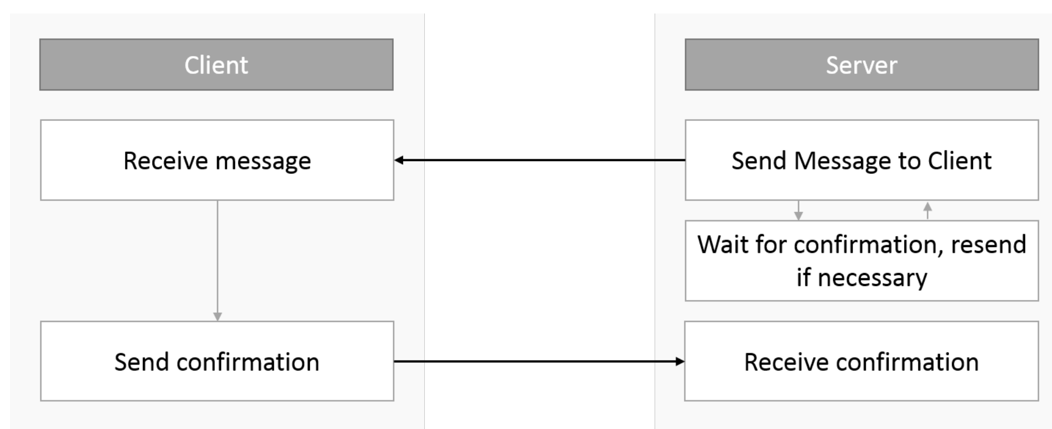


Figura 4.6: Processo de envio de mensagens importantes.

A troca de informação entre os intervenientes deve ser feita como ilustrado na Figura 4.6. O cliente ou servidor envia uma mensagem importante e fica a aguardar por uma mensagem de confirmação do recetor. Por sua vez quem recebe a mensagem deve enviar uma nova mensagem a confirmar a receção da mensagem importante. Na situação em que a mensagem importante não seja recebida pelo recetor ou em que a mensagem de confirmação não chegue ao emissor então o próprio deve reenviar a mensagem importante e todo o processo é repetido.

Outros tipos de comunicação devem ser realizados de forma simples, sem necessidades es-

peciais. Desta forma o sistema LiNGS garante que dados importantes são realmente transmitidos para os vários recetores, funcionando sobre uma camada simples e pequena de forma a evitar *overhead* na transmissão de informação.

## 4.5 Integração

O desenvolvimento de jogos e aplicações é atualmente mais do que nunca feito com auxílio de *frameworks*, tal como indicado no Capítulo 2. Desta forma, a integração do LiNGS é agnóstica à plataforma onde executa, ao tipo de jogo e ao tipo de rede de comunicação onde serão transmitidos os dados do jogo.

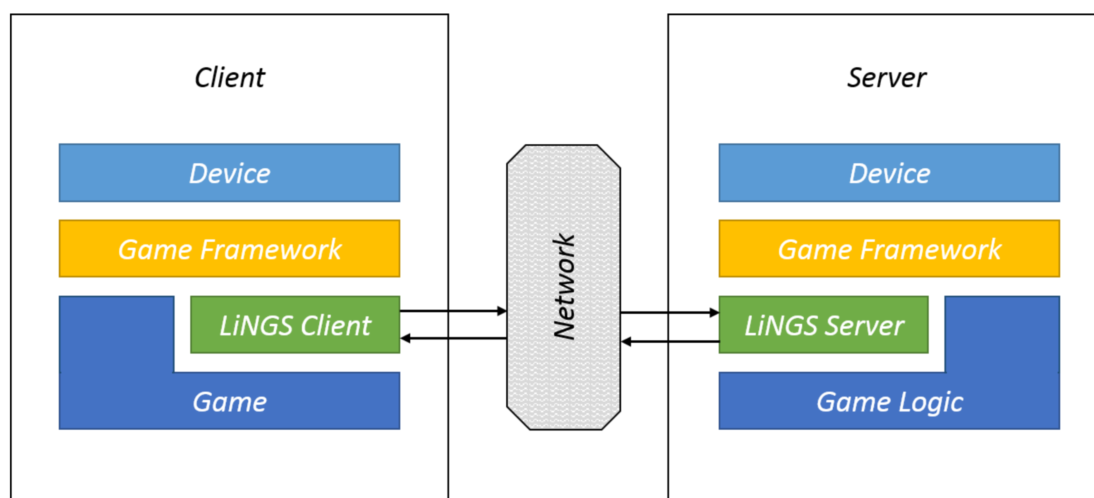


Figura 4.7: Arquitetura do LiNGS - Integração dos componente cliente e servidor.

A integração do LiNGS num jogo não deve implicar a utilização de um sistema operativo ou de uma *framework* de suporte base especifica. Tal como mostra a Figura 4.7, o LiNGS deve ser apenas uma camada auxiliar colocada entre a *framework* de suporte ao jogo e o jogo em si. Assim, a especificação atual não inflige a utilização de mecanismos que restrinjam a implementação e utilização do sistema LiNGS em qualquer dispositivo.

## 4.6 Conclusão

Os jogos de multijogador em rede em dispositivos móveis sofrem de vários problemas afetos ao seu contexto. Assim, de forma a reduzir esses problemas é proposto uma *framework* para jogos multijogador em dispositivos móveis. A *framework* tem como objetivos fazer a ponte entre a lógica de jogo do servidor e os vários clientes, mantendo todos os intervenientes no jogo sincronizados, e abstrair toda a parte de rede, de forma a que a implementação do jogo não tenha de intervir nos problemas diretos da rede, i.e. pouca estabilidade e latência elevada.

No entanto, face aos problemas encontrados neste contexto, a *framework* terá de realizar os objetivos de forma simples e eficaz e ainda de uma forma a reduzir o impacto nos recursos dos dispositivos onde vai executar. Desta forma é necessário separar a arquitetura do sistema LiNGS em cliente e servidor. Estas duas partes são por sua vez separadas em vários componentes que realizam as tarefas necessárias para manter todos o sistema e jogo sincronizados e ainda evitar os problemas de rede.

A especificação do LiNGS é ainda abstrata de forma a que o sistema possa ser implementado e executado em vários sistemas operativos e plataformas. Esta particularidade permite que o LiNGS seja utilizado em conjunto com outras *frameworks* para a implementação de jogos e permite também o suporte a qualquer tipo de jogos.

# Capítulo 5

# Implementação

## 5.1 Introdução

A proposta efetuada no capítulo anterior indica a especificação de uma *framework* com o objetivo de reduzir os problemas das redes e limitações dos dispositivos móveis em jogos multijogador. De forma a colocar o sistema especificado em prática é necessário realizar a sua implementação e integração em ambiente real. Assim, neste capítulo é disseminado toda a fase de implementação do sistema LiNGS seguindo a arquitetura especificada anteriormente.

## 5.2 Requisitos e funcionamento base do LiNGS

De acordo com a especificação apresentada no Capítulo 4, o sistema LiNGS tem várias características derivadas do ambiente onde irá atuar. Tal como analisado anteriormente, no Capítulo 3, os dispositivos móveis e as redes de comunicação que estes utilizam são afetados por vários problemas e limitações, que obrigam as aplicações a utilizarem mecanismos para contornar estes desafios de forma a apresentarem o comportamento esperado. Torna-se assim necessário garantir que o sistema LiNGS consiste num acréscimo útil ao desenvolvimento e execução de jogos e não num acessório que realiza a sua ação, trazendo vários problemas e dificuldades tanto ao desenvolvimento como na execução do próprio jogo.

Desta forma, a implementação do LiNGS tem, de acordo com a especificação, de garantir os seguintes aspetos:

- Suportar a sua integração numa *framework* base para criação de jogos
- Utilizar poucos recursos do dispositivo
- Fornecer as funcionalidades enunciadas da especificação do sistema LiNGS de forma simples e útil para os programadores

A implementação concreta das funcionalidades do sistema LiNGS pode ser feita de diversas formas. O objetivo desta implementação em particular é garantir aos programadores que não existe necessidade de alterar a arquitetura dos seus jogos de uma forma extensa. O intuito principal passa por implementar a *framework* LiNGS para que a sua API (*Application Programming Interface*) não se estenda pela restante implementação do jogo, minimizando as modificações necessárias às classes utilizadas para controlar a lógica do jogo.

No entanto, todo este processo de implementação depende da *framework* base utilizada para implementar o sistema LiNGS e para a sua posterior utilização. De forma a permitir a sua utilização nos mais variados tipos de jogos uma *framework* requer um nível de abstração elevado, permitindo realizar as várias tarefas destinadas, sem implicar modificações excessivas no contexto externo a si mesmo. Assim, o LiNGS é implementado numa *framework* base para suportar a criação de um jogo, tal como será analisado de seguida.

## 5.3 Framework base

As *frameworks* são atualmente uma ferramenta bastante vantajosa na criação de aplicações, tal como concluído no Capítulo 2. A sua utilização é atualmente uma mais-valia para os criadores de jogos. Desta forma, um sistema para acrescento de funcionalidades a jogos deve ser desenvolvido de forma a ser integrado noutros sistemas, principalmente em *frameworks*.

O Unity é atualmente uma das *frameworks* mais utilizadas no desenvolvimento de jogos para dispositivos móveis [46] [90]. As características enunciadas no Capítulo 2 fazem desta *framework* uma ajuda bastante desejada na criação de jogos. Não apenas pela *framework* em si, mas também por todas as ferramentas incluídas no ambiente. O Unity fornece aos programadores um ambiente suficiente para permitir desenvolver qualquer tipo de jogo, de forma rápida e simples.

O Unity Editor é o editor gráfico incluído nesta *framework* e permite editar o jogo a nível visual. Todos os elementos no jogo são manipulados neste editor visual, desde texturas, modelos em três dimensões, emissores de som, partículas, etc. O editor suporta ainda a construção de animações e configuração de sistemas de estados de forma totalmente visual. De forma a auxiliar e agilizar a criação de jogos, o editor suporta ainda a execução direta do jogo, assim este executa diretamente no editor e é possível observar o estado de todos os elementos do jogo e até mesmo efetuar alterações durante a execução. O editor permite ainda configurar as várias opções do jogo, incluindo opções de exportação para as plataformas alvo, sendo a exportação do jogo para as várias plataformas realizada também no editor.

O Unity é assim uma *framework* que de forma utilitária fornece uma ferramenta bastante poderosa para a criação de jogos, o Unity Editor. Incluído no pacote vem ainda o editor de código MonoDevelop. Este editor é utilizado para realizar a edição e depuração do código necessário para controlar os elementos presentes nos jogos.

Em comparação com outras *frameworks*, o Unity fornece todas estas características numa versão gratuita e com suporte a exportação de jogos para múltiplas plataformas móveis. Estes atributos tornaram o Unity bastante popular no desenvolvimento de jogos para plataformas móveis, o Unity é atualmente uma das *frameworks* mais utilizadas por empresas e programadores singulares para a criação de jogos.

Os motivos apresentados levaram à escolhida do Unity como *framework* base para a implementação do sistema LiNGS. As características do Unity permitem o desenvolvimento de jogos de forma ágil e a popularidade desta *framework* permite que a implementação do sistema LiNGS fique disponível para uma grande parte do mercado. O funcionamento interno do Unity e dos seus elementos é de seguida explicado.

### 5.3.1 Funcionamento Interno do Unity

Os jogos são um tipo de aplicação que normalmente tem um nível de complexidade elevado devido à quantidade enorme de possíveis estados de cada elemento presente no jogo. É necessário processar entradas de vários dispositivos (Teclado, Controladores, etc.) e distribuir essa informação para cada elemento de jogo, que por sua vez utiliza essa informação para alterar o seu estado. Depois de todo este processamento é necessário ainda enviar as respetivas saídas para os dispositivos associados, que podem ser de componente audiovisual ou

outros como por exemplo dispositivos de *feedback* por vibração. Tendo em conta que os jogos podem ter milhares de elementos com uma variedade enorme de estados é necessário um sistema para estruturar toda a arquitetura constituinte de um jogo de forma organizada e eficiente.

De forma a criar uma separação lógica de todos os elementos de um jogo o Unity proporciona uma granularidade de 4 níveis, Figura 5.1:

- *Scenes*
- *Game Objects*
- *Components*
- *Scripts*

As *Scenes* representam o estado global do jogo, por exemplo: Cada menu e cada nível de um jogo são `Scenes`. Apenas uma *Scene* pode estar aberta simultaneamente sendo que a transição do estado do jogo é normalmente representada pela transição entre *Scenes*. Dentro de cada *Scene* estão contidos todos os elementos representados na mesma.

Os *Game Objects* são todos os objetos que estão instanciados no jogo, podendo estes serem visuais ou apenas objetos de lógica que trabalham em segundo plano, podendo ainda estar desativados, sem realizar processamento. Estes objetos estão inseridos nas *Scenes* e podem ser inseridos estaticamente, através do editor visual do Unity, o Unity Editor, ou podem ser carregados dinamicamente. Os *Game Objects* por sua vez contêm *Components*.

Os *Components* são anexos que dão vida aos *Game Objects*. Estes são os elementos que realizam as operações necessárias para processar e representar o estado de cada *Game Object*. Os *Components* são agregados aos *Game Objects*, que por sua vez estão contidos nas *Scenes*, e fornecem-lhes uma capacidade específica de acordo com o seu desígnio, por exemplo: um *Component* do tipo "GUI Text" anexado a um *Game Object* permite representar texto 2D no ecrã de jogo, por sua vez um "GUI Texture" permite mostrar uma textura.

O Unity fornece um enorme conjunto de *Components* que realizam uma vasta gama de tarefas [91]. No entanto, cada jogo é diferente e de forma a criar a lógica para estes é necessário utilizar *Scripts*. Os *Scripts* são *Components* criados e programados pelos programadores que

seguem regras estabelecidas pelo Unity [52] de forma a efetuarem operações nos *Game Object* onde estão anexados ou noutros *Game Objects* no jogo. A programação destes *Scripts* é feita no editor MonoDevelop utilizando a linguagem C#, JavaScript ou BOO.

A base para a construção dos *Scripts* é a Mono Framework [64] que por sua vez é baseada na .NET Framework [24]. A .NET Framework é uma *framework* genérica utilizada maioritariamente em sistemas Microsoft. A popularidade desta *framework* levou a que fossem criadas *frameworks* em si baseadas, mas para outros sistemas. Nesta caso o Unity utiliza a Mono Framework. A utilização desta *framework* como base de programação dos *Scripts* fornece um vasto conjunto de funcionalidades úteis para o desenvolvimento de jogos. Estas funcionalidades representam auxílios na construção dos jogos fornecendo aos programadores formas de gestão de objetos, gestão de memória automática, acesso a ficheiros e a outros dispositivos de IO (*Input* / *Output*), etc.

Desta forma, o Unity, como *framework*, permite manter uma gestão eficiente de todos os elementos que compõem um jogo. A separação dos elementos por granularidade permite um encapsulamento flexível e simples.

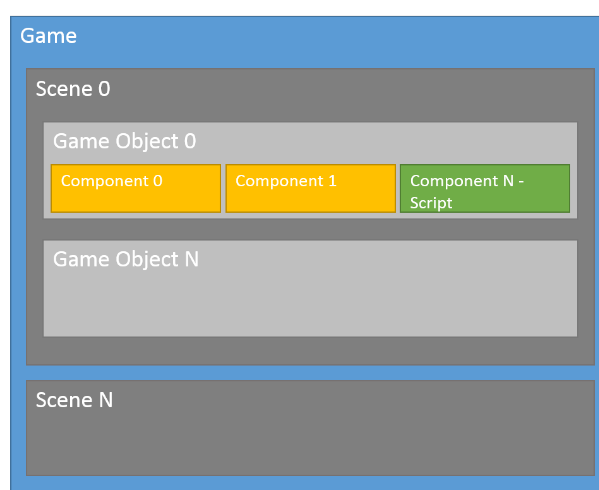


Figura 5.1: Estrutura dos elementos que compõem um jogo em Unity.

A Figura 5.1 representa a organização dos elementos que constituem um jogo realizado no Unity, Tal como em cima referido, a estrutura de um jogo decompõe-se em 4 granularidades, sendo assim possível administrar com facilidade os elementos que devem estar no jogo em

cada estado deste. À exceção dos *Scripts*, todos os elementos de um jogo do Unity são geridos no Unity Editor. Este editor fornece aos seus utilizadores uma representação plena de cada elemento.

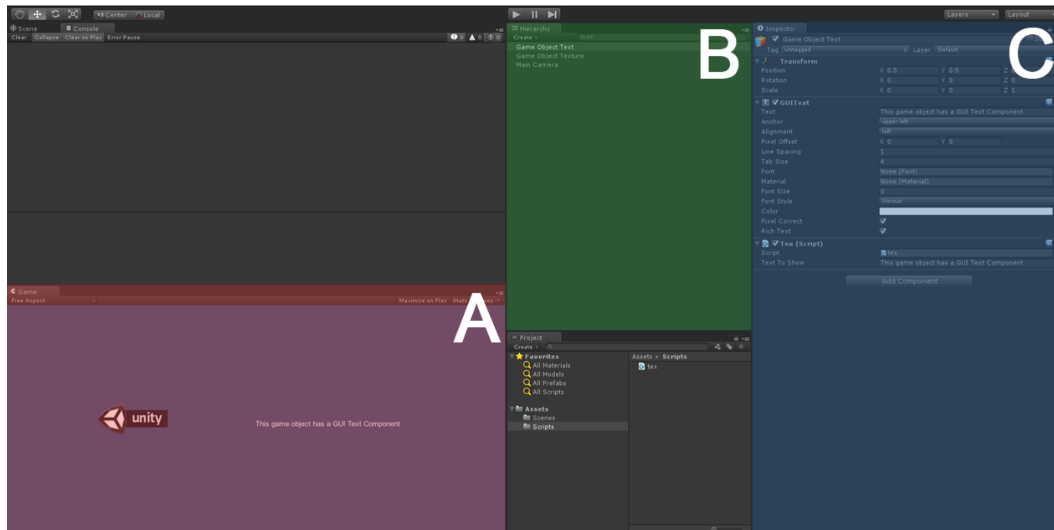


Figura 5.2: Unity Editor com as zonas de configuração de elementos de jogo realçadas.

A Figura 5.2 realça as várias zonas do editor Unity Editor onde os utilizadores podem gerir os elementos de jogo. Neste caso o editor tem uma *Scene* aberta e a representação visual da mesma está realçada na zona **A**, onde é possível obter uma representação visual de cada *Game Object* que tenha forma de se desenhar. Na zona **B** estão enunciados todos os *Game Objects* que se encontram na *Scene* atual. É possível adicionar e remover estes objetos do jogo no editor ou então programaticamente. Na zona representada pela letra **C** encontram-se as propriedades do *Game Object* selecionado. São mostrados os vários *Components*, inclusive *Scripts*, que o *Game Object* tenha e todas as propriedades destes. Aqui o utilizador pode configurar o *Game Object* alterando os componentes e as propriedades respetivas de cada componente.

O Unity fornece assim uma excelente base para o desenvolvimento de jogos e até mesmo outros tipos de aplicações. As funcionalidades do Unity em conjunto com as funcionalidades da integrada .NET Framework serve de plataforma para a criação de jogos de forma rápida e precisa. As ferramentas incluídas no pacote Unity são também uma grande ajuda na criação de jogos e permitem a utilização da *framework* e dos seus componentes de forma visual o que permite um desenvolvimento mais rápido e dinâmico.

Com a *framework* base, Unity, definida e analisada segue-se no próximo capítulo a demonstração e análise da implementação do sistema LiNGS.

## 5.4 Implementação do sistema LiNGS

A utilização do Unity como *framework* base introduz várias limitações e vantagens na implementação do LiNGS. As limitações cingem-se maioritariamente à forma de implementação e à linguagem utilizada. Por sua vez as principais vantagens da utilização do Unity como *framework* base são o suporte para a fácil integração do sistema LiNGS como um *plugin* do Unity e a possibilidade de utilização das funcionalidades da .NET Framework para o desenvolvimento do LiNGS.

A .NET Framework garante o suporte para todas as necessidades do sistema LiNGS, nomeadamente *sockets* para transmissão de dados [92], acesso a ficheiros e funcionalidades de *reflection* [93]. Esta *framework* utiliza, entre outras, a linguagem C# para programação o que também é uma vantagem visto a quantidade de funcionalidades suportadas por esta linguagem [94]. A linguagem C# é uma linguagem de programação orientada a objetos que tem sido desenvolvida e atualizada desde o seu lançamento em 2001. Esta linguagem está fortemente interligada com o .NET Framework e é utilizada num ambiente não nativo com gestão de memória automática. Os projetos realizados na linguagem C# são compilados para um linguagem intermédia, denominada de CIL (*Common Intermediate Language*), que por sua vez é convertida durante a execução para código nativo, adequado à plataforma, pela *framework* .NET.

Na Figura 5.3 está esquematizado o processo de compilação e execução de uma aplicação realizada na linguagem C#. Este processo garante que as aplicações realizadas em C# executem em qualquer sistema desde que exista uma camada que funcione como uma máquina virtual convertendo a linguagem CIL para código nativo e que forneça as várias capacidades da .NET Framework, tais como gestão de memória automática, segurança, tipos base, etc. Desta forma, como o código é compilado de forma nativa em tempo de execução é possível ainda introduzir otimizações que não eram conhecidas durante a compilação para CIL.

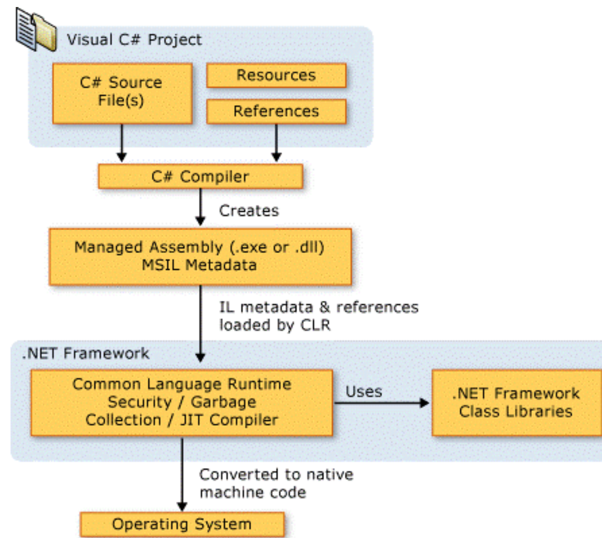


Figura 5.3: Processo de compilação / execução de uma aplicação na linguagem C# [2].

A implementação do LiNGS será assim realizada com a utilização da .NET Framework, versão 3.5, com a linguagem C# e terá como destino principal jogos realizados em Unity.

### 5.4.1 Estrutura de implementação

A implementação do sistema LiNGS está dividida em 3 partes distintas:

- Cliente
- Servidor
- Parte comum

A divisão da implementação do sistema LiNGS em várias partes permite que apenas sejam incluídas nos jogos as partes necessárias para o funcionamento, i.e. o pacote distribuído de um jogo pode não utilizar a parte servidor, sendo que os servidores podem ser exclusivamente alojados em servidores privados, desta forma, e ainda para aumentar a organização do código, a implementação foi separada em 3 partes distintas. A parte `Cliente` tem todas as funcionalidades respetivas a clientes de jogo, a parte `Servidor` é responsável pela lógica do servidor e em situações em que é necessário o mesmo código para ambas as partes existe a `Parte Comum`. Esta parte contém código partilhado entre as partes cliente e servidor de

forma a evitar a duplicação de código e reimplementação de funcionalidades em ambas as partes.

Sendo o LiNGS implementado com base na .NET Framework, então o IDE utilizado para a sua implementação é o Microsoft Visual Studio [30]. Neste IDE o projeto LiNGS é representado por uma solução (ou *solution*) composta por 3 projetos que refletem as diversas partes da implementação, Figura 5.4:

- `LiNGSClient` – Parte cliente
- `LiNGSServer` – Parte servidor
- `LiNGSCommon` – Parte comum

O Projeto `LiNGSCommon` é referenciado nos projetos `LiNGSServer` e `LiNGSClient` para que a sua implementação esteja disponível nesses projetos.

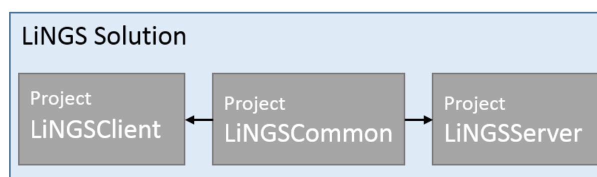


Figura 5.4: Solução LiNGS com os projetos que compõem o sistema LiNGS.

O resultante da implementação do sistema LiNGS será composto por 3 binários que correspondem às 3 partes do sistema. Estes binários, ficheiros *dll* (*Dynamic-link library*), são bibliotecas que são carregadas por outras aplicações, neste caso concreto estas bibliotecas serão carregadas pelo Unity de forma a que as suas funcionalidades fiquem disponíveis para serem utilizadas na implementação de jogos.

### 5.4.2 Arquitetura interna

O sistema LiNGS segue uma arquitetura comum utilizada em jogos digitais. Normalmente as *frameworks*/motores de jogo têm uma arquitetura composta por uma *thread* principal que é executada ciclicamente de forma a atualizar toda a árvore de objetos presentes no jogo. Paralelamente podem existir outras *threads* que atualizam outros sistemas no jogo. A arquitetura do sistema LiNGS está desenhada para executar desta forma na parte cliente como

também na parte servidor.

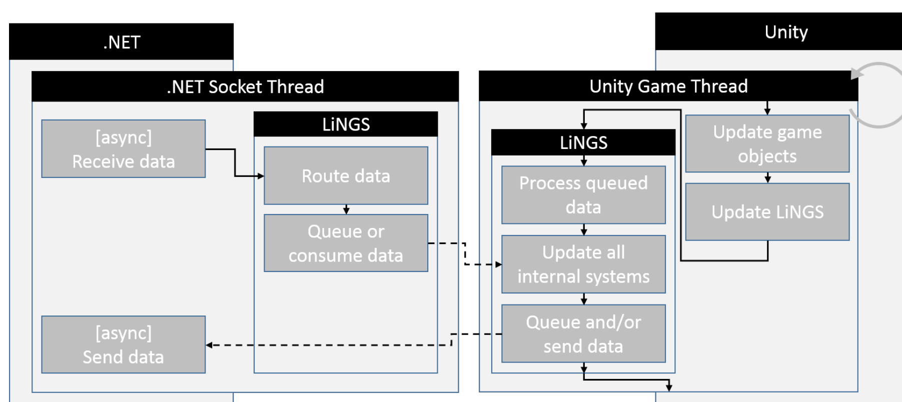


Figura 5.5: Arquitetura de funcionamento do sistema LiNGS quando integrado nas *frameworks* .NET e Unity.

Na Figura 5.5 é apresentada a arquitetura de funcionamento do LiNGS. Este sistema utiliza as capacidades assíncronas da .NET Framework para comunicação [95], nomeadamente os métodos assíncronos da classe *Socket* que permitem o envio e receção de informação para a rede de forma assíncrona e não bloqueante, característica importante na execução de jogos de forma a manter uma velocidade de atualização constante. Desta forma o sistema LiNGS está diretamente integrado na *thread* utilizada pelos sockets quando realiza comunicação. Assim, quando o sistema recebe uma comunicação é efetuada uma análise aos dados recebidos, de forma a determinar o seu tipo e respetivo destino. Estes dados são processados de imediato ou então são guardados para serem processados no próximo ciclo de processamento do sistema LiNGS, que está diretamente ligado ao ciclo de atualização do Unity.

Os restantes componentes do sistema LiNGS são atualizados de forma regular pela *thread* utilizada pelo Unity, que serve para atualizar todos os objetos de jogo. Estes componentes utilizam a informação recebida da parte de comunicação e realizam as ações necessárias para o processamento e delegação da informação recebida. Como a execução do sistema está dividido em 2 *threads* – *thread* de comunicação e *thread* de processamento – é necessário implantar mecanismos de sincronização que garantam que a troca de dados entre as duas partes funciona de forma correta sem perda de informação e dessincronização. A utilizam de mecanismos de sincronização é descriminada nas secções particulares das implementações da parte cliente e servidor do sistema.

Esta arquitetura de execução é partilhada entre a parte cliente e servidor. Desta forma a sua implementação está maioritariamente incluída no projeto `LiNGSCommon` e é referenciada nos outros projetos para ser utilizada.

### 5.4.3 Implementação Comum

A implementação comum do sistema LiNGS corresponde ao projeto `LiNGSCommon` e é composta principalmente pelo mecanismo de atualização dos componentes das partes cliente e servidor e integra também a encapsulação de mensagens do protocolo especificado no capítulo de especificação, Capítulo 4. Este projeto tem o *namespace* de *LiNGS.Common*.

#### Gestor de atualizações

A atualização dos componentes de cada parte da implementação do sistema LiNGS é feita por um gestor de atualizações. Os componentes implementam uma interface – `IUpdatable` ou `ILateUpdatable` – que os identifica como componentes atualizáveis e depois de serem registados no gestor de atualizações estes são atualizados na sua ordem correta de forma transparente.

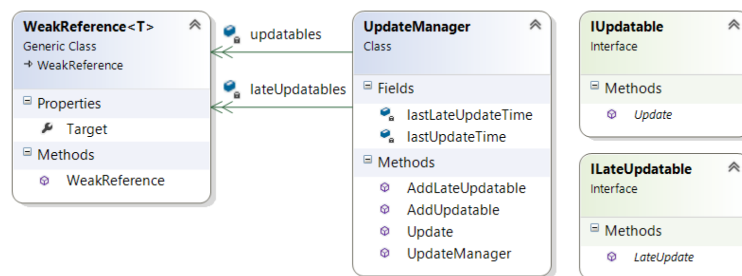


Figura 5.6: Diagrama de classes do sistema de atualizações.

O gestor de atualizações representado no diagrama de classes da Figura 5.6 com o nome `UpdateManager` contém uma lista dos componentes a atualizar. Quando o gestor de atualizações for atualizado pelo método *Update* este atualiza os vários componentes registados.

Tal como indicado previamente os componentes a atualizar têm de implementar uma das interfaces `IUpdatable` ou `ILateUpdatable` que contêm a assinatura dos métodos que vão

invocados para proceder a atualização do componente em si. No entanto, o gestor de atualizações não guarda as referências para esses componentes diretamente. É utilizado uma lista de referências fracas (`WeakReference`) que contêm as referências para os componentes concretos. A classe genérica `WeakReference` contém uma referência para um objeto específico mas esta referência não é tida em consideração para o gestor de memória automática, i.e. se o objeto não for referenciado por outros objetos então o gestor de memória liberta os recursos do objeto. Desta forma os componentes registados no gestor de atualizações não têm de apagar o seu registo quando são terminados, sendo que o gestor de atualizações consegue manter listas inteligentes dos componentes a atualizar, tal como demonstrando na listagem de código que se segue:

```
for (int i = 0; i < updatables.Count; i++)
{
        IUpdatable updatable = updatables[i].Target;
        if (updatable != null)
        {
                updatable.Update(timeSinceLastUpdate);
        }
        else
        {
                updatables.RemoveAt(i--);
        }
}
```

Quando a referência para um componente já não existir, então esse componente é retirado da lista de componentes a atualizar. Assim, o gestor de atualizações consegue operar de forma simples e útil para a restante implementação do LiNGS.

### Protocolo de comunicação

O protocolo de comunicação utilizado é uma das partes comuns entre o cliente e servidor. No projeto comum é implementado a encapsulação das mensagens que são transmitidas entre o cliente e servidor.

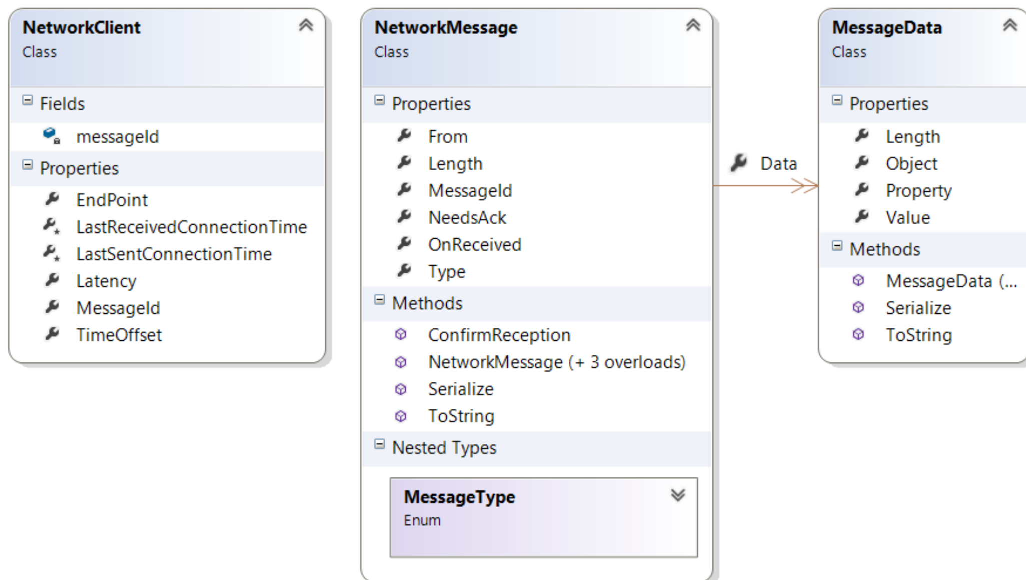


Figura 5.7: Diagrama de classes do encapsulamento de mensagens do LiNGS.

O diagrama de classes da Figura 5.7 representa as classes relativas à comunicação em rede. A classe NetworkMessage encapsula as mensagens que são transmitidas entre o cliente e servidor. Esta classe contém os atributos necessários de acordo com a especificação das mensagens de comunicação do LiNGS, Capítulo 4, tais como o identificador de mensagem, o indicador de aviso de receção e os dados enviados. Os dados que compõem a mensagem são por si encapsulados na classe `MessageData`. Ambas as classes servem de encapsulamento para as mensagens transmitidas devido à forma como a .NET Framework requer os dados para transmissão na rede. Os dados devem vir na forma de um conjunto de bytes, o que não é adequado para a utilizam na restante implementação do LiNGS, assim estas classes realizam o processo de serialização e de-serialização para que os dados das mensagens estejam apropriados para a sua utilização nas diversas partes do sistema LiNGS e até mesmo na implementação do jogo.

Ainda na parte de rede, o projeto comum contém a classe `NetworkClient`. Esta classe representa um utilizador da rede, cliente ou servidor. Nesta classe estão disponíveis propriedades para serem utilizadas pelo sistema LiNGS e pela implementação do jogo que indicam vários atributos de cada utilizador, tais como a latência de ligação, a identificação da rede do utilizador, a identificação da última mensagem recebida, a disparidade entre o tempo da máquina atual e do utilizar e os registos temporais das últimas mensagens recebidas e enviadas.

De referir que o acesso aos vários métodos e atributos das classes é particularizado de forma a permitir que a implementação do jogo apenas tenha acesso aos membros necessários. Desta forma, vários métodos e propriedades estão marcados com acesso *private*, *protected* e *internal* para limitar o acesso apenas à implementação do LiNGS.

### Restante implementação comum

A restante implementação comum inclui ainda a definição que indica quais os objetos de jogos que serão transmitidos e sincronizados pelo LiNGS e ainda a definição de marcadores utilizados internamente pelo sistema LiNGS de forma a realizar ações internas.

**Definição de objetos a sincronizar** A definição dos objetos a sincronizar pelo LiNGS é feita através da interface `INetworkedObject`. Esta interface terá de ser implementada por todos os objetos que sejam sincronizados pelo sistema LiNGS.

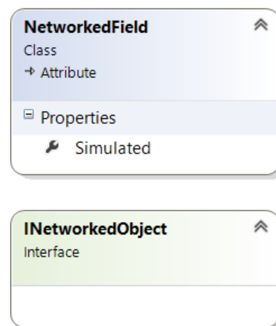


Figura 5.8: Diagrama de classes dos identificadores de objetos a sincronizar.

A Figura 5.8 apresenta o diagrama de classes dos elementos responsáveis por identificar os objetos a sincronizar. A interface `INetworkedObject` não tem membros e serve apenas para identificar os objetos a sincronizar, sendo que não existe necessidade nem o requisito dos programadores finais alterarem o seu código dos objetos para utilizar o sistema LiNGS. Neste diagrama de classes é também apresentado a classe `NetworkedField` que tem o objeto de identificar quais os campos do objeto que devem ser sincronizados. Esta classe é uma subclasse de `Attribute` que representa atributos que podem ser anexados a definições de

classes ou membros de classes. Neste caso particular o atributo `NetworkField` pode ser anexado a campos de forma a indicar que esses campos são sincronizados pelo LiNGS. Este atributo tem ainda a propriedade *Simulated* que indica se o campo deve ser simulado no cliente.

```
public class GameObject : INetworkedObject
{
        public String name;
        public int state;

        [NetworkedField]
        public int positionX;

        [NetworkedField]
        public int positionY;

        [NetworkedField(Simulated = true)]
        public int gameValue;

        ...

}
```

A listagem de código em cima demonstra um objeto de jogo que implementa a interface `INetworkedObject` o que indica que pode ser sincronizado pelo LiNGS e tem os campos a sincronizar marcados com o atributo `NetworkedField`. Os restantes campos do objeto não são sincronizados. Desta simples forma os programadores de jogos podem incluir o sistema LiNGS nos seus objetos.

**Marcadores do sistema LiNGS** Os marcadores do sistema LiNGS são pequenos campos de texto que são enviados junto às transmissões do LiNGS que indicam operações a realizar internamente pelo LiNGS.

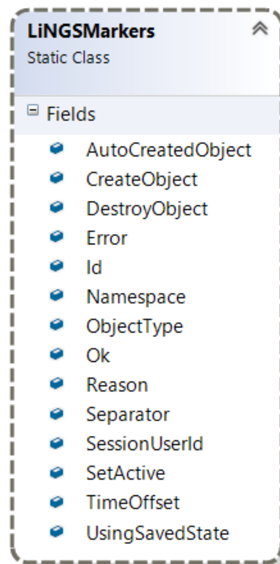


Figura 5.9: Classe com os marcadores para o sistema LiNGS.

A classe na Figura 5.9 inclui todos os marcadores utilizados internamente pelo sistema LiNGS de forma ao sistema sincronizar o seu estado interno entre servidor e clientes. Como exemplo se uma mensagem incluir dados com o marcador *CreateObject* então o sistema irá criar o objeto respetivo.

A restante implementação do sistema LiNGS é feita individualmente para a parte Cliente e Servidor de acordo com a especificação respetiva. Nas próximas secções é feita a disseminação da implementação do restante sistema.

### 5.4.4 Cliente

A parte Cliente do sistema LiNGS é a unidade que vai ficar responsável por realizar as operações de multijogador junto à implementação do jogo nos dispositivos dos jogadores. A implementação da parte Cliente corresponde ao projeto `LiNGSClient` com o *namespace LiNGS.Client*.

A arquitetura utilizada para a implementação desta parte do sistema LiNGS estabelece um núcleo central, composto pela classe `LiNGSClient`, com o propósito de centralizar, facilitar e flexibilizar o funcionamento deste sistema.

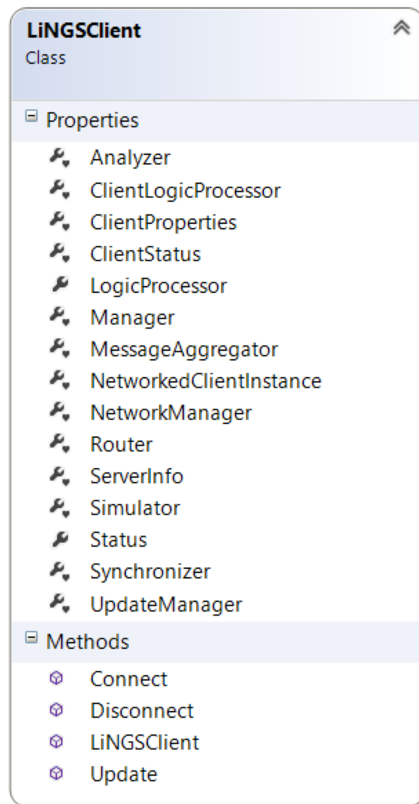


Figura 5.10: Classe central `LiNGSClient`.

A classe `LiNGSClient`, na Figura 5.10, funciona como ponto de entrada para o sistema LiNGS. Esta é a classe que o jogo irá instanciar e referenciar de forma a obter as funcionalidades deste sistema. Esta classe atua como uma *Facade* de acordo com o padrão estrutural utilizado em engenharia de software – *Facade Design Pattern* [96]. De forma a simplificar o acesso ao LiNGS esta classe apenas disponibiliza 4 métodos e 2 propriedades públicas.

A instanciação do sistema é feita pelo construtor público da classe `LiNGSClient` e recebe como parâmetro objetos do tipo `ClientProperties`, `ServerInfo` e `INetworkedClient`, indicados na próxima figura.

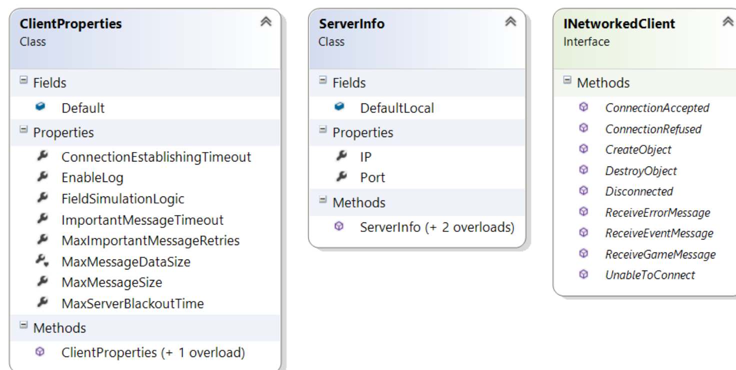


Figura 5.11: Tipos utilizados como parâmetro do construtor da parte cliente do sistema LiNGS.

As classes representadas na Figura 5.11 são os parâmetros necessários para a instanciação do cliente do sistema LiNGS. A classe `ClientProperties` representa as configurações que serão utilizadas pelo sistema para o seu funcionamento interno. Os valores dos elementos desta classe são totalmente personalizáveis, sendo definido um conjunto de propriedades por defeito. A classe `ServerInfo` representa as informações do servidor ao qual será feita a ligação. Esta classe contém o endereço IP do servidor e o porto onde o mesmo está registado. Ainda na inicialização do sistema LiNGS é ainda necessário especificar uma instância da interface `INetworkedClient`.

A instância de um objeto que implemente a interface `INetworkedClient` funciona como uma *Facade*, abstraindo a implementação do jogo, tal como representado na próxima figura.

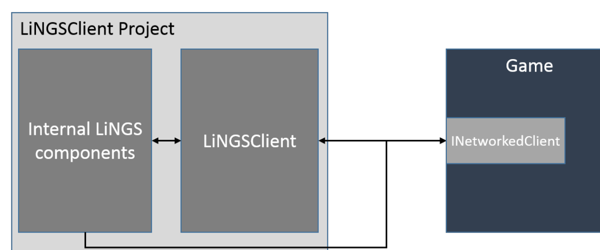


Figura 5.12: Interação do sistema LiNGS com o jogo.

A classe `INetworkedGame`, Figura 5.11, contém vários métodos necessários para o funcionamento do jogo. Estes são invocados quando o sistema LiNGS requer uma ação do jogo referente a algum acontecimento no sistema de multijogador.

A classe `LiNGSClient` serve também como um núcleo interno do sistema LiNGS. A representação desta classe na Figura 5.10 mostra que todos os sistema internos do LiNGS estão em si referenciados, com tipo de acesso *internal*. Estes sistemas têm, por sua vez, uma referência para o objeto do tipo `LiNGSClient`. Assim, a interação entre sistemas passa sempre pela classe `LiNGSClient`, sendo que as referencias dos sistemas apenas se encontram nesta classe, tal como representado na próxima figura.

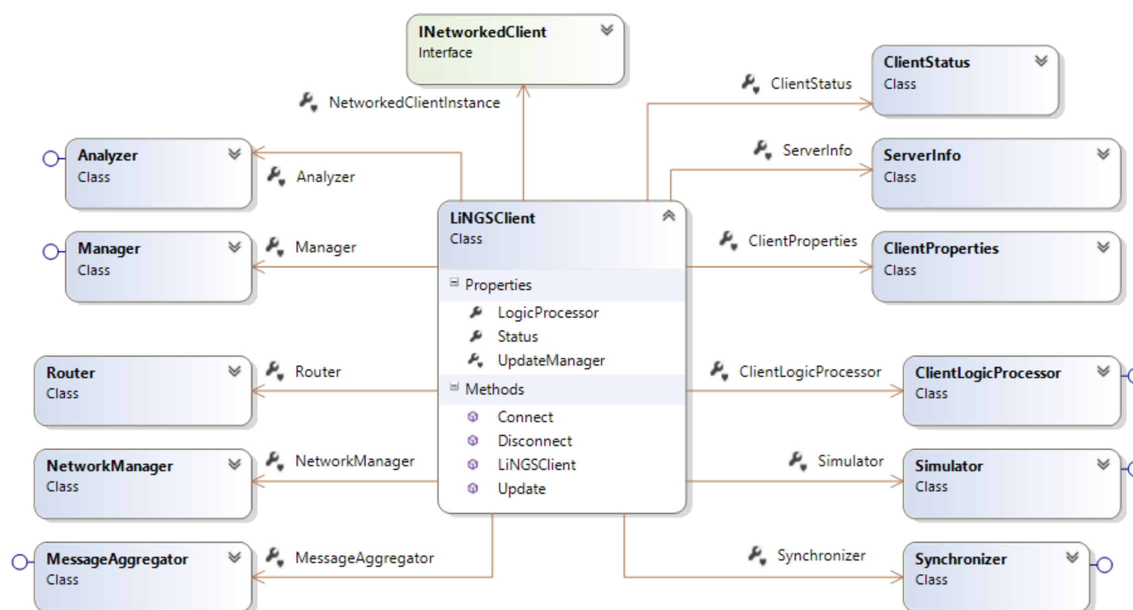


Figura 5.13: Sistema de referências centralizado - `LiNGSClient`.

A Figura 5.13 representa um diagrama de classes simplificado e reduzido do projeto `LiNGSClient` onde está representado a ligação entre os componentes. Estes apenas têm a referência para o objeto `LiNGSClient` em comum e utilizam esse objeto para comunicar entre si, internamente no sistema LiNGS.

A restante implementação da parte cliente do sistema LiNGS está dividida em secções de acordo com a sua funcionalidade.

### Comunicação em rede

No sistema LiNGS a componente de comunicação em rede é um dos aspetos principais. Esta funcionalidade é crucial para o bom funcionamento dos restantes componentes do sistema.

Esta divisão do LiNGS fornece os mecanismos para comunicação em rede realizando a ponte entre o sistema e os protocolos de comunicação utilizados na .NET Framework.

A parte comunicação em rede é composta por 5 classes apresentadas no seguinte diagrama de classes:

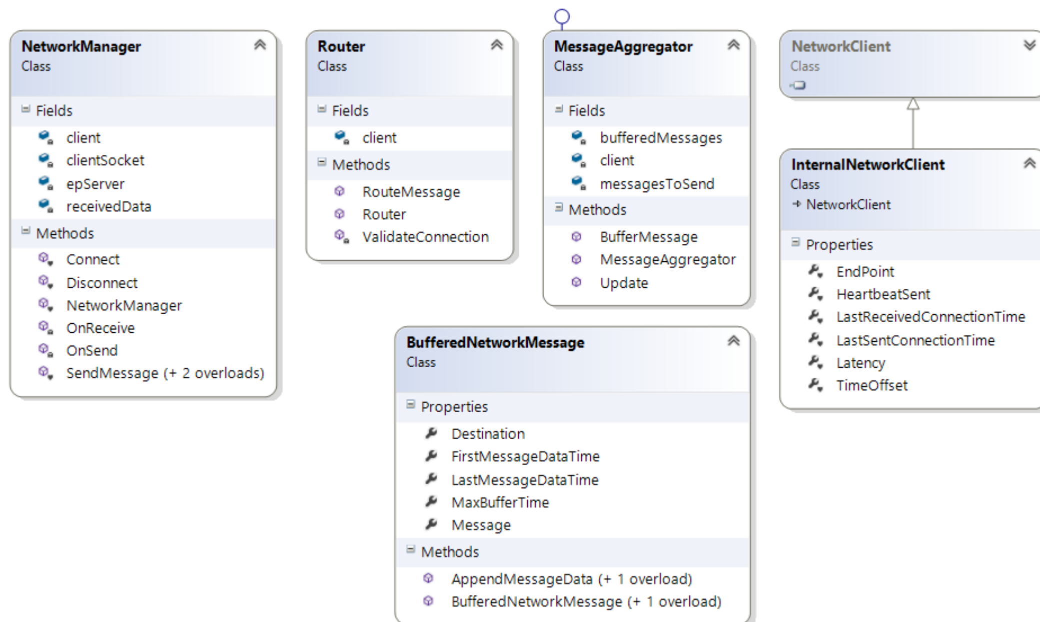


Figura 5.14: Diagrama de classes da parte de comunicação do sistema LiNGS - parte cliente.

O diagrama da Figura 5.14 mostra as várias classes que tratam os aspetos de comunicação em rede. A comunicação em rede está dividida, tal como o restante sistema, em pequenos pacotes de forma a delegar responsabilidades para os objetos respetivos com o objetivo de manter a implementação dos componentes organizada, simples e focada na sua tarefa.

Analisando a figura podemos rapidamente notar a implementação da classe denominada `InternalNetworkClient`, derivada da classe `NetworkClient` implementada no projeto `LiNGSCommon`. A implementação desta classe foi feita de forma a limitar o acesso público das suas propriedades. Como a implementação da classe `NetworkClient` foi feita no projeto `LiNGSCommon` então os vários tipos de acesso das propriedades tem de ser públicos para que a classe possa ser utilizada fora do projeto, no entanto, com o objetivo de bloquear alterações feitas por código externo ao sistema LiNGS foi definido tipos de acesso públicos para leitura – *public* – e para escrita foram definidos acessos protegido – *protec-*

*ted*. Os tipos protegidos autorizam apenas o acesso a classes derivadas, desta forma a classe `InternalNetworkClient` sendo derivada da classe `NetworkClient` pode aceder às propriedades desta para leitura e escrita. Assim, várias propriedades apenas disponíveis dentro do LiNGS para funcionamento interno e outras estão disponíveis de forma pública apenas para leitura.

Continuando a análise do diagrama da Figura 5.14, a classe `NetworkManager` é o ponto de acesso do sistema para a rede. Esta classe tem a responsabilidade de gerir o socket, tipo UDP (*User Datagram Protocol*), para comunicação com o servidor e realizar as operações de receção e transmissão de dados. As mensagens transmitidas são serializadas e de-serializadas pela classe `NetworkMessage` do projeto `LiNGSCommon` e toda a utilização do socket para transmissão é feita assincronamente de forma não bloqueante. Ao receber informação, o `NetworkManager` encaminha-a para o objeto do tipo `Router`. Este objeto determina o tipo das mensagens recebidas e delega-as para os componentes respetivos de forma a serem processadas.

O `MessageAggregator` atua na fase de envio de mensagens e serve para agregar dados de mensagens de forma a juntá-los numa única mensagem e assim reduzir o número de comunicações realizadas. Este componente admite as mensagens a enviar e encapsula-as na classe `BufferedNetworkMessage`. Este encapsulamento permite associar várias informações aos dados da mensagem: informações do destinatário, tempo máximo de espera e tempo de espera. Quando o tempo de espera for maior do que o tempo máximo então é composta e enviada uma mensagem para o destinatário com os dados em espera. Desta forma a implementação realiza os procedimentos especificados no Capítulo 4 para reduzir o número de comunicações e diminuir o consumo energético.

### Gestão interna

A parte de gestão interna da implementação do LiNGSCLient é responsável por garantir as condições de funcionamento do sistema LiNGS.

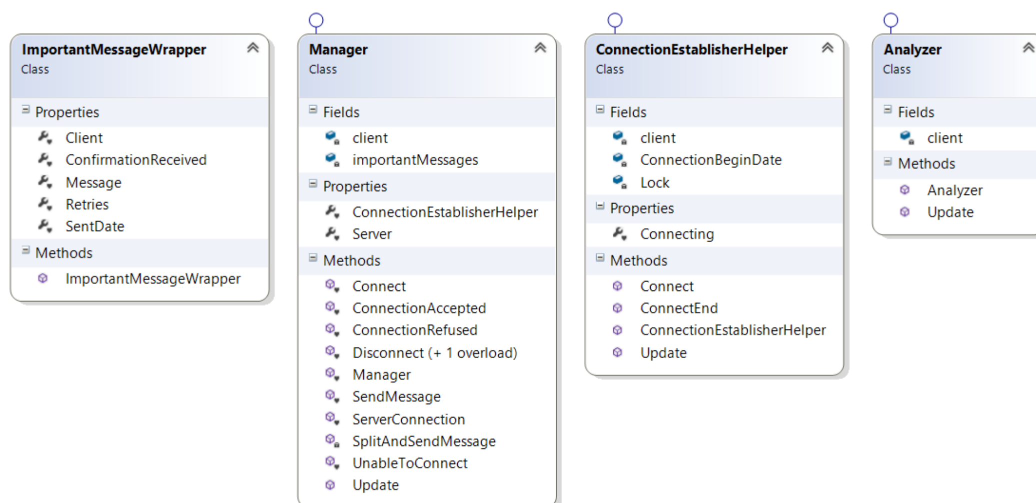


Figura 5.15: Diagrama de classes da parte de gestão interna do sistema LiNGS - parte cliente.

A parte de gestão interna do sistema LiNGS é composta pelas classes apresentadas no diagrama de classes da Figura 5.15. A principal classe deste grupo é a `Manager` onde é feita a maioria da gestão do sistema. Esta classe é responsável por várias ações mas o seu papel principal é ser o núcleo de transmissão de dados do sistema. Qualquer comunicação que passe pelo sistema tem obrigatoriamente de passar também pelo `Manager`. Esta classe, com o auxílio da `ConnectionEstablisherHelper`, negocia a ligação com o servidor e de acordo com o resultado atingido notifica o restante sistema. Várias comunicações do servidor são aqui processadas. Sempre que existe uma comunicação do servidor que requer confirmação de resposta é nesta classe que a resposta é gerada e enviada. A notificação de término da ligação é também processada por esta classe encarregando-se de notificar o restante sistema e de libertar os recursos da ligação. A transmissão de dados também é realizada através da classe `Manager`. Quando o sistema tenciona enviar dados para o servidor então encaminha esse dados para esta classe que de acordo com a importância destes, nomeadamente se a mensagem a enviar requer confirmação ou não, procede com o envio dos dados ou então guarda-os na `MessageAggregator` para posterior envio. Ainda nesta classe é feito a gestão das receções de confirmações de mensagens importantes. Caso uma mensagem importante tenha sido enviada e a confirmação desta não tenha chegado então a mensagem é reenviada apropriadamente.

Um dos problemas na utilização de sockets assíncronos é a sincronização de dados. Tal como referido na secção 5.4.2, da implementação do sistema LiNGS, a utilização da *thread*

de socket e da *thread* de atualização do Unity leva a que os mesmos dados possam ser acedidos simultaneamente. De forma a evitar problemas de sincronização os dados recebidos no `Manager` pela *thread* de sockets são colocados numa lista temporária de forma síncrona e são depois processados no próximo ciclo de atualização pela *thread* do Unity. O acesso à lista temporária é feito dentro de uma declaração síncrona utilizando a expressão *lock* do C# que garante que apenas uma *thread* consegue aceder ao objeto bloqueado de forma simultânea.

Ainda neste grupo de funcionalidades, a classe `Analyzer` realiza um estudo da transmissão de dados e garante que a comunicação entre o cliente e servidor se encontra em pleno funcionamento. Esta classe deteta quando a ligação é interrompida ou quando o servidor deixa de comunicar com o cliente e alerta o restante sistema do problema ocorrido.

**Processamento de dados**

O grupo de classes responsável pelo processamento de dados no `LiNGSClient` é composto por 5 classes que realizam a tarefa de garantir que os dados recebidos do servidor são aplicados de forma correta e certifica também que os dados a enviar são transmitidos para o servidor quando tal é necessário.

Na Figura 5.16 é apresentado o diagrama de classes que corresponde ao processamento de dados da parte cliente do sistema LiNGS. A classe `ClientLogicProcessor` é a classe que realiza a gestão dos objetos que são sincronizados do servidor para o cliente. Quando um objeto deve ser sincronizado o servidor envia uma mensagem para os clientes indicando a necessidade da criação do objeto. Quando esta classe recebe essa mensagem então é invocado o método da implementação do jogo da interface `INetworkedClient` para a criação do objeto. Quando os objetos são criados estes são encapsulados na classe `NetworkedObject` que deteta quais os campos do objeto que são sincronizados, i.e. que tenha o atributo `NetworkedField`, esta classe verifica ainda se o objeto tem o método para se desativar/ativar.

Os objetos sincronizados podem ser desativados quando não estão a ser sincronizados de momento. De forma a notificar o objeto da sua desativação é necessário que o objeto implemente um método com a seguinte assinatura:

```
public void LiNGSSetActive(bool active){ }
```

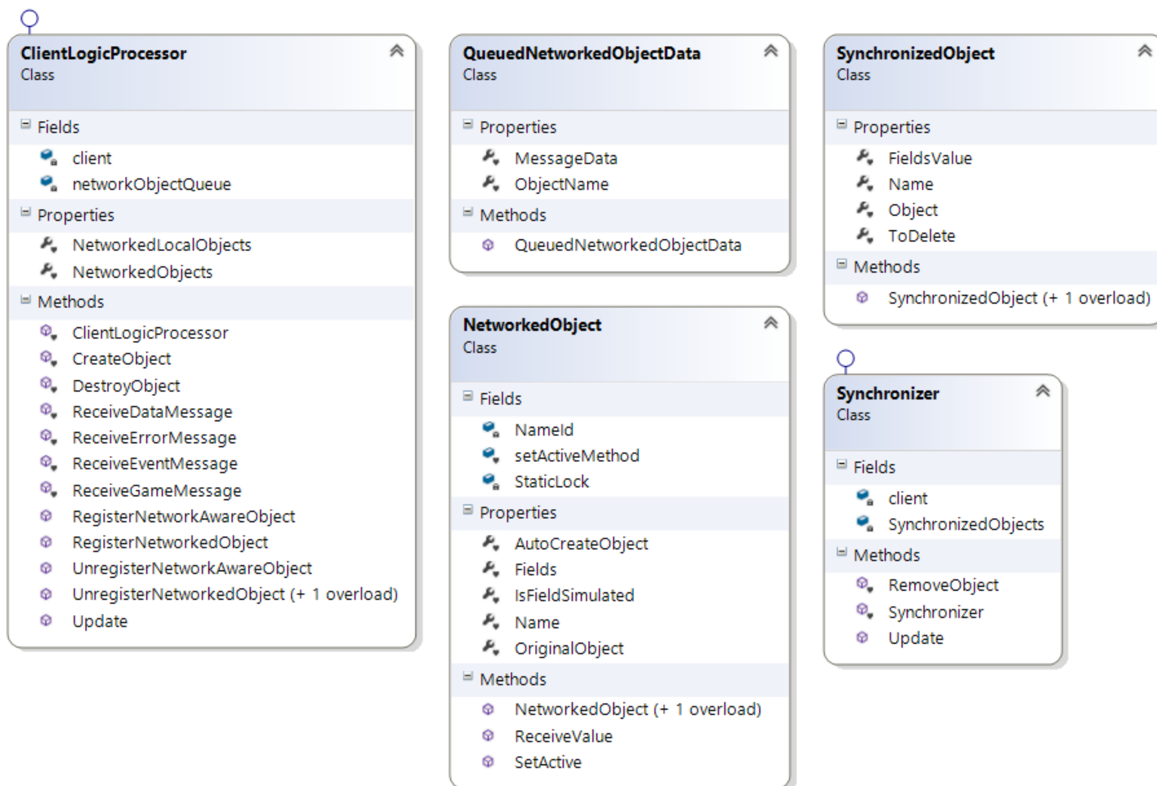


Figura 5.16: Diagrama de classes do grupo de processamento de dados do sistema LiNGS - parte cliente.

Este método é invocado quando o estado de sincronização é alterado de acordo com a sincronização contextual. A implementação deste método não está especificada na interface `INetworkedObject`, que os objetos são obrigados a implementar, e, desta forma, a sua implementação não é obrigatória. Esta decisão deve-se aos objetos poderem não ter um conceito de ativo/não ativo e desta forma não é obrigado que os objetos implementem um método sem lógica. A documentação incluída com o sistema LiNGS indica esta particularidade.

Refletindo as soluções definidas na especificação, a transmissão de dados no sistema LiNGS é feita de forma compacta. A identificação dos objetos automaticamente criados pelo sistema LiNGS é composta por um marcador a identificar o tipo de objeto e por apenas um número, exemplo: *0L:o:1*. A identificação dos campos do objeto é também feita de forma similar. Em vez de identificar os campos pelos seus nomes, o que pode ser bastante dispendioso em termos de tráfego de rede, é feito uma ordenação da lista de campos pelos seus nomes e a

identificação de cada um é baseada no seu índice na lista ordenada. Assim a identificação de um campo é apenas um número inteiro.

Quando as modificações de objetos são recebidas no sistema LiNGS então os novos valores são aplicados diretamente ao objeto via *reflection* [93]. Desta forma, a introdução dos novos valores é transparente para a implementação do jogo e não requer a existência modificações. No entanto existem cuidados com a sincronização dos dados. Tal como na parte de gestão interna, a classe `ClientLogicProcessor` também tem possíveis condições de dessincronização. Mais uma vez a mesma estratégia é aplicada, guardando os dados temporariamente e esperar pelo próximo ciclo de atualização para os processar. Assim é garantido que os dados não estão a ser acedidos por outra *thread* ao mesmo tempo.

Este componente permite assim a sincronização de objetos do servidor para o cliente. A realização da sincronização de objetos no sentido oposto, cliente-servidor, é realizada pela classe `Synchronizer`. Os objetos do cliente são registados na classe `ClientLogicProcessor` e são incluídos na classe `Synchronizer` para serem sincronizados com o servidor. Esta classe guarda uma lista com os objetos a sincronizar, do tipo `SynchronizedObject`, e a cada atualização verifica se existiram mudanças nos valores dos campos a sincronizar dos objetos. Caso existam alterações então apenas as modificações são sincronizadas com o servidor eliminando a necessidade de retransmissão total do objeto.

**Simulação**

A parte de simulação do cliente LiNGS é composta por algoritmos de previsão de estado que servem para reduzir os efeitos provocados pela latência das redes de comunicação e para permitir aumentar o intervalo de tempo entre transmissões. Os problemas de rede como por exemplo, elevada latência na ligação, podem introduzir problemas no jogo, de ordem visual e mesmo lógica, assim a simulação tem um papel importante na implementação de jogo multijogador em rede. Esta componente tem o propósito de esconder esses problemas dos utilizadores.

Este grupo é composto pelas classes apresentadas na seguinte figura:

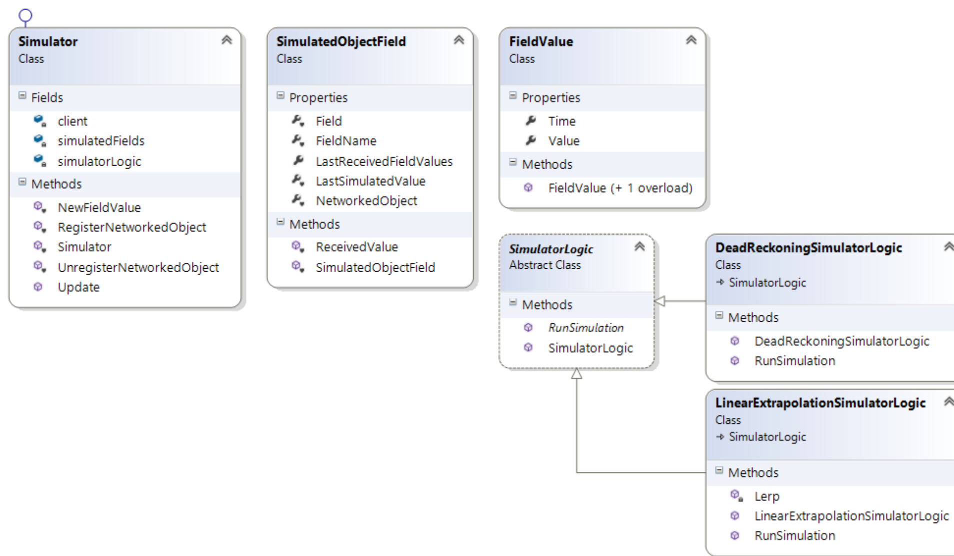


Figura 5.17: Diagrama de classes do grupo de simulação no sistema LiNGS - parte cliente.

A Figura 5.17 apresenta o diagrama com as diversas classes do grupo de simulação. A classe `Simulator` é a classe que controla a tarefa de simulação. Esta classe mantém um registo de todos os campos de objetos sincronizados que sejam simulados, i.e. campos com o atributo `NetworkedField` com a propriedade *Simulated* ativa. Os campos são encapsulados nas classes `SimulatedObjectField` e `FieldValue`. Estas classes mantêm um histórico dos valores que o campo tomou ao longo de um período de tempo, este histórico é utilizado para a realização da simulação pelas classes derivadas de `SimulatorLogic`. Como o sistema LiNGS é um sistema genérico para ser utilizado em diversos tipos de jogos a implementação da lógica de simulação pode depender bastante do tipo de jogo, alguns jogos podem nem utilizar simulação, assim é possível implementar algoritmos personalizados e utiliza-los no LiNGS sem necessidade de alterações ao código do sistema. Basta derivar a classe `SimulatorLogic` e especificar a instância da classe derivada na especificação das propriedades do cliente, classe `ClientProperties`.

A cada atualização do sistema a classe `Simulator` atualiza cada campo executando a lógica de simulação individualmente para cada um. Na implementação do sistema LiNGS vem incluído dois algoritmos de simulação de valores: *Dead Reckoning* [78] e *Linear Extrapolation* [97].

Os vários grupos apresentados compõem assim a implementação cliente do sistema LiNGS. Estes grupos trabalham de acordo com a especificação do LiNGS, de forma simples e organizada. Na próxima secção será feita uma descrição da implementação da parte Servidor que completa o sistema LiNGS.

### 5.4.5 Servidor

A parte Servidor do sistema LiNGS realiza as tarefas de sincronização e gestão de todos os dados do jogo multijogador. Esta parte do sistema LiNGS está acoplada à implementação lógica do jogo de forma a fornecer os mecanismos de multijogador. O projeto que compõe a implementação do servidor é o `LiNGSServer` com o *namespace LiNGS.Server*. A arquitetura e implementação do Servidor são idênticas ao realizado na parte Cliente do sistema LiNGS. A classe de entrada da parte Servidor é a `LiNGSServer`.

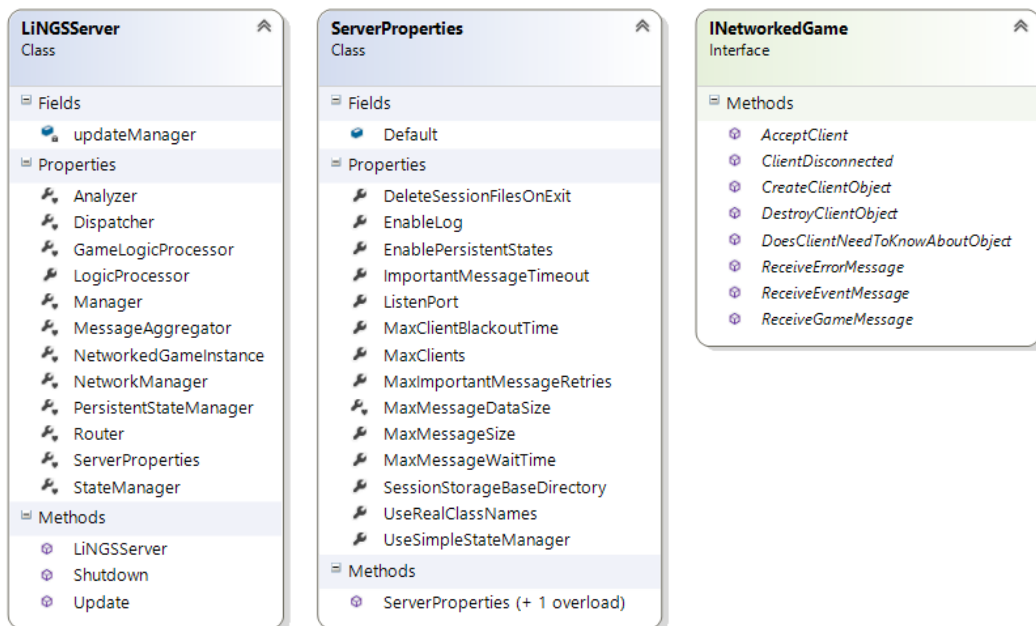


Figura 5.18: Diagrama de classes do ponto de entrada da implementação do sistema LiNGS - parte servidor.

A Figura 5.18 apresenta o diagrama de classes com os tipos a instanciar de forma criar a parte de servidor do sistema LiNGS. A classe `LiNGSServer` é o ponto de entrada para o sistema e, tal como acontece no `LiNGSClient`, esta é a classe que realiza a tarefa de *Facade* ou ponto de

comunicação com os restantes componentes do sistema. Esta classe recebe como parâmetros um objeto do tipo `ServerProperties` e `INetworkedGame`. A classe `ServerProperties` contém todas as propriedades e configurações do sistema LiNGS, estas propriedades alteram o funcionamento dos componentes internos do LiNGS de acordo com os seus valores. Como o LiNGS é um sistema genérico, feito para ser utilizado em qualquer tipo de jogo, existe uma quantidade generosa de configuração do sistema. Vários jogos têm necessidades específicas que podem obrigar a diferentes comportamentos, assim, este sistema foi desenhado para que possa ser facilmente configurado pelos programadores dos jogos.

Ainda na inicialização da classe `LiNGSServer` é requerido um objeto que implemente a interface `INetworkedGame`. Esta interface específica os métodos que a implementação do jogo deve suportar de forma a garantir as funcionalidades do sistema LiNGS. Esta interface contém a assinatura de vários métodos, no entanto, dois métodos são particularmente interessantes. O método *AcceptClient* é invocado quando um cliente se tenta ligar ao servidor e quando o servidor tem as condições necessárias para o receber, ou seja, o número máximo de utilizadores não foi atingido. O retorno deste método é um objeto do tipo `ClientConnectionResponse` onde é indicado se o utilizador é aceite no servidor e se é pretendido utilizar uma sessão prévia desse utilizador. A implementação do sistema LiNGS suporta a suspensão e restauro de sessões de utilizadores e é neste passo que a implementação do jogo indica se deve ou não ser utilizado uma sessão guardada.

Outro método importante na interface `INetworkedGame` é o *DoesClientNeedToKnowAboutObject*. Este método permite que a implementação do jogo informe o sistema LiNGS para que clientes cada objeto de jogo deve ser sincronizado. Através de informações dos objetos de jogo, por exemplo, a posição, a implementação do jogo pode indicar ao sistema LiNGS que certos objetos não devem ser sincronizados para certos clientes naquele momento do jogo. É, desta forma, realizada um sincronização contextual. Esta funcionalidade tem como objetivo reduzir a quantidade de informação trocada entre o servidor e cliente.

De forma semelhante à implementação da parte cliente do sistema LiNGS, a implementação do servidor especifica uma arquitetura centralizada, desta vez na classe `LiNGSServer`.

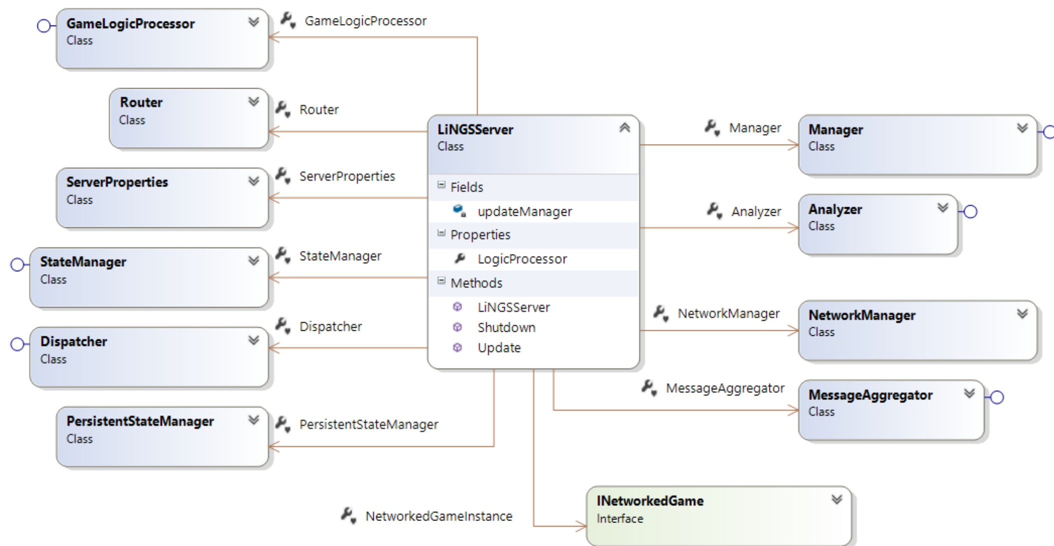


Figura 5.19: Diagrama de classes da arquitetura da implementação do sistema LiNGS - parte servidor.

A Figura 5.19 mostra o diagrama de classes que representa a centralização do sistema na classe `LiNGSServer`. Com o objetivo de manter a implementação organizada e coerente os diversos componentes desta parte do sistema são semelhantes aos componentes da parte de cliente.

Tal como indicado, a classe `LiNGSServer` é o núcleo da parte servidor do sistema LiNGS e esta fornece o ponto de entrada principal para acesso a todas as funcionalidades do LiNGS. A classe `LiNGSServer` realiza ainda a instanciação e inicialização dos vários componentes que compõe esta parte do sistema LiNGS. A decomposição e análise destes será feita de seguida de forma agrupada por funcionalidade.

### Comunicação em rede

A implementação da comunicação em rede do servidor do sistema LiNGS é bastante semelhante à implementação da parte cliente. As classes são idênticas e realizam ações semelhantes mas de forma adequada ao servidor.

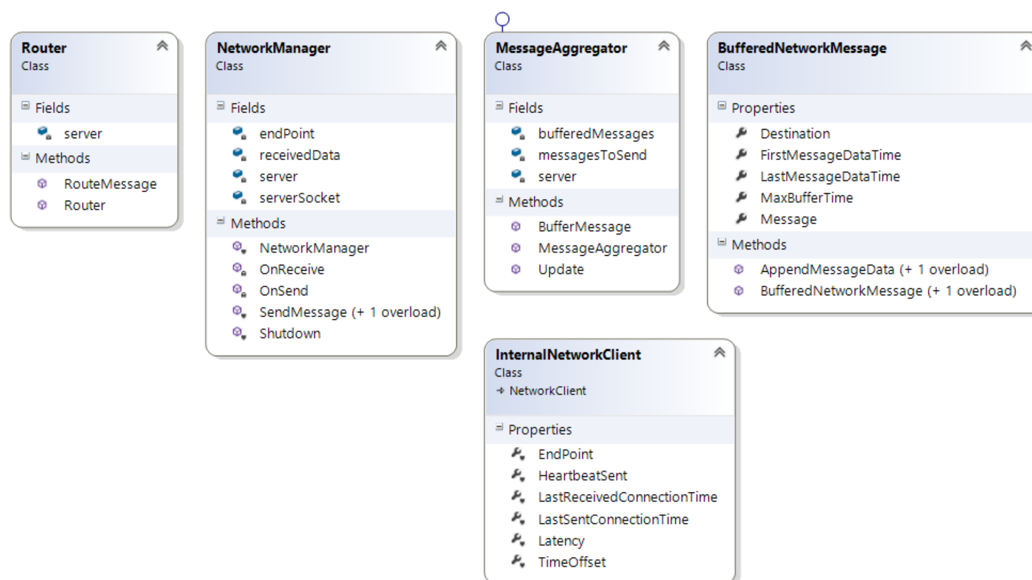


Figura 5.20: Diagrama de classes da parte de comunicação do sistema LiNGS - parte servidor.

A Figura 5.20 representa o diagrama de classes do grupo de comunicação em rede. Comparando esta figura com a Figura 5.14 é possível concluir que a parte de comunicação de ambas as partes (cliente e servidor) do sistema é bastante similar. As mudanças significativas das classes do servidor cingem-se principalmente à adequação para utilização da classe `LiNGSServer` e à implementação de vários métodos apenas específicos à parte servidor. A classe `NetworkManager` realiza a mesma operação que a sua classe relativa, na parte cliente do sistema, mas está adaptada para as necessidades do servidor, suportando comunicação com vários clientes.

A semelhança entre as duas partes do sistema garante que a especificação e implementação do LiNGS é concisa e que permite a introdução fácil de novas funcionalidades, sem a necessidade de refazer as partes do sistema.

### Gestão interna e de clientes

A gestão da parte de servidor do sistema LiNGS é, ao contrário da comunicação em rede, bastante diferente da implementação na parte de cliente. Neste caso existe a necessidade de comunicar com vários clientes e é necessário gerir as transmissões entre o servidor e cada

um destes clientes.

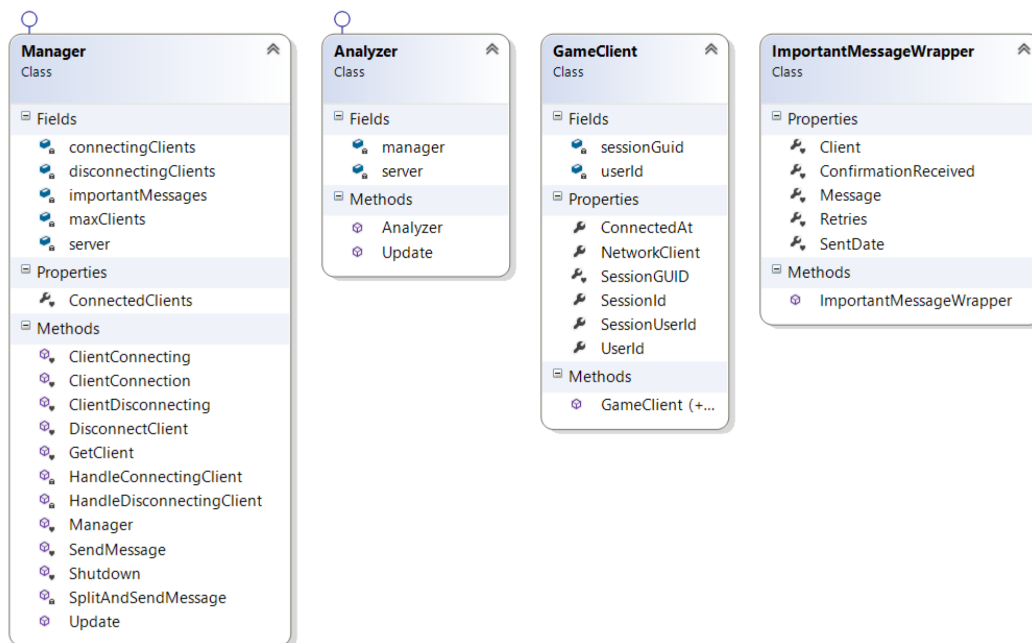


Figura 5.21: Diagrama de classes do grupo de gestão da parte de servidor do sistema LiNGS.

A Figura 5.21 apresenta o diagrama de classes deste grupo de funcionalidades. A classe `GameClient` encapsula os detalhes de cada cliente de forma a os identificar perante o jogo. Cada sessão de jogo é identificada com um GUID (*Globally Unique Identifier*) e cada cliente ligado é identificado com um número sequencial. Desta forma é possível identificar um cliente no jogo. Esta identificação é ainda encaminhada para cada cliente e é utilizada quando o cliente retoma a sessão depois de uma quebra de ligação. Assim, a classe `GameClient` representa um cliente no jogo.

A classe `Manager` é responsável por gerir todos os clientes ligados, processando novas ligações e avisos de desconexão dos vários utilizadores ligados. Quando existe uma nova ligação de um cliente são feitos os procedimentos de início de ligação com base nos dados recebidos pelo cliente e de acordo com o estado atual do servidor e do jogo. Quando um cliente é aceite este entra para a lista de clientes ligados e a sincronização de objetos de jogo é iniciada com o cliente. No caso de restauro de uma ligação de um cliente, depois de uma falha de rede, a sua sessão anterior é utilizada e é feita uma análise da sessão do cliente e a sincronização dos objetos de jogo começa, sendo apenas sincronizado as modificações ocorridas durante

o período em que o cliente não esteve ligado. Quando um cliente se pretende desligar do servidor, este envia uma mensagem que é analisada pela classe `Manager`, que por sua vez aplica as ações necessárias para retirar o cliente do processo de sincronização de dados e avisa o jogo da desconexão do cliente.

Esta classe fornece ainda a gestão do envio de mensagens importantes, registando os envios e as confirmações de receção e ainda, caso seja necessário, volta a reenviar mensagens que não recebidas. Os problemas de sincronização devido a *multithreading*, na classe `Manager`, da parte de cliente do sistema LiNGS são também aqui encontrados e são resolvidos de forma semelhante.

A classe `Analyzer` analisa o comportamento das ligações entre o servidor e todos os clientes ligados de forma a determinar problemas de comunicação. Esta classe verifica os períodos de comunicação entre os clientes e, quando um cliente está algum tempo sem comunicar com servidor é enviada uma mensagem importante para o cliente de forma a obter uma resposta, esta mensagem é do tipo `Heartbeat`. Se o cliente responder então a ligação mantêm-se, caso contrário o servidor pode terminar a ligação visto que a comunicação com o cliente não é possível. Desta forma, é garantido que a ligação com os clientes permanece ativa e em funcionamento e evita que clientes que se tenham desligado sem avisar o servidor, por exemplo, por perda de ligação, não permaneçam no jogo.

#### Processamento de dados e gestão de estados

O processamento de dados e gestão de estados é um dos grupos mais importantes do sistema LiNGS. Esta parte do sistema realiza as operações necessárias para observar os objetos de jogo e realizar a sincronização dos valores de campos destes para os clientes respetivos. O envio de mensagens personalizadas entre o servidor e clientes é também aqui processado.

Este grupo representa o ponto de entrada do sistema LiNGS para o registo de objetos e a realização da sincronização destes. Devido à complexidade desta operação existem vários componentes no processo de sincronização que têm o objetivo de garantir eficácia e fiabilidade nesta operação. Os vários componentes integrantes desta parte do sistema são analisados de seguida.

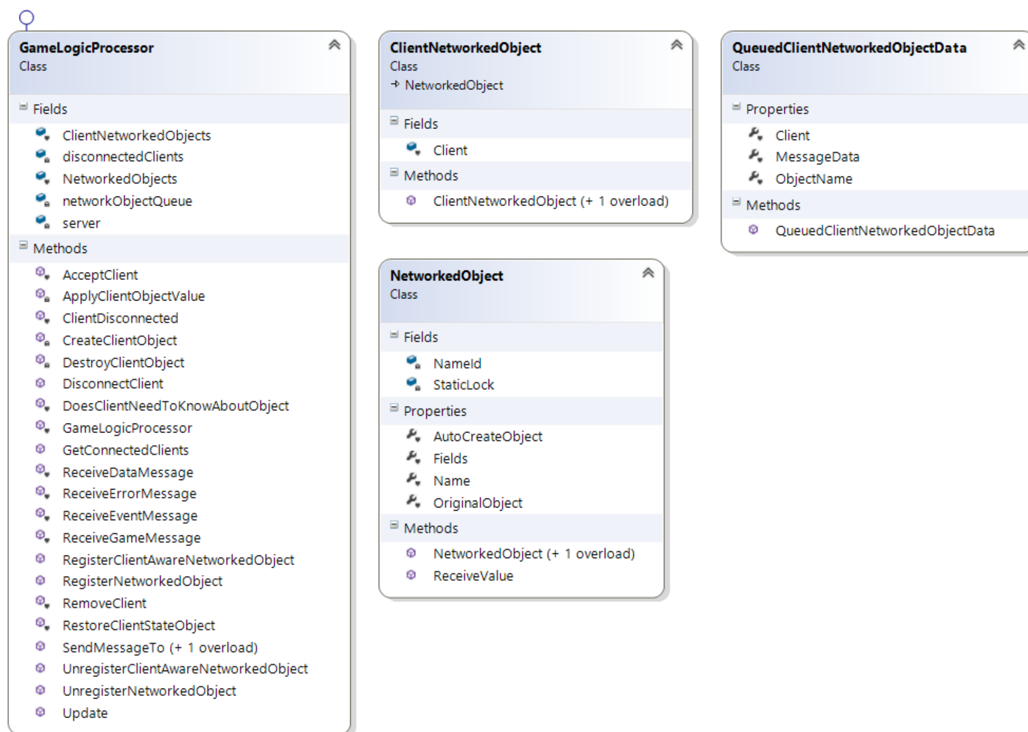


Figura 5.22: Diagrama de classes de processamento de dados do sistema LiNGS - parte servidor.

A Figura 5.22 apresenta o diagrama de classes de processamento do sistema LiNGS. Este grupo é constituído pela classe `GameLogicProcessor` e por outras classes auxiliares que realizam encapsulamento de dados. A classe `GameLogicProcessor` é o ponto do sistema LiNGS onde a implementação do jogo, na parte do servidor, regista os objetos de jogo para serem sincronizados pelos vários clientes. Esta classe mantém uma lista dos objetos a sincronizar, tanto da parte do servidor como objetos criados pelos clientes, e atualiza os valores dos campos desses objetos quando são recebidos novos dados. De forma a agilizar a sincronização de dados esta classe permite o registo de objetos novos que serão automaticamente criados nos clientes e permite também registar objetos que já são conhecidos e já existem nos clientes. A mesma funcionalidade é fornecida para destruir os objetos, sendo possível sincronizar a destruição de objetos pelos clientes ou apenas destruir os objetos localmente.

A classe `GameLogicProcessor` permite ainda o envio e receção de mensagens personalizadas entre o servidor e clientes. A receção de dados está ainda implementada de forma a resolver problemas de concorrência provindos da utilização da *thread* de *sockets* para comu-

nicação e da *thread* do Unity para atualização dos componentes. O método de agendar os dados recebidos para serem processados na próxima atualização é, tal como na parte cliente, utilizado.

A classe `NetworkedObject` é semelhante à classe com o mesmo nome na implementação do cliente e tem um comportamento semelhante fornecendo acesso ao objeto de jogos e aos seus respetivos campos a sincronizar, marcados com o atributo `INetworkedField`.

De forma a realizar a sincronização dos dados é necessário recorrer a outros componentes na implementação.

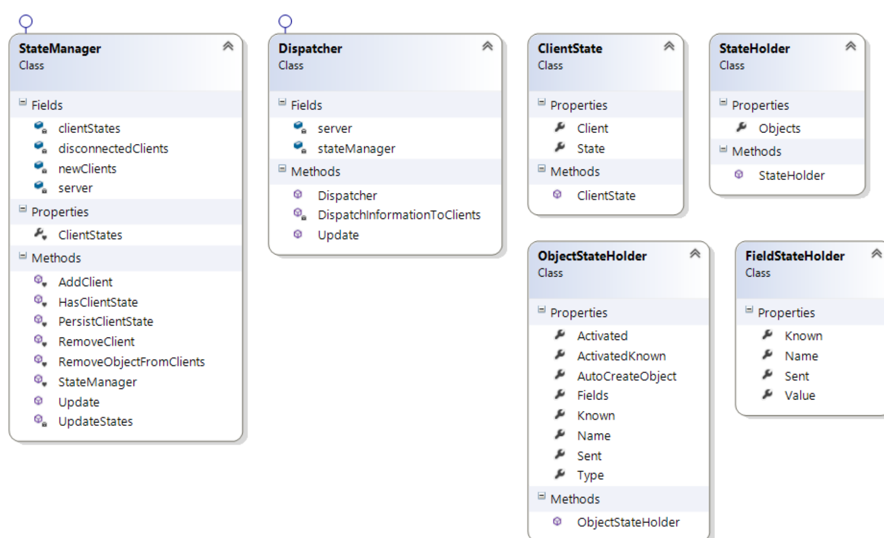


Figura 5.23: Diagrama de classes da parte de sincronização do sistema LiNGS - parte servidor.

As classes responsáveis por manter a sincronização dos vários objetos de jogo estão representadas no diagrama de classes da Figura 5.23. A classe `StateManager` tem a tarefa de manter um estado local de cada cliente e assim permitir que a cada atualização apenas sejam sincronizados os dados que sofreram alterações. Esta sincronização é feita pelo `Dispatcher` que verifica se o estado dos jogadores sofreu alguma alteração e desse ser sincronizado. De forma a manter o estado de cada cliente é necessário manter o estado de sincronização de cada campo de cada objeto, esta operação é feita por dicionários compostos por árvores de classes `ClientState`, `StateHolder`, `ObjectStateHolder` e `FieldStatHolder`. Assim

cada cliente é associado ao seu estado e é possível saber a situação de sincronização de cada objeto de jogo para cada cliente.

A sincronização de objetos é ainda realizada apenas para os clientes necessários. Se um objeto de jogo não for essencial para um cliente então esse objeto não deve ser sincronizado. Desta forma, a quantidade de tráfego e processamento é reduzida.

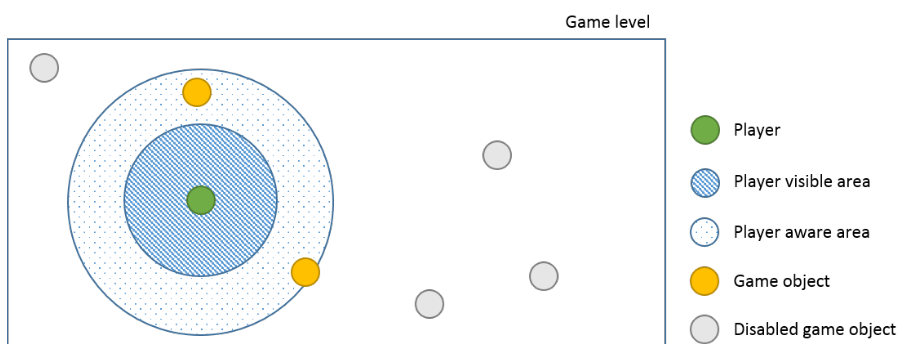


Figura 5.24: Exemplo de sincronização de objetos baseada na distância do cliente.

Um exemplo deste tipo de sincronização parcial é mostrado na Figura 5.24. Apenas são sincronizados os objetos que estejam perto da área de visão do cliente. Os restantes objetos do jogo não são sincronizados até que a sua interação com o cliente posso acontecer. Esta é uma operação de otimização que diminui o tráfego a circular na rede e a quantidade de processamento nos clientes. Esta técnica de otimização é controlada pela implementação do jogo, aumentando assim as potencialidades de otimizações na sincronização e permitindo a existência de objetos sempre sincronizados.

A implementação do sistema LiNGS consegue assim manter sincronização do jogo entre o servidor e os vários clientes.

### Persistência de estados

A perda de ligação é um problema comum nos jogos em rede em dispositivos móveis. Com o objetivo de suprimir os problemas causados por este inconveniente o sistema LiNGS implementa a funcionalidade de gestão persistente de sessões. Esta funcionalidade permite guardar e restaurar a sessão de jogadores.

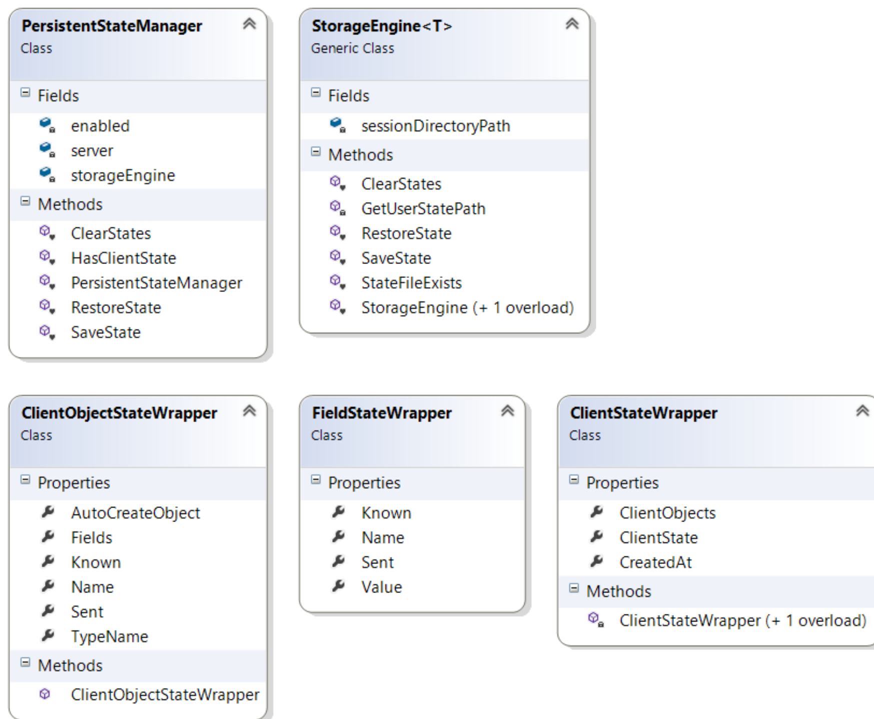


Figura 5.25: Diagrama de classes de persistência de estados do sistema LiNGS - parte servidor.

A funcionalidade de persistência de estados é suportada pelas classes da Figura 5.25. A classe `PersistentStateManager` realiza a gestão da persistência e restaurado do estado de clientes. De forma a ser persistido, o estado de cada cliente é transformado para objetos suportados pela serialização. Estes objetos consistem nos tipos `ClientStateWrapper`, `ClientObjectStateWrapper` e `FieldStateWrapper`. Existem referências de objetos no estado de clientes que não são apropriadas para serem guardadas, nomeadamente referências para objetos de jogo e várias referências utilizadas para *reflection*, torna-se necessário realizar esta transformação e encapsulamento de forma a garantir que os dados a guardar/restaurar conseguem ser suportados pelo mecanismo de serialização e que representam o estado do cliente de forma fiável.

As operações de serialização são realizadas com auxílio à classe `StorageEngine`. Esta classe está desenhada para suportar qualquer tipo de dados. A delegação da tarefa de serialização e persistência dos dados para a classe `StorageEngine` prepara o sistema LiNGS para suportar qualquer tipo de persistência, nomeadamente ficheiros, base de dados ou até

mesmo serviços. A implementação realizada permite a persistência do estado de clientes para ficheiros XML (*Extensible Markup Language*) para uma árvore de diretórios dentro do diretório base definido nas propriedades do servidor.

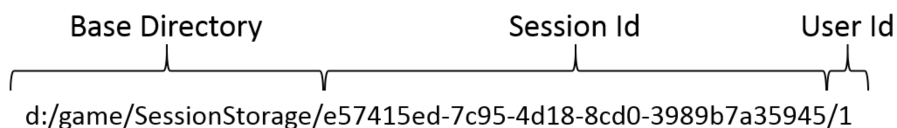


Figura 5.26: Estrutura de diretórios utilizada para persistir as sessões de clientes.

A Figura 5.26 apresenta a estrutura de ficheiros de sessões. O diretório base é especificado nas propriedades do servidor, por defeito é correspondente ao "SessionStorage" dentro do diretório do binário do servidor de jogo. Dentro deste diretório é criado as pastas para cada sessão de jogo, com o nome o identificador da sessão respetiva. Os ficheiros de sessões são guardados dentro dessa pasta com o nome respetivo ao identificador do cliente.

Os ficheiros que contém as sessões são serializados e de-serializados automaticamente pela classe `XmlSerializer` da .NET Framework [98]. Esta classe permite serializar e de-serializar objetos contendo propriedades e campos públicos. De forma a otimizar o espaço gasto pelas sessões persistidas é ainda utilizado uma compressão com o algoritmo gzip, com a classe GZipStream [99].

As sessões são assim persistidas quando um utilizador se desliga do servidor. A implementação do jogo indica se o estado deve ser persistido, sendo que de seguida o estado do cliente é convertido para as classes suportadas pela serialização e os dados são serializados de forma comprimida para ficheiros. O processo de restauro de sessões é feito da forma inversa. Quando um cliente se liga ao servidor é enviado um identificador da sua sessão prévia e o servidor verifica se a sessão do cliente existe e se é válida. A implementação do jogo decide se pretende ou não que o estado seja restaurado e em caso positivo o sistema LiNGS realiza as operações de descompressão, de-serialização, conversão dos dados guardados e retoma o estado do cliente. Se o estado for carregado com sucesso então o cliente é informado que pode utilizar o seu estado, em caso de erros o estado do cliente é sincronizado de forma integral, não sendo utilizado o estado guardado.

## 5.5 Integração com jogos

Um dos aspetos mais importante na implementação de uma *framework* ou sistema aberto é a forma de integração com outros sistemas e aplicações. A inclusão do sistema LiNGS nas várias camadas de integração de um jogo será feita tal como é apresentado na seguinte figura:

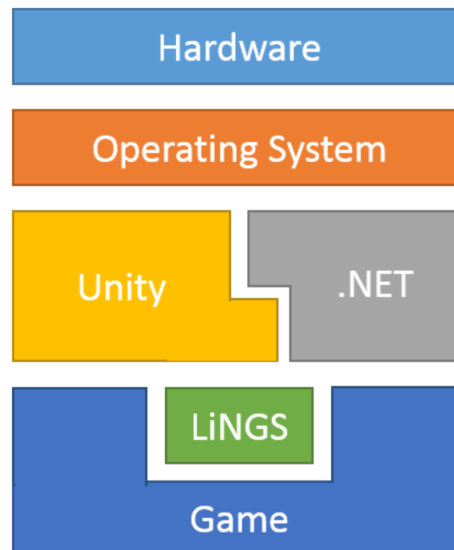


Figura 5.27: Integração do sistema LiNGS nas camadas de um jogo.

A Figura 5.27 demonstra as interações entre as diversas camadas que compõem um sistema até ao jogo em si. O Unity, utilizado como *framework* base para construção do jogo, comunica diretamente com o sistema operativo e toma partido também da utilização da .NET Framework. Por sua vez, os programadores dos jogos quando utilizam o Unity podem utilizar também a *framework* .NET para programação. A integração do LiNGS é feita de forma a ser uma extensão ao motor de jogo Unity, fornecendo várias funcionalidades sem retirar os benefícios presentes. Assim, o sistema LiNGS fica integrado entre a implementação do jogo, que requer as suas funcionalidades, e as *frameworks* base utilizadas para o desenvolvimento do jogo em si.

O cliente e servidor do sistema LiNGS foram desenhados e implementados de forma a serem facilmente integrados em jogos feitos em Unity e até mesmo outros tipos de aplicações. Para integrar o sistema LiNGS basta incluir os 3 *dlls* resultantes da implementação no projeto. Com a implementação do LiNGS referenciada a integração deste sistema passa, tal

como indicado na implementação de cada parte, pela implementação da interface respetiva, `INetworkedClient` e `INetworkedGame`, e pela instanciação da classe `LiNGSClient` ou `LiNGSServer` com as propriedades desejadas.

A integração do restante jogo é também realizada de forma simples, tal como referido anteriormente a arquitetura do sistema LiNGS foi desenhada para permitir uma elevada simplicidade na sua utilização. A integração de objetos de jogo apenas requer que cada objeto implemente uma interface, sem necessidade de implementação de métodos, e que marque os campos a sincronizar com o atributo `NetworkedField`.

**A**

```
public class GameObject
{

        public String name;
        public int state;
        public int positionX;
        public int positionY;
        public int value;


        public GameObject()
        {
           ...
        }

        ...


}
```

**B**

```
public class GameObject : INetworkedObject
{

        public String name;
        public int state;

        [NetworkedField]
        public int positionX;

        [NetworkedField]
        public int positionY;

        [NetworkedField(Simulated = true)]
        public int value;

        public GameObject()
        {
           lingsServer.LogicProcessor.RegisterNetworkedObject(this);
            ...
        }

        ...

}
```

Figura 5.28: Exemplo de integração do sistema LiNGS num objeto de jogo.

A Figura 5.28 representa o processo de integração do sistema LiNGS num objeto de jogo. A letra **A** representa o código do objeto antes da integração do LiNGS e o código anexado à letra **B** representa o código do objeto de jogo com a utilização da API (*Application Programming Interface*) do LiNGS. As modificações necessárias requerem a identificação da classe como um objeto que pode ser sincronizado e a identificação dos campos a sincronizar. De forma a indicar ao LiNGS que o objeto deve ser sincronizado bastante invocar o método `RegisterNetworkedObjet` da classe `GameLogicProcessor` deste sistema.

### 5.5.1 Pontos de acesso - API

O implementação do sistema LiNGS é composta for vários componentes internos, no entanto o acesso ao sistema é feito de forma simples. Os acessos deste sistema permitem a sua integração em jogos de forma acessível, disponibilizado acesso claro apenas aos componentes necessários para o funcionamento e disponibilizando também várias opções para configuração do sistema. Segue-se uma apresentação dos pontos de acesso ao sistema LiNGS que permitem a execução das funcionalidades implementadas.

| Cliente | | |
|---|---|---|
| Tipo | Método / Propriedade | Descrição |
| **LiNGSClient** | | Parte cliente do sistema LiNGS |
| | Connect | Tenta estabelecer a ligação com o servidor |
| | Disconnect | Termina a ligação com o servidor |
| | Update | Atualiza os componentes do sistema LiNGS |
| | LogicProcessor | Acesso ao processador de lógica |
| | Status | Estado do sistema LiNGS |
| **ClientProperties** | | Propriedades configuráveis do sistema LiNGS |
| | ConnectionEstablishingTimeout | Tempo máximo para estabelecimento da ligação com o servidor |
| | FieldSimulationLogic | Lógica para ser utilizada na simulação de valores |
| | ImportantMessageTimeout | Tempo máximo para receber resposta de uma mensagem importante |
| | MaxImportantMessageRetries | Número máximo de tentativas para enviar uma mensagem |
| | MaxMessageSize | Tamanho máximo de mensagens |
| | MaxServerBlackoutTime | Tempo máximo em que não são recebidas comunicação do servidor |
| **ClientStatus** | | Representa o estado do sistema LiNGS |
| | Connected | Indica se o cliente tem uma ligação ativa com o servidor |
| | EndPoint | Detalhes de rede do cliente |
| | SessionUserId | Identificador da sessão atual do cliente |
| **ClientLogicProcessor** | | Realiza as operações relacionadas com a lógica do jogo |
| | RegisterNetworkAwareObject | Regista um objeto de jogo conhecido pelo servidor |
| | RegisterNetworkedObject | Regista um objeto de jogo para ser criado e sincronizado para o servidor |
| | UnregisterNetworkAwareObject | Apaga um objeto previamente registo |
| | UnregisterNetworkedObject | Apaga um objeto previamente registo |
| **INetworkedClient** | | Interface a implementar pelo jogo de forma a receber informações do sistema LiNGS |
| | ConnectionAccepted | A ligação ao servidor foi realizada com sucesso |
| | ConnectionRefused | A ligação ao servidor não foi aceite |
| | CreateObject | Criar novo objeto de jogo |
| | DestroyObject | Destruir objeto de jogo |
| | Disconnected | A ligação com o servidor foi terminada |
| | ReceiveErrorMessage | Recebida mensagem do tipo Error |
| | ReceiveEventMessage | Recebida mensagem do tipo Event |
| | ReceiveGameMessage | Recebida mensagem do tipo Game |
| | UnableToConnect | Tentativa de ligação com o servidor não foi bem sucedida |
| **SimulatorLogic** | | Realiza a operação de simulação de dados |
| | RunSimulation | Simula o próximo valor de um campo com base nos valores previamente recebidos |

Tabela 5.1: Pontos de acesso do sistema LiNGS - parte cliente.

| Servidor | | |
|---|---|---|
| **Tipo** | **Método / Propriedade** | **Descrição** |
| **LiNGSServer** | | Parte servidor do sistema LiNGS |
| | Shutdown | Termina a execução do servidor |
| | Update | Atualiza o sistema LiNGS |
| | LogicProcessor | Acesso ao processador de lógica de jogo |
| **ServerProperties** | | |
| | DeleteSessionFilesOnExit | Indica se o sistema deve eliminar os ficheiros de sessão no seu encerramento |
| | EnablePersistentStates | Ativa a persistência de sessões de utilizadores |
| | ImportantMessageTimeout | Tempo máximo para receber resposta de uma mensagem importante |
| | ListenPort | Porto de escuta do servidor |
| | MaxClientBlackoutTime | Tempo máximo em que não são recebidas comunicação de um cliente |
| | MaxClients | Número máximo de clientes ligados ao servidor |
| | MaxImportantMessageRetries | Número máximo de tentativas para enviar uma mensagem importante |
| | MaxMessageSize | Tamanho máximo de mensagens |
| | MaxMessageWaitTime | Tempo máximo que uma mensagem pode estar em espera para ser enviada |
| | SessionStorageBaseDirectory | Diretoria base para ficheiros de sessões |
| **GameLogicProcessor** | | |
| | DisconnectClient | Termina a ligação com um cliente |
| | GetConnectedClients | Lista de clientes ligados ao servidor |
| | RegisterClientAwareNetworkedObject | Regista um objeto de jogo conhecido pelos clientes |
| | RegisterNetworkedObject | Regista um objeto de jogo para ser criado e sincronizado para os clientes |
| | UnregisterClientAwareNetworkedObject | Apaga um objeto previamente registo |
| | UnregisterNetworkedObject | Apaga um objeto previamente registo |
| **INetworkedGame** | | |
| | AcceptClient | Aceita ou rejeita a ligação de um cliente |
| | ClientDisconnected | Ligação de um cliente foi terminada |
| | CreateClientObject | Criar objeto de jogo de um cliente |
| | DestroyClientObject | Destruir objeto de jogo de um cliente |
| | DoesClientNeedToKnowAboutObject | Indica se o objeto deve ser sincronizado para um cliente |
| | ReceiveErrorMessage | Recebida mensagem do tipo Error |
| | ReceiveEventMessage | Recebida mensagem do tipo Event |
| | ReceiveGameMessage | Recebida mensagem do tipo Game |
| **ClientConnectionResponse** | | |
| | Accept | Indica se a ligação do cliente deve ser aceite |
| | RefuseMessage | Se a ligação foi recusada então contém o motivo |
| | UseSavedState | Indica se o sistema deve utilizar uma sessão existente |

Tabela 5.2: Pontos de acesso do sistema LiNGS - parte servidor.

As Tabelas 5.1 e 5.2 representam os pontos de acesso do sistema LiNGS ou seja a API do sistema. Apenas são indicados os pontos principais de acesso ao sistema que permitem realizar as funcionalidades do LiNGS, sendo que os componentes internos e tipos complementares não estão aqui representados.

## 5.6 Conclusão

A implementação de sistemas complexos para a utilização em ambientes diversos e com várias características é um grande desafio. A necessidade de solidez interna e uma interface de programação simples e útil são os principais requisitos deste tipo de sistemas, sendo que pode ser necessário iterar a arquitetura do sistema até obter um desenho sólido que concretize todos os objetivos. A implementação do sistema LiNGS foi realiza de forma ágil com a adição de funcionalidades de forma sequencial com o objetivo de manter um núcleo sólido e organizado. Tal é refletido na estrutura e organização da implementação do LiNGS onde os vários componentes do sistema estão dispostos por *namespaces* apropriados e cada objeto tem a sua responsabilidade bem definida.

A linguagem de programação C# e a .NET Framework permitiram implementar todas as funcionalidades da especificação do LiNGS de forma conveniente e flexível. A utilização de funcionalidades como o LINQ [100] e expressões Lambda [101] revelaram-se uma forma de agilizar e reduzir o tempo de programação. Estes mecanismos de desenvolvimento rápido e ágil fizeram da linguagem C# uma ótima escolha para o desenvolvimento do LiNGS.

O resultado da implementação do sistema LiNGS são três ficheiros binários que podem ser incluídos em jogos e outras aplicações em qualquer plataforma com suporte à .NET Framework. A implementação do sistema é também distribuída de forma *open-source* com o intuito de permitir que qualquer programador possa explorar e alterar o sistema de forma a mudar o comportamento de componentes ou acrescentar funcionalidades.

De uma forma geral a implementação do sistema LiNGS foi feita com sucesso e reflete as necessidades e requisitos da especificação previamente efetuada. De forma a assegurar o correto funcionamento do sistema é ainda necessário a realização de testes num envolvente real. No próximo capítulo serão apresentados os testes e resultados da utilização do sistema LiNGS implementado.

# Capítulo 6

# Análise de funcionamento

## 6.1 Introdução

O desenvolvimento de um sistema requer normalmente uma fase de teste de forma a demonstrar o funcionamento do sistema e as características. Nos capítulos anteriores foram realizadas a especificação e a análise da implementação do sistema LiNGS. Trata-se de um sistema complexo que propõe solucionar vários problemas relacionados com tecnologias externas e sem controlo direto, sendo este um sistema desenhado para executar como *framework* em jogos multijogador em rede para dispositivos móveis.

Neste capítulo serão realizados diversos testes e respetivas análises, de forma demonstrar o funcionamento do sistema LiNGS no ambiente real.

## 6.2 Metodologia de teste

Testar a especificação e implementação de uma *framework* impõe a criação de uma aplicação para utilizar as funcionalidades que esta fornece. No caso do sistema LiNGS é necessário implementar um jogo em Unity que consiga testar as várias funcionalidades do sistema. A especificação do sistema LiNGS deve também ser testada com a implementação do jogo, ou seja, a realização dos testes não pretende apenas determinar se as várias funcionalidades do sistema se encontram implementadas de forma correta, mas pretende também determinar se a especificação desenhada combate os problemas assinalados no Capítulo 4.

Existe assim a necessidade de realizar o desenho e implementação de um jogo multijogador

utilizando o sistema LiNGS e a *framework* Unity. O jogo deverá ser realizado tendo como destino dispositivos móveis atuais.

### 6.2.1 Implementação de jogo para testes

Com o objetivo de testar o sistema LiNGS de forma real, é proposta a criação de um jogo pequeno e visualmente simples, mas que tenha funcionalidades que obriguem a ações de rede constantes e com informação importante em continua transição. O objetivo não é criar um jogo complexo de forma a testar a *framework* Unity, mas sim utilizar o Unity como *framework* para implementar um jogo para testar o sistema LiNGS.

Com base nestes fundamentos, é proposto criar um jogo que teste as seguintes vantagens do sistema LiNGS:

- Sincronização de dados
- Delegação de processamento para o servidor
- Redução do processamento local, redução de tráfego e consumo de energia
- Suporte para falhas de rede

#### Descrição do jogo

O jogo criado de raiz para testar o sistema LiNGS tem o nome de Striker e é um jogo em 2D com uma perspetiva *top-down*. Este tipo de perspetiva representa o jogo de forma 2D com uma visão por cima do mapa e dos objetos de jogo, sendo a representação destes feita com texturas estáticas e animadas.

A representação do jogo na Figura 6.1 demonstra a perspetiva utilizada no jogo. Este tipo de perspetiva foi escolhida de forma a permitir evidenciar várias funcionalidades do sistema LiNGS como por exemplo a sincronização de objetos pelo contexto do jogo, neste caso particular apenas os inimigos próximos do jogador são sincronizados.

#### Funcionamento

O Striker é um jogo desenvolvido com o objetivo de testar o sistema LiNGS. Assim, este jogo funciona com multijogador e está desenvolvido para suportar até 2 jogadores reais e 4

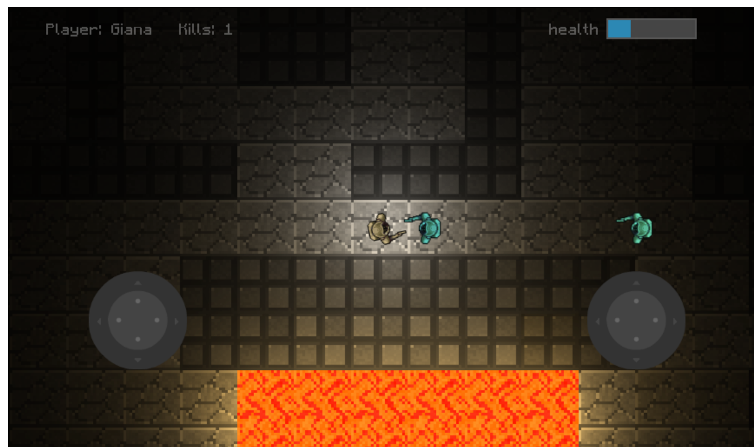


Figura 6.1: Representação do jogo Striker utilizado para testar o sistema LiNGS.

jogadores controlados pelo servidor – NPC (*Non-player character*). A utilização de jogadores controlados pelo servidor é feita de forma a evidenciar a delegação de processamento para o servidor.

O objetivo de jogo consiste em matar os inimigos até um limite de 15 mortes. O jogo é realizado num mapa estático onde os vários jogadores podem navegar livremente, no entanto os jogadores têm uma visão reduzida do mapa de jogo e apenas conseguem ver uma pequena parcela à sua volta. Desta forma, os jogadores são obrigados a estar em constante movimento para encontrar e matar os inimigos. Os jogadores conseguem movimentar-se, apontar e disparar tiros de forma a acertar e matar inimigos. Cada jogador tem ainda de se desviar dos tiros dos inimigos de forma proteger a sua vida.

**Detalhes de implementação**

A implementação de um jogo multijogador impõe a criação e utilização de uma arquitetura separada com um servidor e um cliente. A implementação do Striker é feita dessa forma, com o servidor feito em .NET para poder ser executado num computador ou diretamente no jogo, e o cliente implementado em Unity, versão 4.3.3f1.

A implementação do servidor é realizada num projeto no Microsoft Visual Studio denominado de Striker tendo como base a .NET Framework 3.5 e utilizando a linguagem de programação C#. É importante referir que a implementação do servidor poderia ter sido realizada

diretamente no Unity, mas dessa forma não existia uma separação lógica do servidor e cliente e o servidor requeria o Unity para ser executado. Assim, o servidor pode ser executado em qualquer aplicação .NET, até mesmo de forma remota, estando alojado num servidor. Esta forma de implementação tem as vantagens enunciadas, mas requer a replicação de código no cliente.

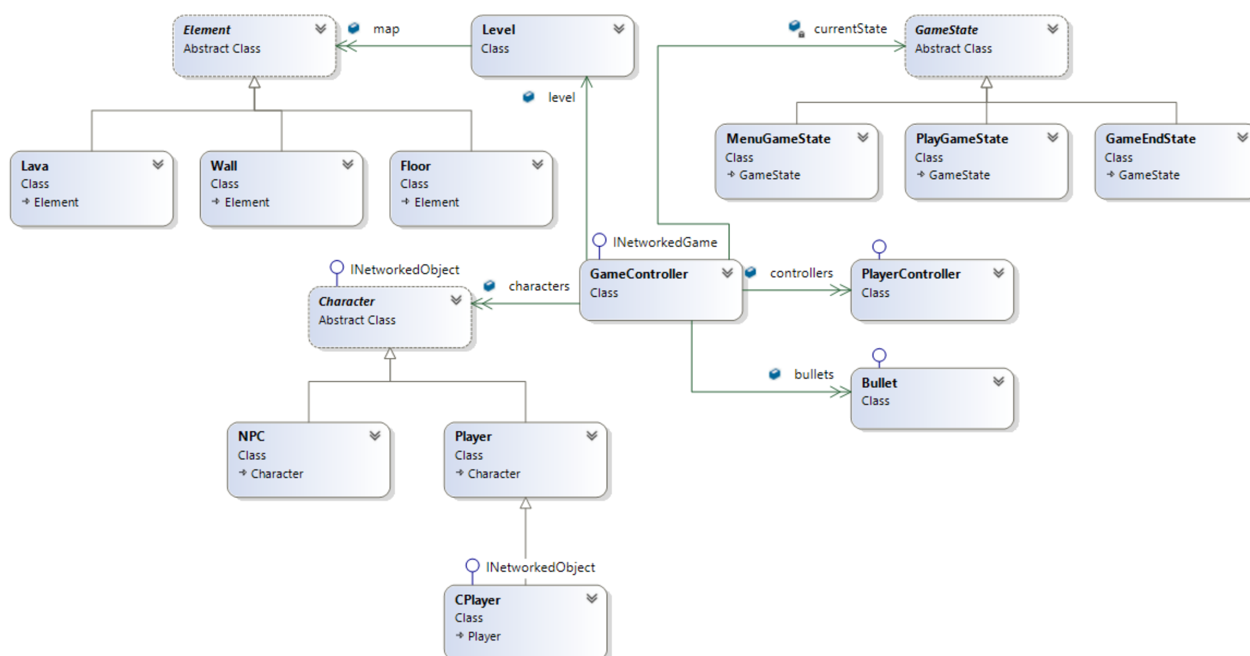


Figura 6.2: Diagrama de classes da implementação do servidor para o jogo Striker.

A Figura 6.2 mostra o diagrama de classes da implementação do servidor. A implementação é feita de forma clássica com o objeto `GameController` a englobar todos os objetos de jogo, incluindo o mapa e os seus elementos, representados pela classe `Level` e pelas classes `Element`, `Lava`, `Wall` e `Floor`. O mapa é representado por um ficheiro de texto em que cada caracter representa um tipo de elemento. Assim é possível personalizar o mapa facilmente alterando os caracteres do ficheiro de texto e os seus índices no ficheiro.

Os jogadores são representados pelas classes `Character`, `NPC`, `Player`. Estas classes contêm os campos que representam o jogador, tais como a sua posição, pontuação, nome. Os jogadores clientes são representados pela classe `CPlayer` e `PlayerController` que permitem a sincronização dos campos do cliente e das entradas de movimentos que o utilizador

realiza no jogo. Outro elemento que está presente no jogo são os tiros que são representados pela classe `Bullet`. A classe `GameController`, tal como indicado, mantém referência para todos os objetos presentes no jogo.

De forma a realizar a lógica de jogos nos vários elementos, existem as classes `GameState`, `MenuGameState`, `PlayGameState` e `GameEndState`. Estas classes realizam a lógica implementada para cada estado do jogo. O servidor inicia com o estado `MenuGameState`. Quando os clientes se ligam ao servidor e iniciam o jogo, o estado passa para o `PlayGameState`. Este estado realiza os procedimentos normais do jogo, atualiza as posições dos `NPCs`, cria `Bullets` para representar os tiros dos jogadores, verifica colisões das `Bullets` com os vários jogadores e verifica ainda as condições de final de jogo.

A movimentação dos `NPCs` é realizada com um algoritmo de inteligência artificial. Se o `NPC` tiver um inimigo na sua zona visível então o seu destino é a posição do inimigo, caso o NPC não tenha inimigos na sua área de visão, então este movimenta-se para uma posição aleatória no mapa. A movimentação para o destino é feita com o algoritmo A* (*A-Star*) [102]. Este algoritmo realiza uma pesquisa de forma a determinar o melhor caminho para o `NPC` chegar à posição de destino. A delegação deste algoritmo para o servidor implica que a sua computação pesada não tenha de ser feita em todos os clientes e permite assim minimizar o consumo de energia dos dispositivos.

A verificação de colisões é também um aspeto importante de um jogo multijogador. No Striker existem dois tipos principais de colisões: colisão com o mundo/mapa e colisão com as `Bullets`. A colisão dos `NPC` com o mundo está diretamente implementada no algoritmo de IA (Inteligência Artificial) e não permite que o `NPC` se desloque para um elemento que não seja `Floor`, que representa o chão do mapa. A colisão dos clientes é feita localmente na parte do cliente. São utilizados mecanismos de verificação de colisão do Unity que permitem facilmente detetar a colisão dos jogadores com o mapa. Assim, a posição dos jogadores, que é sincronizada para o servidor, nunca é inválida, ou seja, está sempre num elemento do tipo `Floor`.

A colisão e atualização das `Bullets` é um caso diferente. Como o número de `Bullets` simultâneas no jogo pode ser alto é necessário utilizar técnicas que permitam reduzir o número de transmissões de atualizações dos campos de cada `Bullet`. Assim, quando estas são criadas, é indicada a sua posição, rotação e velocidade para que os clientes consigam

movimentar as balas sem necessidade de atualizações constantes do servidor. A verificação de colisão é realizada no servidor, que verifica se as balas colidem com jogadores ou se colidiram com uma parede do mapa. Quando tal acontece, a bala é eliminada de todos os pontos de jogo – servidor e clientes. A deteção deste tipo de colisões no servidor garante que as colisões são fiáveis para todos os jogadores.

A recuperação do estado de clientes é ainda uma funcionalidade do sistema LiNGS utilizada neste jogo. O jogo suporta o restauro do estado de jogadores. Para tal a classe `GameController`, que implementa a interface do LiNGS `INetworkedGame`, indica ao sistema que a sessão de jogadores que se desliguem deve ser guardada. Internamente, o jogo mantém os objetos dos clientes, mas identifica-os como pertencentes a um jogador desligado. Com o objetivo de limitar a utilização desta funcionalidade para benefício de jogadores, estes permanecem ativos durante 5 segundos depois de serem desligados. Assim, os inimigos conseguem continuar a atingir o jogador, se for o caso, e não permite que o jogador se desligue propositadamente do jogo para fugir a um combate.

A implementação da parte cliente do jogo é feita diretamente no Unity. O jogo está dividido em duas *Scenes*:

- `SceneInit`
- `SceneGame`

A `SceneInit` inicia automaticamente com o jogo e contém os menus e navegação do jogo. Os jogadores podem iniciar um servidor ou ligar-se a um jogo utilizando para isso os menus implementados. Quando a fase de jogo inicia então a *scene* `SceneGame` é carregada e o jogo carrega o mapa que é indicado pelo servidor. Apenas o nome do mapa é transmitido e os dados deste são carregados de um ficheiro de texto presente no jogo.

Os restantes objetos do cliente são inicializados, nomeadamente os `joysticks` de controlo do jogador e outros de controlo. De seguida a sincronização entre o cliente e o servidor começa. O servidor sincroniza os vários jogadores do jogo e indica aos clientes qual o jogador que vão controlar. O cliente começa também a sincronizar os controlos introduzidos pelo jogador e o jogo inicia.

A implementação do cliente de jogo está preparada para permitir o restauro da sessão do jogador caso a ligação seja interrompida. Quando tal acontece é mostrado um aviso ao jogador e é dado a escolher entre tentar ligar novamente ou então sair do jogo. Na tentativa de ligação, o cliente envia o identificador de sessão que lhe foi enviado no início do jogo e, caso o servidor já se encontre contactável, o jogo é retomado.

A implementação do jogo permite assim testar praticamente todas as funcionalidades do sistema LiNGS.

## 6.3 Resultados e análise

A implementação do jogo Striker permite realizar vários testes ao sistema LiNGS. Os testes mais importantes consistem em comprovar o funcionamento do sistema e a sua fiabilidade como sistema para jogos multijogador.

A implementação do jogo foi testada durante o decorrer do seu desenvolvimento utilizando o Unity Editor. Depois da implementação concluída o jogo foi exportado para ser utilizado em dispositivos móveis.

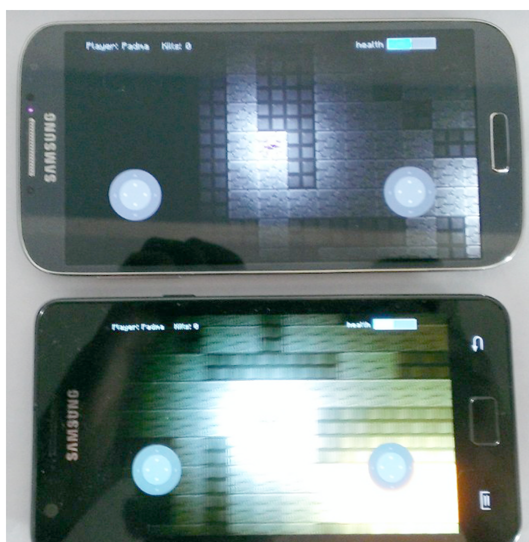

Figura 6.3: Jogo Striker a executar em dois dispositivos móveis.

Na Figura 6.3 é demonstrado o jogo Striker a executar em dois dispositivos móveis. O dispositivo em cima é um Samsung Galaxy S4 com o sistema operativo Android 4.4.2 ligado a rede *wireless* através da norma *N*. O dispositivo em baixo é um Samsung Galaxy S2 com o sistema operativo Android 4.1.2 e encontra-se ligado a uma rede *wireless* através da norma *G*.

A exportação do jogo para os dispositivos é relativamente simples utilizando as ferramentas do Unity. De forma a testar as mecânicas de multijogador, o servidor do Striker é executado num computador ligado à mesma rede *wireless* através da norma *N*. Os primeiros testes indicam que que a implementação do sistema LiNGS se encontra a funcionar corretamente. A sincronização dos elementos de jogo é realizada com sucesso e a lógica de jogo executada no servidor é corretamente representada nos dispositivos. A sincronização entre cliente-servidor-cliente é também realizada com sucesso. Os vários clientes são corretamente representados nos dispositivos de outros jogadores.

De forma a testar as consequências de uma ligação com problemas, é simulada uma quebra de ligação num dos dispositivos. A execução do sistema LiNGS neste jogo está configurada para desligar a ligação ao servidor quando este não envia qualquer resposta para o jogo em 5 segundos consecutivos. Respetivamente, o servidor está configurado para desligar o cliente quando este não responde a pedidos durante um período de 10 segundos. O teste efetuado consistiu em desligar a ligação *wireless* do dispositivo por um período maior do que 10 segundo e de seguida tentar realizar uma nova tentativa de ligação ao servidor.

Figura 6.4: Processo de deteção de falha de ligação e restabelecimento da ligação.

O processo de perda da ligação ao servidor é demonstrado na Figura 6.4, onde o jogador se encontra no jogo e é mostrada uma mensagem a indicar o problema na ligação. A falha de rede é, tal como indicado previamente, simulada com o término da ligação que está a ser utilizada pelo jogo. A implementação do sistema LiNGS deteta o problema de ligação, sendo

que não é possível realizar comunicações dentro do tempo configurado para tal.

O jogo permite ao utilizador escolher se pretende sair do jogo ou tentar restabelecer a ligação. Caso o jogador tente restabelecer a ligação, o sistema tenta realizar a ligação ao servidor indicando o identificador de sessão previamente recebido. Se a nova ligação for realizada com sucesso, então o jogo é retomado e o jogador continua com as mesmas características que tinha quando a sua ligação foi terminada.

Com o objetivo de analisar o funcionamento do sistema e os seus benefícios foram realizados vários testes. O primeiro teste consistiu em realizar várias partidas do jogo de forma a obter dados sobre o funcionamento do sistema LiNGS.

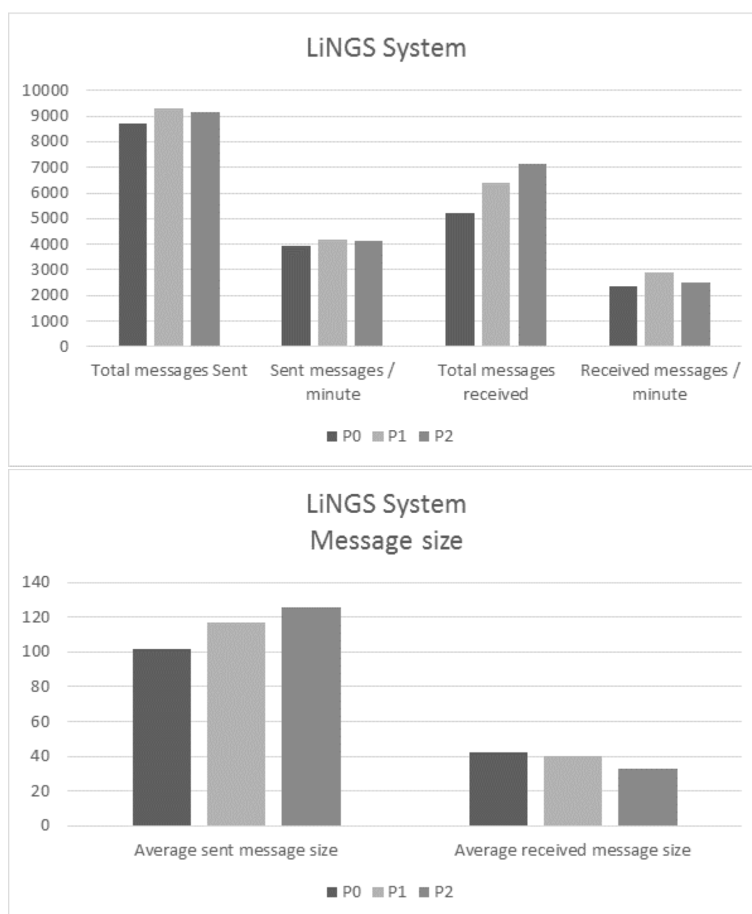


Figura 6.5: Dados recolhidos do sistema LiNGS em funcionamento normal.

A Figura 6.5 demonstra os dados recolhidos do sistema LiNGS de 3 partidas (representados por P0, P1 e P2) realizados com o jogo Striker. Os jogos tiveram uma duração média de 2,39 minutos com um jogador e 4 NPC. Os dados apresentados foram recolhidos do servidor do sistema LiNGS. Nos gráficos da Figura 6.5 é possível obter várias informações sobre o funcionamento do sistema LiNGS no jogo Striker. Sendo este um jogo em tempo real com constantes movimentações são necessárias constantes trocas de informação entre os clientes e o servidor, de forma a sincronizar os elementos de jogo. Os jogos tiveram uma média de 9036 mensagens enviadas e 6244 recebidas. Estes dados equivalem a cerca de 4085 mensagens enviadas por minuto e 2588 mensagens recebidas por minuto de jogo. É também apresentado no gráfico o tamanho das mensagens de jogo. As mensagens enviadas têm um tamanho médio de 115 bytes e as mensagens recebidas 38 bytes.

Estes são os valores por defeito do sistema LiNGS em pleno funcionamento com todos os sistemas ativos e com um agregador de mensagens com um tempo de espera de pelo menos 10ms. Com base nestes dados é possível comprovar a eficácia destes sistemas na redução de processamento e poupança energética nos clientes.

O próximo teste é realizado com o sistema de agregação de mensagens desativado. As mensagens são transmitidas assim que sejam formuladas.

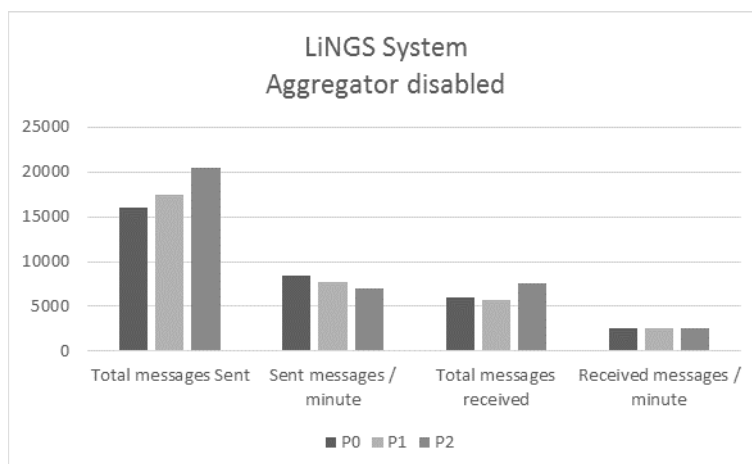


Figura 6.6: Dados relativos ao número de mensagens, recolhidos do sistema LiNGS com o sistema de agregação de mensagens desativado.

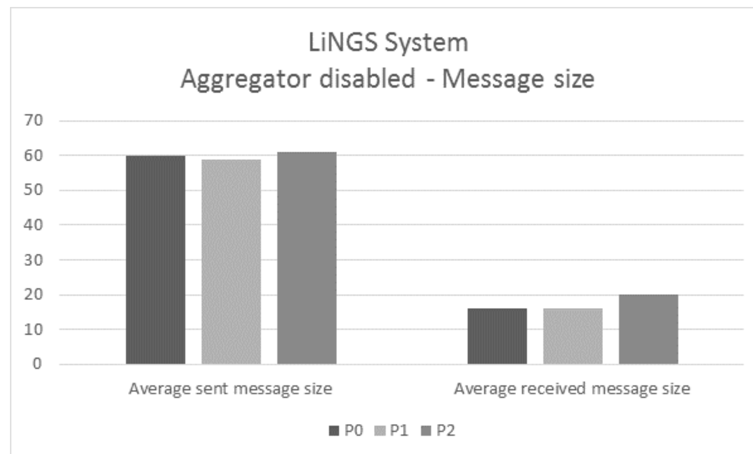


Figura 6.7: Dados relativos ao tamanho de mensagens, recolhidos do sistema LiNGS com o sistema de agregação de mensagens desativado.

Os gráficos das Figuras 6.6 e 6.7 mostram resultados esperados com o sistema de agregação de mensagens desativado. Novamente foram realizadas 3 partidas (P0, P1, P2) com uma duração média de 2,34 minutos com configurações idênticas ao teste anterior. Os resultados indicam que existiu um aumento de cerca de 98% do número de mensagens enviadas, sendo o número médio de mensagens enviadas por jogo de 17976, e um aumento insignificativo nas mensagens recebidas de 2.9%, sendo recebidas 6429 mensagens por jogo. Os valores de mensagens transmitidas por minuto aumentaram de forma correspondente, sendo enviadas uma média de 7773 mensagens por minuto e recebidas 2598 mensagens por minuto.

Estes números demonstram que o sistema LiNGS é bastante mais eficiente utilizando um agregador de mensagens. Continuando a análise da Figura 6.7, é possível concluir que o tamanho das mensagens transmitidas baixou consideravelmente. O tamanho das mensagens enviadas e recebidas é agora cerca de 50% mais pequeno do que o primeiro teste. Estes resultados são normais, visto que a informação do jogo é agora transmitida de forma separada.

Comparando os resultados dos testes anteriores é possível concluir que um sistema sem um agregador de mensagens pode prejudicar bastante os recursos do sistema, tanto a nível de processamento como de consumo de energia, mas pode também influenciar o funcionamento da rede. A quantidade de pacotes enviados na rede pode ser de tal forma alto que os controladores de rede podem começar a realizar *traffic shaping* [103] afetando o número de pacotes transmitidos com sucesso e atrasando a entrega destes. Com o agregador de mensa-

gens o sistema LiNGS consegue juntar várias informações para o mesmo cliente na mesma mensagem e assim baixar o número de mensagens transmitidas.

A utilização de sincronização dos elementos de jogo pelo contexto é também uma das principais funcionalidades do sistema LiNGS. De forma a testar a eficiência deste sistema foi realizado um teste com os vários componentes do LiNGS em funcionamento mas todos os objetos de jogo eram sincronizados.

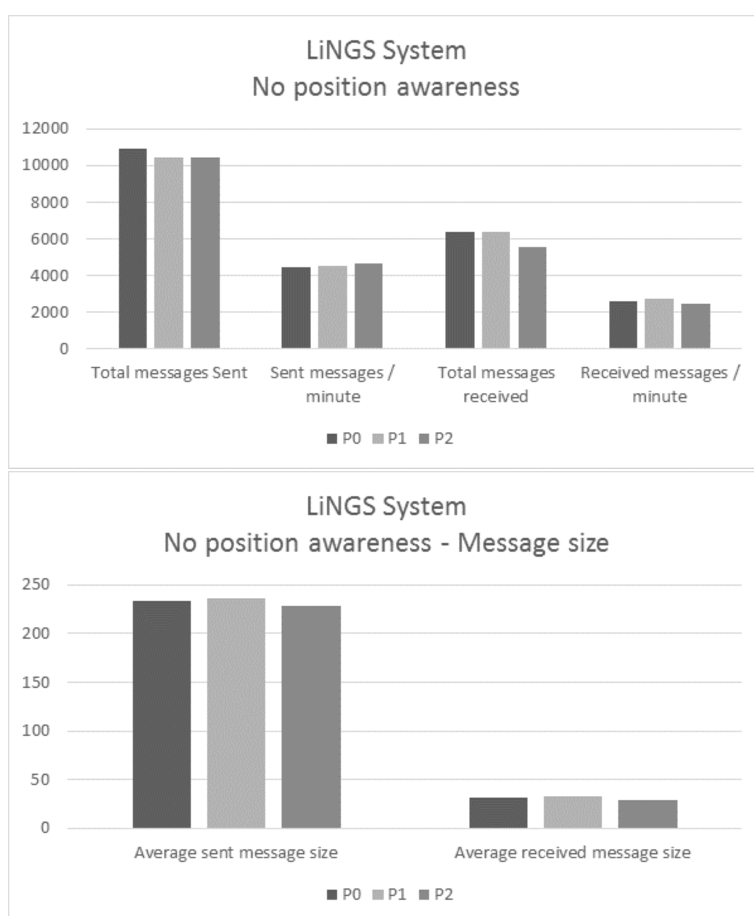


Figura 6.8: Dados recolhidos do sistema LiNGS sem a utilização de sincronização contextual.

Este teste foi concretizado com a realização de 3 jogos com a mesma configuração utilizada nos testes anteriores.

Os resultados do teste sem sincronização contextual são demonstrados na Figura 6.8 e revelam que esta funcionalidade tem um impacto significativo nos dados transmitidos. O número de mensagens enviadas pelo servidor teve um aumento de 17% relativamente ao teste com o funcionamento do sistema LiNGS de forma normal. Ainda mais significativo é o tamanho das mensagens enviadas que teve um aumento de mais de 100%. Este teste revela portanto que a sincronização contextual de objetos beneficia bastante o funcionamento de jogos multijogador. Sem esta funcionalidade não são apenas transmitidas mais mensagens mas o tamanho das mesmas é significativamente maior, elevando bastante a quantidade de informação transmitida.

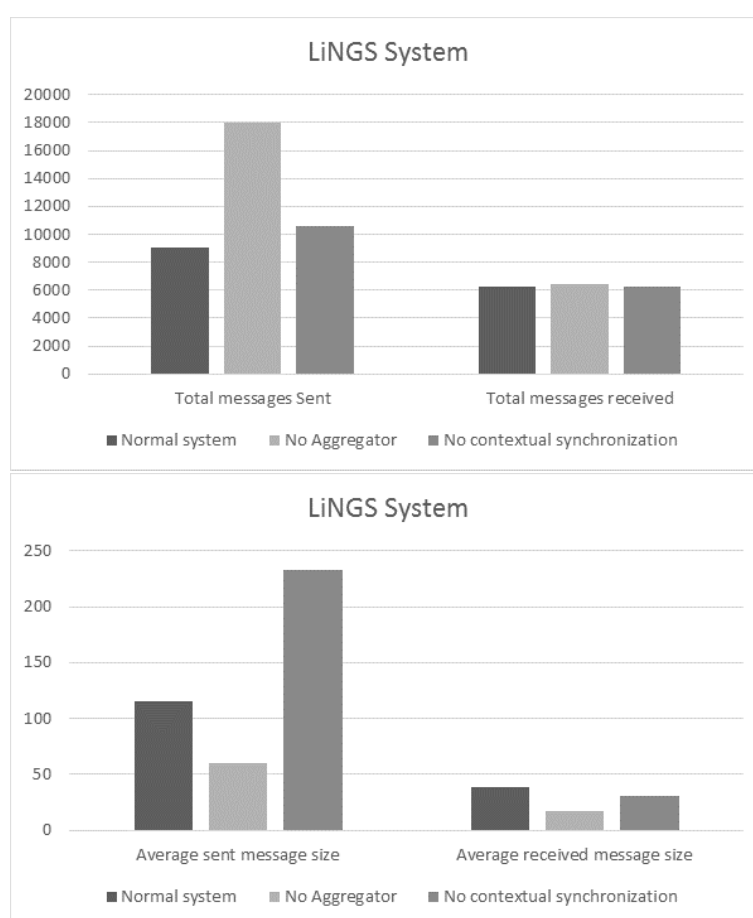


Figura 6.9: Comparação dos resultados dos testes realizados.

A comparação direta dos testes realizados demonstra concretamente a análise realizada individualmente. A Figura 6.9 inclui dois gráficos de comparação dos dados recolhidos pelos

testes anteriores. No primeiro gráfico é possível observar que a utilização do sistema LiNGS sem a agregação de mensagens e sem a sincronização contextual aumenta significativamente o número de pacotes enviados pelo servidor. A alteração nos números de pacotes recebidos é pouco significativa devido ao baixo número de clientes ligados ao servidor.

O segundo gráfico da Figura 6.9 mostra que o tamanho das mensagens enviadas (em bytes) é relativamente mais pequeno quando o agregador de mensagens não está ligado e que quando a sincronização dos elementos de jogo é feita de forma total o tamanho das mensagens é significativamente maior. Mais uma vez a comparação das mensagens recebidas é pouco significativa devido ao baixo número de clientes ligados.

Esta comparação demonstra assim que a utilização normal do sistema LiNGS é mais eficaz com as diversas funcionalidades ativas e que as funcionalidades implementadas conseguem aumentar a eficácia dos sistemas multijogador em rede. As funcionalidades conseguem um termo de eficácia que garante o menor número de pacotes transmitidos na rede tendo estes um tamanho ideal pois apenas transportam as informações estritamente necessárias.

### 6.3.1 Testes de utilização

A realização de testes de utilização permite complementar os testes já realizados ao sistema LiNGS. Nestes testes é proposto a utilização real do jogo Striker por vários utilizadores de forma a verificar o funcionamento do sistema LiNGS num ambiente real e com utilizadores que não conhecem a implementação do sistema. Este tipo de teste é denominado de *Black-box* [104].

Foram realizados 5 testes que consistiram na realização de uma partida no jogo Striker. Os utilizadores que realizaram os testes estão apresentados na Tabela 6.1.

| **Utilizadores** | |
|---|---|
| Programador | Utilizador regular |
| U0 | U1 |
| U2 | U4 |
| U3 | |

Tabela 6.1: Utilizadores que realizaram testes ao jogo Striker.

Os utilizadores escolhidos têm idades compreendidas entre os 22 e 30 anos. Os utilizadores

da categoria 'Programador' são utilizadores com experiência de programação e vasto conhecimento e experiência na utilização de dispositivos móveis de forma pessoal e profissional. Os utilizadores 'Utilizador regular' são utilizadores com experiência de utilização de dispositivos móveis na ótica de utilização pessoal.

Foi indicado aos utilizadores o funcionamento do jogo e os objetivos necessários para o jogo terminar. O objetivo destes testes não é testar a implementação do jogo mas sim a camada de multijogador em rede, nomeadamente o sistema LiNGS. Assim, foram recolhidos vários dados do sistema durante a realização dos testes.

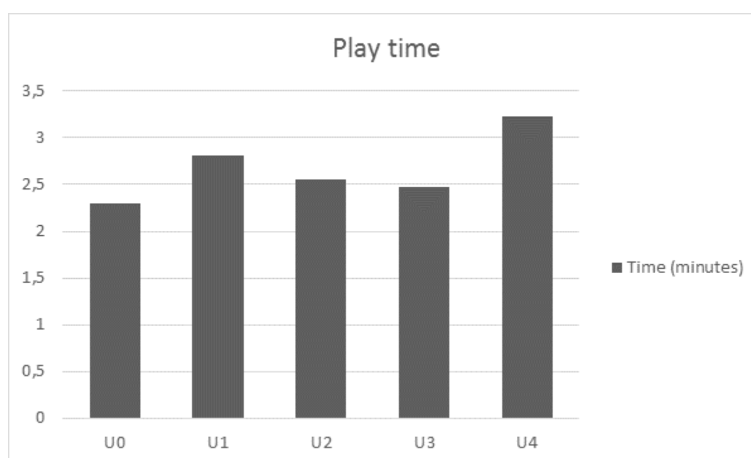


Figura 6.10: Duração dos vários testes realizados.

Os testes tiveram uma duração média de 2,6 minutos, tal como apresentado na Figura 6.10, sendo que a duração mínima registada foi de 2,2 minutos e a máxima de 3,23 minutos. Durante estes testes os utilizadores jogaram de acordo com as suas capacidades, características e objetivos. A forma de jogar e a componente aleatória do jogo tornam variável o tempo total de cada jogo.

Durante a realização dos testes foram encontrados vários problemas com a implementação do jogo e com o sistema LiNGS. Vários jogadores demonstraram problemas com a utilização dos controlos do jogo, portanto na parte de interface com o utilizador. O jogo utiliza dois *joysticks* para controlo da posição e rotação do jogador, sendo que um desses é também utilizado para disparar a arma quando o jogador carrega intermitentemente. Os testes revelaram esta forma de controlo pouco eficaz e o jogo foi alterado para resolver o problema. As

modificações realizadas implicam agora que o jogador apenas tenha de tocar no *joystick* de controlo da rotação para que a arma seja disparada.

Outra problema detetado, desta vez relacionado indiretamente com o sistema LiNGS, foi bloqueios nas redes de comunicação. Durante a realização de vários testes numa rede partilhada, foi detetado que a rede bloqueia a comunicação de certas aplicações que utilizem certos portos de ligação. A resolução deste tipo de problemas está relacionadas com equipamentos e configurações de terceiros, para a conclusão dos testes foi criada uma rede dedicada sem bloqueios.

Ainda durante o progresso dos vários jogos realizados foram recolhidos diversos dados com o objetivo de observar o funcionamento interno do sistema LiNGS. As diferente formas de jogar, aplicadas pelos vários utilizadores, podem influenciar o funcionamento do sistema provocando consequências na comunicação e tratamento de dados em situações que não foram previstas.

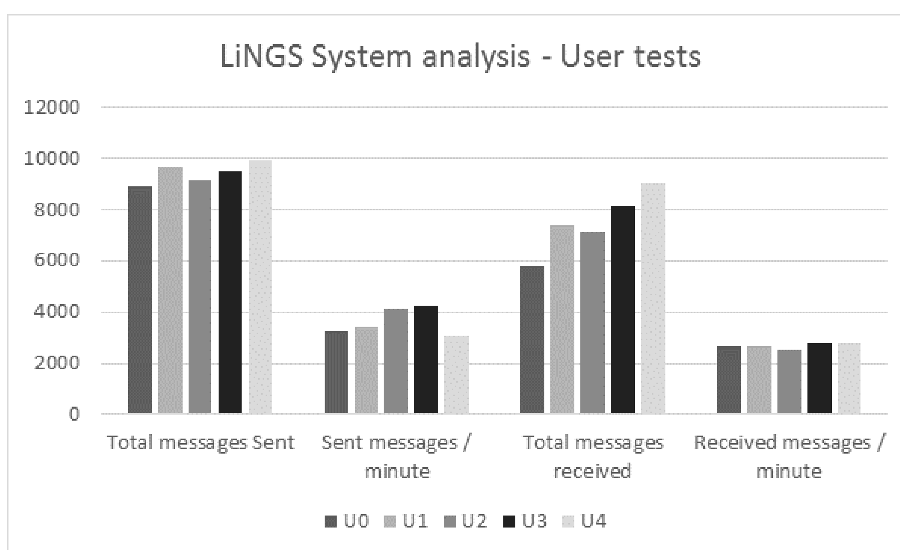


Figura 6.11: Duração dos vários testes realizados.

Os dados apresentados na Figura 6.11, capturados na parte servidor do sistema, mostram com detalhe o funcionamento do sistema LiNGS nas várias situações de jogo. Os dados recolhidos são positivamente consistentes na medida em que não se verificou alterações significativas do funcionamento do sistema durante a realização dos vários testes.

Na comparação entre os dois tipos de utilizadores é possível verificar que a duração dos jogos para os utilizadores regulares (U1, U4) é um pouco superior aos restantes. Estes valores refletem-se na forma de jogar aplicada por estes jogadores, em que a velocidade de movimentação e interação era notavelmente inferior. É ainda possível visualizar os efeitos desta forma de jogar, nos dados da Figura 6.11, nomeadamente no número de mensagens enviadas por minuto. O tempo de jogo dos utilizadores regulares é superior aos restantes mas o número de mensagens enviadas é bastante semelhante, sendo que o número de mensagens enviadas por minuto é inferior. A interação mais lenta com os elementos do jogo implicou uma taxa de transmissão de mensagens menor.

A realização destes testes permitiu analisar o funcionamento da *framework* LiNGS em diversas condições de jogo aplicadas por vários utilizadores. A influência do estilo de jogo reflete-se no funcionamento do LiNGS mas de forma mínima. Quando a interação dos utilizadores é mais lenta existe menos mensagens a circular entre as várias partes dos sistema (cliente e servidor) devido a diminuição de comandos enviados e também pela sincronização contextual.

## 6.4 Conclusão

A implementação de um sistema complexo é, de certa forma, também complexa de testar. O LiNGS é um sistema genérico com bastantes funcionalidades suportadas por vários componentes. Este sistema pode ser utilizado por qualquer tipo de jogo. Logo é difícil encontrar um bom teste que consiga prever todos os casos de utilização. De forma a testar e analisar o funcionamento do sistema LiNGS foi implementado o jogo Striker.

O jogo Striker foi implementado com o Unity e é um jogo 2D em multijogador para dispositivos móveis, sendo a vertente multijogador totalmente suportada pelo sistema LiNGS. A implementação deste jogo serviu como uma forma de testar as funcionalidades do sistema e serviu também para encontrar novas funcionalidades que podem ser incluídas neste. A implementação do jogo foi realizada de forma a tirar partido de várias funcionalidades do LiNGS, nomeadamente a sincronização contextual, restauro de sessões e delegação de processamento para o servidor.

Com a implementação terminada foi possível testar o sistema LiNGS. Os primeiros testes realizados foram feitos de forma a garantir a operabilidade do sistema. O jogo Striker foi exportado para dois dispositivos móveis ligados ao servidor, a executar num computador, via rede Wifi e foram realizadas várias partidas de forma a comprovar o funcionamento do sistema. A sincronização dos elementos de jogo é feita com sucesso e a lógica é transmitida pelos vários clientes, e de forma a permitir o correto funcionamento do jogo.

Os próximos testes envolveram testar funcionalidades específicas do sistema LiNGS para comprovar a sua eficácia. Os testes realizados demonstraram que os componentes desenhados e implementados no LiNGS conseguem obter uma eficácia de cerca de 50% relativamente ao sistema com os componentes testados desativados. Esta eficácia é notória na frequência de transmissão de dados e na quantidade de dados transmitidos, o que beneficia bastante os clientes e a rede utilizada para comunicação.

Foram ainda realizados testes com utilizadores, que mais uma vez comprovaram a funcionalidade do sistema e ajudaram a compreender o funcionamento deste em situações de utilização real. Estes testes mostraram alterações mínimas no comportamento do sistema de acordo com o tipo de jogo aplicado pelos utilizadores, sem prejuízo para o jogo. Foram encontrados vários problemas relacionados com a implementação do jogo e com a utilizam do sistema LiNGS em redes partilhadas, os quais foram resolvidos. Refiro ainda que estes problemas apesar de não estarem diretamente relacionados com a *framework* implementada, foram resolvidos para permitir a realização dos testes com utilizadores.

# Capítulo 7

# Conclusões e trabalho futuro

A execução deste projeto tinha como principal objetivo o desenho e implementação de uma *framework* para suporte a jogos multijogador em rede para dispositivos móveis. Para tal o primeiro passo consistiu na realização de um levantamento de informações relativas a estes dispositivos e aplicações. Foi realizado um trabalho de pesquisa sobre o desenvolvimento para dispositivos móveis de forma a entender o estado atual do desenvolvimento para estes dispositivos. A conclusão obtida indica que o desenvolvimento para dispositivos móveis é atualmente bastante popular sendo que as lojas de aplicações dos sistemas operativos móveis estão inundadas com aplicações e principalmente jogos. As lojas digitais trazem facilidade e comodidade aos programadores de aplicações e aos utilizadores finais tornando o processo de distribuição e instalação de aplicações bastante simples. Os programadores e empresas podem facilmente publicar aplicações para milhões de utilizadores e estes podem facilmente descarregar e utilizar essas aplicações. Desta forma, com tanta concorrência, torna-se necessário implementar aplicações de forma ágil, com qualidade e com funcionalidades superiores a aplicações existentes. As *frameworks* são um fator importante, pois permitem o desenvolvimento de aplicações reduzindo o custo e tempo de desenvolvimento.

Foi ainda realizada uma análise do desenvolvimento de jogos com capacidades multijogador em rede. O funcionamento deste tipo de jogos é complexo e normalmente estes requerem vários sistemas sofisticados de forma a obter comportamentos assertivos. O desenvolvimento deste tipo de jogos sofreu várias evoluções ao longo dos anos de acordo com os dispositivos e tecnologias para os quais eram desenvolvidos. Da pesquisa realizada foram encontradas várias *frameworks* com suporte a jogos multijogador em rede, mas estas apenas suportam jogos em dispositivos fixos, tais como computadores pessoais e consolas de jogos. Existem

também algumas implementações para dispositivos móveis, no entanto as potencialidades das novas tecnologias dos dispositivos não são exploradas por estas *frameworks*. Várias técnicas utilizadas não se adequam aos jogos e não resolvem os problemas de forma concreta, deixando o tratamento dos problemas para a implementação do jogo. Em certas situações as implementações existentes conseguem obter bons resultados, permitindo reduzir o consumo energético e reduzir os efeitos dos problemas de rede. Noutros casos os problemas implicam a perda de funcionalidades no jogo.

A especificação do sistema LiNGS foi realizada no sentido de criar um sistema totalmente genérico (para suportar qualquer tipo de jogo) que realize as operações de tratamento de problemas e que ajude a reduzir as consequências da sua operação no seu ambiente. O sistema especificado reflete assim exatamente estes fatores e ainda outro que é considerado bastante importante – a facilidade de utilização e integração do sistema no ambiente real.

A especificação realizada estabelece a arquitetura do sistema e o seu comportamento deixando vários fatores dependentes da implementação realizada. O objetivo não passa por construir um sistema fechado mas sim um sistema que possa ser implementado para utilizar em qualquer plataforma/sistema e que suporte qualquer tipo de jogo.

De acordo com a especificação realizada foi implementado o sistema LiNGS para a *framework* Unity. Esta *framework* base foi escolhida de acordo com as funcionalidades que disponibiliza e por ser atualmente uma das mais utilizadas. Esta escolha e a utilização da .NET Framework revelaram-se boas opções, pois permitiram realizar a implementação e integração do sistema LiNGS de forma ágil.

A implementação do sistema seguiu rigorosamente a especificação previamente realizada e focou-se em fornecer as funcionalidades necessárias e permitir uma fácil integração com jogos. A estrutura do sistema implementado é bastante modular e toda a implementação está organizada por componentes. A implementação está documentada e a sua organização permite que possam ser realizadas alterações de forma simples.

Como comprovação do funcionamento do sistema, foram realizados vários testes que implicaram o desenvolvimento de um jogo multijogador em rede, denominado de Striker, para utilizar o sistema LiNGS. Os testes principais consistiram em comprovar o funcionamento do LiNGS como um sistema multijogador que permite a sincronização de dados entre os

vários dispositivos. Foram também realizados testes que permitiram comprovar o funcionamento do restauro de sessão em casos de problemas da rede. O sistema LiNGS superou os testes realizados, visto que a sincronização dos dados de jogo é feita de forma correta e o restauro da sessão de um jogador, depois de uma quebra de ligação, é realizado com sucesso.

De forma a comprovar também a eficácia de vários componentes do sistema LiNGS, nomeadamente os sistemas de redução do consumo de energia e redução de tráfego na rede de comunicação, foram realizadas análises a componentes internos de forma a entender as suas ações no funcionamento do LiNGS. Os vários componentes do sistema conseguem reduzir o número de mensagens enviadas em cerca de 50%. Esta ação é bastante importante pois evita congestionamentos na rede de comunicação e permite que os componentes de comunicação estejam em repouso durante mais tempo, evitando assim o desperdício de energia.

A quantidade de dados enviados é também significativamente reduzida. O jogo de teste obteve reduções na quantidade de dados transmitidos na ordem dos 40% com os componentes de sincronização contextual ativos. Esta redução significativa permite reduzir o processamento nos vários dispositivos móveis e, por conseguinte, reduzir o consumo energético.

É assim possível concluir que o desenvolvimento de jogos multijogador em rede para dispositivos móveis é atualmente possível utilizando as tecnologias correntes. Mais do que possível, este tipo de jogos é atualmente fiável, e os grandes problemas destes podem ser limitados e resolvidos com sistemas com comportamento semelhante ao LiNGS.

Os jogos multijogador para dispositivos móveis são escassos nas lojas digitais mas a possibilidade do seu desenvolvimento, demonstrada neste projeto, pode trazer esse tipo de funcionalidade a mais jogos e pode ainda criar novas formas de interação que não eram possíveis até ao momento. A implementação do jogo Striker, realizado como forma de teste ao sistema LiNGS, demonstra que o desenvolvimento deste tipo de jogos é possível de ser realizado.

O resultado da implementação do sistema LiNGS está ainda disponibilizado de forma aberta sobre uma licença *open-source* e com documentação associada [105]. A licença *open-source* utilizada é a MIT [106] que permite a modificação livre do projeto. A necessidade de personalização é algo normal no desenvolvimento de jogos, desta forma o sistema LiNGS é lançado de forma aberta permitindo que qualquer programador possa realizar alterações de acordo com as suas necessidades.

**Trabalho Futuro**

A corrente implementação do sistema LiNGS permite realizar as funcionalidades pretendidas com bons resultados. No entanto, o aperfeiçoamento do sistema é algo que pode e deve ser realizado. A exploração de outras técnicas para melhorar o sistema e introduzir novas funcionalidades é algo bastante importante no contexto atual onde as tecnologias sofrem evoluções constantes.

A integração do sistema LiNGS num sistema central, *master-server*, que agilize o processo de ligação dos utilizadores aos servidores de jogo, é uma funcionalidade que deve ser explorada. Atualmente, a especificação atual não prevê qualquer tipo de funcionalidade para a descoberta de jogos na rede. Assim, os utilizadores têm de inserir os dados (IP e porto) do servidor de forma manual ou então a implementação do jogo tem de realizar esta tarefa. A inclusão de um sistema central permitiria que os utilizadores se ligassem e que os servidores de jogos disponíveis fossem anunciados para os clientes de forma automática.

Deverão ser ainda desenvolvidas novas funcionalidades para o sistema LiNGS, tais como suporte a RPC (*Remote Procedure Call*). A inclusão de RPCs permite realizar a invocação de métodos do cliente a partir do servidor, ou vice-versa. Esta funcionalidade permite agilizar a implementação de jogos, já que os programadores podem facilmente invocar funções remotas em qualquer cliente de forma direta, sem necessidade de troca de mensagens manuais a indicar que as funções devem ser invocadas.

Em paralelo ao desenvolvimento de novas funcionalidades é ainda necessário analisar com mais detalhe os vários pormenores da implementação multijogador em rede de forma a identificar os pontos problemáticos e quais as melhorias que se podem introduzir no sistema.

# Bibliografia


[1] Nielsen. Play before work games most popular mobile app category in US. http://www.nielsen.com/us/en/newswire/2011/games-most-popular-mobile-app-category.html, Publicado em 2011. Acedido a 22 de abril de 2014.

[2] Microsoft. Introduction to the C# language and the .NET Framework. http://msdn.microsoft.com/en-us/library/z1zx9t92.aspx. Acedido a 19 de agosto de 2014.

[3] Adam Redsell. SEGA: A soothsayer of the games industry. http://www.ign.com/articles/2012/05/20/sega-a-soothsayer-of-the-games-industry, Publicado em 20 de Maio de 2012. Acedido a 26 de agosto de 2014.

[4] Blizzard Entertainment. World of warcraft. http://us.battle.net/wow/en/. Acedido a 26 de agosto de 2014.

[5] Levent Ozler. Taneli armanto: Snake creator receives special recognition. http://www.dexigner.com/news/4785, Publicado em 16 de Junho de 2005. Acedido a 26 de agosto de 2014.

[6] Gartner. Gartner says smartphone sales grew 46.5 percent in second quarter of 2013 and exceeded feature phone sales for first time. http://www.gartner.com/newsroom/id/2573415, Publicado em 14 de Agosto de 2013. Acedido a 22 de abril de 2014.

[7] Entertainment Software Association. The evolution of mobile games. http://www.theesa.com/games-improving-what-matters/The_Evolution_of_Mobile_Games.pdf, Publicado em 2012. Acedido a 22 de abril de 2014.

[8] Clint Smith and Daniel Collins. *3G Wireless Networks*. McGraw-Hill Telecom Professional, 2002.

[9] Apple. Finger tips. http://manuals.info.apple.com/MANUALS/0/MA617/en_US/iPhone_Finger_Tips_Guide.pdf, Publicado em 2009. Acedido a 22 de abril de 2014.

[10] Apple. iPhone. https://www.apple.com/iphone/. Acedido a 22 de abril de 2014.

[11] Apple. App store. http://www.apple.com/iphone-5s/app-store/. Acedido a 22 de abril de 2014.

[12] Microsoft. Xbox LIVE indie games. http://xbox.create.msdn.com/en-US. Acedido a 22 de abril de 2014.

[13] Microsoft. Arcade xbox games store. http://marketplace.xbox.com/en-US/Games/XboxArcadeGames?SortBy=ReleaseDate. Acedido a 22 de abril de 2014.

[14] Google. Google play. https://play.google.com/store. Acedido a 28 de junho de 2014.

[15] Google. Android. http://www.android.com/. Acedido a 28 de junho de 2014.

[16] Microsoft. Windows phone store. http://www.windowsphone.com/en-US/store. Acedido a 28 de junho de 2014.

[17] Microsoft. Windows phone. http://www.windowsphone.com/en-us. Acedido a 28 de junho de 2014.

[18] Google. Get the android sdk. http://developer.android.com/sdk/index.html?hl=sk. Acedido a 17 de maio de 2014.

[19] BlackBerry. Runtime for android apps. http://developer.blackberry.com/android/. Acedido a 17 de maio de 2014.

[20] IEEE. POSIX - austin joint working group. http://standards.ieee.org/develop/wg/POSIX.html. Acedido a 26 de abril de 2014.

[21] Google. Introduction to android. http://developer.android.com/guide/index.html. Acedido a 30 de abril de 2014.

[22] Apple. Develop apps for ios. https://developer.apple.com/technologies/ios/. Acedido a 30 de abril de 2014.

[23] Box2D. Box2d - a 2D physics engine for games. http://box2d.org/. Acedido a 30 de abril de 2014.

[24] Microsoft. Microsoft .net. http://www.microsoft.com/net. Acedido a 30 de abril de 2014.

[25] Marmalade. Marmalade sdk. https://www.madewithmarmalade.com/index. Acedido a 30 de abril de 2014.

[26] Unity. The unity editor. http://unity3d.com/unity/workflow/integrated-editor. Acedido a 30 de abril de 2014.

[27] .NET Framework Blog. How your feedback is shaping .NET. http://blogs.msdn.com/b/dotnet/archive/2014/04/09/how-your-feedback-is-shaping-net.aspx, Publicado em 2014. Acedido a 27 de abril de 2014.

[28] Android. Google play game services. http://developer.android.com/google/play-services/games.html. Acedido a 30 de abril de 2014.

[29] Marmalade Developer. Marmalade supports all open native operating systems. https://developer.madewithmarmalade.com/develop/supported-platforms. Acedido a 30 de abril de 2014.

[30] Microsoft. Visual studio. http://www.visualstudio.com/. Acedido a 30 de abril de 2014.

[31] Apple. Xcode. https://developer.apple.com/xcode/. Acedido a 30 de abril de 2014.

[32] Marmalade. Showroom. https://www.madewithmarmalade.com/showroom. Acedido a 30 de abril de 2014.

[33] Epic Games. Unreal engine. https://www.unrealengine.com/. Acedido a 30 de abril de 2014.

[34] Tim Sweeney Epic Games. Welcome to unreal engine 4. https://www.unrealengine.com/blog/welcome-to-unreal-engine-4, Publicado em 19 Março 2014. Acedido a 30 de abril de 2014.

[35] Epic Games. Unreal engine features. https://www.unrealengine.com/products/unreal-engine-4. Acedido a 30 de abril de 2014.

[36] Epic Games. Unreal engine showcase. https://www.unrealengine.com/showcase. Acedido a 30 de abril de 2014.

[37] Epic Games. UE4 editor. https://www.youtube.com/watch?v=QMsFxzYzFJ8. Acedido a 30 de abril de 2014.

[38] Epic Games. Unreal engine frequently asked questions. https://www.unrealengine.com/faq. Acedido a 30 de abril de 2014.

[39] Epic Games. Unreal engine frequently asked questions - billing. https://www.unrealengine.com/faq#billing. Acedido a 30 de abril de 2014.

[40] Epic Games. Unreal engine showcase. https://www.unrealengine.com/showcase?category=Mobile. Acedido a 30 de abril de 2014.

[41] Corona. Corona sdk. http://coronalabs.com/products/corona-sdk/. Acedido a 30 de abril de 2014.

[42] Sublime HQ. Sublime text. http://www.sublimetext.com/. Acedido a 30 de abril de 2014.

[43] Corona. Corona editor. http://coronalabs.com/products/editor/. Acedido a 30 de abril de 2014.

[44] Corona. Why is corona 10x faster? http://coronalabs.com/products/corona-sdk/corona-is-10x-faster/. Acedido a 30 de abril de 2014.

[45] Corona Labs. Corona showcase. http://coronalabs.com/blog/topics/showcase/. Acedido a 30 de abril de 2014.

[46] Dice. How unity3d became a game-development beast. http://news.dice.com/2013/06/03/how-unity3d-become-a-game-development-beast/, Publicado em 3 Junho 2013. Acedido a 30 de abril de 2014.

[47] Unity. Unity - multiplatform. http://unity3d.com/unity/multiplatform. Acedido a 30 de abril de 2014.

[48] Unity. Unity 4.3: 2D game development overview. http://blogs.unity3d.com/2013/11/12/unity-4-3-2d-game-development-overview/, Publicado em 12 Novembro 2013. Acedido a 30 de abril de 2014.

[49] Unity. Unity - plugins. http://docs.unity3d.com/Documentation/Manual/Plugins.html. Acedido a 30 de abril de 2014.

[50] MonoDevelop. Monodevelop. http://monodevelop.com/. Acedido a 30 de abril de 2014.

[51] CodeHaus. Boo. http://boo.codehaus.org/. Acedido a 30 de abril de 2014.

[52] Unity. Unity - documentation - using scripts. http://docs.unity3d.com/412/Documentation/Manual/Scripting.html. Acedido a 30 de abril de 2014.

[53] Unity. Made with unity. http://unity3d.com/showcase/gallery/games. Acedido a 30 de abril de 2014.

[54] Cocos2D. Cocos2d-iphone. http://www.cocos2d-iphone.org/. Acedido a 30 de abril de 2014.

[55] Cocos2D. Cocos2d-x. http://www.cocos2d-x.org/. Acedido a 30 de abril de 2014.

[56] Cocos2D. Cocos2d-x manual. http://www.cocos2d-x.org/wiki/Cocos2d-x. Acedido a 30 de abril de 2014.

[57] The Brenwill Workshop. Cocos 3d. http://brenwill.com/cocos3d/. Acedido a 30 de abril de 2014.

[58] Cocos2D. Cocostudio. http://www.cocos2d-x.org/wiki/CocoStudio. Acedido a 30 de abril de 2014.

[59] Thorbjorn Lindeijer. Tiled. http://www.mapeditor.org/. Acedido a 30 de abril de 2014.

[60] CodeAndWeb. Texture packer. http://www.codeandweb.com/texturepacker. Acedido a 30 de abril de 2014.

[61] Cocos2d-x. Showcase. http://cocos2d-x.org/games. Acedido a 30 de abril de 2014.

[62] Microsoft. XNA game studio 4.0 refresh. http://msdn.microsoft.com/en-us/library/bb200104.aspx. Acedido a 30 de abril de 2014.

[63] Monogame. Monogame. http://www.monogame.net/. Acedido a 30 de abril de 2014.

[64] Xamarin. Mono. http://www.mono-project.com/Main_Page. Acedido a 30 de abril de 2014.

[65] Xamarin. Xamarin for visual studio. https://xamarin.com/visual-studio. Acedido a 30 de abril de 2014.

[66] Monogame. About monogame. http://www.monogame.net/about/. Acedido a 30 de abril de 2014.

[67] Monogame. Monogame - github. https://github.com/mono/MonoGame. Acedido a 30 de abril de 2014.

[68] Brookhaven National Laboratory. The first video game? http://www.bnl.gov/about/history/firstvideo.php. Acedido a 30 de abril de 2014.

[69] Cisco. Appletalk. http://docwiki.cisco.com/wiki/AppleTalk. Acedido a 30 de abril de 2014.

[70] Atari Protos. Midi maze. http://www.atariprotos.com/8bit/software/midimaze/midimaze.htm. Acedido a 30 de abril de 2014.

[71] Andrew Swift. An introduction to MIDI. http://www.doc.ic.ac.uk/~nd/surprise_97/journal/vol1/aps2/. Acedido a 30 de abril de 2014.

[72] Moby Games. Doom. http://www.mobygames.com/game/doom. Acedido a 30 de abril de 2014.

[73] John Carmack. Development on doom classic. http://web.archive.org/web/20110725100250/http://www.idsoftware.com/doom-classic/doomdevelopment.htm. Acedido a 30 de abril de 2014.

[74] Novel. The ipx protocol. http://www.novell.com/documentation/nw6p/?page=/documentation/nw6p/ipx_enu/data/hjyh8yg8.html. Acedido a 30 de abril de 2014.

[75] Doom Wiki. Doom in workplaces. http://doom.wikia.com/wiki/Doom_in_workplaces. Acedido a 30 de abril de 2014.

[76] id Software. Quake. http://www.idsoftware.com/games/quake/quake. Acedido a 30 de abril de 2014.

[77] John Carmack. Quake. http://fabiensanglard.net/quakeSource/johnc-log.aug.htm. Acedido a 30 de abril de 2014.

[78] Jesse Aronson. Dead reckoning: Latency hiding for networked games. http://www.gamasutra.com/view/feature/3230/dead_reckoning_latency_hiding_for_.php. Acedido a 30 de abril de 2014.

[79] Bradley Mitchell. Bluetooth. http://compnetworking.about.com/cs/bluetooth/g/bldef_bluetooth.htm. Acedido a 30 de abril de 2014.

[80] Nokia. Games over bluetooth: Recommendations to game developers. http://fivedots.coe.psu.ac.th/Software.coe/J2ME/bluetooth/Nokia%20Articles/Games_over_B_tooth_v1_0_en.pdf, 2003. Acedido a 06 de setembro de 2014.

[81] 3GPP. LTE. http://www.3gpp.org/technologies/keywords-acronyms/98-lte. Acedido a 30 de abril de 2014.

[82] IEEE Computer Society. IEEE STANDARD 802.11n - 2009. *IEEE Standard for Information technology – Local and metropolitan area networks*, 2009.

[83] Christian Hiedels, Christian Hoff, Steffen Rothkugel, and Ulf Wehling. Ubisettlers - a dynamically adapting mobile P2P multiplayer game for hybrid networks. *IEEE – International Conference on Pervasive Computing and Communications Workshops*, pages 109–103, 2007.

[84] Yung-Wei Kao, Pin-Yin Peng, Sheau-Ling Hsieh, and Shyan-Ming Yuan. A client framework for massively multiplayer online games on mobile devices. *IEEE – International Conference on Convergence Information Technology*, pages 48–53, 2007.

[85] Ahmad Hoirul Basori, Riyanarto Sarno, and Surya Widyanto. The development of 3D multiplayer mobile racing games based on 3D photo satellite map. *IEEE – 5th IFIP International Conference on Wireless and Optical Communications Networks*, pages 1–5, 2008.

[86] Alf Inge Wang, Eivind Sorteberg, Martin Jarrett, and Anne Marte Hjemas. Issues related to mobile multiplayer real-time games over wireless networks. *IEEE – International Symposium on Collaborative Technologies and Systems*, pages 237–246, 2008.

[87] Bhojan Anand, Karthik Thirugnanam, Le Thanh Long, and Duc-Dung Pham. ARIVU: Power-aware middleware for multiplayer mobile games. *IEEE – 9th Annual Workshop on Network and Systems Support for Games (NetGames)*, pages 1–6, 2010.

[88] Bhojan Anand, Jeena Sebastian, Soh Yu Ming, Akhihebbal L. Ananda, Mun Choon Chan, and Rajesh Krishna Balan. PGTP: Power aware game transport protocol for multi-player mobile games. *IEEE – International Conference on Communications and Signal Processing*, pages 399–404, 2011.

[89] IETF. Internet layer - 3.3.2 reassembly. `http://tools.ietf.org/html/rfc1122#page-56`. Acedido a 12 de julho de 2014.

[90] Unity3D. Public relations. `http://unity3d.com/public-relations`. Acedido a 19 de agosto de 2014.

[91] Unity3D. Unity manual. `http://docs.unity3d.com/Manual/index.html`. Acedido a 19 de agosto de 2014.

[92] Microsoft. Socket class. `http://msdn.microsoft.com/en-us/library/system.net.sockets.socket%28v=vs.90%29.aspx`. Acedido a 19 de agosto de 2014.

[93] Microsoft. Reflection overview. `http://msdn.microsoft.com/en-us/library/f7ykdhsy%28v=vs.90%29.aspx`. Acedido a 19 de agosto de 2014.

[94] Microsoft. C# programming guide. `http://msdn.microsoft.com/en-us/library/67ef8sbd.aspx`. Acedido a 19 de agosto de 2014.

[95] Microsoft. Using an asynchronous client socket. http://msdn.microsoft.com/en-us/library/bbx2eya8%28v=vs.90%29.aspx. Acedido a 19 de agosto de 2014.

[96] SourceMaking. Facade design pattern. http://sourcemaking.com/design_patterns/facade. Acedido a 19 de agosto de 2014.

[97] University of Waterloo Douglas Wilhelm Harder. Extrapolation. https://ece.uwaterloo.ca/~dwharder/NumericalAnalysis/06LeastSquares/extrapolation/complete.html. Acedido a 19 de agosto de 2014.

[98] Microsoft. XmlSerializer Class. http://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlserializer%28v=vs.110%29.aspx. Acedido a 19 de agosto de 2014.

[99] Microsoft. Gzipstream class. http://msdn.microsoft.com/en-us/library/system.io.compression.gzipstream%28v=vs.90%29.aspx. Acedido a 19 de agosto de 2014.

[100] Microsoft. Language-integrated query (LINQ). http://msdn.microsoft.com/en-us/library/bb397926%28v=vs.90%29.aspx. Acedido a 19 de agosto de 2014.

[101] Microsoft. Lambda expressions. http://msdn.microsoft.com/en-us/library/bb397687.aspx. Acedido a 19 de agosto de 2014.

[102] Carnegie Mellon University Andrew W. Moore, School of Computer Science. A* heuristic search. http://www.cs.cmu.edu/~cga/ai-course/astar.pdf. Acedido a 21 de agosto de 2014.

[103] A10 Networks. Traffic shaping. http://www.a10networks.com/glossary/traffic_shaping.php. Acedido a 22 de agosto de 2014.

[104] Software Testing. Black box testing. http://softwaretestingfundamentals.com/black-box-testing/. Acedido a 06 de setembro de 2014.

[105] Valter Costa – Github. LiNGS - Lightweight Networked Game System. https://github.com/valterc/lings. Acedido a 06 de setembro de 2014.

[106] Open Source Initiative. The MIT License (MIT). http://opensource.org/licenses/MIT. Acedido a 06 de setembro de 2014.

# Apêndice A

# Diagramas de classes

## A.1 Diagrama de classes da implementação da parte comum da *framework* LiNGS

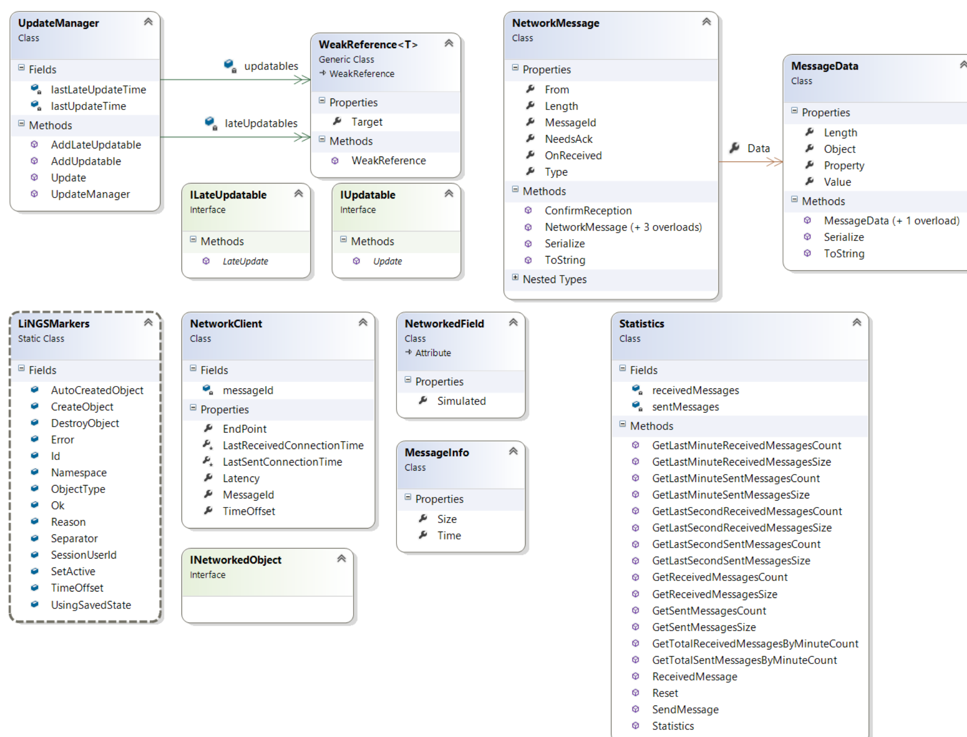


Figura A.1: Diagrama de classes da implementação comum da *framework* LiNGS.

## A.2 Diagrama de classes da implementação da parte cliente da *framework* LiNGS

Figura A.2: Diagrama de classes da implementação da parte cliente da *framework* LiNGS. (Versão em alta definição desta imagem está disponível na entrega digital do documento)

## A.3 Diagrama de classes da implementação da parte servidor da *framework* LiNGS

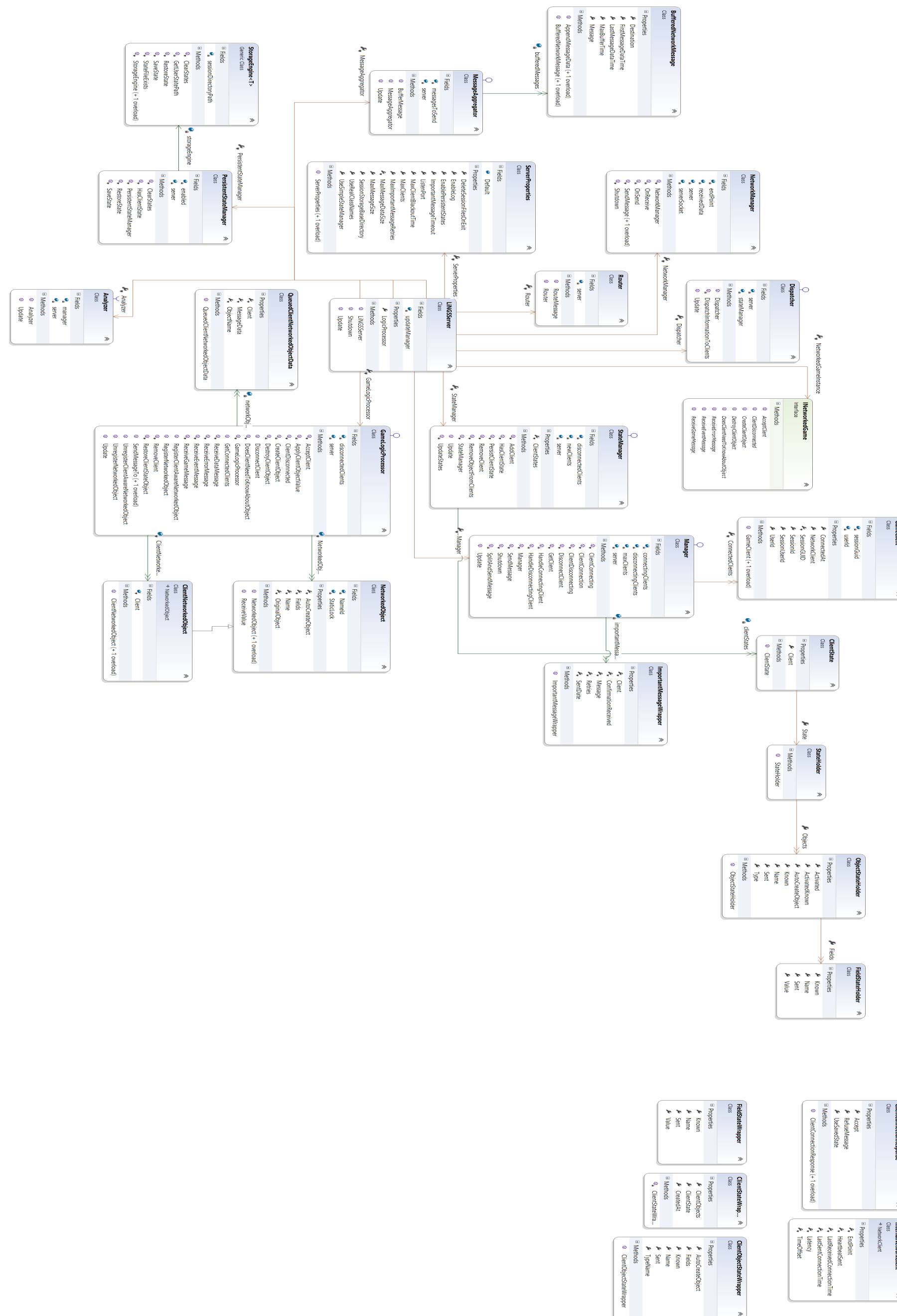

Figura A.3: Diagrama de classes da implementação da parte servidor da *framework* LiNGS. (Versão em alta definição desta imagem está disponível na entrega digital do documento)