Implementação de uma arquitetura multijogador num videojogo (The Abyss)

um Projecto/Dissertação da autoria de

João Henrique Queirós Peneda

e supervisionado por

Cláudia Sofia Borlido de Freitas

Professor Assistente, ISMAI

Agostinho Gil Teixeira Lopes

Professor Auxiliar, ISMAI

Mestrado em Tecnologias da Informação, Comunicação e Multimédia na Universidade da Maia - ISMAI (ISMAI)

19 September, 2022

Resumo

No decorrer deste projeto, é descrita a história por de trás dos videojogos, o que estes são, como evoluíram, os seus géneros e como se planeia programar um jogo com recurso ao Unity Engine e ao Netcode for GameObjects (solução de Networking para Unity). Exploram-se também alguns motores de jogos disponíveis no mercado e também apresenta-se uma breve história da evolução desses motores e algumas especificações dos mesmos com comparação entre eles. Dá-se mais ênfase à parte multiplayer do projeto, que ferramentas estão disponíveis no mercado, e algumas diferenças entre elas. Fala-se também nos diferentes modos possíveis de operação de uma infraestrutura multijogador com as devidas funcionalidades de cada modo.

Palavras-chave: Unity. NetCode for Gameobjects. multijogador. Motor.

Abstract

During the course of this project it will be discussed what videogames are, how they evolved, their genres and how it is planned to program a game using the Unity Engine and Netcode for GameObjects (Networking solution for Unity). Some game engines available in the market are also explored and a brief history of the evolution of these engines is presented, as well as some of their specifications with comparison between them. More emphasis is given to the multiplayer part of the project, which tools are available on the market, and some differences between them. It's also talked about the different possible modes of operation of a multiplayer infrastructure with the proper functionalities of each mode.

Keywords: Unity. NetCode for Gameobjects. Multiplayer. Engine.

Agradecimentos

Gostaria de agradecer aos meus pais que sempre estiveram ao meu lado durante o percurso académico e que mesmo nas dificuldades me apoiaram. Agradeço também aos meus orientadores Cláudia Sofia Borlido de Freitas e Agostinho Gil Teixeira Lopes por se mostrarem sempre disponíveis para ajudar. Agradeço, também, ao professor Pedro Pinto, pela disponibilidade e apoio durante as aulas.

Por fim gostaria de agradecer aos meus avôs maternos Conceição Alves e Bernardino Queirós por estarem sempre presentes na minha vida, por se preocuparem sempre com o meu bem estar e por me ajudarem em momentos difíceis.

A todos mencionados OBRIGADO por tudo.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Excertos de Código

Siglas

API Application Programming Interface

Co-Op Cooperative

Epic Games Epic Games, Inc

FPS First Person Shooter

frames/s Frames per second

GPU Graphical Processing Unit

ISMAI Universidade da Maia - ISMAI

LAN Local Area Network

MIT Massachusetts Institute of Technology

MM Match Making

MMO Massive Multiplayer Online

MMORPG Massive Multiplayer Online Role Playing Game

MTICM Mestrado em Tecnologias de Informação Comunicação e Multimédia

NGO Netcode for GameObjects

NPC Non Playable Character

P2P Peer-to-Peer

PUN Photon Unity Networking

RPC Remote Procedure Call

RPG Role Playing Game

SSD Solid State Drive

TPS Third Person Shooter

Capítulo 1

Introdução

O tema deste projeto é o desenvolvimento da infraestrutura multijogador do jogo "Abyss". Para o desenvolvimento do jogo a nossa equipa foi dividido em dois artistas e dois programadores, em que cada um ficou encarregue de uma parte especifica do desenvolvimento, como referido na figura 1.1. A visão geral para o jogo é ser multijogador *Cooperative (Co-Op)* com elementos de *Role Playing Game (RPG)* em estilo *Third Person Shooter (TPS)*.

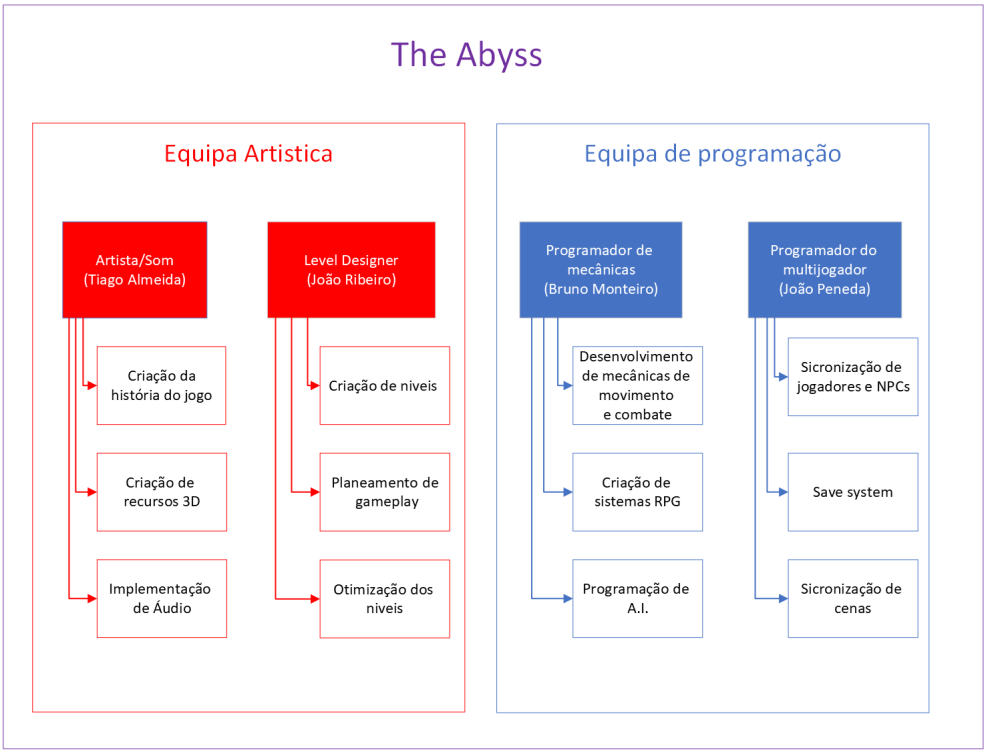


Figura 1.1: Divisão de tarefas

1.1 Definição de Videojogos

Segundo Wolf [1] o termo videojogo parece bastante simples, mas o seu significado tem variado muito ao longo dos anos e de um lugar para outro. Wolf começa por observar as duas palavras presentes; o seu estatuto de "jogo"e a sua utilização da "tecnologia de vídeo". O autor refere ainda que o termo "jogos de computador" também é usado por vezes como sinónimo de videojogos. Ainda no seguimento do pensamento de Wolf, embora as definições de "jogo"possam variar, os elementos que seriam de esperar encontrar num jogo são conflitos (contra um adversário ou circunstâncias), regras (determinar o que pode e não pode ser feito e quando), o uso de alguma habilidade do jogador (como habilidade, estratégia ou sorte) e algum tipo de recompensa (como ganhar vs. perder, ou alcançar a pontuação mais alta ou o tempo mais rápido para completar uma tarefa). Todos estes estão geralmente presentes em videojogos, embora em graus e formas diferentes. Nos videojogos, a atribuição de pontos, o respeito às regras e a exibição visual do jogo são todos monitorizados por um computador em vez de humanos. O computador também pode controlar o oponente ou personagens dentro de um jogo, tornando-se um participante e também um árbitro. Grande parte dos videojogos são jogos single-player (de um só jogador) em que o jogador enfrenta adversários e situações controladas pelo computador. Devido à velocidade quase instantânea a que um computador pode processar a entrada (input) do utilizador, responder com reacções e exibir a acção no ecrã, os videojogos são muitas vezes concebidos para exigir uma acção rápida e reflexos elevados, tal como no desporto. Nos olhos de Keith Feinstein, dono do museu Videotopia, de acordo com a entrevista conduzida por Wolf [1], o computador deve ser mais do que um árbitro ou encenador que controla o mundo dos videojogos, mas um adversário activo que compete com o jogador humano, isto é possível pois ao atribuir uma identidade ao jogador e criar uma situação 'um-a-um' contra o computador dentro do jogo, a competição torna-se possível e emocional.

1.2 Problemática e Motivação

Para alcançar os objectivos, será necessário ultrapassar várias dificuldades, para tentar mitigar o problema de tempo, a equipa concordou em fazer apenas um demo, contendo

este pelo menos um nível terminado e com as funcionalidades online funcionais. Um problema que enfrentamos é o uso de ferramentas *"bleeding-edge"* (ferramentas novas, ou sem grande documentação) como é o caso do Netcode for GameObjects (NGO) que opera em vários modos, entre eles está o modo clássico de cliente-servidor, Peer-to-Peer (P2P), ou até mesmo Cliente-Anfitrião que requerem um pouco de experimentação para fazer a sua implementação.

Devido ao facto de o jogo ser online, existem também outros problemas espectáveis como a latência dos jogadores e sincronização dos ativos em cena. Esta comunicação entre os jogadores em cena tem de ser capaz de sustentar comunicações constantes de áudio, movimentos, interacções, animações, etc., de forma a não quebrar a imersão dos jogadores.

A boa implementação do sistema multijogador é crucial pois a programação das mecânicas depende bastante da qualidade do sistema de rede pois qualquer falha no sistema de rede pode causar uma grande perda de imersão no jogo.

Têm-se visto uma evolução nos jogos ano após ano, mas nunca se soube como os criadores eram capazes de criar estes jogos ou como estes poderiam controlar cada pequeno detalhe no comportamento das personagens envolvidas. Ser capaz de saber e entender o que estava a acontecer por de trás de um jogo, foi o que motivou a desenvolver este projeto.

1.3 Objectivos

Neste projeto, ficaram definidos os seguintes objetivos:

- Implementar um jogo do estilo *Co-Op* para quatro pessoas com recurso ao *NGO*.
- Desenvolver um servidor que os jogadores podem hospedar na nuvem ou até mesmo em casa .
- Desenvolver um sistema para a sincronização dos aspetos e armas equipadas pelos jogadores.
- Desenvolver um sistema para a sincronização de ataques efetuados pelos jogadores e Non Playable Character (NPC)'s.
- Desenvolver um sistema para guardar o progresso dos jogadores em rede.

- Desenvolver um sistema para a sincronização de itens adquiridos nos combates (*Drops*).

1.4 Organização

O capitulo 1 dá-nos uma breve definição de videojogos, explica o conceito do projeto, menciona a problemática e objetivos e contêm também a organização de capítulos.

No capitulo 2 foi realizada uma pesquisa bibliográfica acerca dos componentes para uma implementação de um jogo em multijogador. Este capitulo descreve o que é um motor de jogo, os motores mais comuns e as suas funcionalidades, algumas diferenças entre uma arquitetura multijogador local e uma em rede com implementações prévias e por fim as ferramentas mais conhecidas para a implementação de uma arquitetura multijogador nos motores mencionados.

O capitulo 3 descreve o método de investigação usado no desenvolvimento do projeto, da implementação do projeto, dos recursos necessários para o desenvolvimento, os testes que irão ser realizados e do cronograma com a distribuição de tarefas.

O capitulo 4 menciona as ferramentas usadas no decorrer do projeto, as suas configurações, os problemas encontrados e soluções para os mesmos e a lógica operacional do jogo a desenvolver.

O capitulo 5 demonstra os testes efetuados ao desempenho do jogo e os comentários obtidos no inquérito efetuado.

No capitulo 6 é apresentada a conclusão do projeto e o possível trabalho futuro.

Capítulo 2

Estado da Arte

Neste capítulo será apresentada a investigação relacionada com os componentes para uma implementação de um jogo multijogador. Este capitulo descreve o que é um motor de jogo, os motores mais comuns e as suas funcionalidades, algumas diferenças entre uma arquitetura multijogador local e uma em rede com implementações prévias e por fim as ferramentas mais conhecidas para a implementação de uma arquitetura multijogador nos motores mencionados.

São mencionados também os componentes existentes numa arquitetura multijogador como as interações, as frequências dessas interações e os seus tipos existentes, dinâmicas de grupo, boas práticas e jogabilidade assimétrica.

2.1 Constituinte principal de um videojogo (Motor)

Os motores de jogo são uma parte vital da produção de jogos, mas existe uma imprecisão nas definições sobre os limites dos componentes num motor de jogo e as restantes ferramentas de produção usadas num ambiente de desenvolvimento de jogos [2]. Produzir um jogo inclui uma infinidade de competências. Programadores, artistas gráficos, engenheiros de som, designers narrativos e engenheiros de jogos são todas competências com várias necessidades baseadas na produção e tarefas relacionadas com o trabalho num cenário de desenvolvimento de jogos [3, 4, 5, 6]. Uma vez que os jogos não são apenas um produto para entretenimento, mas também um complexo sistema técnico com o objetivo de providenciar ao utilizador final uma experiência satisfatória, muitas vezes divertida,

o desenvolvimento de jogos é uma tarefa complexa onde a engenharia do sistema e as competências criativas na arte e no design devem ser tratadas na mesma infraestrutura de projeto [2].

A cadeia de distribuição tradicional (lojas físicas) na indústria dos *Triple-A* [7, 8] não é hoje em dia a única forma de lançar um jogo. Isto abriu caminho para o setor dos *Indie*, onde equipas mais pequenas com orçamentos mais pequenos podem sobreviver através da venda de jogos [9, 10]. Conforme refere [2], desde o advento da distribuição digital, mesmo as produções de pequena escala (*Indie*) têm a possibilidade de entrar no mercado e chegar a grandes grupos de consumidores. Acrescidamente os métodos de produção no mercado dos *Triple-A* e no *Indie* diferem. Independentemente do tamanho, ambas as definições de produção usam algum tipo de motor de jogo.

Até agora parece claro que um motor de jogo é uma peça de *software* que permite a criação de videojogos. No entanto, continuamos a não dispor de uma definição adequada do que se pode esperar nestes quadros. O principal objectivo de um motor de jogo é abstrair funcionalidades comuns de videojogos permitindo a reutilização de códigos e activos de jogo em diferentes jogos [11]. Para tal, este refere, que as seguintes funcionalidades são tipicamente encontradas num motor de jogo:

- Um motor de renderização de gráficos, 2D ou 3D;
- Tratamento de entradas (Input) (teclado, rato e outro *hardware*);
- Ciclo de Jogo (rotina que corre em todas as imagens);
- Motor de físicas, que pode incluir ou não colisões;
- Motor de som;
- Um gráfico de cenas (gestão gráfica dos elementos no ecrã e em cena);
- Motor de Animação (animações para texturas 2D e modelos 3D);
- Gestão de memoria ou recolha de lixo;
- Múltiplos processos para otimizar o uso do processador).

Outras funcionalidades podem incluir:

- Programação de *scripts*;
- Inteligência Artificial;
- Suporte a rede;
- *Streaming* (para serviços como a *Twitch*);
- Suporte linguístico;
- Publicar para diferentes plataformas;

2.2 Concorrência de motores

Em 2012, existiam 334 motores de jogo presentes na base de dados DevMaster.net [12]. Este web-site contém muita informação sobre motores de jogo, mas esta informação não está filtrada entre plataformas móveis e outras [13].

Actualmente observa-se duas grandes tendências nos motores de videojogos. Em primeiro lugar, os motores de jogo estão agora muito mais propensos a usar linguagens de alto nível como Java, C# ou Python, o que muitas vezes resulta num aumento de produtividade para os criadores. Tendo em conta o *hardware* actual, a sobrecarga proveniente do uso de linguagens mais sofisticadas tornou-se insignificante e o desempenho do jogo está principalmente relacionado com o poder da placa gráfica e processador disponível. A segunda grande tendência para alguns motores de jogo é a possibilidade de permitir a publicação do jogo em múltiplas plataformas diferentes, mantendo a mesma programação. A maioria dos motores de jogo são, portanto, actualmente capazes de publicar para ambientes de computador, mas também para os principais sistemas operativos de dispositivos móveis e para diferentes sistemas de consolas de jogos [11].

2.2.1 Unreal

Em 2021 o "Unreal Engine 4" é a iteração atual de um dos primeiros grandes motores de jogo a ser disponibilizada ao público. A Epic Games, proprietária do Unreal, reviu

recentemente os planos de monetização do Unreal Engine, fornecendo assim a sua tecnologia gratuitamente para uso não comercial, e para todos os outros casos licenciando o *software* por uma pequena taxa de *royalties*. Inicialmente desenvolvido para apoiar o jogo "Unreal Tournament", rapidamente tornou-se num motor muito poderoso capaz de suportar qualquer género de jogo (incluindo jogos em 2D). O Unreal Engine também tem uma comunidade considerável e uma loja onde podemos comprar ativos. A linguagem usada para programar neste motor é C++, o que pode ser um inconveniente em comparação com as linguagens de alto nível que outros motores fornecem. Para ajudar novos criadores a versão mais recente do Unreal Engine introduziu uma funcionalidade bastante inovadora designada de "Blueprint Visual Scripting", como é possível observar na figura 2.2. Estas Blueprints (diagramas) consistem numa interface baseada em "nodes" (nós) para criar elementos de jogabilidade dentro do editor, como a figura 2.1 mostra, sem a necessidade de programação. A partir da sua documentação, o objetivo desta abordagem é permitir que os criadores experimentem praticamente toda a gama de conceitos geralmente apenas disponíveis para programadores. Esta ferramenta é descrita como sendo tão poderosa que todos os jogos em Unreal podem ser construídos usando-a exclusivamente [11].

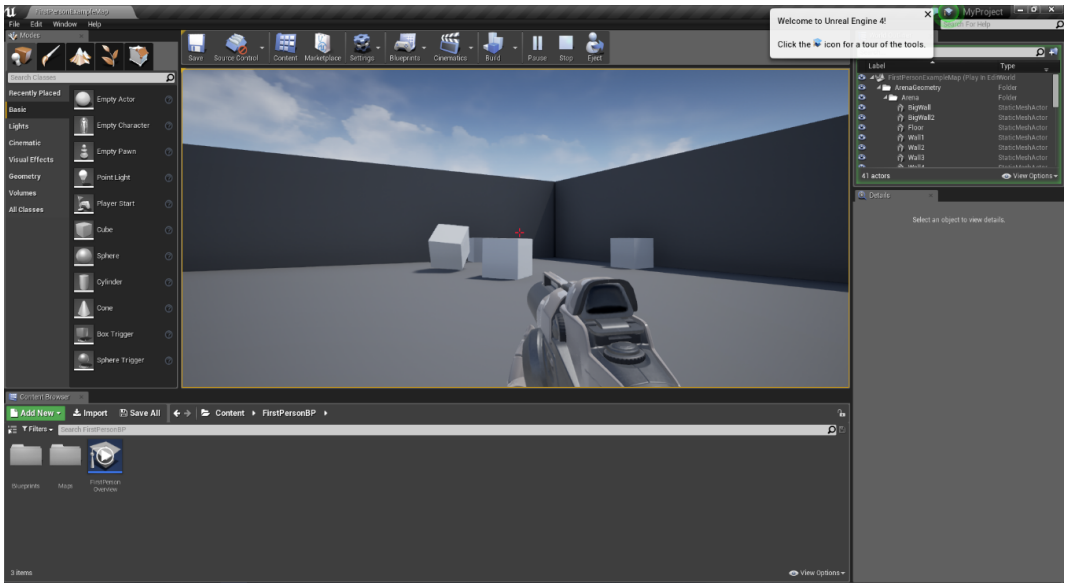


Figura 2.1: Editor do Unreal Engine

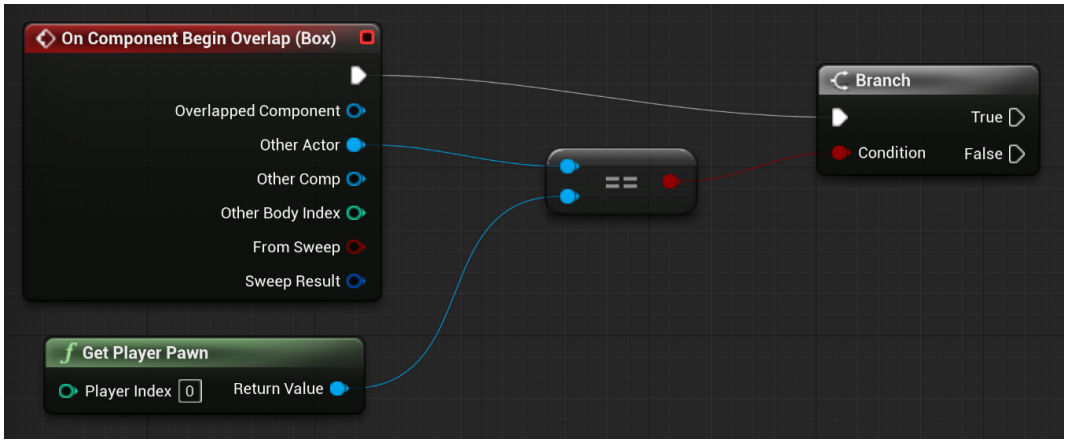


Figura 2.2: Sistema de Blueprints do Unreal Engine

2.2.2 Unity

O motor de jogos Unity foi anunciado pela primeira vez na conferência mundial de criadores da Apple em 2005 e desde então introduziu grandes mudanças na indústria dos videojogos [11].

Especificamente, à data deste documento é capaz de exportar para 20 plataformas (com a mesma programação), entre elas estão: iOS, Android (AndroidTV), tvOS, Windows, Mac, Linux, WebGL, PlayStation (PS4, PS5), Xbox (XboxOne, XboxSeriesX/S), Nintendo Switch, Stadia, Oculus, PlayStationVR, Google's ARCore, Apple's ARKit, WindowsMixedReality (HoloLens), MagicLea, SteamVR e GoogleCardboard [14]. No entanto, é preciso ter em mente que a publicação para consolas de videojogos geralmente requer uma licença de programador do proprietário desse sistema, que muitas vezes é dispendioso e que pode requerer um portfólio de jogos pré-existente [11].

O Unity, é de uso gratuito para criadores *Indie* ou empresas com um volume de negócios anual não superior a €100.000 . Uma versão Pro também está disponível, que inclui uma série de funcionalidades para uso avançado, como o acesso a um nível mais profundo de modificação gráfica, bem como recursos específicos para uma optimização de utilização de recursos [11]. Esta versão é também gratuita para estudantes, dando assim mais ferramentas para a aprendizagem [15]. Na figura 2.3 é possível observar o aspeto do editor do Unity.

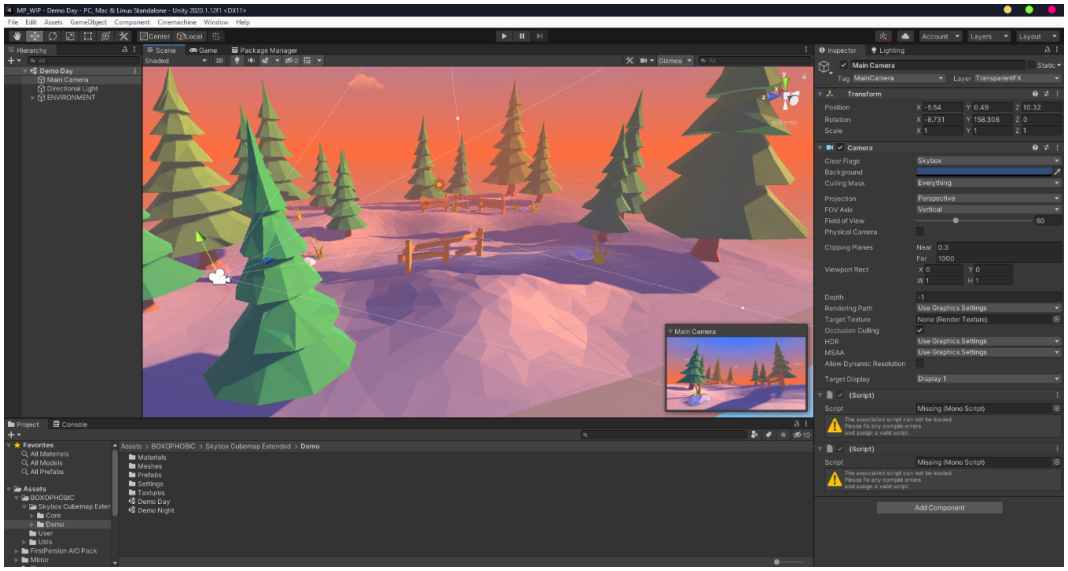


Figura 2.3: Editor do Unity Engine

2.2.3 Godot

O Godot é um motor de jogos multiplataforma, gratuito e de código aberto lançado sob a licença MIT. Foi inicialmente desenvolvido pelos criadores de software argentinos Juan Linietsky e Ariel Manzur [16, 17] para várias empresas na América Latina antes do seu lançamento público [17, 18]. O ambiente de desenvolvimento corre em múltiplos sistemas operativos, incluindo Linux, BSDs, macOS, e Microsoft Windows. Foi concebido para criar jogos 2D e 3D direccionados para PC, plataformas móveis, e web. O Godot pretende oferecer um ambiente de desenvolvimento de jogos totalmente integrado, como é possível observar na figura 2.4. Permite aos programadores criar um jogo, não necessitando de outras ferramentas para além das utilizadas para a criação de conteúdos (recursos visuais, música, etc.). A arquitectura do motor é construída em torno do conceito de uma árvore de "nós". Os nós são organizados dentro de "cenas", que são reutilizáveis, exemplificáveis, hereditárias, e grupos de nós destacáveis. Todos os recursos do jogo, incluindo *scripts* e recursos gráficos, são guardados como parte do sistema de ficheiros do computador (em vez de numa base de dados). Esta solução de armazenamento destina-se a facilitar a colaboração entre equipas de desenvolvimento de jogos utilizando sistemas de controlo de versões de software (exemplo o Github ou Gitea) [17, 19].

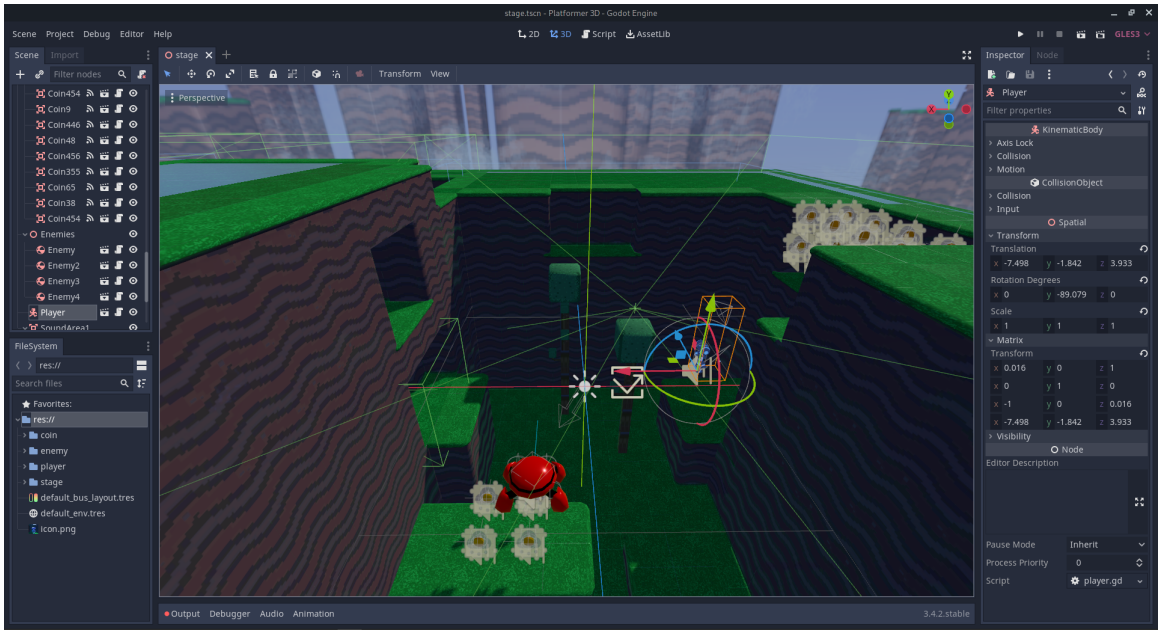


Figura 2.4: Editor do GODOT

2.2.4 Comparação dos motores

A tabela 2.1 apresenta as diferenças entre os três motores acima mencionados com as devidas linguagens de programação suportadas, as suas plataformas e o tipo de monetizarão do motor. Todas estas informações foram retiradas dos respectivos websites. [20, 21, 22].

Tabela 2.1: Motores

Engines	Linguagens	Plataformas	Monetização
Unreal Engine	C++/Blueprints	Windows PC, PlayStation 5, PlayStation 4, Xbox Series X, Xbox One, Nintendo Switch, Google Stadia, macOS, iOS, Android, AR, VR, Linux, SteamOS, HTML5	Gratuita para lançamentos na EpicGames/ 5% royalties noutras lojas
Unity Engine	C#/JavaScript/Bolt	iOS, Android (AndroidTV), tvOS, Windows, Mac, Linux, WebGL, PlayStation (PS4, PS5), Xbox (XboxOne, XboxSeriesX/S), Nintendo Switch, Stadia, Oculus, PlayStationVR, Google's ARCore, Apple's ARKit, WindowsMixedReality (HoloLens), MagicLea, SteamVR e GoogleCardboard	Versão Gratuita e Versão Paga
GODOT	GDScript,C++,C#	Linux, BSDs, macOS, e Microsoft Windows	Gratuita e de Código Livre

2.3 Arquitetura Multijogador

Alguns dos primeiros videojogos eram para dois jogadores. Os primeiros exemplos de jogos multijogador em tempo real foram desenvolvidos para o sistema PLATO por volta de 1973 [23].

Gauntlet (1985) e Quartet (1986) introduziram o jogo cooperativo de 4 jogadores nos salões de jogos [24]. Os jogos tinham consolas mais amplas para permitir quatro conjuntos de controlos.

Os jogos multijogador tornaram-se muito populares [25]; O Battlefield 2042 [26] e o Counter-Strike: Global Offensive [27] têm pouca (ou nenhuma) jogabilidade de um jogador. Com o tempo, o número de consumidores que jogam videojogos tem vindo a aumentar. Segundo a ESA (Associação de softaware para entretenimento) em 2020, a maioria dos lares nos Estados Unidos tem pelo menos um ocupante que joga videojogos, e 65% desses jogadores jogam jogos multijogador com outros utilizadores, quer online quer pessoalmente [28].

2.3.1 Multijogador em Rede

Daniel Cook num *blog* para a GameDeveloper.com [29] afirma que jogos multijogador online oferecem o benefício da distância, mas também vêm com os seus próprios desafios únicos.

Daniel Cook [29] menciona ainda que os jogadores referem-se à latência usando o termo "ping", (a wikipédia refere-se à latência como *lag* [30]) e que um jogador numa ligação com um ping de 50 ms pode reagir mais rapidamente do que um utilizador de com uma latência de 350 ms. Afirma ainda que existem outros problemas como perda de pacotes (Packet Loss) e asfixia (Network Choke), que podem impedir um jogador de "registar" as suas acções num servidor.

Em jogos de tiro em primeira pessoa, este problema aparece quando as balas atingem o inimigo sem provocar dano [30, 31].

Infraestrutura

Brian Wirtz menciona que a Local Area Network (LAN) permitiu ligações de sistema onde jogadores jogam entre si. *"A LAN não se limitava aos cibercafés; de facto, era muito utilizada em dormidas, festas, e em casas de amigos a jogar uns com os outros."* [32].

Brian Wirtz afirma ainda que *"Costumava ser um verdadeiro "trabalho" levar a Xbox até à casa de um amigo para que nos pudesse-mos ligar a uma rede local. Mas fizemo-lo funcionar, e criámos algumas memórias divertidas (por vezes apenas a tentar pôr a LAN funcionar em geral)."* [32].

Outro caminho para jogos multijogador são os servidores dedicados. Estes são normalmente criados pelos criadores de jogos ou pela empresa que o distribui. Estes servidores funcionam automaticamente, não fazendo de ninguém, em particular, o anfitrião. A empresa tem controlo total sobre os servidores, largura de banda, e *hardware* utilizado nos mesmos [32, 33].

Os servidores dedicados não têm que ser propriamente hospedados pela empresa, existem alguns jogos que permitem que os jogadores hospedem o seu próprio servidor como é o caso do Minecraft [34] e do Valheim [35].

Interações

Daniel Cook explica que é possível dividir qualquer jogo multijogador numa série de interações [29]. Uma interação por exemplo, é sempre que os jogadores interagem uns com os outros através de um sistema dentro do jogo, quer seja conversar, combate, etc.

Se observarmos uma interação aleatória num espaço de tempo podemos obter o seguinte:

- O jogador inicia a interação.
- O jogador termina a interação.
- O jogador espera por uma resposta por parte do servidor.
- Se não houver resposta, o jogador por vezes fica frustrado e sai.

O mesmo autor refere [29] que estes conceitos remontam à teoria da comunicação que Chris Crawford [36] adaptou à teoria do design de jogos na década de 1980 e menciona

que este tópico é algo fundamental que todos os *designers* de jogos profissionais devem saber.

Frequência de Interações

Qual é a frequência de interacção necessária para dar a impressão de fluidez? Em geral, quanto maior for a frequência das interacções, mais informação será comunicada entre os jogadores [32]. Simplesmente ao alterar o espaçamento entre interações, obtém-se formas de jogo radicalmente diferentes (e desafios logísticos associados). Por exemplo, ao mudar o *TTK* (Time to Kill) de um inimigo, pode-se obter jogos completamente diferentes [37].

Cook entra em mais detalhe acerca deste tópico no artigo *"Loops and Arcs"* [38]. Tynan Sylvester, um dos criadores do jogo *Bioshock Infinite* tem algumas ideias que divergem das presentes em *"Loops and Arcs"* e escreveu uma resposta para Daniel Cook no seu blog [39].

Tipos de Interações

Segundo Raph Koster e Daniel Cook, existe uma variedade de tipos de interacção [29, 40]:

- Interação com avatares: Dois ou mais avatares interagem um com o outro.
- Interação com o ambiente virtual: No Minecraft, os jogadores constroem estruturas que outros jogadores podem depois explorar.
- Decoração e exibição: Os jogadores afirmam o seu estatuto, afiliações e história através do que vestem ou da forma como ornam as suas armas, animais de estimação e casas. Dois exemplos de jogos onde este fenómeno pode ser observado são o Guild Wars 2 e o Warframe onde os jogadores têm mesmo desfiles de moda com prémios para quem se vestir melhor e com mais estilo [41]. No caso do Warframe este fenómeno é reconhecido pelos criadores do jogo, onde os mesmos por vezes mencionam a prática "Fashion Frame" no site oficial do jogo [42].
- Económico: Os jogadores dão, trocam ou pagam por vários recursos a outros jogadores. Warframe é um jogo cuja economia é completamente gerida pelos jogadores.

A economia é tão livre que existe mesmo um site para os jogadores trocarem itens uns com os outros [43]. Joris Dormans e Ernest Adams, entram em mais detalhe acerca da economia dos jogos no livro "Game Mechanics: Advanced Game Design" [44].

- Texto: O método mais comum de introduzir a linguagem num jogo online é através de texto. Esta abordagem tem tendência para ser de baixo custo e existem um conjunto de ferramentas (filtros de spam, convenções estilísticas) para lidar com questões comuns.

- Voz: A voz oferece nuances adicionais, incluindo emoções, idade, sexo e muito mais. É notoriamente fraca quando se trata de filtragem e pode originar mesmo conflitos sociais e toxicidade nos jogos [45].

Dinâmicas de Grupo e Perigos de Grandes Grupos

Multijogador é uma opção popular para videojogos com muitas vantagens que superam os contras. No entanto, a implementação deste sistema é algo a que se tem de prestar atenção no momento e segundo Josh Bycer *"É melhor não ter um modo multijogador do que ter um modo mal implementado que as pessoas odeiam"* [46].

Pode ser tentador fazer jogos multijogador com suporte para milhares de jogadores em simultâneo, no entanto, os custos da tecnologia e do design são elevados e os benefícios são fracos. De acordo com Dunbar [47] o numero máximo cognitivo possível de relações sociais é de 150, onde após excedido todas as pessoas extra acabam por ser tratadas apenas como número ou abstracções pelos jogadores, em vários jogos dá-se a este fenómeno o nome de *Zerg* [48, 49] onde o objetivo é a vitória através de números. O youtuber Enardo é famoso por fazer *Zergs* no jogo Rust [50].

Boas Práticas

Daniel Cook deixa recomendações para novos criadores de videojogos [29]:

- Minimizar a fragmentação da comunidade devido aos modos de jogo (*Matchmaking*).

- Sempre que possível, utilizar a técnica *"Room Based"* em vez de partidas singulares.

- A persistência do mundo virtual deve ser usada se o jogo for pensado para tal, uma vez que permite jogabilidade assíncrona.
- As relações com os jogadores devem ser cultivadas, uma vez que aumentam a taxa de retenção dos jogadores.
- Fazer protótipos cedo e lidar com os problemas emergentes enquanto a densidade populacional do jogo é pequena.

Jogabilidade Assimétrica

A jogabilidade assimétrica é uma mecânica de jogo em que os jogadores podem ter papéis ou habilidades significativamente diferentes uns dos outros o que leva a uma experiência de jogo significativamente diferente para os jogadores [51]. Em jogos com leve assimetria, os jogadores partilham algumas mecânicas básicas (tais como movimento e morte), mas têm papéis/habilidades diferentes no jogo. Esta é uma característica comum dos jogos de tiros com heróis como o Overwatch e o Apex Legends [23].

2.3.2 Implementações prévias

Em [52] Mika Anttonen, compara diferentes abordagens para o desenvolvimento de um jogo multijogador e implementa esses sistemas num jogo feito em Unity.

Mika Anttonen usou no seu projeto o Photon Unity Networking (PUN). Anttonen afirma que embora os resultados da jogabilidade não fossem ideais, os resultados sobre as funcionalidades multijogador excederam as expectativas e deram uma boa orientação de como o desenvolvimento futuro deve continuar. Anttonen refere que em 2019, *"O Unity está no meio de uma mudança nas suas ferramentas multijogador e o Photon lançou um novo ativo para substituir o atualmente utilizado no X-Craft. Um súbito anúncio do Unity tornou os primeiros meses de desenvolvimento e teste das funções multijogador irrelevantes e forçou um recomeço completo do desenvolvimento multijogador. Se o desenvolvimento da parte multijogador do jogo fosse iniciado agora, o PUN 2 seria provavelmente utilizado."* [52].

Em [53] Chatziagapis Alexandros, refere que os jogos desenvolvidos com recurso ao UNET têm uma quantidade limitada de formas para ser implementados. Afirma ainda

que têm de se usar as Remote Procedure Call (RPC) para conseguir a sincronização e evitar problemas de rede. Alexandros menciona que é preciso optimizar o jogo de uma forma onde só se utilize RPC através da rede quando é absolutamente necessário para o jogo. Alexandros afirma o mesmo que Anttonen no que toca ao UNET [53], e que para o melhoramento do seu projeto, o UNET devia ser substituido pelo PUN. Segundo Alexandros para garantir que não haveriam perdas de ligação com o UNET, teria que pagar ao Unity e ao utilizar a versão gratuita do PUN obteria a mesma qualidade.

Em [54] Timo Jetsonen informa que os métodos para a implementação do projeto foram a investigação da própria tecnologia de rede disponibilizada pelo Unity e um dos populares *plugins* de rede o PUN. O objectivo da investigação era no fim obter funcionalidades semelhantes com as duas implementações. Como resultado, afirma que conseguiu dois sistemas de *lobby* com implementações separadas e a funcionar correctamente. Refere ainda que o resultado foi um empate entre as duas abordagens *"Devido às limitações das ferramentas, o único atributo comparável e digno de confiança que se pode obter tanto do Unity Profiler como do Photon Stats Gui foi o "round trip time"(RTT). A distribuição de pontos resultou num empate, indicando que ambos os métodos de implementação são excelentes em diferentes áreas-chave, enquanto a Rede Unity teve tempos de ida e volta globalmente mais elevados."*[54]. Em [55] Polančec e Mekterović fazem recurso do *Photon Cloud service* para gerir a sessão e o servidor de jogo, mas não exploram a sua implementação, apenas informam que devido à simplicidade do jogo criado, existe apenas uma base de dados para a gestão do jogo, que é acedida utilizando a API JDBC (Java Database Connectivity) através do servidor "Spring".

Em [56] Akshay Gautam refere que para uma implementação rápida e eficaz fizeram recurso ao *Match Making* do PUN para Unity. De acordo com [56], o PUN foi escolha ideal para o desenvolvimento uma vez que estava disponível documentação para a mesma, mas não mencionam como fizeram a sua implementação do PUN dentro do Unity.

2.4 Ferramentas para implementação de uma arquitetura multijogador

Uma biblioteca de rede é necessária em jogos multijogador porque cada cliente do jogo precisa de comunicar com outro cliente. Existem muitos modelos de rede para jogos multijogador, mas o mais complicado para um jogo multijogador é a de um Massive Multiplayer Online (MMO) porque muitas pessoas podem jogar no mesmo mundo virtual ao mesmo tempo [13]. Por exemplo, "World of Warcraft" é um dos jogos MMO mais bem sucedidos, onde pode suportar mais de 1 milhão de utilizadores em simultâneo [57]. Contudo existem soluções como P2P Networking onde as mensagens são enviadas diretamente de um jogador para outro sem a necessidade de um servidor [58].

2.4.1 Motores mais usados

De acordo com Marcus Toftedahl [2, 59] em 2019, dos 6743 jogos identificados na Steam, 1726 (25.6%) utilizam Unreal Engine e 889 (13.2%) utilizam o Unity Engine.

Os números do Unity aparentam ser muito mais pequenos do que os do Unreal, mas isto é se apenas levarmos em consideração os jogos da Steam. Se analisarmos os dados da Itch.io que é a maior plataforma para jogos Indie verificamos números completamente diferentes. No Itch.io o Unity é responsável por 24200 (47.3%) jogos enquanto que o Unreal tem 1458 (2.8%) jogos disponíveis.

Mesmo assumindo que na Itch.io estão todos os jogos já presentes na Steam a disparidade de números continua muito alta, o que mostra que o Unity é o motor preferido para iniciantes ou pequenas equipas. O mesmo diz Toftdahl [59], que ficou curioso após ler uma citação no site do Unity, onde este afirmava, "ser a plataforma líder a nível mundial"para o desenvolvimento de videojogos e que o Unity era responsável "por criar metade dos jogos do mundo".

Devido a estas estatísticas este documento dá apenas ênfase ás ferramentas disponíveis para Unreal e Unity.

2.4.2 Ferramentas para Unreal

Este subcapítulo descreve três ferramentas possíveis para o desenvolvimento multijogador em Unreal.

Multiplayer com Unreal Blueprints

"As experiências multijogador modernas requerem a sincronização de grandes quantidades de dados entre grandes números de clientes espalhados por todo o mundo. Que dados são enviados e como são enviados é extremamente importante para proporcionar uma experiência convincente aos utilizadores, uma vez que estes podem afectar drasticamente a forma como o jogo funciona e se sente.[60] " -Epic Games, Inc (Epic Games)

De acordo com a Epic Games [60, 61] em Unreal Engine, a replicação é o nome para o processo de sincronização de dados e chamadas de procedimentos (RPC) entre clientes e servidores. O sistema de replicação proporciona uma abstracção de nível superior juntamente com uma personalização de baixo nível para facilitar o tratamento de todas as várias situações que criadores possam encontrar ao criar um projecto concebido para múltiplos utilizadores simultâneos.

A Epic Games afirma ainda que "é fácil configurar um jogo básico através de Blueprint's que funciona através de uma rede" [61].

No site disponibilizado pela Epic Games [61], esta refere que é importante compreender os papéis das principais classes de jogabilidade que são fornecidas pelo motor e como funcionam em conjunto e especialmente como funcionam num contexto multijogador.

Segundo a mesma, as classes mais importantes são:

- *GameInstance*
- *GameMode*
- *GameState*
- *Pawn* (e *Character*, que herda de *Pawn*)
- *PlayerController*
- *PlayerState*

O Unreal, por padrão, permite que seja criado um jogo no modo de *hosting* (P2P) e *dedicated server*[60, 61].

Multiplayer com Steam API

O Unreal Engine fornece também o subsistema online para Steam API [62] que permite lançar aplicações Unreal Engine 4 (UE4) para a plataforma Steam da Valve. O principal objectivo do módulo Steam é ajudar a distribuir aplicações com um conjunto de características (tais como matchmaking e leaderboards) aos utilizadores da Steam. Além disso, o módulo Steam implementa várias das interfaces expostas pelo Subsistema Online, suportando a maior parte do que é oferecido pelo Kit de Desenvolvimento de Software Steamworks (SDK).

Para utilizador o subsistema Steam é esperado do utilizador ler e perceber como o Steamworks disponibilizado pela Valve funciona [62].

Algumas das funções disponibilizadas pela Steam API são [62, 63]:

- *Matchmaking* (*Lobbies* e *APIs* de *GameServer*)
- *Leaderboards*
- *Achievements*
- *Voice*
- *UserCloud*
- *SharedCloud*
- *External UI*

VisGM

o VisGM é uma framework em crescimento para a criação de jogos multijogador em Unreal Engine 4 [64]. Fornece mais de 200 funções, mais de 30 macros, sistemas expansivos, e uma estrutura que ajuda a tornar o desenvolvimento de jogos multijogador mais rápido.

De acordo com a Visualistic Studios, criadores desta framework [64]: *"Em vez de se concentrar na construção da estrutura para o seu próximo jogo multijogador, irá concentrar-se*

na criação das funcionalidades. Tudo é facilmente instalado, expandido, modificado, e integrado."

A Visualistic Studios afirma [64] ainda que com o crescente conjunto de sistemas VisGM os criadores irão usufruir de um avanço que acelerará o processo de criação de jogos multijogador. Referem ainda que os dados são acessíveis a partir de quase qualquer parte do projecto e que o sistema integra-se bem com outros plugins e redes e que a utilização de largura de banda por cada jogador é optimizada, e é facilmente estendida e modificada devido ao seu design.

Para concluir o VisGM não fornece funcionalidade multijogador a Visualistic Studios avisa que é necessário ter um sistema Host/Join já configurado, o que não é difícil com as BluePrints do Unreal [60, 61, 64].

2.4.3 Ferramentas para Unity

Este subcapítulo descreve quatro das ferramentas mais comuns para o desenvolvimento multijogador em Unity.

Photon

O PUN é um pacote disponível para Unity desenvolvido para a criação de jogos multijogador. O Match Making (MM) flexível disponibilizado pelo PUN leva os jogadores a salas onde os objetos em cena podem ser sincronizados na rede. A lista de funcionalidades do PUN inclui, RPCs e propriedades personalizadas de objetos. A comunicação rápida e fiável é feita através de servidores proprietários e dedicados disponibilizados pelo próprio Photon [65]. O pacote PUN é constituído por três camadas de Application Programming Interface (API)s [65]:

- O nível mais alto é o código PUN, que implementa funcionalidades específicas do Unity como objectos em rede, RPCs, entre outros.
- O segundo nível contém a lógica para trabalhar com os servidores do Photon, como por exemplo, fazer "MM" e "callbacks".
- O nível mais baixo é composto por ficheiros DLL, que contêm a serialização e protocolos.

Mirror

O Mirror é uma API de rede de alto nível desenvolvida para Unity, com suporte a diferentes transportes de baixo nível. Este é desenvolvido e testado para *MMO Scale Networking*, ou seja, para projetos em grande escala, pelos criadores do uMMORPG, uSurvival e Cubica. O Mirror facilita, agrega e mantém a ligação em rede. Referem os criadores [66] que *"O uMMORPG foi possível com menos de 6000 linhas de código. Precisávamos de uma biblioteca de rede que nos permitisse lançar os nossos jogos, ponto final." Com o Mirror, o servidor e cliente são um único projeto e assim surgiu o nome Mirror. Em vez de termos um código para o servidor e um para o cliente, simplesmente usamos o mesmo código para ambos no mesmo projeto."* O Mirror [66] descreve as seguintes marcações para o código:

- [Server] / [Client] estas marcações servem para correr código exclusivamente no servidor ou no cliente respetivamente.
- [Command] é usado para comunicação do cliente para o servidor.
- [ClientRpc] / [TargetRpc] são usados para comunicação do servidor para o(s) cliente(s).
- [SyncVar]s e [SyncLists] são usadas automaticamente para sincronizar estados.

SteamWorks.NET

O Steamworks.NET é um wrapper escrito em C# para a API steamworks originalmente escrita em C++ da Valve e é completamente gratuito e de código aberto sob a licença permissiva do Massachusetts Institute of Technology (MIT). Pode-se usar o Steamworks.NET com projetos Unity ou qualquer outro projeto .NET.

O objetivo principal do Steamworks.NET é ser o mais fácil possível de utilizar apesar de ser concebido para seguir o mais próximo possível a API nativa fornecida em C++ e possui uma cobertura completa da API nativa steamworks. O Steamworks.NET disponibiliza assim algumas ferramentas aos criadores como estatísticas e troféus, *Leaderboards* para que os jogadores possam ver quem é o mais rápido, o mais forte ou apenas o mais mortal, autenticação do utilizador e propriedade, *Matchmaking*, *Community* que permite o uso

do *Overlay* da Steam, *P2P Networking* onde é criada uma camada de abstracção em rede que é fornecida para tornar a difícil logística do envio de dados pela Internet mais simples, *Steam Cloud* que fornece uma forma mais simples de sincronizar os dados de um jogo na nuvem, permitindo que os jogadores mantenham o seu progresso no jogo sem problemas ao alternarem entre dispositivos e finalmente o *Valve Anti-Cheat* (VAC) que proporciona uma camada adicional de segurança nas experiências multijogador competitivas [67].

Netcode for Gameobjects

O Netcode for GameObjects é o successor do UNET e tem como origem o MLAPI que foi comprado pelo Unity. O Netcode for GameObjects é um pacote para Unity que fornece capacidades de rede aos GameObject's e MonoBehaviour's do Unity. Esta framework é interoperável com muitos transportes de baixo nível, incluindo o pacote oficial de transporte do Unity e também com o transporte da SteamWorks [68].

Capítulo 3

Planeamento

O método de investigação a ser usado neste projeto é o "Practice-Based Research" que se foca no processo de aprendizagem com base numa investigação original com o propósito de adquirir conhecimento nas áreas investigadas. Como este projeto consiste num videojogo categorizado como RPG, e este é um conceito já bem conhecido e com métodos padronizados, é sem dúvida o método que contribui mais para o crescimento do conhecimento. A ênfase deste projeto está no processo criativo e na obra que será gerada. Neste sentido, a prática e a pesquisa atuam juntas de forma a gerar novos conhecimentos. Tem-se como objetivo principal deste projeto obter novos conhecimentos devido à prática e resultados dessa prática.

Este método de investigação apresenta características muito positivas na aquisição de novos conhecimentos em diferentes áreas e actualmente é um termo muito discutido pelas vantagens que apresenta no crescimento via conhecimento adquirido através da prática [69].

3.1 Projeto a desenvolver

O projeto "The Abyss" consiste num videojogo RPG visto em terceira pessoa, e com multijogador. Para tal, foi escolhido o motor de jogo Unity 3D onde é utilizada a linguagem de programação C# para o desenvolvimento dos mecanismos de jogo e o NGO que permite a implementação de multijogador.

Numa fase inicial foram realizados testes de sincronização com a ferramenta multijoga-

dor denominada "Mirror". Ao ser utilizado o protocolo P2P disponibilizado pela Steam, foi possível colocar a pronto dois jogadores em simultâneo num ecrã (figura 3.1) comum através da Internet, onde as animações oriundas das interações estão em sincronia. Apesar desta abordagem ter sido alterada, serviu como aprendizagem para o trabalho final.

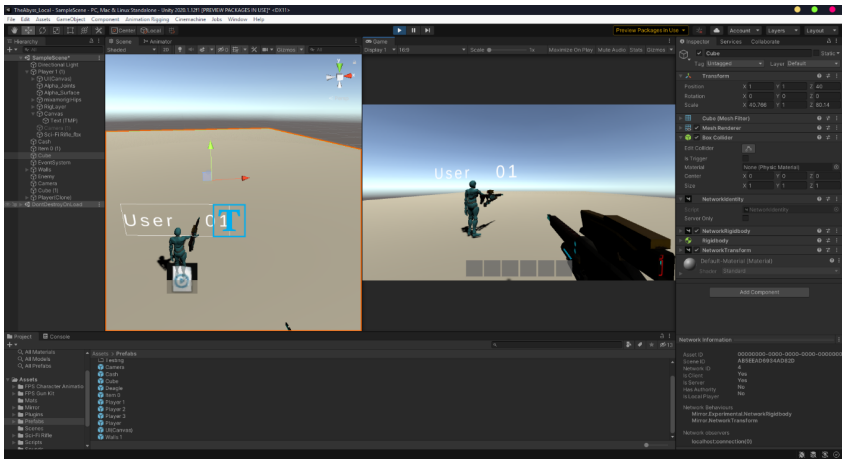


Figura 3.1: Implementação inicial

3.1.1 Recursos

- **Visual Studio Code** - É um ambiente de desenvolvimento integrado da Microsoft usado para desenvolver programas de computador.
- **SourceTree/GitHub Desktop** - Git GUI para a representação visual de repositórios e controlo de versões.
- **GitHub/Gitea** - O GitHub e o Gitea são serviços de hospedagem de repositórios Git.
- **Unity** - Motor de jogos 2D/3D.
- **Netcode for Gameobjects** - Ferramenta para implementação de multijogador em rede para o motor Unity.

3.2 Testes

Na fase final do desenvolvimento foi disponibilizada uma versão de teste a diferentes utilizadores com o intuito de avaliar o estado do projeto e obter críticas ao mesmo através de um questionário, apresentado no subcapítulo 5.2.

3.3 Requisitos

Os requisitos definidos para a implementação da infraestrutura multijogador foram os seguintes:

- R1: Multijogador Co-Op com suporte de um a quatro jogadores.
- R2: Posição dos jogadores sincronizada em rede.
- R3: Ataques dos jogadores sincronizados em rede.
- R4: Posição dos inimigos sincronizados em rede.
- R5: Ataques dos inimigos sincronizados em rede.
- R6: Equipamento dos jogadores sincronizado em rede.
- R7: Sistema para guardar o progresso dos jogadores.
- R8: Itens adquiridos sincronizados em rede.
- R9: Áudio sincronizado em rede.
- R10: Servidor para os jogadores poderem jogar.

3.4 Cronograma

Baseado na figura abaixo, as tarefas de cor laranja são as tarefas gerais do jogo e as tarefas de cor verde são as referidas neste projeto. Estas tarefas são a sincronização dos jogadores e NPC's, a criação de uma interface básica para o jogo, sincronização de cenas e o desenvolvimento de um sistema para guardar o progresso dos jogadores.

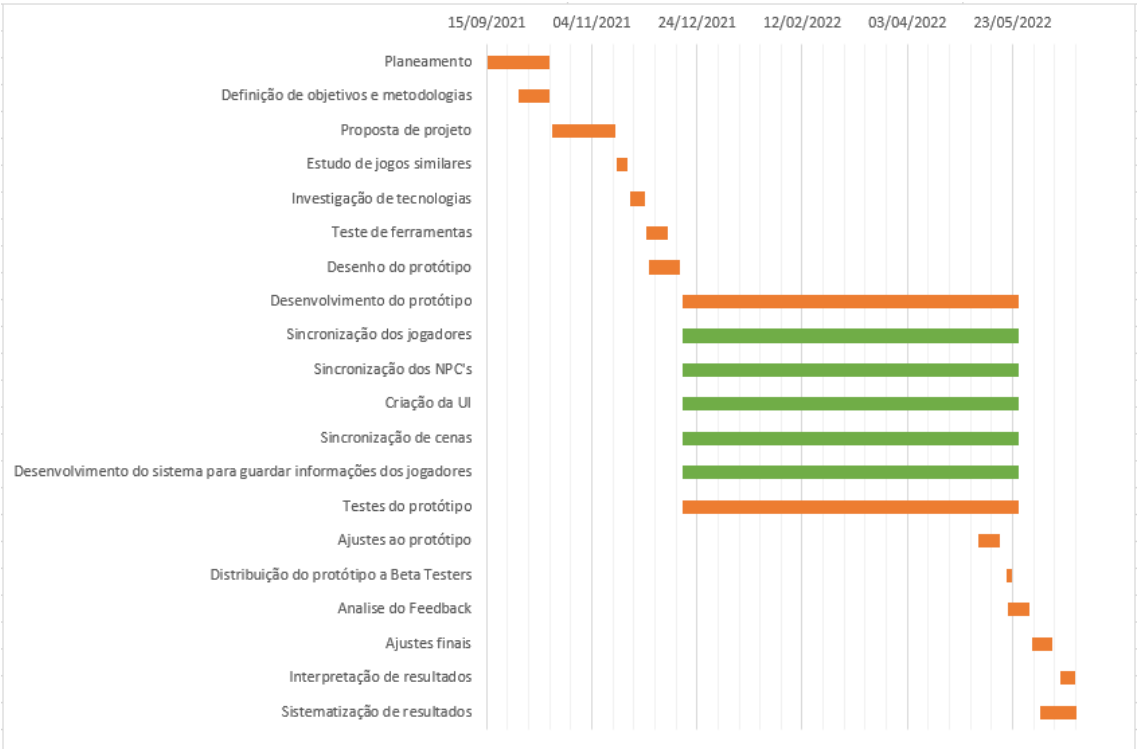


Figura 3.2: Cronograma

Capítulo 4

Desenvolvimento

Este capitulo descreve todo o processo usado para a criação do projeto e inclui a instalação, a preparação, as ferramentas disponíveis para a sincronização de objetos e variáveis, como foi realizada a gestão de cenários em rede, os problemas encontrados ao longo do desenvolvimento, bem como algumas soluções para estes. Também se apresenta o sistema criado para efetuar o registo do progresso dos jogadores e finalmente, a, lógica operacional do jogo.

4.1 Instalação

À escrita deste documento o Netcode (NGO) pode ser instalado nas versões 2020.3, 2021.1, 2021.2, e 2021.3 do Unity. Para se efetuar a instalação usa-se o *package manager* do Unity, e seleciona-se *"add package by name"* e adiciona-se *"com.unity.netcode.gameobjects"*, isto irá instalar o pacote base do Netcode. Para se obter o pacote extra adiciona-se também "com.unity.multiplayer.samples.coop", este pacote contem os *scripts* necessários para se ter o movimento dos jogadores com controlo local nos clientes.

4.2 Preparação

Com os dois pacotes instalados, pode-se começar a preparar o projeto para funcionamento *online*. Para tal, na primeira cena do jogo, que neste caso é um ecrã de inicio, criou-se um *GameObject* vazio e adicionou-se os componentes de *NetworkManager* e *UNET Transport*, como demonstra a figura 4.1. Outros transportes estavam disponíveis para utilização mas como se queria fazer um servidor dedicado optou-se pelo UNET. (Na versão 2022.2 o Unity mudou o transporte por padrão para o Unity Transport e marcou o transporte UNET como descontinuado).

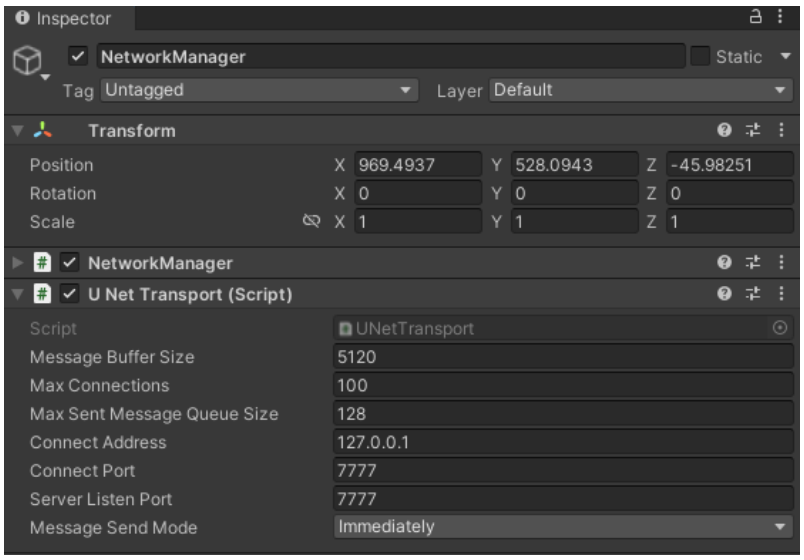


Figura 4.1: NetworkManager

No *NetworkManager* adiciona-se todos os *prefabs* que serão ser instanciados em rede. Neste caso, os *prefabs* dos jogadores, inimigos e consumíveis estão presentes na lista de *NetworkPrefabs*, como é possível ver na figura 4.2.

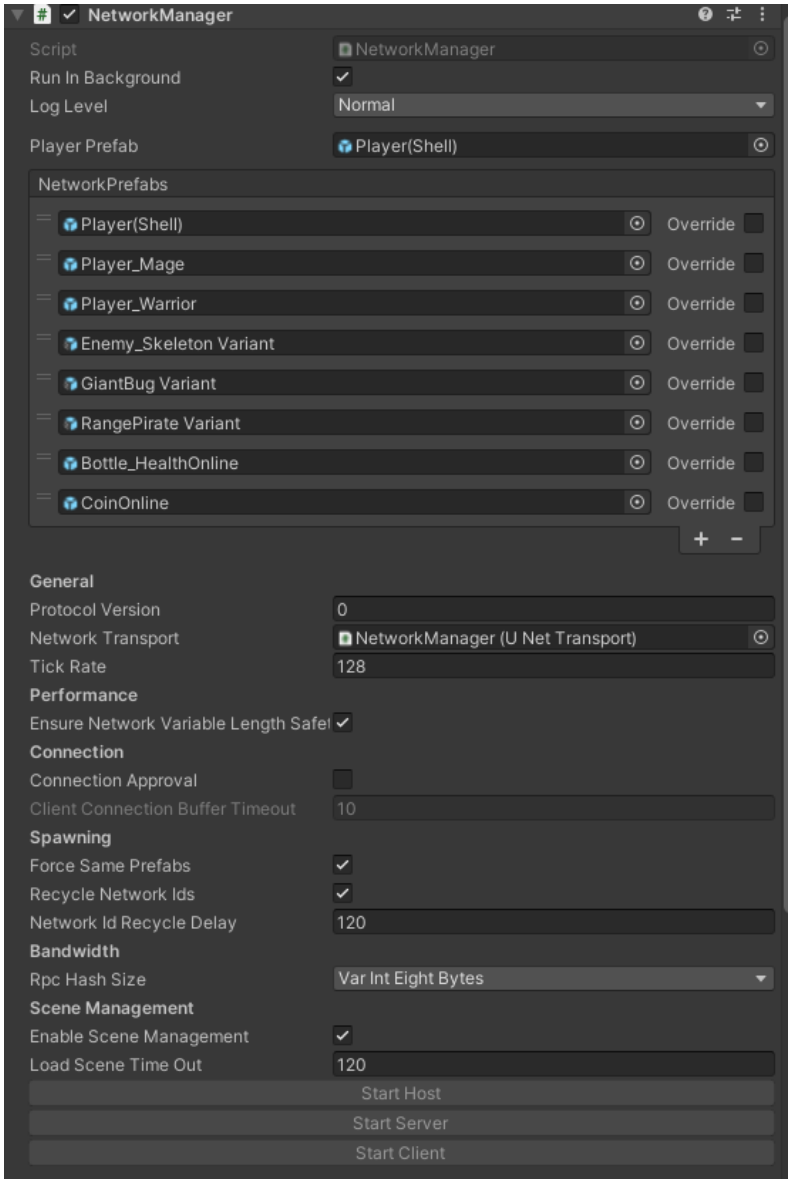


Figura 4.2: NetworkManager com os Prefabs

Como é possível ver na figura 4.3, nos *prefabs* dos jogadores adicionou-se os componentes de *NetworkObject*, *NetworkAnimator* e *Client Network Transform*, este ultimo apenas é necessário pois o movimento dos jogadores é controlado do lado do cliente e não do servidor, caso se estivesse a fazer um jogo completamente controlado do lado do servidor não seria necessário.

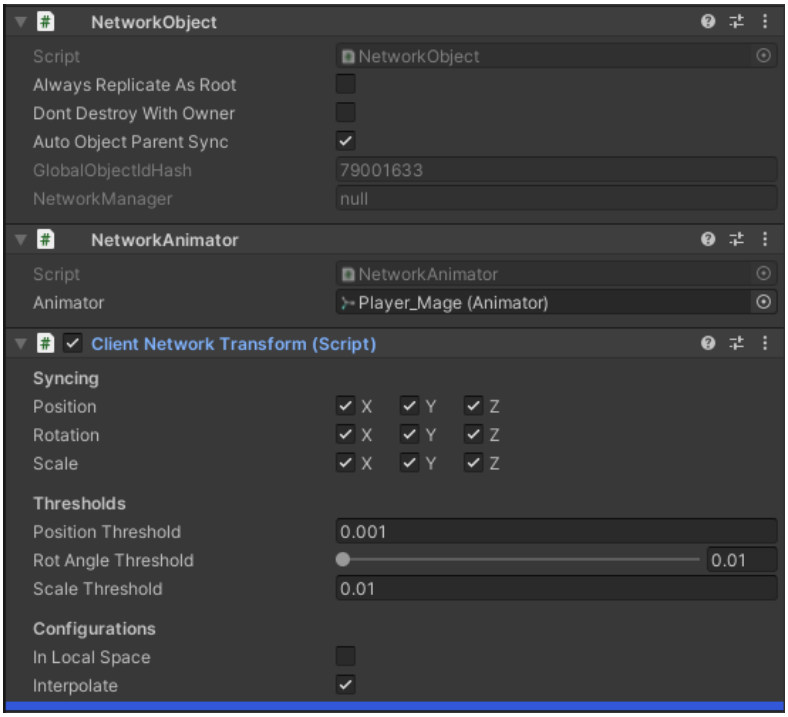


Figura 4.3: Componentes de rede dos jogadores

Para se poder testar o jogo online precisa-se de ter dois computadores na mesma rede ou então pode-se usar o pacote *ParrelSync* que cria um clone do projeto do UNity, com a finalidade de servir como servidor. Com o *ParrelSync* instalado na barra do Unity aparece uma opção nova com o nome do pacote. Nesta opção clica-se em *Clone Manager* e cria-se um clone do projeto. Como a figura 4.4 demonstra, no *Clone Manager*, verifica-se que o clone foi criado com sucesso, sendo possível abrir o mesmo num novo editor.

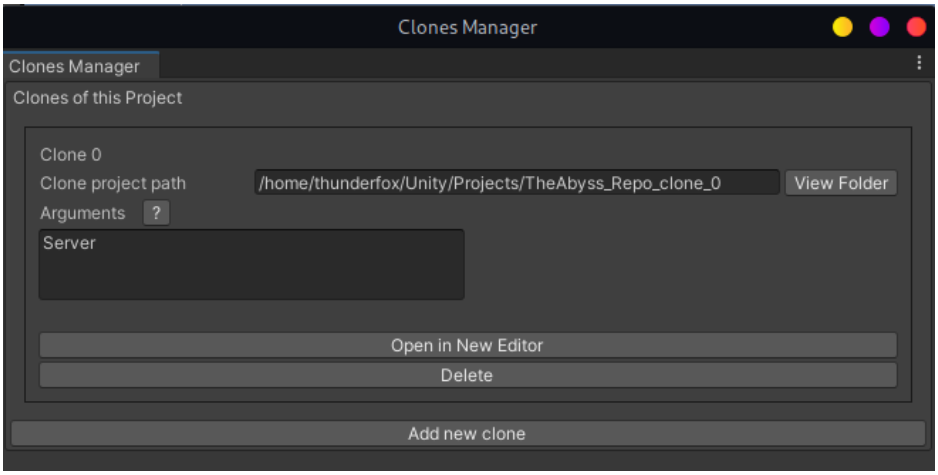


Figura 4.4: Gestor de Clones ParrelSync

Agora com o clone criado e aberto simultaneamente, pode-se começar o servidor manualmente através do editor do Unity no componente do *NetworkManager*. Como era necessário o servidor começar automaticamente, criou-se um *script* (Apêndice A) que verifica se o jogo está em *Headless Mode* ou se é um clone. Assim sempre que se entra em modo de jogo no clone ou se se compilar um servidor dedicado o servidor do NGO começa automaticamente.

4.3 Limitar o numero máximo de jogadores

Como o jogo foi planeado para suportar quatro jogadores é necessário limitar a capacidade máxima do servidor. Para tal é usada uma função do NGO chamada de *ConnectionApprovalCallback*. Para tal no nosso *script* de verificação de servidor (Apêndice A) adicionamos uma função nova (Apêndice B). Esta função encarrega-se de verificar se o numero de jogadores conectados ao servidor excede a lotação máxima que neste caso é quatro. Depois de escrita a função no *script* vai-se ao *NetworkManager* e seleciona-se a opção de *Connection Approval*, quando se ligar o servidor verifica-se que apenas quatro clientes se podem ligar ao mesmo.

4.4 Métodos para sincronização de objetos e variáveis

Para a sincronização de objetos e variaveis através da rede o NGO oferece principalmente duas opções, as *Network Variables* e *RPC's*.

Os RPC servem como modo de enviar uma notificações de eventos, bem como uma maneira de lidar com a comunicação direta entre um servidor e um cliente (ou viceversa). Isto é útil quando se pretende que um ou mais clientes possam comunicar com um *NetworkObject*.

Tal como se observa na figura 4.5, um *ServerRpc* pode ser usado por um cliente para notificar o servidor de que o jogador está a tentar usar um objeto no mundo (ou seja, uma porta, um veículo...), ou neste caso pode ser também usado para indicar ao servidor que um cliente pretende iniciar uma animação, e o servidor através de um *ClientRpc* comunica aos outros clientes o estada da animação. A figura 4.6 retrata um exemplo de uma ordem dada por um servidor para os seus clientes.

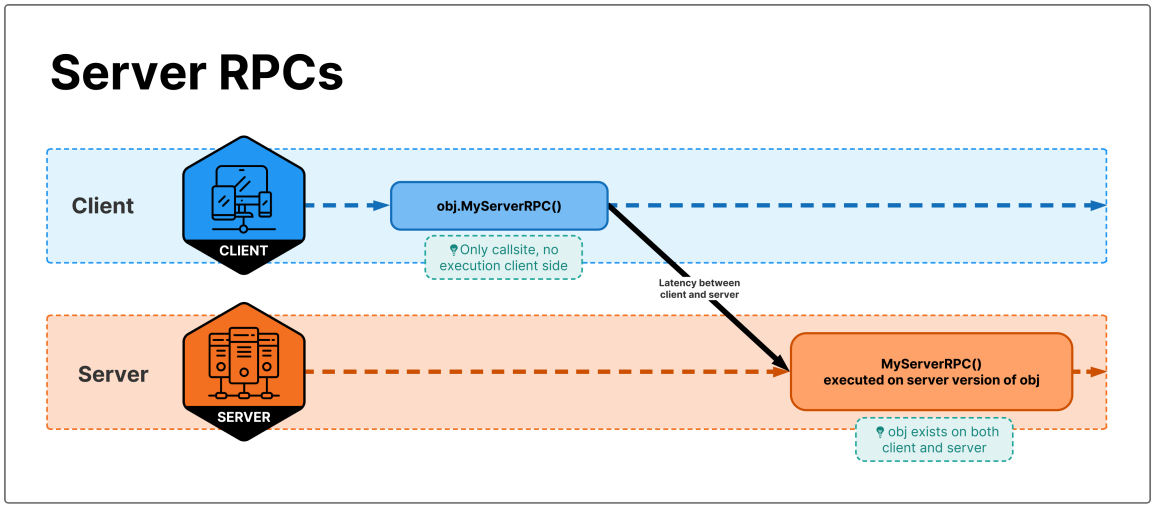


Figura 4.5: Exemplo ServerRpc [68]

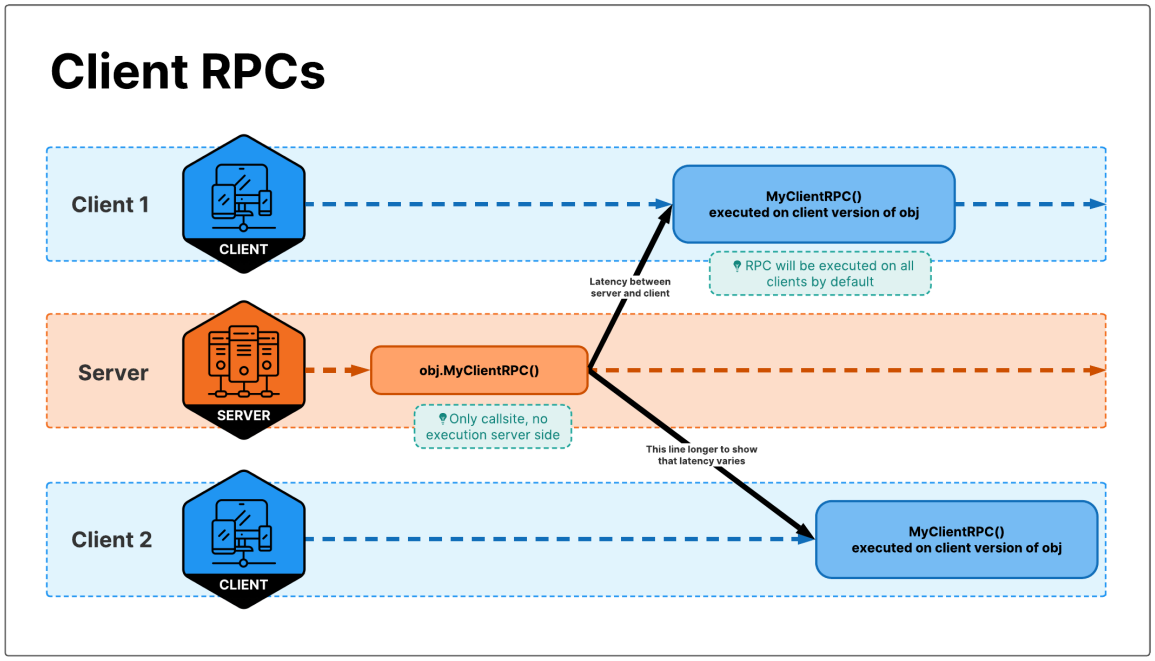


Figura 4.6: Exemplo ClientRpc [68]

Uma *NetworkVariable* é um mecanismo para sincronizar uma variável entre um servidor e clientes sem ter que usar *RPC*. O excerto 4.1, retrata um exemplo do uso de uma *NetworkVariable*.

```
private NetworkVariable<int> m_SomeValue = new
    NetworkVariable<int>();
```

Listing 4.1: Utilização de Network Variables

As *NetworkVariables* são úteis para vários propósitos como, por exemplo, sincronizar

o estado do contador de vida dos inimigos, a experiência que vão dar ao jogador depois de morrer, que consumíveis podem dar aos jogadores ,as chances de deixarem para trás algum consumível e até *"Cooldowns"* de habilidades. Normalmente só o servidor é que pode modificar estas variáveis mas pode-se mudar este comportamento através da forma como se declara estas variáveis, como mostra o excerto 4.2. Esta modificação permite ao dono da variável (neste caso o jogador) atualizar a mesma com o valor relevante.

```
public NetworkVariable<int> Health = new
    NetworkVariable<int>(default,
         NetworkVariableReadPermission.Everyone,
            NetworkVariableWritePermission.Owner);
```

Listing 4.2: Utilização de Network Variables Cliente

4.5 Instanciação do objeto do jogador

O NGO é capaz de realizar uma gestão automática do objeto do jogador, bastando apenas selecionar o mesmo no *NetworkManager*. Esta gestão apenas acontece se o objeto for comum a todos os clientes do jogo.

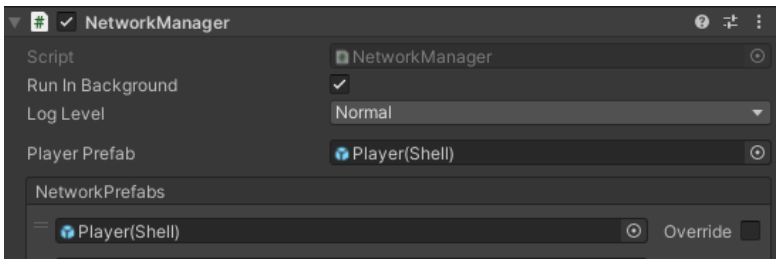


Figura 4.7: Objeto do jogador no NetworkManager

Se for a primeira vez que o jogador abre o jogo, é lhe apresentado um ecrã de seleção de personagem. O jogador faz a sua escolha e é inserido numa cena *offline* de tutorial. Em ligações futuras ao servidor o jogador já não vê a cena *offline* e é levado para a cidade automaticamente. Como o videojogo possui duas personagens diferentes, o objeto do jogador (PlayerShell) precisa de um componente para efetuar a realização da escolha da respetiva da personagem. Este componente (Apêndice C) é responsável por armazenar as preferências do jogador, sendo que, quando este é criado pelo NGO, pede ao servidor

para instanciar a personagem correspondente à escolha do jogador. Como demonstra o excerto 4.3, dependendo da cena corrente, o objeto (PlayerShell) chama uma função para a instanciação do objeto do jogador.

```
public void Spawn() {
        ...
        if(SceneManager.GetActiveScene().name == "CityLevel"){
            switch (CharID)
            {
                case 0:
                    SpawnPlayerServerRpc(NetworkManager.Singleton.
                    LocalClientId,0,new Vector3(0,540,181));
                    break;
                case 1:
                    SpawnPlayerServerRpc(NetworkManager.Singleton.
                    LocalClientId,1,new Vector3(0,540,181));
                    break;
            }
        }
```

Listing 4.3: Função de instanciação do jogador

4.6 Gestão de cenas em rede

Segundo o Unity [70] a gestão de cenas deve ser feita na maioria dos casos de modo integrado pelo o NGO. Isto significa que a cena que estiver carregada no momento pelo servidor é a cena que é apresentada aos jogadores.

O NGO oferece um componente denominado de *NetworkSceneManager*. Este componente sincroniza automaticamente as cenas e os objetos para os clientes durante a fase de sincronização aquando da ligação ao servidor.

O excerto 4.4 descreve a forma como o servidor deteta os jogadores presentes e carrega a nova cena.

```
if(CurrentScene.name == "CityLevel"){
            if(other.tag == "Player"){
                ...
                if(NetworkManager.Singleton.
                ConnectedClientsIds.Count == PlayerIDs.Count){
                    NetworkManager.Singleton.
                    SceneManager.LoadScene("FirstDungeon",
                    UnityEngine.SceneManagement.
                    LoadSceneMode.Single);
```

Listing 4.4: Função do servidor para carregar uma nova cena

O *NetworkSceneManager* disponibiliza também *SceneEvents* que permitem ao servidor e aos clientes subscreverem a eventos como, *LoadComplete* e *UnloadComplete*, *Load* e *Unload*. Estes eventos permitem invocar certas ações quando um evento é chamado quer pelo servidor como pelos clientes. Com o excerto 4.5, exemplifica-se a função executada quando um servidor pede para carregar uma cena, com recurso a esta função pode-se ativar o *Loading Screen* do jogo nos clientes, tal como referido no Anexo C.

```
public void Start(){
        NetworkManager.SceneManager.OnSceneEvent +=
           SceneManager_OnSceneEvent;
    }
private void SceneManager_OnSceneEvent(SceneEvent sceneEvent){
        ...
        switch (sceneEvent.SceneEventType){
        case SceneEventType.Load:
            //Ativar o Loading Screen
            break;
        }
        ...
    }
```

Listing 4.5: Código executado pelo jogador aquando de um carregamento de nova cena

4.7 Problemas encontrados

No decorrer do desenvolvimento deste projeto foram encontrados alguns problemas. Este subcapitulo menciona estes mesmos problemas e as consequências dos mesmos para o projeto.

4.7.1 Mirror e MLAPI

O projeto começou por ser desenvolvido com recurso ao Mirror mas como foi necessário atualizar a versão do Unity para corrigir alguns problemas presentes na versão 2020 LTS, o Mirror na altura não suportava as versões 2021, então desistiu-se do Mirror e mudou-se a implementação para MLAPI que era um "sucessor"do UNET.

O projeto foi reescrito em MLAPI, e passado alguns meses desta implementação, o MLAPI foi comprado pelo Unity e reestruturado com um novo nome *NetCode for GameObjects*. Isto levou à reescrita do projeto mais uma vez visto que a API do agora NGO tinha sido mudada substancialmente.

4.7.2 Sincronização de animações dos clientes

O NGOfoi pensado para ser intuitivo e fácil de implementar mas, quando se começou a utilizar esta ferramenta, encontrou-se um grande problema. O sistema de sincronização de animações não funciona corretamente se for utilizado o componente *ClientNetworkTransform*. Este componente permite programar os objetos de jogadores com controlo do lado do cliente em vez do lado do servidor, o que à partida deveria ser mais fácil de implementar. No entanto quando o mesmo é utilizado os clientes não sincronizam os estados das animações uns com os outros. Como mostra a imagem 4.8, a equipa deste projeto abriu um pedido de resolução de problemas no GitHub no dia 23 de Outubro de 2021 mas, até à data deste documento, o problema ainda não foi resolvido apesar de ter prioridade alta atribuída pelos programadores do Unity.

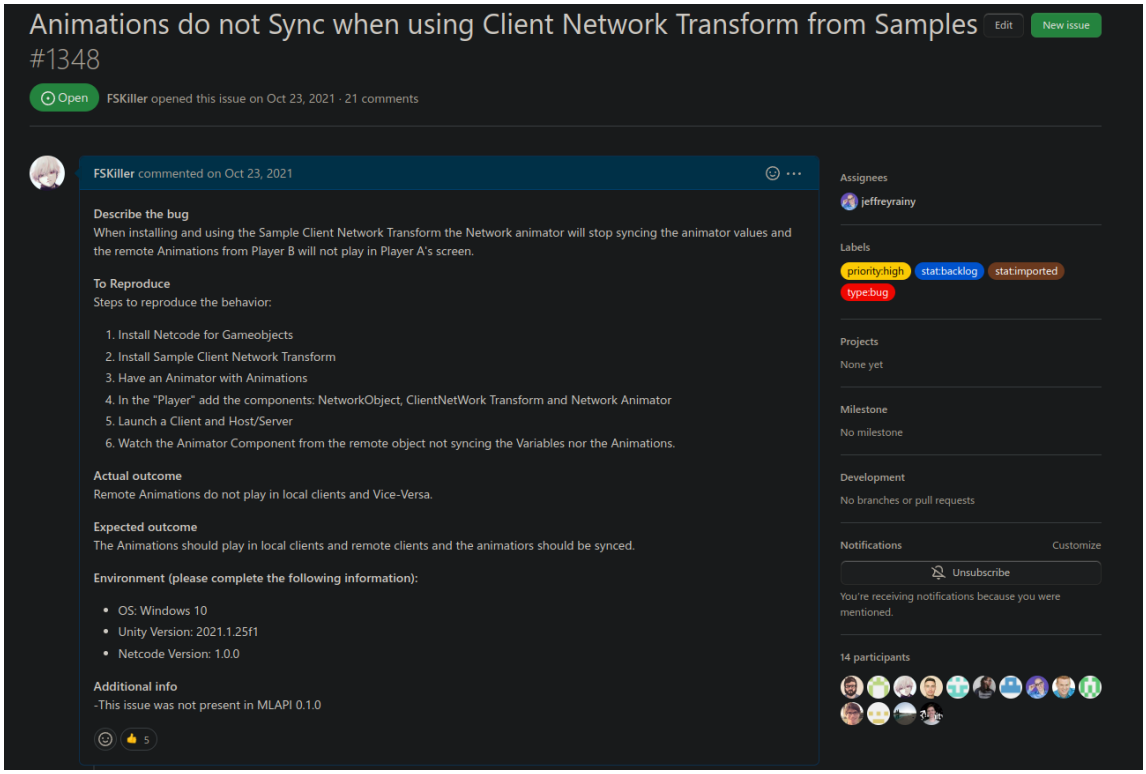


Figura 4.8: Problema aberto no GitHub

4.7.3 Dedicated Server Build

Esta funcionalidade foi introduzida no dia 20 de Julho de 2021 [71] na versão 2021.2 do Unity e, teoricamente, deveria permitir uma compilação muito mais rápida de um servidor dedicado visto que deveria remover todo o conteúdo que não seria utilizado pelo servidor como texturas, modelos 3D e *Shaders*. No entanto esta funcionalidade, à data de escrita deste documento ainda não se encontra completa pois, apesar de se poder compilar um servidor dedicado, este continua a possuir todos os componentes que deveriam ser removidos o que leva a um tempo de compilação muito elevado.

4.7.4 Unity

Por vezes o Unity recusa-se a instalar componentes selecionados pelo utilizador, que origina uma instalação incompleta. Identificou-se este problema quando se tentou compilar um servidor do jogo e o Unity disse que o componente para tal não se encontrava instalado. Para mitigar temporariamente o problema, visto que ainda não existe solução [72], editou-se na pasta do editor do Unity o ficheiro "modules.json"e na secção ""selected":true,",

como pode ser observado na figura 4.9, altera-se o valor para “false”e abre-se novamente o UnityHub para instalar novamente o componente.


```
{
  "id": "windows-server",
  "name": "Windows Dedicated Server Build Support",
  "description": "Allows building your Unity projec
  "downloadUrl": "https://download.unity3d.com/down
  "category": "Platforms",
  "installedSize": 1785545000,
  "downloadSize": 550610964,
  "visible": true,
  "selected": true,
  "destination": "{UNITY_PATH}/Editor/Data/Playback
  "checksum": "ee348ced5e363c75190bc7809c68858c",
  "preselected": false
}
```


Figura 4.9: modules.json

4.8 Solução alternativa para a sincronização de animações

Para dar a volta ao problema de sincronização de animações mencionado no subcapítulo 4.7.2, implementpu-se, com recurso a RPC, um método de sincronização. A desvantagem desta abordagem é que sempre que algum cliente entra ou sai de um estado de animação, é enviado um RPC para o servidor, que por sua vez envia para todos os clientes um RPC de volta para que estes atualizem o estado da animação atual.

```
if (_hasAnimator){
        if(_animationBlend < 0.01f){
            AnimatorSetFloatServerRpc(_animIDSpeed, 0.0f);
        }
        ...
```

Listing 4.6: Pedido ao servidor para atualizar o estado de animações

4.9 Autoridade de cliente e servidor

Neste projeto, grande parte do controlo está do lado do cliente, isto deve-se ao facto de se querer que o jogador experienciasse uma jogabilidade suave sem requer muito tráfego de rede.

Os movimentos do jogador, as estatísticas do mesmo e o inventário estão todos programados para serem do controlo do cliente. Já o servidor controla as mudanças de cena, os objetos de rede, o movimento, as animações, a posição , vida e ataques dos inimigos. O servidor controla também o estado das animações dos jogadores devido ao problema descrito no subcapítulo 4.7.2.

O NGO permite usar o mesmo *script* no cliente e no servidor, bastando apenas identificar os blocos de código que pertencem a cada um. Isto é possivel pois o NGO fornece várias variáveis que nos permitem identificar se somos um cliente (IsClient ou IsLocalClient) ou servidor (IsServer). Ao colocar código especifico para cada uma das partes pode-se segmentar o mesmo código e ter a execução do mesmo divido entre cliente e servidor tal como é demonstrado no excerto 4.7.

```
public void Start(){
        if(IsServer){
            return;
        }
        if(IsLocalPlayer){
            Spawn();
        }
    }
```

Listing 4.7: Exemplo de diferenciação entre cliente e servidor

4.10 Sistema para guardar o progresso dos jogadores

Como era necessário um sistema que permitisse alterações rápidas aos endereços IP do servidor e cliente optou-se por juntar a configuração de IP do cliente com o progresso do mesmo. As alterações rápidas são úteis pois fornecem uma maneira rápida de realizar alterações ao nível, classe, vida e inventário de um jogador através de uma edição de um ficheiro.

Para a implementação usou-se uma ferramenta chamada *SharpConfig*. Esta ferramenta permite guardar múltiplos tipos de dados incluindo *Strings*, *Integers*, *Floats*, *Booleans*, *Arrays* e até dias da semana. Para usufruir da mesma é apenas necessário efetuar o

download da mesma e importar para o editor do Unity.

Para criar um novo ficheiro de configuração usa-se a função presente no excerto 4.8.

```
...
using SharpConfig;
...
public void initSave(int customID){
    var myConfig = new Configuration();
    switch (customID)
    {
        case 0:
            myConfig["Server"]["IP"].StringValue = "127.0.0.1";
            myConfig["Server"]["Port"].IntValue = 7777;
            myConfig["Player"]["CharID"].IntValue = customID;
            myConfig["Player"]["CurrentHealth"].IntValue = 150;
            myConfig["Player"]["MaxHealth"].IntValue = 150;
            myConfig["Player"]["Level"].IntValue = 1;
            ...
            myConfig["Player"]["Money"].IntValue = 0;
            myConfig["Player"]["HealthPotions"].IntValue = 1;
            myConfig.SaveToFile("Abyss.ini");
            break;
        ...
    }
```

Listing 4.8: Função para guardar uma configuração

Para carregar uma configuração já existente usa-se a função presente no excerto 4.9.

```
    public void loadStats(){
        ...
        var config = Configuration.LoadFromFile("Abyss.ini");
        PlayerCharacter.level = config["Player"]["Level"].IntValue;
        ...
        PlayerStats.agility = config["Player"]["Agility"].IntValue;
        PlayerInventory.SetCurrency(
        config["Player"]["Money"].IntValue);
```

```
        PlayerInventory.SetPotions(
        config["Player"]["HealthPotions"]
        .IntValue);
    }
```

Listing 4.9: Função para carregar uma configuração

Como se pode observar na figura 4.10, no final obtém-se um ficheiro que possui as informações para a ligação ao servidor e as estatísticas do jogador. Este ficheiro reside na pasta onde o executável do jogo se encontra. Qualquer alteração realizada no conteúdo deste ficheiro irá refletir-se em jogo. Esta abordagem permite uma alteração valores rápida num ambiente de testes.

```
[Server]
IP=127.0.0.1
Port=7777

[Player]
CharID=1
Level=1
MaxExperience=125
CurrentExperience=74
CurrentHealth=159
MaxHealth=250
Inteligence=2
Strenght=7
Agility=4
Money=800
HealthPotions=997
```

Figura 4.10: Ficheiro de configuração do jogador

4.11 Lógica operacional

Quando o jogo é aberto pela primeira vez, é mostrado ao jogador uma cena inicial (*TitleScreen*). Esta cena, ao ser carregada, possui um objeto que verifica se já existe um ficheiro de configurações. Se este não estiver presente e o jogador carregar no botão de *Play*, é levado para uma cena onde pode escolher a personagem desejada (*Mage* ou *Warrior*). Quando o jogador escolhe uma personagem é carregada uma cena de "tutorial"(*PrisonLevel*) para o jogador se poder ambientar às mecânicas do jogo. Estas três cenas (*TitleScreen*,*ClassSelection* e *PrisonLevel*) funcionam no modo *offline* e não requerem ligação a um servidor.

Quando o jogador chega ao fim da cena da *PrisonLevel* é feita a ligação com o servidor e é carregada a cena principal do jogo a *CityLevel*. Se existir um ficheiro de configuração, o jogador é levado automaticamente para a cidade. Nesta cena o jogador já pode ver outros jogadores ligados ao servidor. Quando um jogador entra no portal localizado nesta cena, o servidor verifica se o número de pessoas no portal é igual ao número de jogadores no servidor. Se a contagem for igual, então este carrega a próxima cena e envia uma mensagem de *On Scene Event Load* para todos os clientes para que estes possam carregar também a cena.

A cena da *Dungeon* contém os inimigos e o *Boss* final que os jogadores têm de derrotar para progredir no jogo. Ao serem derrotados, os inimigos têm uma probabilidade de deixar para trás poções ou moedas que os jogadores podem apanhar. Estas moedas e poções estão sincronizadas em rede o que significa que o primeiro jogador a apanhar estes consumíveis é que fica com eles. No caso das poções se o jogador já possuir três então não pode apanhar mais, dando assim oportunidade aos restantes jogadores para as ir buscar. No fim deste nível os jogadores encontram numa sala o inimigo final (*Boss*), ao entrar uma grades que impedem o jogador de sair do encontro com o *Boss*.

Ao derrotar o inimigo final, este deixa também para trás moedas e poções, e as grades da sala abrem dando ao jogador oportunidade de regressar à cidade através de outro portal. Apesar de o *Boss* estar sincronizado em rede, as grades presentes na sala do mesmo não estão, isto dá oportunidade aos jogadores que ainda não tenham entrado na sala para o fazer e impede os jogadores presentes na sala de sair.

Capítulo 5

Testes e discussão de resultados

Concluída a fase de desenvolvimento verificou-se a existência de algum impacto no desempenho devido à implementação do sistema de rede. Para tal, nesta análise, usou-se as estatísticas do Unity para efetuar uma comparação da contagem dos *Frames per second (frames/s)* nos modos *online* e *offline*.

O projeto foi enviado a um grupo de doze pessoas para testarem o jogo. A este mesmo grupo foi feito o convite para responder a um questionário acerca do estado atual do jogo.

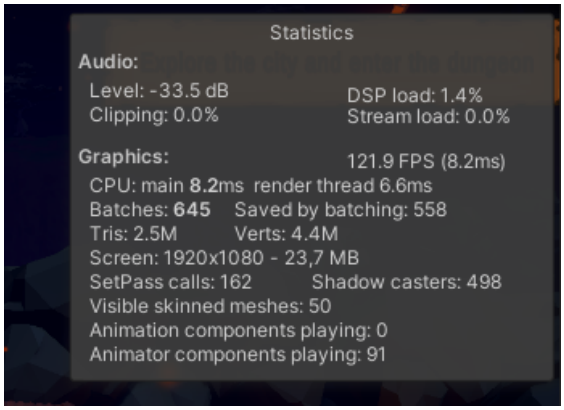
5.1 Impacto do sistema multijogador no desempenho do jogo

Uma das grandes preocupações para os jogadores é o desempenho do jogo e, segundo a análise realizada, não se encontrou um impacto significativo no desempenho do jogo, aqui medido em *frames/s* do jogo.

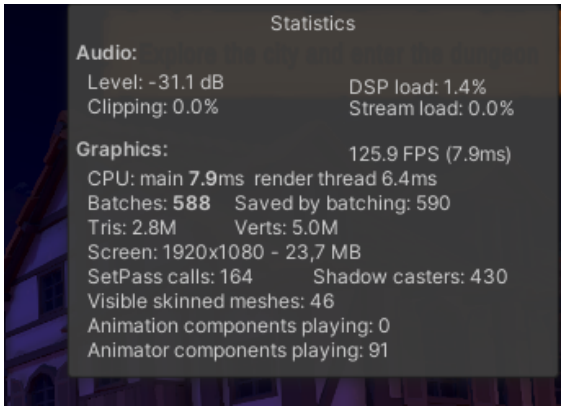
Como se observa nas sub-figuras (a) e (b) da figura 5.1, a diferença de desempenho na cena da cidade, que é o maior mapa para os jogadores explorarem, é negligível, com apenas uma diferença de quatro imagens por segundo. Visto que em média obtém-se 110 imagens por segundo, a diferença é impercetível sem o uso de um monitor de estatísticas. Já na cena da *dungeon*, observa-se uma diferença de sete *frames/s*, tal como se pode verificar nas sub-figuras (c) e (d).

Em tráfego de rede, verificou-se, através do *profiler* do Unity, que, em média, cada cliente envia 121 bytes para o servidor e recebe do mesmo em média 400 bytes a cada

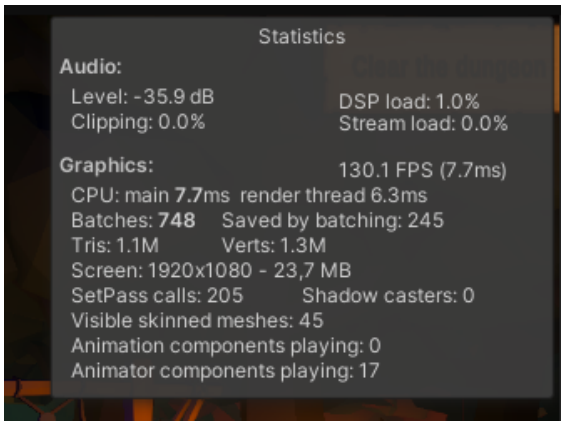
frame do jogo. Para um jogador que tenha 60 *frames/s* este envia cerca de 7,26 K por segundo para o servidor e recebe do mesmo em média 24 K, para um total de tráfego por segundo de 31 KB/s.



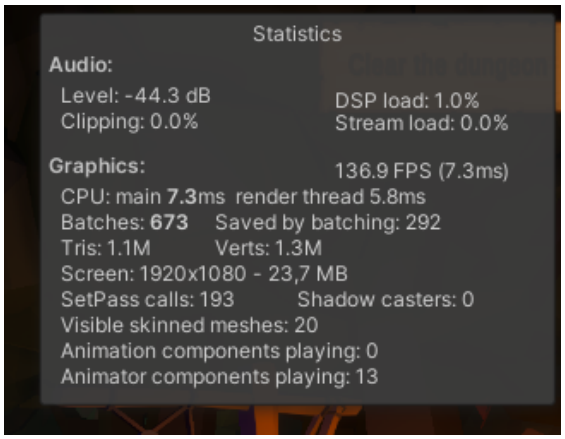
(a) Desempenho *online* na cidade



(b) Desempenho *offline* na cidade



(c) Desempenho *online* na *dungeon*



(d) Desempenho *offline* na *dungeon*

Figura 5.1: Desempenho dos níveis em rede

5.2 Inquérito sobre o estado atual do jogo

Para obter opiniões sobre o estado atual do projeto, foi disponibilizada uma versão de teste do videojogo a um grupo de doze pessoas que jogam com frequência. Aquando do término da demonstração, os participantes deste grupo foram convidados a responder a um breve questionário sobre cada área de desenvolvimento do videojogo. Esta secção expõe as questões efetuadas e as suas respostas.

Dos participantes, 92% reportaram que não experienciaram nenhum problema relacionado com o sistema implementado, tal como é observado na figura 5.2. No questionário pediu-se que, caso existisse algum problema em rede, para este ser descrito numa caixa de texto. Infelizmente a pessoa que colocou que observou problemas não preencheu este campo.

Figura 5.2: Problemas relacionados com o sistema de rede

Na figura 5.3 verifica-se que a grande maioria, com 83%, obteve um desempenho satisfatório em rede. Contudo, duas pessoas esperavam mais, o que indica que ainda existe espaço para melhoramento do projeto.

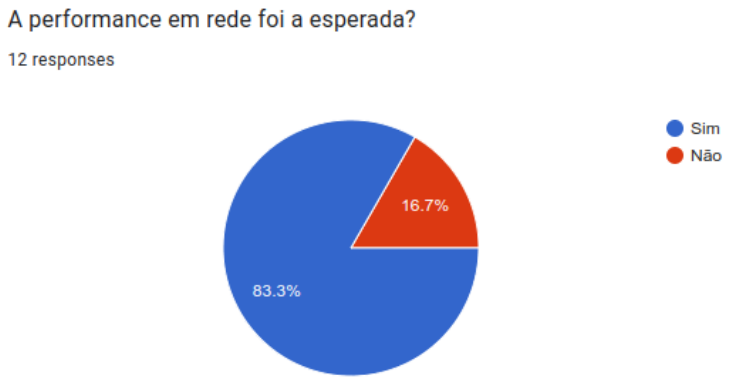


Figura 5.3: Desempenho em rede

Com este questionário determinou-se, ainda que a grande maioria dos jogadores preferia que estivesse disponível um sistema P2P como, por exemplo, através de convites na Steam. Apesar de 83% dos inquiridos ter gostado da implementação em servidor dedicado, como é possível verificar na figura 5.5, é de notar o interesse na funcionalidade P2P.

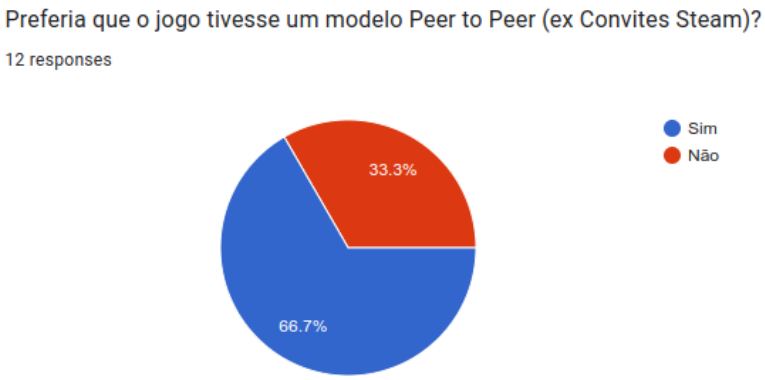


Figura 5.4: P2P ou servidor dedicado?

Figura 5.5: Taxa de aceitação do servidor dedicado

Na última pergunta relacionada com a implementação do sistema de rede, obteve-se um resultado misto de 50/50. Como se verifica na figura 5.6, estes valores indicam que talvez uma abordagem onde o jogador possa escolher se quer que o seu servidor seja mostrado ou não numa lista de servidores seja o ideal para satisfazer ambas as partes.

Por fim, foram colocadas duas perguntas sobre o estado geral do jogo, para as quais as respostas foram bastante positivas. Tal como a figura 5.7 demonstra, com uma pontuação média de quatro valores em cinco, dez pessoas reportaram que a experiência tinha sido positiva e, como se observa na figura 5.8, todos os inquiridos responderam que gostariam de acompanhar o desenvolvimento do jogo.

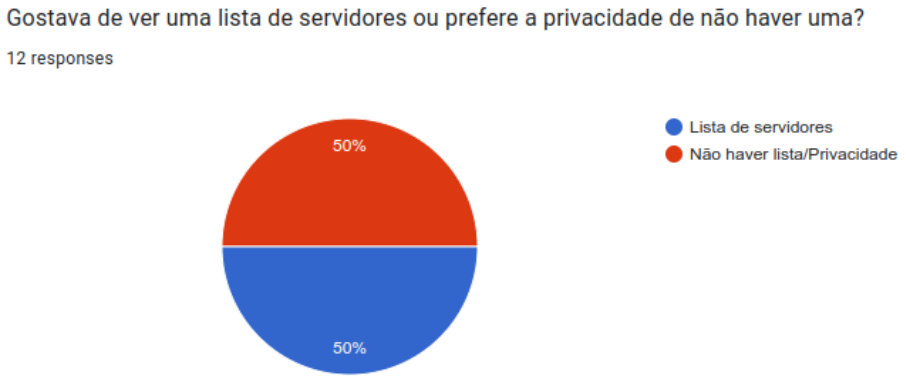


Figura 5.6: Lista de servidores

Figura 5.7: Experiência geral do jogo

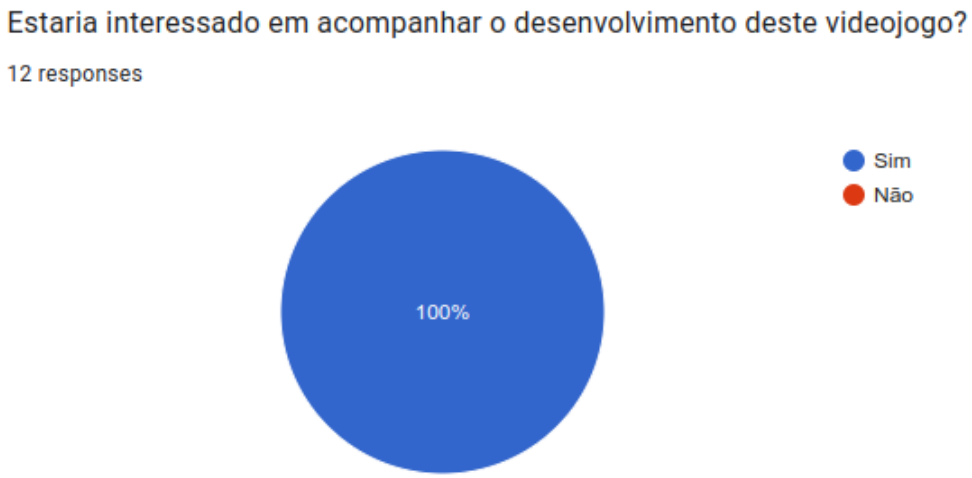


Figura 5.8: Interesse no jogo

Capítulo 6

Conclusão

Pode-se concluir que, para o desenvolvimento de um jogo, são necessárias várias competências.

Com o advento das lojas de jogos digitais, o processo de distribuição encontra-se mais facilitado, permitindo, assim novos criadores no mercado dos videojogos. Um motor é sempre uma peça fundamental para o desenvolvimento de um jogo. Existem vários concorrentes possíveis no mercado dos motores e com funcionalidades distintas, onde o Unity é utilizado por cerca de 50% dos jogos.

Conclui-se, ainda, que dentro dos motores existem várias formas para se construir uma implementação multijogador e que esta requer atenção em várias frentes, nomeadamente, interações, tamanho dos grupos formados dentro do jogo e tipos de jogabilidade, como por exemplo a assimétrica.

Devido ao facto de se ter utilizado ferramentas *bleeding-edge*, foram encontrados alguns problemas, como referido no subcapítulo 4.7, mas estes foram superados mesmo que com alguma dificuldade. Graças ao uso do NGO, que se encontra ainda nas primeiras fases de desenvolvimento, foi possível uma aprendizagem sobre como irá funcionar esta ferramenta no futuro e ajudar no seu desenvolvimento. Conclui-se, ainda, que apesar de o NGO se encontrar numa fase inicial, já possui um leque bastante grande de ferramentas para o desenvolvimento de um jogo num formato multijogador. Relativamente ao projeto desenvolvido, os comentários recebidos foram bastante positivos, o que indica que a metodologia e ferramentas usadas têm potencial para o futuro. Ajudou, também, a perceber que existe um grande interesse na utilização de um sistema P2P para o jogo e dos

problemas atualmente presentes com o mesmo.

6.1 Trabalho Futuro

Como o desenvolvimento deste projeto se baseou em experimentação, algumas partes poderão, possivelmente, ser reescritas para melhorar o desempenho de alguns componentes. Os componentes dos inimigos e jogadores poderão ser melhorados para adicionar uma maior modularidade.

Planeia-se migrar o método atual de transporte de rede de UNET, que foi descontinuado, para Unity Transport e reescrever o sistema desenvolvido para guardar o progresso dos jogadores, visto que o método atual permite uma modificação de todas as informações do jogador, o que poderia levar ao uso indevido desta funcionalidade.

Planeia-se, ainda, investigar a possibilidade de implementação de um sistema P2P. De acordo com os resultados obtidos, esta funcionalidade demonstra um grande interesse por partes dos membros do grupo de teste.

Referências

[1] Mark J. P. Wolf. *Before the Crash: Early Video Game History (Contemporary Approaches to Film and Media Series)*. Wayne State University Press, 2012, p. 272. ISBN: 0814334504.

[2] Marcus Toftedahl e Henrik Engström. "A Taxonomy of Game Engines and the Tools that Drive the Industry". Em: (2019). URL: http://www.digra.org/wp-content/uploads/digital-library/DiGRA_2019_paper_164.pdf.

[3] Ulf Hagen. "Where do game design ideas come from? invention and recycling in games developed in Sweden". Em: *Breaking New Ground: Innovation in Games, Play, Practice and Theory - Proceedings of DiGRA 2009* (2009), pp. 1–11.

[4] Casey O'Donnell. "The everyday lives of video game developers: Experimentally understanding underlying systems/structures". Em: *Transformative Works and Cultures* 2 (fev. de 2009). DOI: 10.3983/twc.2009.073. URL: https://journal.transformativeworks.org/index.php/twc/article/view/73/76%20https://journal.transformativeworks.org/index.php/twc/article/view/73.

[5] F Ted Tschang e Janusz Szczypula. "Idea Creation, Constructivism and Evolution as Key Characteristics in the Videogame Artifact Design Process". Em: *European Management Journal* 24.4 (2006), pp. 270–287. URL: https://econpapers.repec.org/RePEc:eee:eurman:v:24:y:2006:i:4:p:270-287.

[6] Peter Zackariasson, Alexander Styhre e Timothy L. Wilson. "Phronesis and Creativity: Knowledge Work in Video Game Development". Em: *Creativity and Innovation Management* 15.4 (dez. de 2006), pp. 419–429. ISSN: 14678691. DOI: 10.1111/j.1467-8691.2006.00400.x. URL: https://onlinelibrary.wiley.com/doi/full/10.1111/j.1467-8691.2006.00400.x%20https://onlinelibrary.wiley.com/

doi/abs/10.1111/j.1467-8691.2006.00400.x%20https://onlinelibrary.wiley.com/doi/10.1111/j.1467-8691.2006.00400.x.

[7] Simon Egenfeldt-Nielsen, Jonas Heide Smith e Susana Pajares Tosca. *Understanding Video Games*. Routledge, dez. de 2015. DOI: 10.4324/9781315725161. URL: https://www.taylorfrancis.com/https://www.taylorfrancis.com/books/mono/10.4324/9781315725161/understanding-video-games-simon-egenfeldt-nielsen-jonas-heide-smith-susana-pajares-tosca.

[8] Minako O'Hagan e Carmen Mangiron. *Game Localization*. Vol. 106. Benjamins Translation Library. Amsterdam: John Benjamins Publishing Company, ago. de 2013. ISBN: 978 90 272 2456 9. DOI: 10.1075/btl.106. URL: http://www.jbe-platform.com/content/books/9789027271860.

[9] Matthew Thomas Payne e Gregory Steirer. "Redesigning game industries studies". Em: *Creative Industries Journal* 7.1 (2014), pp. 67–71. ISSN: 17510708. DOI: 10.1080/17510694.2014.892292.

[10] Leônidas S. Pereira e Maurício M.S. Bernardes. "Aspects of independent game production: An exploratory study". Em: *Computers in Entertainment* 16.4 (2018). ISSN: 15443981. DOI: 10.1145/3276322.

[11] A. Andrade. "Game engines: a survey". Em: *EAI Endorsed Transactions on Game-Based Learning* 2.6 (2015), p. 150615. DOI: 10.4108/eai.5-11-2015.150615.

[12] DevMaster.net. *DevMaster.net - Engines Listing*. 2012. URL: https://web.archive.org/web/20110817040956/http://www.devmaster.net/engines/list.php (acedido em 03/06/2021).

[13] Akekarat Pattrasitidecha. "Comparison and evaluation of 3D mobile game engines AKEKARAT PATTRASITIDECHA Comparison and evaluation of 3D mobile game engines". Em: February (2014), p. 45. URL: http://publications.lib.chalmers.se/records/fulltext/193979/193979.pdf.

[14] Unity. *Unity Real-Time Development Platform — 3D, 2D VR & AR Engine*. 2021. URL: https://unity.com/ (acedido em 03/06/2021).

[15] Unity. *Get a Unity Student plan today — Free 3D & VR software for students — Unity - Unity Store*. 2021. URL: https://store.unity.com/academic/unity-student (acedido em 03/06/2021).

[16] Nestor Gomez. *Godot Engine, el Motor de Videojuegos Open Source más completo*. 2017. URL: https://www.headsem.com/godot-engine-el-motor-de-videojuegos-open-source-mas-completo/.

[17] Wikipédia. *Godot (game engine)*. URL: https://en.wikipedia.org/w/index.php?title=Godot_(game_engine)&oldid=1062894393.

[18] Juan Linietsky. *Godot 2.0: Talking with the Creator*. 2016. URL: https://80.lv/articles/godot2-interview/.

[19] Godot Docs. *Getting started*. 2019. URL: https://docs.godotengine.org/en/stable/getting_started/step_by_step/filesystem.html.

[20] EPIC GAMES. *Unreal ENgine*. 2022. URL: https://www.unrealengine.com/en-US/.

[21] Unity. *Unity*. 2022. URL: https://unity.com/.

[22] Godot. *Godot*. 2022. URL: https://godotengine.org/.

[23] Wikipédia. *Multiplayer Video Game*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Multiplayer_video_game&oldid=1065213288.

[24] Wikipédia. *Gauntlet (1985 video game)*. 2021. URL: https://en.wikipedia.org/w/index.php?title=Gauntlet_(1985_video_game)&oldid=1062073028.

[25] SteamCharts. *Steamcharts TOP*. 2022. URL: https://steamcharts.com/top.

[26] Valve e Electronic Arts. *Battlefield™ 2042*. 2022. URL: https://store.steampowered.com/app/1517290/Battlefield_2042/.

[27] Valve. *Counter-Strike: Global Offensive*. 2012. URL: https://store.steampowered.com/app/730/CounterStrike_Global_Offensive/.

[28] ESA. *2020 Essential Facts About the Video Game Industry*. 2020. URL: https://www.theesa.com/resource/2020-essential-facts/.

[29] Daniel Cook. *What I've learned about designing multiplayer games so far*. 2014. URL: https://www.gamedeveloper.com/design/what-i-ve-learned-about-designing-multiplayer-games-so-far.

[30] Wikipédia. *Lag(Video Games)*. 2021. URL: https://en.wikipedia.org/w/index.php?title=Lag_(video_games)&oldid=1060902525.

[31] Wikipédia. "Hitscan". Em: (2022). URL: https://en.wikipedia.org/w/index.php?title=Hitscan&oldid=1064958205.

[32] Bryan Wirtz. *Intro to Multiplayer Games and How to Create Your Own*. 2022. URL: https://www.gamedesigning.org/learn/multiplayer/.

[33] Wikipédia. *Game Server*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Game_server&oldid=1065235456.

[34] Mojang. *How to Setup a Minecraft: Java Edition Server*. 2022. URL: https://help.minecraft.net/hc/en-us/articles/360058525452-How-to-Setup-a-Minecraft-Java-Edition-Server.

[35] Valheim.fandom.com. *Valheim Dedicated Server*. 2022. URL: https://valheim.fandom.com/wiki/Valheim_Dedicated_Server.

[36] Chris Crawford. *The Art of Computer Game Design*. Berkeley, California: McGraw-Hill/Osborne Media, 1984. ISBN: 0-88134-117-7.

[37] Nathan Lawrence. *Understanding TTK and why it was important for BFV to revert its changes*. 2018. URL: https://www.redbull.com/au-en/understanding-ttk-and-why-it-was-important-for-bfv-to-revert-its-changes#:$%5Csim$:text=Time%20to%20Kill%20(TTK)%20is,your%20opponent%20will%20kill%20you..

[38] Daniel Cook. *Loops and Arcs*. 2012. URL: https://lostgarden.home.blog/2012/04/30/loops-and-arcs/.

[39] Tynan Sylvester. *Response to Danc's Loops and Arcs*. 2012. URL: https://tynansylvester.com/2012/04/response-to-dancs-loops-and-arcs/.

[40] Raph Koster. *Social Mechanics*. 2011. URL: https://www.raphkoster.com/wp-content/uploads/2011/02/Koster_Social_Social-mechanics_GDC2011.pdf.

[41] r/WarframeRunway/. *Warframe Runaway*. 2022. URL: https://www.reddit.com/r/WarframeRunway/.

[42] AlexandraLive. *FASHION FRAME: NOT JUST END GAME*. 2018. URL: https://www.warframe.com/news/fashion-frame-not-just-end-game.

[43] Waframe.market. *Warframe Market*. 2022. URL: https://warframe.market/.

[44] Ernest Adams e Joris Dormans. *Game Mechanics: Advanced Game Design (Voices That Matter)*. 2012, p. 360. ISBN: 0321820274. URL: http://www.amazon.com/Game-Mechanics-Advanced-Design-Voices/dp/0321820274.

[45] Modulate.ai. *Gaming, Voice Chat, and Transcending Toxicity*. 2021. URL: https://www.modulate.ai/blog/gaming-voice-chat-and-transcending-toxicity.

[46] Josh Bycer. *Designing for Multiplayer*. 2018. URL: https://superjumpmagazine.com/designing-for-multiplayer-4f37f6905140.

[47] Wikipédia. *Dunbar's number*. 2021. URL: https://en.wikipedia.org/w/index.php?title=Dunbar%27s_number&oldid=1060112158.

[48] Rust.fandom.com. *Jargon*. 2021. URL: https://rust.fandom.com/wiki/Jargon?oldid=97574.

[49] Techopedia. *Zerg*. 2022. URL: https://www.techopedia.com/definition/27053/zerg.

[50] Enardo. *Zerging a Rich Clan Until They're Poor - Rust*. 2021. URL: https://www.youtube.com/watch?v=B87_Ab-YVjM&ab_channel=Enardo.

[51] Josh Bycer. *Asymmetrical Game Design*. 2019. URL: https://superjumpmagazine.com/asymmetrical-game-design-2d3ccbc2b4ab.

[52] Mika Anttonen. "Online Multiplayer on Mobile Game". Em: *TURKU UNIVERSITY OF APPLIED SCIENCES Information* (2019).

[53] Chatziagapis Alexandros. "KILLSHOT : Online multiplayer FPS / TPS game in Unity3D". Em: (2019), pp. 1–86.

[54] Timo Jetsonen. "Development of online game prototype with Unity engine : Multiplayer solutions". Em: *Undefined* May (2016).

[55] D. Polancec e I. Mekterovic. "Developing MOBA games using the Unity game engine". Em: *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2017 - Proceedings* (2017), pp. 1510–1515. DOI: 10.23919/MIPRO.2017.7973661.

[56] Akshay Gautam et al. "Battle Royale: First-Person Shooter Game". Em: *SSRN Electronic Journal* (2021). DOI: 10.2139/ssrn.3863304.

[57] Mike Schramm. *Chinese WoW hits 1 million concurrent players*. Abr. de 2008. URL: https://web.archive.org/web/20150203081030/http://wow.joystiq.com/2008/04/11/chinese-wow-hits-1-million-concurrent-players/ (acedido em 03/06/2021).

[58] Valve. *Peer-to-Peer Networking and Sharing Your IP Address - Steam Support*. 2021. URL: https://support.steampowered.com/kb_article.php?ref=9929-UAFN-9329 (acedido em 03/06/2021).

[59] Marcus Toftedahl. *Which are the most commonly used Game Engines?* 2019. URL: https://www.gamedeveloper.com/production/which-are-the-most-commonly-used-game-engines-.

[60] EPIC GAMES. *Networking and Multiplayer*. 2022. URL: https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/Networking/.

[61] EPIC GAMES. *Multiplayer in Blueprints*. 2022. URL: https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/Networking/Blueprints/.

[62] EPIC GAMES. *Online Subsystem Steam*. 2022. URL: https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Online/Steam/.

[63] Valve. *Steamworks Documentation ¿ Features*. 2022. URL: https://partner.steamgames.com/doc/features.

[64] EPIC GAMES e Visualistic Studios. *VisGM - Deathmatch - A Multiplayer Game Framework*. 2022. URL: https://www.unrealengine.com/marketplace/en-US/product/visgm-deathmatch?sessionInvalidated=true.

[65] Photon. *Introduction — Photon Engine*. 2021. URL: https://doc.photonengine.com/en-us/pun/current/getting-started/pun-intro (acedido em 03/06/2021).

[66] Mirror Networking. *Mirror Networking – Open Source Networking for Unity*. 2021. URL: https://mirror-networking.com/ (acedido em 03/06/2021).

[67] Steamworks.NET. *Steamworks.NET*. 2021. URL: https://steamworks.github.io/ (acedido em 03/06/2021).

[68] Unity. *Getting Started with Netcode*. 2022. URL: https://docs-multiplayer.unity3d.com/docs/getting-started/about.

[69] Linda Candy e Ernest Edmonds. "Practice-Based Research in the Creative Arts". Em: (2018), p. 7. DOI: 10.1162/LEON_a_01471.

[70] Unity. *Using NetworkSceneManager*. 2022. URL: https://docs-multiplayer.unity3d.com/netcode/current/basics/scenemanagement/using-networkscenemanager.

[71] Unity. *Unity 2021.2: Dedicated Server target and stripping optimizations now live! Please share feedback!* 2021. URL: https://forum.unity.com/threads/unity-2021-2-dedicated-server-target-and-stripping-optimizations-now-live-please-share-feedback.1143734/.

[72] Unity. *"DEDICATED SERVER SUPPORT FOR WINDOWS IS NOT INSTALLED"ERROR IN BUILD SETTINGS WINDOW*. 2021. URL: https://issuetracker.unity3d.com/issues/dedicated-server-support-for-windows-is-not-installed-error-in-build-settings-window.

Apêndices

Apêndice A

Código de deteção de servidor

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Unity.Netcode;
using UnityEngine.SceneManagement;
using Unity.Netcode.Transports.UNET;
using SharpConfig;
using System.IO;
using ParrelSync;
public class IsServer : MonoBehaviour
{
    UNetTransport transport;
    string ip;
    int port;
    // Start is called before the first frame update
    void Start()
    {
        bool isHeadless = IsHeadlessMode();
        if (isHeadless || ClonesManager.IsClone()) {
            try
            {
                var config =
                    Configuration.LoadFromFile("AbyssServer.ini");
```

```
                var section = config["Server"];
                ip = section["IP"].StringValue;
                port = section["Port"].IntValue;
                Debug.Log("IP: " + ip + " " + "Port: " + port);
            }
            catch (FileNotFoundException)
            {
                var myConfig = new Configuration();

                myConfig["Server"]["IP"].StringValue = "127.0.0.1";
                myConfig["Server"]["Port"].IntValue = 7777;
                myConfig.SaveToFile("AbyssServer.ini");
                ip = "127.0.0.1";
                port = 7777;
            }
            transport = NetworkManager.Singleton.
            GetComponent<UNetTransport>();
            transport.ConnectAddress = ip;
            transport.ServerListenPort = port;
            //NetworkManager.Singleton.ConnectionApprovalCallback
                += ApprovalCheck;
            NetworkManager.Singleton.StartServer();
            ClearServerConsole();
            var sceneManager =
                NetworkManager.Singleton.SceneManager;
            sceneManager.LoadScene("CityLevel",
                LoadSceneMode.Single);
        }
        else
        {
            enabled = false;
        }
    }

    public static bool IsHeadlessMode()
```

```
    {
        return UnityEngine.SystemInfo.graphicsDeviceType ==
            UnityEngine.Rendering.GraphicsDeviceType.Null;
    }

    private void ClearServerConsole()
    {
        Debug.Log("Server Started On: " + ip + ":" + port);
    }
}
```

Apêndice B

Numero Máximo de jogadores

```
    private void
        ApprovalCheck(NetworkManager.ConnectionApprovalRequest
        request, NetworkManager.ConnectionApprovalResponse response)
{
    // The client identifier to be authenticated
    var clientId = request.ClientNetworkId;

    // Additional connection data defined by user code
    var connectionData = request.Payload;

    if(NetworkManager.Singleton.ConnectedClientsIds.Count <4){
        response.Approved = true;
        response.CreatePlayerObject = true;
    }

    // The prefab hash value of the NetworkPrefab, if null the
        default
    NetworkManager player prefab is used
    response.PlayerPrefabHash = null;

    // If additional approval steps are needed, set this to true
        until the additional steps are complete
    // once it transitions from true to false the connection
```

```
    approval response will be processed.
    response.Pending = false;
}
```

Apêndice C

Script para instanciar os jogadores

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Unity.Netcode;
using SharpConfig;
using System.IO;
using UnityEngine.SceneManagement;
using StarterAssets;

public class SpawnPlayerClient : NetworkBehaviour
{
    [SerializeField] private GameObject playerPrefabA; //add
       prefab in inspector
    [SerializeField] private GameObject playerPrefabB; //add
       prefab in inspector
    private int CharID;

    private bool isSpawned = false;
    private bool hasMovedScene = false;

    public void Start(){
        if(IsServer){
            return;
```

```
        }
        if(IsLocalPlayer){
            NetworkManager.Singleton.SceneManager.OnSceneEvent +=
                SceneManager_OnSceneEvent;
            Spawn();
        }
    }

    private void SceneManager_OnSceneEvent(SceneEvent sceneEvent){
    // Both client and server receive these notifications
    switch (sceneEvent.SceneEventType){
        case SceneEventType.Load:
        {
            if(!IsServer){
                GameObject loadingScreen =
                    GameObject.FindGameObjectWithTag("LoadingUI");
                loadingScreen.GetComponent<LoadingUI>().screen.SetActive(true);
                loadingScreen.GetComponent<LoadingUI>().slider.value
                    = sceneEvent.AsyncOperation.progress;
                Debug.Log("Calling Loading Screen");
            }
            break;
        }
        case SceneEventType.LoadComplete:
            {
                if (!IsServer)
                {
                    Debug.Log("Player Moved Scenes");
                    hasMovedScene = true;
                }
                break;
            }
    }
}
    public void Update(){
```

```
        if(IsServer){
            return;
        }
        if(hasMovedScene){
            hasMovedScene = false;
            UpdatePlayers();
        }
    }

    public void UpdatePlayers(){
        SaveSystem SaveSystem =
           GameObject.FindGameObjectWithTag("SaveSystem").gameObject.
        GetComponent<SaveSystem>();
        CharID = SaveSystem.loadCharID();
        switch (CharID)
        {
            case 0:
                NetworkManager.Singleton.LocalClient.PlayerObject.
                GetComponent<ThirdPersonController>().Start();
                break;
            case 1:
                NetworkManager.Singleton.LocalClient.PlayerObject.
                GetComponent<ThirdPersonControllerWarrior>().Start();
                break;
            default:
                NetworkManager.Singleton.LocalClient.PlayerObject.
                GetComponent<ThirdPersonController>().Start();
                break;
        }

    }
    public void Spawn() {
        isSpawned = true;
        SaveSystem SaveSystem =
           GameObject.FindGameObjectWithTag("SaveSystem").gameObject.
```

```
GetComponent<SaveSystem>();
CharID = SaveSystem.loadCharID();
if(SceneManager.GetActiveScene().name == "CityLevel"){
    switch (CharID)
    {
        case 0:
            SpawnPlayerServerRpc(NetworkManager.Singleton.
            LocalClientId,0,new Vector3(0,540,181));
            break;
        case 1:
            SpawnPlayerServerRpc(NetworkManager.Singleton.
            LocalClientId,1,new Vector3(0,540,181));
            break;
        default:
            SpawnPlayerServerRpc(NetworkManager.Singleton.
            LocalClientId,0,new Vector3(0,540,181));
            break;
    }
}
if(SceneManager.GetActiveScene().name == "FirstDungeon"){
    switch (CharID)
    {
        case 0:
            SpawnPlayerServerRpc(NetworkManager.Singleton.
            LocalClientId,0,new Vector3(-5,46,-96));
            break;
        case 1:
            SpawnPlayerServerRpc(NetworkManager.Singleton.
            LocalClientId,1,new Vector3(-5,46,-96));
            break;
        default:
            SpawnPlayerServerRpc(NetworkManager.Singleton.
            LocalClientId,0,new Vector3(-5,46,-96));
            break;
    }
```

```
        }

    }

    [ServerRpc(RequireOwnership=false)] //server owns this object
       but client can request a spawn
    public void SpawnPlayerServerRpc(ulong clientId,int prefabId,
       Vector3 pos) {
        GameObject newPlayer;
        if (prefabId==0)
             newPlayer=(GameObject)Instantiate(playerPrefabA,pos,
             Quaternion.identity);
        else
            newPlayer=(GameObject)Instantiate(playerPrefabB,pos,
               Quaternion.identity);
        NetworkObject
           netObj=newPlayer.GetComponent<NetworkObject>();
        newPlayer.SetActive(true);
        netObj.SpawnAsPlayerObject(clientId,false);
    }
}
```