UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Pedro Henrique Braga Moraes

Estudo Comparativo das Soluções de Rede e de Dados para Jogos Massivos Online Desenvolvidos no Unity

Natal
2020, Dezembro

Pedro Henrique Braga Moraes

Estudo Comparativo das Soluções de Rede e de Dados para Jogos Massivos Online Desenvolvidos no Unity

Trabalho de Conclusão de Curso submetido à avaliação pelo corpo docente do curso de Engenharia Elétrica da Universidade Federal do Rio Grande do Norte a fim de obter o diploma de Engenheiro Eletricista

Universidade Federal do Rio Grande do Norte

Centro de Tecnologia

Departamento de Engenharia Elétrica

Curso de Graduação em Engenharia Elétrica

Orientador: Charles Andryê Galvão Madeira

Natal

2020, Dezembro

Pedro Henrique Braga Moraes

Estudo Comparativo das Soluções de Rede e de Dados para Jogos Massivos Online Desenvolvidos no Unity

Trabalho de Conclusão de Curso submetido à avaliação pelo corpo docente do curso de Engenharia Elétrica da Universidade Federal do Rio Grande do Norte a fim de obter o diploma de Engenheiro Eletricista

Trabalho aprovado. Natal, 22 de dezembro de 2020:

Charles Andryê Galvão Madeira
Orientador
Instituto Metrópole Digital - IMD
Universidade Federal do Rio Grande do Norte - UFRN

Agostinho de Medeiros Brito Junior
Examinador
Departamento de Engenharia de Computação e Automação
Universidade Federal do Rio Grande do Norte - UFRN

André Luiz de Souza Brito
Examinador
Instituto Metrópole Digital - IMD
Universidade Federal do Rio Grande do Norte - UFRN

Natal
2020, Dezembro

Aos meus pais.

Agradecimentos

Agradeço à Rosiane, a melhor mãe que eu poderia ter. Obrigado pelas horas de conversas durante o confinamento devido à pandemia, pelo suporte psicológico, por tudo. Sem você eu não teria sequer saído da estaca zero. Meu muito obrigado ao meu pai, Luís Carlos, minha inspiração e meu ideal. Espero um dia ser para meus filhos o que você é para mim. Obrigado à minha irmã, Luane, por sempre me apoiar em minhas escolhas e sempre me incentivar a arriscar nas coisas em que acredito. Agradeço a todos os membros da minha família, que sempre estiveram comigo, incluindo meu cachorro, Theodoro.

Obrigado aos meus amigos, pelos inúmeros momentos que dividimos ao longo desses anos de graduação, bons e ruins. As conversas, os trabalhos em grupo, os lanches, os sofrimentos, os filmes, os estudos, os jogos. Todos esses momentos foram muito importantes para mim e eu não poderia ter dividido com pessoas melhores.

Obrigado a todos os educadores que passaram pela minha vida, por todo o conhecimento que me proporcionaram. Agradeço especialmente ao meu orientador, Charles, por toda a orientação durante as horas de reuniões virtuais que possibilitaram a realização deste trabalho.

Por fim, agradeço a todos os livros que eu li e a todos os filmes que assisti. Alimentos para mente e para a alma.

"Percebi que colocando as primeiras coisas em primeiro lugar, teremos as segundas a seguir, mas, colocando as segundas em primeiro, perdemos ambas."

(C. S. Lewis)

Resumo

Os sistemas de *rede* para jogos online, além de complexos, estão sempre demandando soluções adequadas e modernas. A crescente quantidade de usuários e dados nos jogos massivos online, e a evolução tecnológica vista nos jogos digitais em geral, demandam uma evolução à altura nas respectivas infraestruturas de *rede* e de *dados*. Tendo em vista que há essa demanda, o presente trabalho apresenta um estudo comparativo entre diferentes soluções de rede e de dados para jogos online desenvolvidos no *motor de jogos Unity*. Nesse contexto, o *Photon Server* foi escolhido como solução para servidor auto-hospedado, o *Photon Unity Networking* foi utilizado para possibilitar o acesso ao *Photon Server*, e o *Azure PlayFab* foi selecionado como solução para armazenamento dos dados de usuários. A partir dessas escolhas, uma proposta de solução foi apresentada para contemplar funcionalidades básicas dos jogos massivos online. O modelo foi implementado em um protótipo e diferentes experimentos foram realizados para a validação da proposta. Dentre esses, foram realizados testes de cadastro e autenticação de usuários, armazenamento de dados persistentes, criação de salas de jogo e compartilhamento de informações, entre os usuários, em um mesmo *ambiente virtual*. Resultados satisfatórios foram obtidos nos testes realizados com o protótipo, o que demonstrou a viabilidade da proposta como solução de rede e de dados para jogos.

Palavras-chave: Jogos Multijogador; Aplicativo Cliente-Servidor; Unity.

Abstract

Network systems for online games, in addition to being complex, are always demanding adequate and modern solutions. The growing amount of users and data in massive online games, and the technological evolution seen in digital games in general, demand an up-to-date evolution in *network* and *data* infrastructures. In view of this demand, the present work presents a comparative study between different network and data solutions for online games developed on the *Unity game engine*. In this context, *Photon Server* was chosen as a solution for self-hosted server, *Photon Unity Networking* was used to provide access to *Photon Server*, and *Azure PlayFab* was selected as the solution for data storage of users. From these choices, a solution proposal was presented to contemplate basic features of massive online games. The model was implemented in a prototype and different experiments were performed for the validation of the proposal. Among these, registration and authentication of users, persistent data storage, creation of game rooms and information sharing among users in the same *virtual environment* tests were carried out. Satisfactory results were obtained in the tests performed with the prototype, which demonstrated the viability of the proposal as a network and data solution for games.

Keywords: Multiplayer Games; Client-Server Application; Unity.

Lista de figuras

Lista de tabelas

Lista de abreviaturas e siglas

IMD	Instituto Metrópole Digital
UFRN	Universidade Federal do Rio Grande do Norte
PLATO	Programming Logic for Automatic Teaching Operations
MUD	Multi-User Dungeon
LAN	Local Area Network
MMOG	Massive Multiplayer Online Game
MMORPG	Massively Multiplayer Online Role-Playing Games
ARPG	Action Role Playing Game
MMORTS	Massively Multiplayer Online Real-Time Strategy
FPS	First-Person Shooter
TPS	Third-Person Shooter
MOBA	Multiplayer Online Battle Arena
ARAM	All Random All Mid
UNET	Unity Networking
API	Application Programming Interface
HLAPI	High-Level Scripting Application Programming Interface
SDK	Software Development Kit
SaaS	Software as a Service
CCU	Concurrent Users
PUN	Photon Unity Networking
IP	Internet Protocol
IOCP	Input/Output Completion Port
UDP	User Datagram Protocol

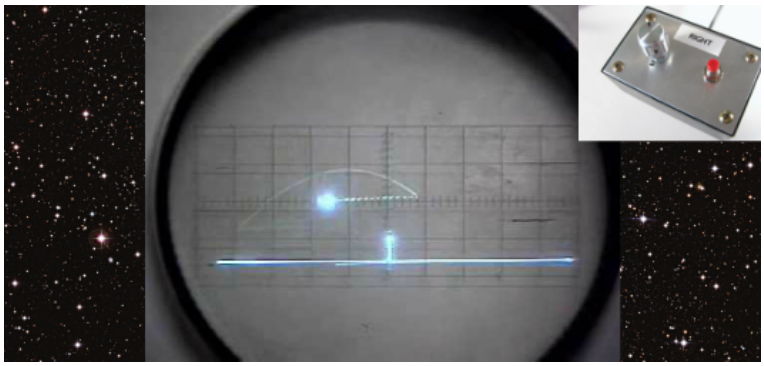
TCP	Transmission Control Protocol
HTTP	Hypertext Transfer Protocol
DLL	Dynamic Link Library

Sumário

1 Introdução

São diversas as categorias de *jogos digitais multijogador* atualmente. Contudo, os pioneiros foram os chamados jogos multijogador locais, nos quais geralmente o console ou computador é conectado a uma tela (televisão ou monitor), permitindo que vários jogadores participem de uma partida através do uso de controladores. Essa forma de jogar surgiu em 1958, com o desenvolvimento de *Tennis for Two* (1958). Criado pelo físico Willian A. Hinginbotham, o jogo consistia em uma simulação de um jogo virtual de tênis, utilizando um osciloscópio e controladores de alumínio personalizados (THE HISTORY OF VIDEO GAMES, 2020). Na Figura 1.1 pode ser visto o jogo e o controlador que era utilizado.

Figura 1.1 – O primeiro jogo virtual multijogador, *Tennis for Two*, era jogado utilizando um osciloscópio e controladores de alumínio personalizados.



Fonte: (THE HISTORY OF VIDEO GAMES, 2020)

Com o desenvolvimento das tecnologias disponíveis, os jogos multijogador evoluíram de forma a possibilitar a conexão de diferentes dispositivos através da rede. Uma das primeiras comunidades de rede online foi o PLATO (*Programming Logic for Automatic Teaching Operations* - Lógica de Programação para Operações de Ensino Automático), sistema no qual usuários poderiam se conectar aos servidores e utilizar mecanismos de comunicação que hoje fazem parte do dia a dia; como e-mail e chat (ARMITAGE; CLAYPOOL; BRANCH, 2006). Em meados dos anos setenta, surgiram os percursores dos *jogos online* no PLATO. Um dos mais populares foi *Empire* (1973), um jogo de simulação espacial do gênero *shoot'em up* e provavelmente o primeiro jogo multijogador de ação em rede.

Os MUDs (*Multi-User Dungeons*) se popularizaram logo após a difusão dos PLATOs. Consistiam em salas de chat online, sem gráficos, com elementos de jogabilidade básicos e

que existem até hoje nos jogos online; como múltiplas áreas para os usuários se moverem e interagirem, elementos de combate, armadilhas, *puzzles*, magia e economia.

Os anos setenta e começo dos anos oitenta, contudo, foram marcados pelos *jogos arcade*. Sendo um dos primeiros o chamado *Pong* (1972), semelhante ao *Tennis for Two*, mas desenvolvido para fliperamas e mais acessível para o público geral. O jogo foi o primeiro *video game* que foi um sucesso comercial (ARMITAGE; CLAYPOOL; BRANCH, 2006). A *Atari*, que desenvolveu *Pong*, emplacou diversos outros jogos multijogador de sucesso, como *Football* (1978) e *Gauntlet* (1985), que permitiam até quatro jogadores dividindo uma máquina arcade.

Nos anos noventa, os chamados jogos em LAN (*Local Area Network* - Rede de área local) se popularizaram. *Doom* (1993) foi um dos pioneiros no uso da tecnologia, a qual permitia a conexão entre diferentes dispositivos através de uma rede local para partidas multijogador. No jogo, até quatro jogadores poderiam jogar simultaneamente. *Doom 2* (1994), sequência do primeiro jogo, veio com uma arquitetura de rede mais robusta, permitindo até oito jogadores em LAN. Surgiram na mesma época softwares que permitiam o acesso de jogos à internet, entretanto, uma vez que esses jogos não tinham sido desenvolvidos visando partidas online, os problemas de alta latência eram frequentes.

Muitos jogos da mesma época foram precursores na popularização dos *jogos multijogador online* como são conhecidos hoje. *Quake* (1996) era compatível com partidas através da internet e possuía servidores que estavam sempre disponíveis para jogadores de todo o mundo se conectarem e jogarem entre si. *Warcraft II* (1995) permitia partidas multigador online com alta performance, possibilitando assim o surgimento de ligas competitivas; mesmo com os efeitos de latência, limitação da taxa de bits e limitações nas velocidades de internet. *Ultima Online* (1997), inicialmente suportando cinquenta jogadores simultâneos, foi o primeiro exemplo de sucesso de um *jogo multijogador massivo online*. *Final Fantasy XI* (2002), foi o primeiro do gênero a ser multiplataforma, com jogadores de PCs e consoles dividindo um mesmo mundo virtual (ARMITAGE; CLAYPOOL; BRANCH, 2006).

Hoje, os MMOGs (*Massive Multiplayer Online Games* - Jogos Multijogador Massivos Online) tornaram-se extremamente populares, contando com centenas de milhões de jogadores espalhados pelo mundo. Esses jogos costumam ser baseados em inscrição e possuir conteúdo persistente. Há uma enorme geração de dados, tanto em tamanho, quanto em quantidade (TSIPIS; KOMIANOS; OIKONOMOU, 2019). *Soluções* são necessárias para que seja possível manter a funcionalidade de um jogo digital dessas proporções.

1.1 Motivação

Existem três principais arquiteturas para hospedar jogos online: *cliente-servidor*, *peer-to-peer* (ponto a ponto) e *arquitetura multi-servidor*. Na arquitetura cliente-servidor,

as publicadoras (companhias responsáveis pela publicação dos jogos digitais) gerenciam os servidores e cedem ou vendem o jogo para os jogadores (clientes). Esse tipo de arquitetura permite um bom controle do conteúdo por parte da publicadora, porém requer uma considerável infraestrutura computacional para a gestão e manutenção do servidor (CHAMBERS; FENG; FENG, 2017). Na arquitetura ponto a ponto, a funcionalidade dos servidores é movida para os clientes, que se conectam uns aos outros diretamente. Para a arquitetura multi-servidor, o mundo do jogo é separado em várias zonas e cada uma é gerenciada por um servidor diferente, permitindo uma maior escalabilidade e flexibilidade (TSIPIS; KOMIANOS; OIKONOMOU, 2019).

Os sistemas de rede para jogos online, além de complexos, estão sempre demandando soluções adequadas e modernas. A crescente quantidade de usuários e dados nos jogos massivos online, e a evolução tecnológica vista nos jogos digitais em geral, demandam uma evolução à altura nas *infraestruturas de rede e de dados*.

Atualmente, existem diversas tecnologias disponíveis para o desenvolvimento de jogos digitais. Desde os *motores de jogos* (*game engines* em inglês) até os diversos *plugins* responsáveis por prover funcionalidades adequadas às intenções dos desenvolvedores na criação de jogos.

Portanto, o presente trabalho tem como motivação as demandas por atualizações e implantações de novas funcionalidades nas infraestruturas de rede e dados dos jogos multijogador online com grande quantidade de usuários. Mais especificamente, das demandas por soluções para jogos desenvolvidos com o motor de jogos *Unity*.

1.2 Objetivos

1.2.1 Objetivo Geral

A proposta do presente trabalho é verificar a viabilidade de soluções de rede e dados para Jogos Multijogador Massivos Online desenvolvidos no motor de jogos Unity. O objetivo citado será alcançado através de *estudos comparativos*, *prototipagem* e *experimentação*.

1.2.2 Objetivos Específicos

O objetivo geral deste trabalho pode ser subdividido nos seguintes objetivos específicos:

- Realizar um estudo comparativo entre diferentes *soluções de rede* para Unity;
- Realizar um estudo comparativo entre diferentes *soluções de dados* para Unity;
- Desenvolver um protótipo para testes das soluções de rede e de dados escolhidas;

- Verificar a viabilidade das soluções utilizadas através de baterias de testes.

1.3 Contribuições

As contribuições deste trabalho são:

- Documentação de um estudo comparativo entre as diferentes soluções de dados e rede para jogos multijogador massivos online desenvolvidos utilizando o motor de jogos Unity;
- Desenvolvimento de um documento de referência para uso das tecnologias utilizadas no presente trabalho;
- Desenvolvimento de um protótipo de jogo funcional com recursos de rede e dados.

1.4 Organização do Trabalho

No *Capítulo 1* é realizada uma contextualização e justificativa do trabalho, apresentando os objetivos do mesmo. O *Capítulo 2* é composto por uma fundamentação teórica sobre jogos multijogador online, além de apresentar um estudo comparativo entre diferentes tecnologias. No *Capítulo 3* são mostradas a proposta do projeto desenvolvido e as tecnologias utilizadas para tal. O *Capítulo 4* é composto pela descrição do desenvolvimento do protótipo. No *Capítulo 5* são descritos os testes realizados no protótipo. Enfim, no *Capítulo 6*, são apresentados os resultados dos experimentos e as perspectivas para futuros trabalhos.

2 Fundamentação Teórica

Este capítulo é composto, primeiramente, por um referencial teórico relativo aos jogos multijogador online e uma contextualização a respeito de algumas das problemáticas a serem resolvidas nesse gênero de jogos. Em seguida, há a seção correspondente aos diferentes subgêneros de jogos multijogador online. A apresentação das tecnologias de rede e dados e os estudos comparativos entre elas concluem o capítulo.

2.1 Jogos Multijogador Massivos Online

Jogos que localmente ou através da internet permitem que múltiplas pessoas possam jogar no mesmo ambiente ao mesmo tempo, são chamados jogos multijogador. Esse tipo de jogo é levado a um novo patamar quando uma quantidade massiva de jogadores é capaz de interagir com o jogo e entre si, através de avatares, em um ambiente gráfico 2D ou 3D, o que define MMOG (*Massive Multiplayer Online Game* - Jogo Multijogador Massivo Online) na literatura científica (CHAMBERS; FENG; FENG, 2017).

Na subseção seguinte serão apresentadas as principais problemáticas envolvendo jogos multijogador massivos online e jogos multijogador online em geral.

2.2 A Problemática dos Dados e do Número de Jogadores

Os MMOGs, embora extremamente populares e rentáveis, necessitam de uma contínua criação de conteúdo e uma custosa hospedagem para manter os jogadores ativos. Custos com hardware, software e manutenção da arquitetura para garantir a escalabilidade de hospedagem e o funcionamento adequado de funções como chat e mercado, são crescentes e entre os maiores para a administração de um MMOG.

CHAMBERS; FENG; FENG (2017) dizem que os casos genéricos de Jogos Massivos Online são baseados no controle de um avatar com habilidades disponíveis. Conforme o jogador completa tarefas nesse mundo, há a progressão de poder, possessões e habilidades do personagem. Essas tarefas variam de acordo com o gênero dos jogos, que serão discutidos na seção 2.3. O jogador recebe recompensas conforme completa seus objetivos, costumando começar o jogo com poucos recursos disponíveis e, após horas de *gameplay*, ter seus recursos multiplicados.

Esse conteúdo persistente (habilidades, possessões e níveis), denominado *loot*, é de extrema importância para manter o engajamento dos jogadores, que valorizam as horas investidas no jogo e sempre buscam novas recompensas. Faz-se necessário então uma

infraestrutura de rede segura e estável para armazenar esse tipo de informação proveniente de um grande número de jogadores.

Há geração de dados do momento que o jogador inicializa o jogo até o encerramento da partida. Uma vez armazenados, esses dados podem ser utilizados com diferentes fins. ODIERNA; SILVEIRA (2018) desenvolveram um estudo utilizando mineração de dados, tratando dados tais como avatar ID, guilda, nível, raça e zona do jogo onde o jogador se encontra, para classificar jogadores de acordo com a Taxonomia de Bartle (Matadores, Exploradores, Conquistadores e Socializadores). Outro estudo envolvendo mineração de dados buscou detectar jogadores mais propensos a gastar dinheiro em jogos de interpretação de personagens online e em massa (MOREIRA et al., 2017).

Só haverá investimento de tempo e dinheiro dos usuários, a longo prazo, se eles tiverem a confiança de que estão investindo em uma plataforma segura e confiável. Portanto, é necessário, ademais, um processo de autenticação para a adequada associação entre *loot* e *usuário*. Nas seções a respeito das *tecnologias de rede* serão discutidas algumas das tecnologias disponíveis no mercado para customização de servidores e bancos de dados.

2.3 Gêneros de Jogos Multijogador Online

A seguir serão apresentados diferentes subgêneros dos jogos multijogador online.

2.3.1 Jogos de Interpretação de Personagens Online e em Massa

Uma categoria mais específica dos MMOGs é a dos MMORPGs (*Massively Multiplayer Online Role-Playing Games* - Jogos de Interpretação de Personagens Online e em Massa). O gênero é caracterizado por uma boa narrativa, rica construção de personagens e pela imersão do jogador em um mundo fantasioso, com regras próprias predeterminadas (LIMA, 2018).

World of Warcraft, MMORPG lançado em 2004 pela *Blizzard Entertainment*, continua extremamente popular em 2020 com mais de 70 milhões de jogadores ativos (MMO POPULATIONS, 2020). Jogos do mesmo gênero, mas com uma base de jogadores bem reduzida, também possuem seu espaço no mercado, como é o caso do jogo *Grand Fantasia*, mostrado na Figura 2.1, lançado em 2009 pela *Aeria Games*.

2.3.2 Jogos Multijogador Massivos Online de Estratégia em Tempo Real

Outro subgênero dos MMOGs é o MMORTS (*Massively Multiplayer Online Real-Time Strategy* - Multijogador Massivo Online de Estratégia em Tempo Real). Os Jogos de Estratégia em Tempo Real são caracterizados pelas simulações onde diversos jogadores disputam e gerenciam espaços e recursos disponíveis em uma área simulada. Eles precisam

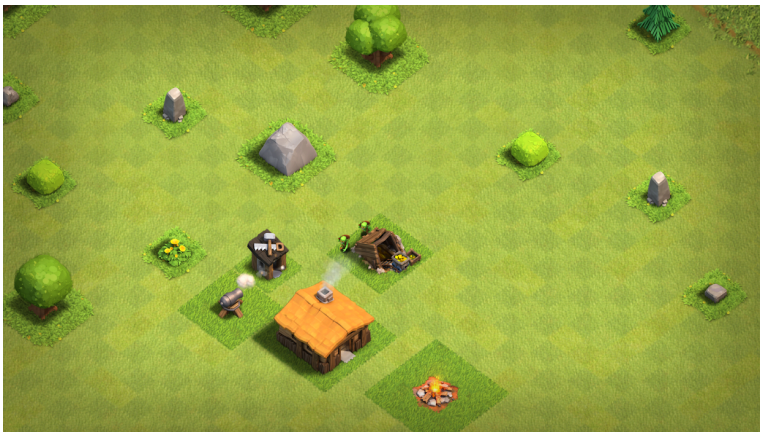
Figura 2.1 – Cenário do MMORPG *Grand Fantasia*.

Fonte: Própria

utilizar diferentes estratégias para construir exércitos, orientá-los em batalha e desenvolver uma economia, visando derrotar oponentes em combates em tempo real (SANTOS, 2017).

Clash of Clans é um exemplo de MMORTS. Lançado em 2012 pela desenvolvedora finlandesa *Supercell*, está disponível para iOS e Android. O cenário inicial do jogo pode ser visualizado na Figura 2.2.

Figura 2.2 – Cenário inicial de *Clash of Clans*.



Fonte: Própria

2.3.3 Jogos Online de Tiro em Primeira Pessoa

A categoria Online FPS (*Online First-Person Shooter* - Tiro em Primeira Pessoa Online) também precisa suportar um grande número de jogadores simultâneos, contudo, divididos em diversas pequenas seções com poucos jogadores (BARRI; GINÉ; ROIG,

2010). Nesse tipo de jogo, o jogador normalmente controla um personagem e tem visão em primeira pessoa, como se estivesse vendo o cenário através dos olhos do personagem. Frequentemente, esse tipo de jogo envolve armas e monstros fictícios inseridos em um universo 3D (JUNIOR; OLIVEIRA; ALECRIM, 2013).

Overwatch é um FPS Online lançado em 2016, desenvolvido e publicado pela *Blizzard Entertainment*. A visão em primeira pessoa, que o jogador tem ao jogar *Overwatch* no modo de jogo *Heróis Misteriosos*, é mostrada na Figura 2.3.

Figura 2.3 – Heróis Misteriosos: modo de jogo do MMOFPS *Overwatch*.

Fonte: Própria

2.3.4 Jogos de Batalha Real

Jogos de Batalha Real (*Battle Royale* em inglês) começaram a ganhar popularidade em 2017. A classificação é originária do filme japonês homônimo, lançado em 2000. O gênero mistura elementos de exploração, sobrevivência e procura por equipamentos e armas, característicos do filme. O objetivo das partidas geralmente é ser o último sobrevivente e eliminar oponentes em um mapa que encolhe com o passar do tempo (CHOI; KIM, 2018).

Os Jogos de Batalha Real obtiveram uma parcela considerável da receita de 120.1 bilhões de dólares proveniente da indústria de jogos em 2019. *Fortnite*, um jogo do gênero, obteve uma receita de aproximadamente 1.8 bilhão de dólares no mesmo ano (MMORPG, 2020). No principal modo de jogo do *Fortnite*, o objetivo é ser o último sobrevivente em uma ilha com 100 outros jogadores. O começo de uma partida pode ser visto na Figura 2.4.

2.3.5 Jogos de Arena de Batalha Multijogador Online

Os jogos MOBA (*Multiplayer Online Battle Arena* - Arena de Batalha Multijogador Online) precisam também lidar com uma grande quantidade de jogadores e de dados. São

Figura 2.4 – Começo de uma partida de *Fortnite*

Fonte: Própria

caracterizados por partidas nas quais equipes de jogadores competem entre si em um mapa, visando alcançar um objetivo, que costuma ser a destruição da estrutura defensiva da equipe oposta (POLANCEC; MEKTEROVIC, 2017).

League of Legends é um exemplo de MOBA bem-sucedido. Desenvolvido e publicado pela *Riot Games* em 2009, o jogo teve uma receita de aproximadamente 1.5 bilhão de dólares no ano de 2019 (MILLENIUM, 2020). Pode ser vista na Figura 2.5, parte do cenário de uma partida de *League of Legends* no modo de jogo chamado ARAM (*All Random All Mid - Todos Aleatórios, Todos no Meio*).

Figura 2.5 – Partida de ARAM: um dos modos de jogo de *League of Legends*

Fonte: Própria

2.4 Motor de Jogos Unity

O Unity é um motor de jogos desenvolvido em 2005 pela empresa *Unity Technologies*. A ferramenta pode ser utilizada para desenvolver jogos, 2D ou 3D, para diferentes plataformas, como computadores, consoles, dispositivos móveis e navegadores de internet (JETSONEN, 2016).

O motor possui versão gratuita e é um dos três mais populares disponíveis, os outros dois sendo a *Unreal Engine* e a *CryEngine*. Além disso, a comunidade de desenvolvedores Unity é bem ativa, existindo uma grande quantidade de conteúdo e recursos para aprendizagem disponíveis online (POLANCEC; MEKTEROVIC, 2017).

2.5 Soluções de Rede para Jogos Massivos Online no Unity

Conforme citado na apresentação deste trabalho, há uma demanda por soluções de rede nos jogos multijogador online. Na presente subseção, serão apresentadas diferentes possibilidades para a resolução da problemática, seguidas por uma comparação entre as possíveis soluções e a justificativa da escolha da tecnologia adequada.

2.5.1 Unity Networking

Unity Networking (UNET) é um conjunto de ferramentas e serviços designados para jogos multijogador no Unity. Tudo já vem integrado com a *engine* e serve tanto para desenvolvedores que pretendem criar um jogo multijogador simples, quanto para desenvolvedores que precisam de uma infraestrutura de rede complexa.

O UNET contém uma HLAPI (*High-Level Scripting Application Programming Interface* - Interface de Programação de Aplicações de Script de Alto Nível) que cobre a maior parte dos requisitos mínimos para jogos multijogador, assim como uma API de script de baixo nível, denominada Transport Layer, que permite ao desenvolvedor criar seus próprios sistemas de rede com características mais específicas ou avançadas (UNITY, 2020b).

Contudo, o serviço está descontinuado e sendo substituindo por um novo sistema mais completo e transparente, que proporcione segurança, estabilidade e performance. De toda forma, o Unity continuará dando suporte para alguns recursos do UNET até o final de 2021 (UNITY, 2020a).

2.5.2 Photon Server

Photon Server é um SDK multiplataforma auto-hospedado para backends customisáveis. Propondo alta escalabilidade, multiplataforma, além de uma licença gratuita

para até 100 CCU (Concurrent Users - Usuários Simultâneos) e licenças pagas para mais usuários (PHOTON, 2020e).

Há ainda outras licenças, como por exemplo, licenças para empresas ou licenças de compra única para 60 meses. Os detalhes podem ser encontrados na página oficial de preços da ferramenta.

Os usuários podem se beneficiar dos mais de 15 anos de desenvolvimento desta ferramenta, podendo criar uma lógica customizada a partir das aplicações de demonstração inclusas no SDK, ou a partir do zero (PHOTON, 2020e).

Albion Online (2017), MMORPG desenvolvido pela *Sandbox Interactive*, é um exemplo de jogo que utiliza o Photon Server como solução de rede.

2.5.3 Photon Cloud

Muitos dos serviços Photon utilizam a Photon Cloud, que é uma solução SaaS (*Software as a Service* - Software como Serviço) para servidores de jogos multijogador, permitindo que os desenvolvedores possam focar em suas aplicações no lado cliente, enquanto as operações de servidores ficam na responsabilidade da *Exit Games*, empresa desenvolvedora do Photon. Há diversas licenças, sendo a opção para até 20 CCU gratuita, e as opções para 500 CCU ou mais pagas (PHOTON, 2020b).

2.5.4 Photon Realtime

O SDK Photon Realtime é a API básica para acessar todos os serviços em nuvem do Photon, conforme mostrado na Figura 2.6. É independente do Unity e funciona como uma base para SDKs de mais alto nível, como PUN, BOLT e QUANTUM que serão apresentados ainda nesta subseção.

O Photon Realtime e as aplicações para as quais ele funciona como base possuem alta escalabilidade e flexibilidade, permitindo a criação de jogos multijogador multiplataforma a um preço acessível (PHOTON, 2020d). O SDK é, ademais, compatível com diversos motores de jogos, frameworks, plataformas e linguagens de programação.

2.5.5 Photon Unity Networking

Photon Unity Networking (PUN) é um pacote para o motor de jogos Unity designado para jogos multijogador. É recomendado para jogos baseados em salas, como FPS, MOBA, jogos de corrida, entre outros. O desenvolvedor pode escolher utilizar Photon Cloud ou servidores auto-hospedados, enquanto utiliza PUN no lado-cliente.

O pacote, denominado PUN 2, a versão mais atual, pode ser facilmente importado para o projeto do desenvolvedor através da Asset Store do Unity, que é uma loja virtual da

Figura 2.6 – O *Photon Realtime* é a API para acessar os serviços da *Photon Cloud*

Fonte: (PHOTON, 2020d)

Unity Technologies, utilizada para distribuir diversos conteúdos para jogos, denominados *assets*. PUN 2 contém também o SDK Photon Realtime Unity e é intermediário na conexão entre os clientes e os servidores.

99Vidas (2016), desenvolvido pela *QUByte Interactive*, é um exemplo de jogo multiplataforma online que utiliza o Photon Unity Networking.

2.5.6 Bolt

Bolt é outro pacote para Unity, sendo bastante popular na comunidade devido à abstração dos complexos recursos de rede por trás de uma simples e acessível interface. Ideal para jogos FPS, TPS, *Battle Royale* e jogos de ação em geral.

Está disponível em dois formatos. O Bolt Pro tem como recursos a conexão direta entre os jogadores através do IP, sem precisar de servidores, além de ser possível o uso de servidores dedicados (PHOTON, 2020a). O Bolt Free, disponível na Asset Store do Unity, requere que as conexões entre os usuários sejam mediadas pelo serviço em nuvem do Photon.

O Photon Realtime está contido dentro do pacote Bolt, e é utilizado para fazer a conexão com a Photon Cloud.

2.5.7 Quantum

O Quantum é um framework determinístico de alta performance designado para o desenvolvimento de jogos multijogador online de diversos gêneros, como MOBA, RTS,

ARPG (*Action Role Playing Game* - RPG Eletrônico de Ação), Esporte, Luta, FPS, Jogos por Turno e *Battle Royale*. Os sistemas internos do Quantum foram todos desenvolvidos visando velocidade, baixa largura de banda, proteção contra trapaças e ausência de Netcode (PHOTON, 2020c).

Jogos determinísticos são aqueles nos quais os clientes trocam apenas informações de entrada, com simulações rodando localmente em todos os clientes. Jogos do gênero RTS, por exempo, costumavam utilizar *determinismo com lockstep*, em que cada cliente esperava as informações de entrada de cada jogador antes de atualizar os frames da simulação (PHOTON, 2020c).

A *Exit Games* propõe fornecer *determinismo sem lockstep* por meio do Quantum, de forma que os clientes não precisam esperar para fazer simulações localmente, utilizando previsão de entrada em conjunto com um avançado sistema de rollback para restaurar o estado do jogo e re-simular qualquer previsão errônea.

Considerado pela própria empresa como uma tecnologia de ponta, o Quantum é o SDK mais moderno da *Exit Games*. As licenças disponíveis são mais caras do que as das outras tecnologias Photon. Os desenvolvedores que compram licenças recebem acesso às demonstrações de jogos com código fonte, tutoriais em vídeo e suporte exclusivo, através de um canal no Discord (PHOTON, 2020c).

2.5.8 Atavism Online

Atavism Online é uma plataforma para desenvolvimento de MMORPGs compatível com Unity. Possui diversos recursos básicos para jogos online em geral, como plugin para banco de dados, gerenciamento de contas, software de servidor, entre outros. Disponibiliza também recursos mais específicos para MMORPGs, como um sistema de quests e de raças. Há diversas licenças mensais, para diferentes números de CCU, que vão desde 100 CCU até 10000 CCU (ATAVISM, 2020). Existe ainda a opção de licença única.

O pacote está disponível para a compra na Asset Store do Unity e é bem avaliado na loja, mesmo não sendo simples de utilizar. Contudo, ainda não existem jogos de grandes desenvolvedoras que utilizam a solução.

2.5.9 Comparativo entre as Soluções de Rede

Tendo em vista que faz-se necessário escolher uma solução dentre as listadas na subseção anterior, foi realizado um comparativo visando selecionar as tecnologias adequadas e viáveis para a utilização neste trabalho. As seguintes características foram levadas em conta:

- **Confiabilidade:** Se jogos massivos online de grandes proporções, desenvolvidos

por empresas estabelecidas no mercado, já utilizaram a solução;

- **Fácil Integração:** Se a integração da solução com a *game engine* Unity é simples ou complexa;
- **Cliente/Servidor:** Se a tecnologia é utilizada para implantar o lado-cliente ou o lado-servidor do jogo;
- **Versão Gratuita Acessível:** Se existe uma versão gratuita disponível para desenvolvedores/estudantes;
- **Liberdade Lado-servidor:** Se é possível customizar a lógica no lado-servidor do jogo desenvolvido utilizando a tecnologia;
- **Suporte:** Se a empresa responsável está disponível para dar o suporte adequado aos usuários da tecnologia.

Para ter maior liberdade de customização no lado-servidor, foi visada a escolha de uma tecnologia para servidores auto-hospedados, em detrimento de tecnologias que utilizam servidores em nuvem. A partir do estudo comparativo visto na Tabela 2.1, o Photon Server foi escolhido como solução para o lado-servidor. Ele possui a opção de licença gratuita para até 100 CCU, é produto de uma empresa que tem histórico de confiabilidade, é fácilmente integrável e é uma tecnologia na qual os utilitários recebem suporte adequado.

Tabela 2.1 – Comparativo entre diferentes soluções de rede disponíveis para Unity.

Soluções	Confiabilidade	Fácil Integração	Cliente/Servidor	Versão Gratuita Acessível	Liberdade Lado-servidor	Suporte
UNET	Não	Sim	Ambos	Sim	Sim	Sim (mas está descontinuado)
Photon Server	Sim	Sim	Servidor	Sim	Sim	Sim
Photon Cloud	Sim	Sim	Servidor	Sim	Não	Sim
Realtime	Sim	Sim	Cliente	Sim	Cliente	Sim
PUN	Sim	Sim	Cliente	Sim	Cliente	Sim
Bolt	Sim	Sim	Cliente	Sim	Cliente	Sim
Quantum	Sim	Sim	Cliente	Não	Cliente	Sim
Atavism	Não	Não	Servidor	Não	Sim	Sim

De maneira análoga, o Photon Unity Networking foi escolhido como solução para o lado-cliente: sendo facilmente importável para o projeto, possuindo uma versão gratuita, e apresentando a confiabilidade dos produtos Photon.

No capítulo 3 será apresentado um estudo mais detalhado quanto ao PUN e ao Photon Server.

2.6 Soluções de Dados para Jogos MMO

Existem diversas soluções que permitem tratar a problemática dos dados nos jogos multijogador online. Nesta subseção serão apresentadas algumas opções e a justificativa da escolha de uma solução viável e de acordo com as demandas desse gênero de jogos.

2.6.1 Azure PlayFab

O PlayFab é uma plataforma completa para serviços backend em jogos multijogador online. A plataforma permite escalabilidade instantânea dos servidores, sistemas para autenticação, criação de placares de líderes, utilização de serviços de comércio para genrenciar lojas virtuais, rastrear moedas virtuais e processar pagamentos, entre outras diversas funcionalidades (PLAYFAB, 2020). Apresentando grande versatilidade, o PlayFab é compatível com diversos consoles, plataformas e motores de jogos, além de fornecer ferramentas de monitoramento de dados em tempo real.

A plataforma utiliza a Azure Cloud da *Microsoft*. A licença gratuita permite até 10 títulos em desenvolvimento e até 100.000 contas de jogadores por título. Para contas de usuário ilimitadas é necessária uma licença paga (PLAYFAB, 2020).

O Azure PlayFab é confiável, tendo sido utilizado por grandes empresas em jogos como *Rainbow Six Siege* (2015) da *Ubisoft* e *Sea of Thieves* (2017) da *Rare*. O PlayFab permite também a integração com o Photon, permitindo a autenticação de jogadores.

2.6.2 Chilli Connect

De maneira similar ao Azure PlayFab, o Chilli Connect é uma plataforma de serviços backend para jogos. Permite aos desenvolvedores, a adição de recursos sociais e escalabilidade para milhões de jogadores. Criação de contas para os usuários, gerenciamento da economia do jogo, dados em nuvem, leaderboards e integração com Photon, por exemplo, são recursos inclusos no SDK Chilli Connect (CHILLI CONNECT, 2020).

O SDK ainda conta com funções para gerenciamento de dados, como *dashboards*, dados diários disponíveis para exportação, rastreamento dos comportamentos dos usuários, e diversas outras funções (CHILLI CONNECT, 2020).

O Chilli Connect foi adquirido pela *Unity Technologies* e conta com diversas opções de licenças pagas. Existem licenças para desenvolvedores independentes, startups, estúdios e grandes empresas.

2.6.3 Game Sparks

Game Sparks é outro exemplo de solução para a problemática de dados em jogos multijogador. Com funcionalidades semelhantes às do Azure PlayFab e do Chilli Connect,

o Game Sparks é uma opção confiável para backend de jogos digitais online, sendo utilizada por grandes empresas como *Square Enix*, *Namco Bandai Entertainment*, *Ubisoft* e *Amazon Game Studios*.

A solução é compatível com diversas game engines como Unreal Engine, Construct 2 e o próprio Unity. É possível baixar o Unity SDK Package direto da Asset Store, ou importá-lo para projetos, com opção de licença gratuita para até 100 CCU e opções pagas para mais usuários (GAME SPARKS, 2020).

2.6.4 Multiplay

Multiplay é um serviço multijogador compatível com diversas *engines* e recomendado pela *Unity Technologies* como solução para hospedagem de servidores em nuvem. O serviço também propõe a remoção de complexidade de criação e operação do backend dos jogos multijogador, provendo ainda uma economia financeira de aproximadamente 30% em comparação com outras soluções em nuvem (MULTIPLAY, 2020).

São fornecidos alguns benefícios pelo serviço, como o auxílio de especialistas em jogos multijogador, e o fornecimento e análise de informações do servidor. Para utilizar o serviço, todavia, é necessário entrar em contato com a empresa, a qual foi adquirida pela *Unity Technologies* em novembro de 2017 (MULTIPLAY, 2020).

2.6.5 Comparativo entre as Soluções de Dados

Dentre as opções disponíveis apresentadas nesta subseção, foi realizado um estudo comparativo visando escolher a solução mais adequada e viável para a utilização no presente trabalho. A análise foi realizada a partir dos seguintes parâmetros:

- **Confiabilidade:** Se jogos massivos online de grandes proporções, desenvolvidos por empresas estabelecidas no mercado, já utilizaram a solução;
- **Fácil Integração:** Se a integração da solução com o motor de jogos Unity é simples ou complexa;
- **Versão Gratuita Acessível:** Se existe uma versão gratuita disponível para desenvolvedores/estudantes;
- **Recursos:** Se a solução provê uma grande diversidade de recursos backend para o jogo;
- **Suporte:** Se a empresa responsável está disponível para dar o suporte adequado aos usuários da tecnologia.

O estudo pode ser visto na Tabela 2.2. O fator decisivo da escolha do Azure PlayFab foi a ferramenta possuir a versão gratuita com mais recursos dentre as analisadas, que permite até 100.000 contas de usuários e 10 títulos em modo de desenvolvimento.

Tabela 2.2 – Comparativo entre diferentes soluções de dados para Unity.

Soluções	Confiabilidade	Fácil Integração	Versão Gratuita Acessível	Recursos	Suporte
PlayFab	Sim	Sim	Sim	Sim	Sim
Chilli Connect	Sim	Sim	Não	Sim	Sim
Game Sparks	Sim	Sim	Sim	Sim	Sim
Multiplay	Sim	Sim	Não	Sim	Sim

A plataforma, ademais, é confiável e pode ser utilizada em conjunto com o Photon Server e o PUN, tecnologias que serão utilizadas como soluções de rede para o presente trabalho. O PlayFab possui ainda um SDK que pode ser importado de maneira prática e simples para o projeto no Unity.

No próximo capítulo será apresentado um estudo mais detalhado quanto ao Azure PlayFab.

3 Tecnologias Utilizadas

O presente capítulo é composto primeiramente pela proposta de projeto a ser desenvolvida por meio das tecnologias escolhidas a partir do estudo comparativo do Capítulo 2. Em sequência, são apresentadas com mais detalhes as tecnologias que serão utilizadas: o Photon Server, o Photon Unity Networking e o Azure PlayFab.

3.1 Proposta de Projeto

A ideia geral do projeto é desenvolver um protótipo pronto para testes e que possua algumas das propriedades primordiais dos jogos multijogador massivos online. É esperado que seja possível implantar funcionalidades de *distribuição de usuários dentro do jogo*, *cadastro e autenticação*, e *armazenamento dos dados correspondentes a cada jogador*.

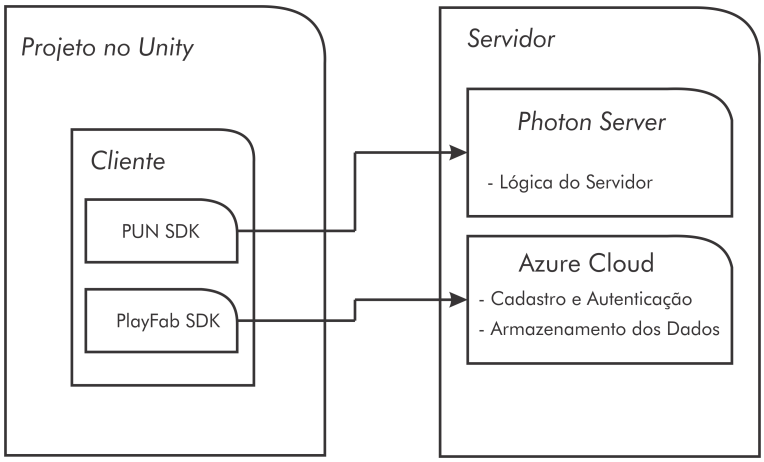
O projeto será desenvolvido no motor de jogos Unity, com os SDKs PUN e PlayFab. Através do PUN será possível implementar a lógica do acesso a um servidor Photon Server, o qual será o responsável pelo balanceamento das cargas de usuários, conforme será explanado mais detalhadamente ao longo deste capítulo. Com o PlayFab, será programada a lógica de cadastro e autenticação de usuários e as requisições de envio e recebimento de dados da Azure Cloud.

Os usuários que jogarem o protótipo navegarão por um sistema de menus, em que poderão realizar diversas ações, por exemplo:

- Realizar cadastro na plataforma PlayFab;
- Realizar autenticação;
- Criar Salas de Jogo (*Game Rooms)*;
- Visualizar as *Game Rooms* disponíveis;
- Entrar nas *Game Rooms* abertas;
- Enviar e receber dados para outros jogadores através da Azure Cloud;
- Começar uma partida multijogador;

Dentro de uma partida, será possível que os usuários interajam entre si em um ambiente virtual 3D. Haverão algumas mecânicas de gameplay que serão mostradas ao longo do trabalho. Um diagrama representando a *proposta de projeto* pode ser visto na Figura 3.1.

Figura 3.1 – Proposta de Projeto



Fonte: Própria

3.2 Photon Server

O Photon Server é um servidor local em tempo real e um framework para desenvolvimento de jogos multijogador multiplataforma online. A ferramenta possui diferentes aplicações, as quais podem ser usadas de maneira nativa ou customizável. Na Figura 3.2 está apresentada a arquitetura do Photon Server.

O Photon Core (núcleo da ferramenta) é escrito em C++, visando o aprimoramento da performance. O núcleo utiliza IOCP (Input/Output completion port - Porta de conclusão de entrada/saída) para gerenciamento de soquetes com alta performance, além de suportar UDP, TCP, HTTP e Web Sockets.

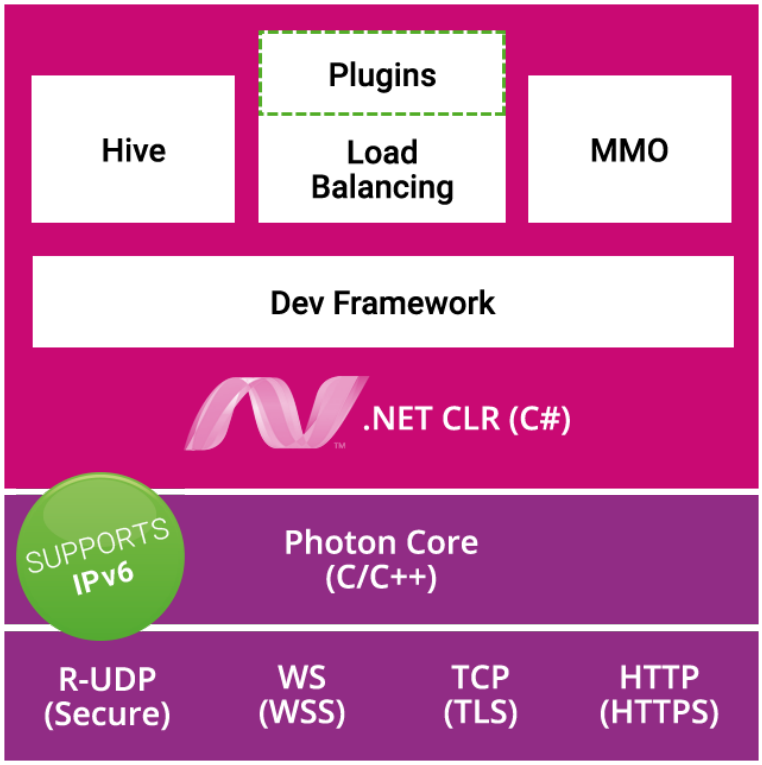
A lógica de negócios é executada em uma camada superior, na máquina virtual .NET CLR, e é escrita em C# ou alguma outra linguagem .NET. Nessa camada estão contidas as aplicações, que serão explicadas mais detalhadamente na subseção seguinte.

3.2.1 Aplicações Base

Algumas aplicações que servem como base para o desenvolvimento estão inclusas no SDK do Photon Server. As três aplicações são:

- **LoadBalancing:** Consiste em duas aplicações: *Master Server* e *Game Server*, as quais serão explicadas em detalhes ao longo do capítulo. Uma aplicação semelhante é utilizada nos servidores em nuvem do Photon (Photon Cloud);
- **MMO:** Esta aplicação foi descontinuada e não recebe mais suporte. É uma solução para jogos nos quais todos os jogadores compartilham um mundo virtual grande;

Figura 3.2 – Arquitetura do Photon Server



Fonte: (PHOTON, 2020d)

- **Lite:** É uma aplicação simplificada. Antes da Versão 4 do Photon Server, era utilizada para jogos multijogador simples, que não requeriam lógica de servidor. Muitas vezes, também era utilizada como base para projetos mais complexos. Porém, atualmente, é recomendado pela empresa que a aplicação seja utilizada apenas como uma introdução aos conceitos básicos do Photon, e não mais em projetos.

Load Balancing será a aplicação utilizada no presente trabalho, uma vez que ela não foi descontinuada e é compatível com as outras tecnologias que serão usadas.

3.2.2 LoadBalancing

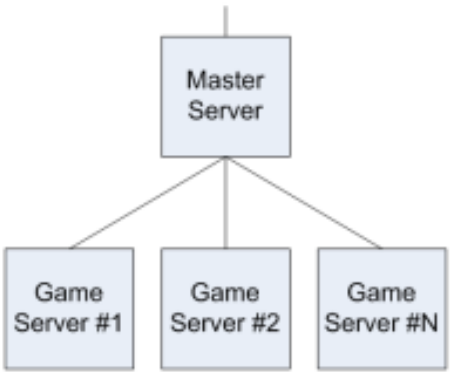
A configuração básica do LoadBalancing consiste em um Servidor Mestre (*Master Server*) responsável por vários Servidores de Jogo (*Game Servers*). O Master Server é reponsável por diversas tarefas:

- Controla os jogos que estão em andamento nos Game Servers;
- Controla o fluxo de trabalho dos Game Servers conectados, atribuindo os pares aos servidores adequados;

- Mantém e atualiza uma lista de salas para os clientes no *lobby* (local onde jogadores podem inspecionar as sessões de jogo disponíveis);
- Encontra salas e direciona os endereços dos Game Servers para os clientes.

Já os *Game Servers* gerenciam as salas de jogo e reportam regularmente suas respectivas cargas de trabalho para o *Master Server*. A configuração básica do LoadBalancing pode ser vista na Figura 3.3.

Figura 3.3 – Configuração básica do LoadBalancing



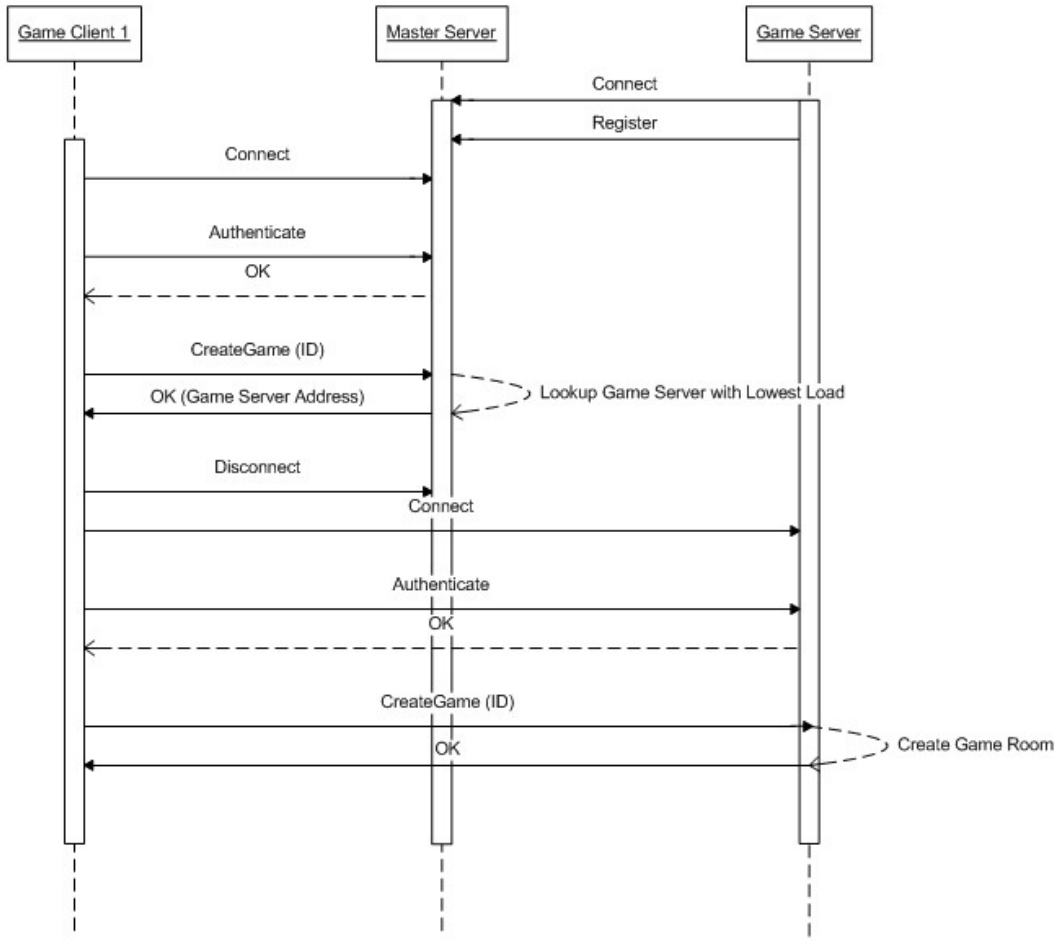
Fonte: (PHOTON, 2020d)

A dinâmica básica entre o Cliente do Jogo (*Game Client*), *Master Server* e o *Game Server* se dá da seguinte maneira:

1. *Game Clients* se conectam ao *Master Server*, de onde é possível se juntar ao lobby e ter acesso a uma lista de jogos abertos (visíveis);
2. Os clientes podem realizar operações de criação de jogo (*CreateGame*) no *Master Server*. Quando fazem isso, o jogo ainda não é criado. O *Master Server* identifica o *Game Server* com menos carga de trabalho e retorna o respectivo IP para o cliente;
3. Os clientes também podem realizar operações para se juntar a *Game Servers* específicos (*JoinGame*) ou aleatórios (*JoinRandomGame)* no *Master Server*. Neste caso, o *Master Server* busca pelo *Game Server* no qual o jogo está acontecendo e retorna seu IP para o cliente;
4. Em ambos os casos (*CreateGame* e *JoinGame*), o cliente é desconectado do *Master Server* e conectado ao *Game Server*, de onde eles chamam as mesmas operações, criando ou se juntado às Salas de Jogo.

É possível ver na Figura 3.4 um diagrama representando o fluxo de trabalho descrito acima.

Figura 3.4 – Fluxo de trabalho básico do LoadBalancing



Fonte: (PHOTON, 2020d)

3.3 Photon Unity Networking

O Photon Unity Networking (PUN) é uma biblioteca para desenvolvimento de jogos multijogador com Unity. Existem duas versões disponíveis: uma paga e a outra gratuita. A versão paga contém um plano para 100 CCU quando utilizando a Photon Cloud, porém, uma vez que o Photon Server será utilizado neste trabalho, a versão gratuita será suficiente.

O pacote é constituído por três camadas de API's:

- A camada de mais alto nível é a programação realizada no PUN. Nesta camada são implementadas funcionalidades específicas do Unity;
- A camada intermediária contém a lógica para trabalhar com os servidores Photon (*Photon Servers*);

- A camada mais baixa é composta por arquivos DLL.

Para o desenvolvimento do projeto serão realizadas implementações na camada de mais alto nível do PUN, em que são utilizados os chamados *mecanismos de sincronização*.

3.3.1 Mecanismos de Sincronização

Uma vez que o Photon Server apenas fornece os meios necessários para que os jogadores possam trocar dados entre si, o PUN fica responsável pela utilização de certos mecanismos de sincronização de estados de jogo, são eles:

- **Custom Properties (Propriedades Customizadas):** São valores sincronizados entre os clientes através de *Hashtables* (Tabelas de Dispersão) chave-valor. As *custom properties* podem ser atribuídas aos *players* (jogadores) e às *rooms* (salas);
- **Object Synchronization (Sincronização de Objetos):** Atribuindo um componente chamado *PhotonView* aos objetos em cena, é possível sincronizar diferentes informações através da rede. Este componente deve ser configurado para observar e transmitir um outro componente. O *PhotonView* pode ser configurado, por exemplo, para transmitir o *Transform* de um objeto, sincronizando assim posição, rotação e escala deste. Para esse tipo de sincronização, pode ser utilizado o protocolo TCP ou UDP. Geralmente o protocolo UDP é utilizado para garantir a responsividade do jogo;
- **RPC's (Remote Procedure Calls - Chamadas de Procedimento Remoto):** Através das RPC's é possível chamar métodos em outros clientes, bastando utilizar a assinatura *[PunRPC]* sobre o método. Um método *TakeDamage()* assinalado como *[PunRPC]* e responsável por reduzir a energia de um jogador, por exemplo, poderia ser chamado nos clientes de todos os jogadores simultaneamente. As RPC's sempre utilizam o protocolo TCP, de forma a garantir que os eventos sejam sincronizados corretamente.

3.3.2 *Photon Callbacks*

As *Photon Callbacks* são ferramentas muito úteis para a implementação com o PUN. A partir delas é possível desencadear ações chaves a partir de eventos disparados dentro do jogo.

Para poder utilizar as *Photon Callbacks* é necessário herdar da classe *MonoBehaviourPunCallbacks* e sobrescrever as respectivas *callbacks*. Alguns exemplos estão listados a seguir:

- *OnConnectedToMaster():* Chamada quando o cliente se conecta ao *Master Server*. Essa *callback* poderia ser sobrescrita de forma a fazer o jogador tentar criar uma partida depois de conectado ao servidor.
- *OnJoinedRoom():* Chamada quando o cliente entra na sala. Um exemplo de uso seria sobrescrevê-la de modo a atualizar a tela do jogador no momento que ele se junta a uma sala.
- *OnCreatedRoom():* Chamada quando um cliente cria uma sala e entra nela. *OnJoinedRoom()* é chamada também.

Existem diversas *Photon Callbacks*. Outros exemplos serão apresentados na prática, ao longo do trabalho.

3.4 Azure PlayFab

Neste trabalho, o Azure PlayFab será utilizado mais especificamente como banco de dados, armazenando dados de cadastro (e-mail, nome de usuário e senha) e dados correspondentes aos usuários (*status*, inventário, etc.).

3.4.1 Autenticação

O registro de usuários utilizando o PlayFab é feito através de requisições. Neste caso, os cadastros serão relizados através de e-mail, *password* (senha) e *username* (nome de usuário). Na Figura 3.5 é possível observar um exemplo de como é solicitado o cadastro de um novo usuário no PlayFab.

Figura 3.5 – Exemplo de requisição para registro de usuários

```
var registerRequest = new RegisterPlayFabUserRequest { Email = userEmail, Password = userPassword, Username = username };
PlayFabClientAPI.RegisterPlayFabUser(registerRequest, OnRegisterSuccess, OnRegisterFailure);
```

Fonte: Própria

Os parâmetros citados são passados no corpo da requisição e ela é enviada para os servidores do PlayFab, juntamente às duas *callbacks*, respectivamente para o sucesso e falha no cadastro de um novo usuário.

O *login* é realizado de forma semelhante, através do envio de requisição, embora existam diferentes formas de realizar a conexão utilizando o PlayFab, cada uma com uma requisição específica (*LoginWithSteam*, *LoginWithFacebook*, *LoginWithEmailAddress*, *LoginWithGoogleAccount*, etc.).

3.4.2 Dados de Usuário

É possível ademais, enviar e receber dados do PlayFab através de requisições. Sendo os dados enviados sob forma de *Dictionaries*, como pares chave/valor (*key/value pairs*). Para obter os dados armazenados no PlayFab, podem ser feitas requisições passando como parâmetro o ID do Jogador (*PlayerID*), que é um código único gerado pelo PlayFab para cada jogador. Na Figura 3.6 abaixo pode ser visto um exemplo de requisição para obter dados de um usuário.

Figura 3.6 – Exemplo de requisição para obter dados de um usuário a partir do *PlayerID*.

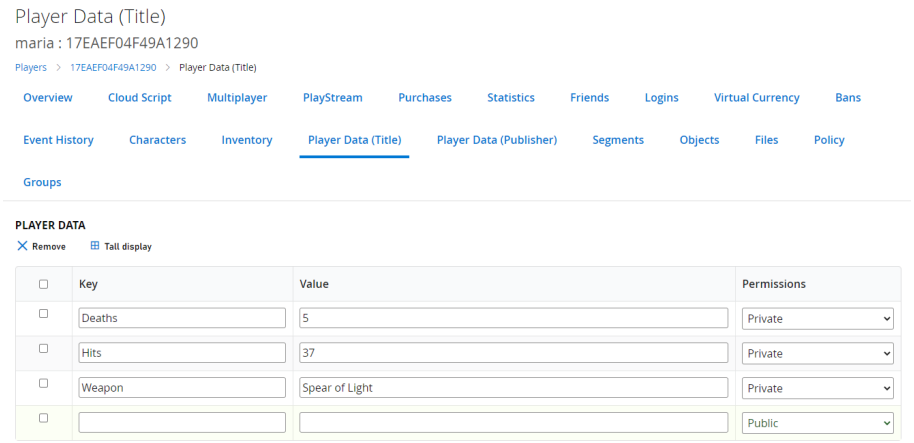
```
var request = new GetUserDataRequest() { PlayFabId = playerID, Keys = null };
PlayFabClientAPI.GetUserData(request, GetUserDataSuccess, GetUserDataFailure);
```

Fonte: Própria

Esses dados ficam disponíveis para os desenvolvedores e podem ser visualizados e alterados por eles a partir do *Game Manager*: o *gerenciador online do PlayFab*, o qual pode ser acessado pelo navegador. Na Figura 3.7 abaixo é mostrado um exemplo da visualização dos dados armazenados.

No exemplo, o nome do usuário é *maria* e seu *PlayerID* é: *17EAF04F49A1290*. Três informações estão armazenadas sob a forma de pares chave/valor: *Deaths* (mortes) associada ao valor *5*, *Hits* (acertos) associada ao valor *37* e *Weapon* (arma) associada ao valor *Spear of Light*.

Figura 3.7 – Exemplo de visualização dos dados armazenados no gerenciador online do PlayFab.



Fonte: Própria

4 Protótipo no Unity

Este capítulo é composto por uma descrição do protótipo desenvolvido. Primeiro é apresentado um panorama, e em seguida, cada parte do protótipo é descrita em detalhes.

4.1 Panorama

O protótipo consiste em duas cenas: uma correspondente ao sistema de menus e outra a uma cena jogável. Ambas foram desenvolvidas a partir das tecnologias descritas no capítulo 3, juntamente ao motor de jogos Unity. No sistema de menus é possível realizar todas as ações enumeradas na seção 3.1. Na cena jogável, os usuários podem jogar uma partida multijogador, onde é possível trocar informações com outros jogadores, os quais compartilham o mesmo mundo virtual. É possível também realizar uma *quest*; o jogo terá um estado para os usuários que a completarem e outro para os que estiverem com a *quest* em andamento.

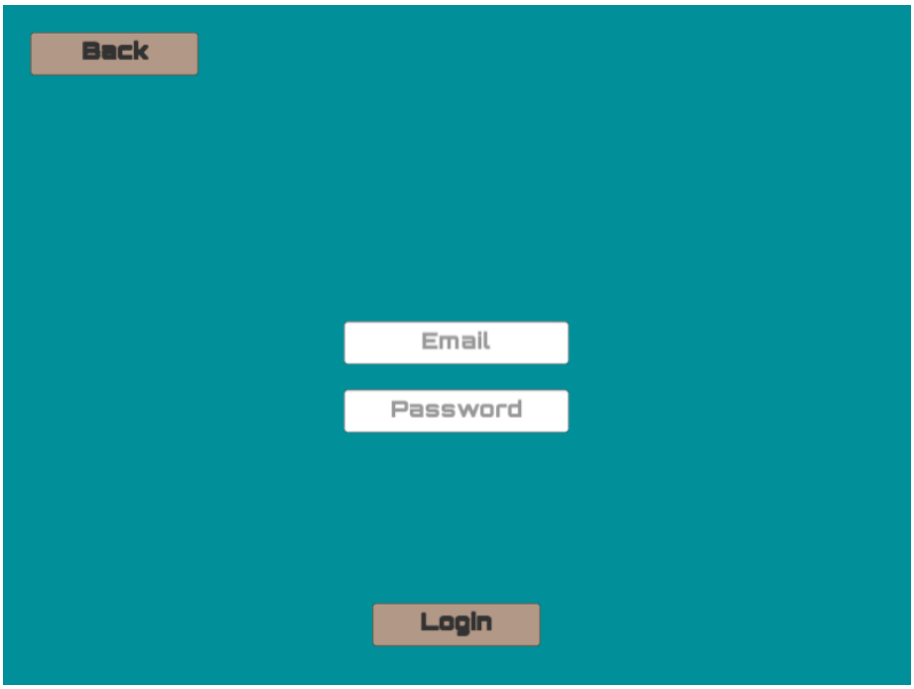
4.2 Autenticação e Cadastro

Ao executar o protótipo, o usuário será apresentado a uma tela onde ele poderá escolher entre fazer *login* (autenticação) ou *sign up* (cadastro). Em caso de sucesso no cadastro, o usuário recebe uma notificação visual em texto. Em caso de sucesso na autenticação, existem duas possíveis telas que serão apresentadas ao jogador:

- Quando o jogador está realizando o primeiro *login*, ele é direcionado a um painel onde pode escolher uma *Weapon*. No caso deste protótipo, a *Weapon* é uma variável *string* a ser armazenada nos servidores do PlayFab, simbolizando um item de inventário;
- Caso não seja a primeira autenticação, o usuário é levado diretamente a um painel no qual pode realizar diferentes ações que serão apresentadas na seção 4.3.

Se a autenticação, o cadastro ou o envio dos dados falhar, há também uma notificação visual em texto para o usuário. Na Figura 4.1 é apresentado o painel de autenticação do protótipo implementado.

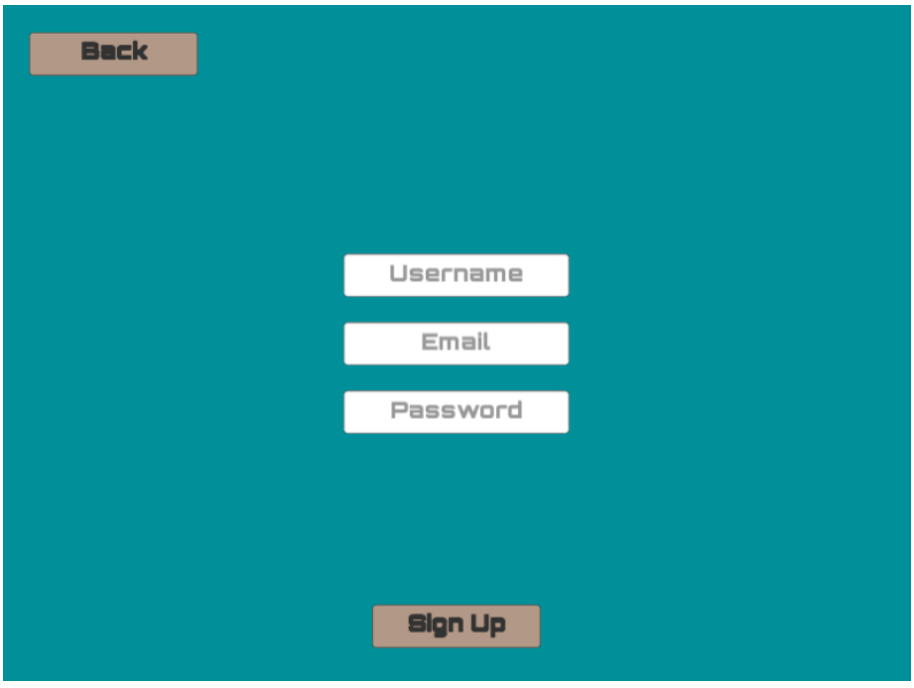
Figura 4.1 – Painel para *login* de usuário.



Fonte: Própria

É possível visualizar na Figura 4.2 o painel para cadastro de novo usuário.

Figura 4.2 – Painel para cadastro de novo usuário.



Fonte: Própria

4.2.1 PlayFab (Login, Sign Up, Dados)

Conforme detalhado na subseção 3.4.2, o acesso aos servidores do PlayFab ocorre por meio de requisições. No protótipo, a requisição para autenticação é composta por e-mail e senha (*password*). A requisição para cadastro é composta por e-mail, senha e nome de usuário. Ambas as requisições podem ser vistas respectivamente nas figuras 4.3 e 4.4.

Figura 4.3 – Código para autenticação de usuário.

```
var request = new LoginWithEmailAddressRequest { Email = userEmail, Password = userPassword };
PlayFabClientAPI.LoginWithEmailAddress(request, OnLoginSuccess, OnLoginFailure);
```

Fonte: Própria

Figura 4.4 – Código para cadastro de novo usuário.

```
var registerRequest = new RegisterPlayFabUserRequest { Email = userEmail, Password = userPassword, Username = username };
PlayFabClientAPI.RegisterPlayFabUser(registerRequest, OnRegisterSuccess, OnRegisterFailure);
```

Fonte: Própria

Os dados são enviados e recebidos de maneira semelhante. Para envio, é criada uma requisição composta pelos dados a serem enviados. Os dados enviados são armazenados na forma de *Dictionaries*. Para obter os dados armazenados no PlayFab, utiliza-se o ID do jogador alvo. Na Figura 4.5 há um exemplo de declaração e envio de dados para o PlayFab. Na Figura 4.6 pode ser visto como são obtidos os dados de um usuário.

Figura 4.5 – Código para envio de dados a serem armazenados no *PlayFab*.

```
Dictionary<string, string> data = new Dictionary<string, string>()
{
    {"Weapon", weaponData },
    {"MissionComplete", "NO" },
    {"Deaths", 0.ToString() }
};

var request = new UpdateUserDataRequest() { Data = data };
PlayFabClientAPI.UpdateUserData(request, SetUserDataSuccess, SetUserDataFailure);
```

Fonte: Própria

Figura 4.6 – Código para solicitação de dados de determinado usuário.

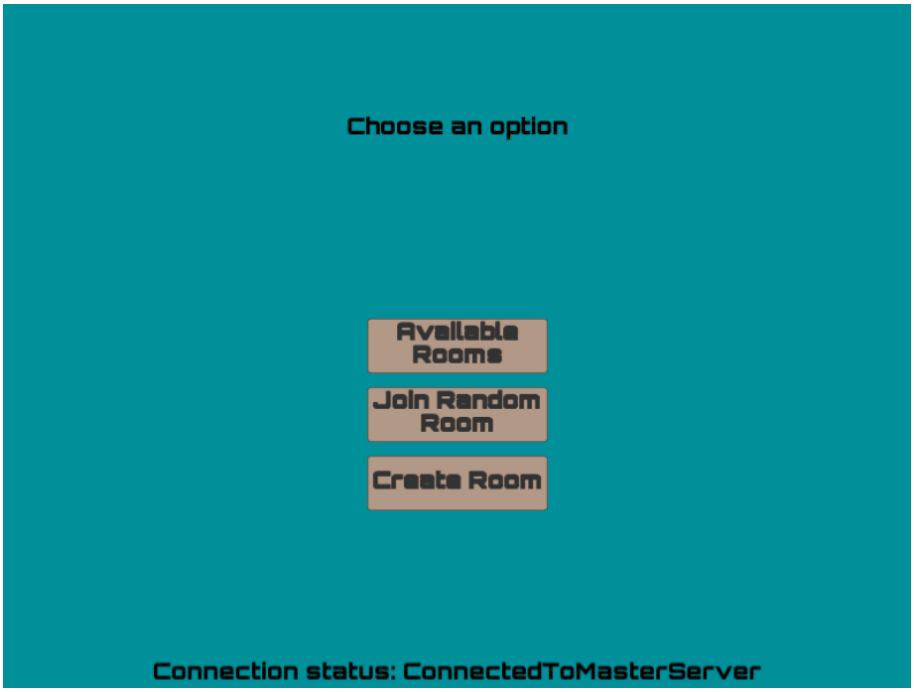
```
var request = new GetUserDataRequest() { PlayFabId = playerID, Keys = null };
PlayFabClientAPI.GetUserData(request, GetUserDataSuccess, GetUserDataFailure);
```

Fonte: Própria

4.2.2 Acesso ao Servidor Photon Server

Uma vez que o usuário clica em *Login*, ele se conecta também ao servidor Photon Server, mais especificamente à aplicação *LoadBalancing*, explicada com detalhes na subseção 3.2.2. A conexão se dá pelo método *PhotonNetwork.ConnectUsingSettings()*. O estado de conexão é apresentado na parte inferior do painel, conforme na Figura 4.7. O estado esperado, em caso de sucesso na conexão à aplicação *LoadBalancing*, é *ConnectedToMasterServer*.

Figura 4.7 – Tela apresentada ao usuário em caso de sucesso na conexão. Na parte inferior, *Connection status: ConnectedToMasterServer*, indica sucesso na conexão.



Fonte: Própria

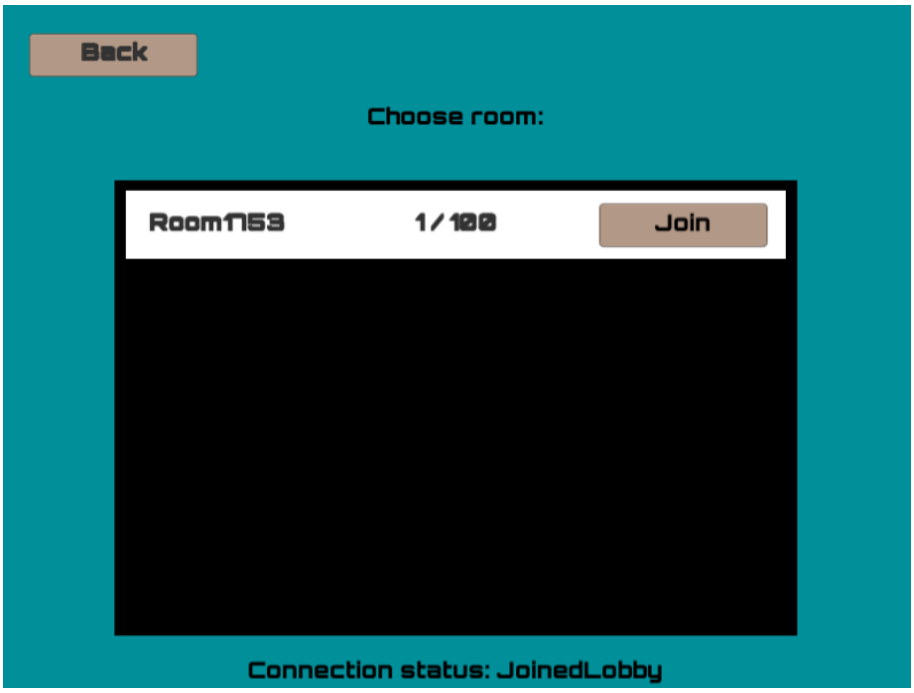
4.3 Salas de Jogo

Uma vez que o usuário é autenticado, o mesmo é apresentado a três opções disponíveis:

- *Available Rooms* (Salas Disponíveis): Mostra uma lista com as salas disponíveis;
- *Join Random Room* (Entrar em Sala Aleatória): Entra em uma sala disponível aleatoriamente;
- *Create Room* (Criar Sala): Cria uma nova sala e entra na mesma automaticamente.

A lista das salas disponíveis apresenta o nome de cada sala, a lotação e um botão no qual o usuário pode clicar para entrar na sala específica. Um exemplo pode ser visto na Figura 4.8.

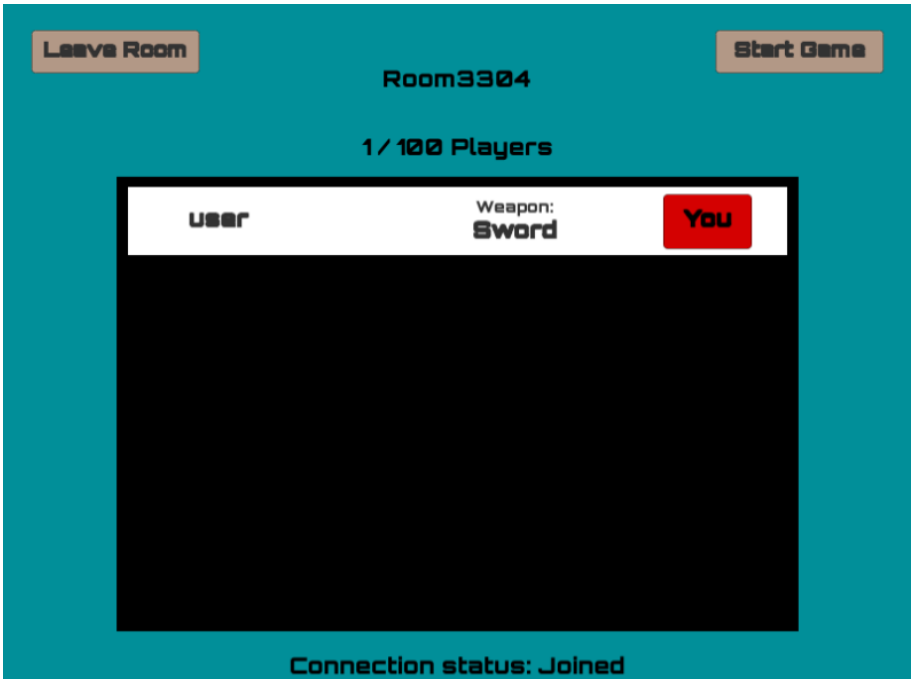
Figura 4.8 – Lista das salas disponíveis. Na parte inferior da interface é mostrado o estado da conexão: *Joined Lobby* (Entrou no Lobby).



Fonte: Própria

Dentro da sala há uma lista com os usuários e seus respectivos *itens de inventário*. Um sinalizador, com o texto *YOU*, identifica o próprio usuário. Há um botão para começar o jogo (*Start Game*), o qual só fica disponível para o criador da sala. A Figura 4.9 ilustra a interface interna da Sala de Jogo.

Figura 4.9 – Lista de usuários que pode ser vista pelos jogadores de dentro da sala. O botão para começar o jogo se encontra na região superior direita. O estado de conexão *Joined* indica que o jogador entrou na sala com sucesso.



Fonte: Própria

4.3.1 Criando e Entrando em Game Rooms

A criação de salas utilizando C# e o pacote do PUN é feita com o método *PhotonNetwork.CreateRoom(randomRoomName, roomOptions)*. São passados como parâmetros o nome da sala (*randomRoomName*) e as opções da sala (*roomOptions*). Dentro de *roomOptions* são especificadas opções como visibilidade, número máximo de jogadores, se a sala está aberta ou não, entre outros parâmetros. O código para criar uma *Game Room* pode ser visualizado na Figura 4.10.

O jogador que cria uma sala, automaticamente entra nela. Ademais, existem duas outras formas de entrar nas salas:

1. *PhotonNetwork.JoinRandomRoom()*: O usuário entra em uma sala aberta e visível aleatoriamente;

2. *PhotonNetwork.JoinRoom(roomName)*: O usuário entra em uma sala específica a partir do nome da sala (*roomName*).

Figura 4.10 – Criação de uma sala com o método *PhotonNetwork.CreateRoom*, passando como parâmetros nome (*randomRoomName*) e opções (*randomRoomOptions*).

```
public void CreateAndJoinRoom()
{
    string randomRoomName = "Room" + Random.Range(0, 10000);

    RoomOptions roomOptions = new RoomOptions();
    roomOptions.IsOpen = true;
    roomOptions.IsVisible = true;
    roomOptions.MaxPlayers = 100;

    PhotonNetwork.CreateRoom(randomRoomName, roomOptions);
}
```

Fonte: Própria

4.3.2 Começando a Partida

Apenas o criador da sala (*Master Client*) pode começar uma partida. É importante ressaltar que a sincronização das cenas entre diferentes jogadores na mesma sala é feita através do parâmetro booleano *PhotonNetwork.AutomaticallySyncScene*, assinalado como *true* (verdadeiro). Quando o *Master Client* começa o jogo, ele e todos os jogadores da sala carregam a cena *ForestQuest*. Na Figura 4.11 pode ser visto o método para começar uma partida:

Figura 4.11 – Quando o *Master Client* clica em *Start Game* a cena *ForestQuest* é carregada.

```
public void OnClickStartGame()
{
    PhotonNetwork.LoadLevel("ForestQuest");
}
```

Fonte: Própria

4.4 Game Scene

Quando o usuário que criou a sala começa a partida, o cenário *ForestQuest* é carregado. Esse cenário, originalmente, consiste em um pequeno mapa no qual o jogador controla um personagem e precisa realizar uma missão. O cenário está disponível neste link. Neste trabalho, ele foi adaptado para contemplar funcionalidades multijogador online e será explicado em detalhes.

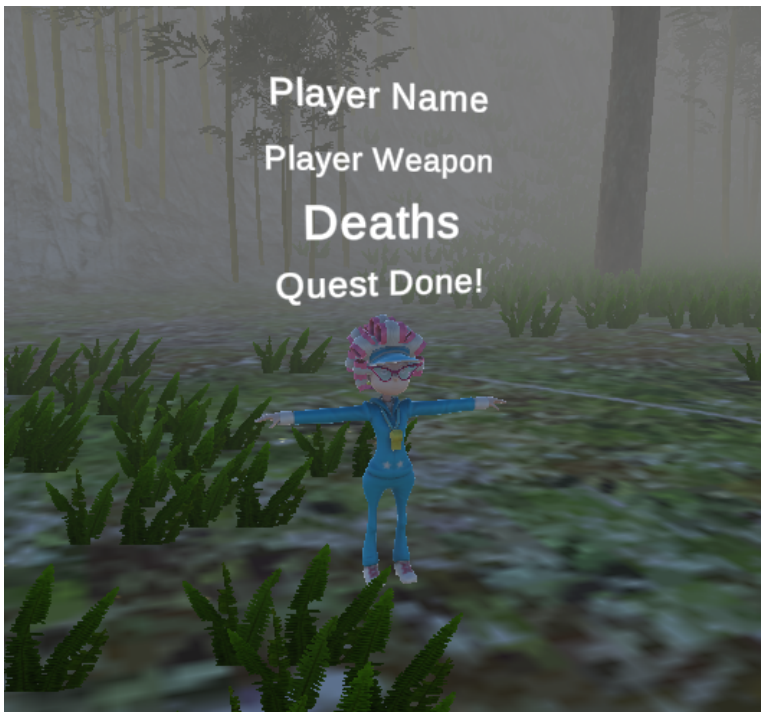
Uma interface, que é visível apenas para os outros jogadores, foi criada para o

personagem. A interface apresenta quatro informações, as quais ficam armazenadas no PlayFab e são dados persistentes:

- *Player Name* (Nome do Jogador): Nome do usuário, cadastrado na criação da conta;
- *Player Weapon* (Arma do Jogador): Item de inventário, cadastrado no primeiro *login*;
- *Deaths* (Mortes): Número de mortes do jogador;
- *Quest Done!* (Missão Completa!): Sinalizador de missão completa.

Há também uma interface visível apenas para o usuário. Mais detalhes a respeito da interface, das mecânicas e da missão serão apresentados na próxima subseção, *Quest*. Na Figura 4.12 é mostrado o personagem dentro da cena *ForestQuest*.

Figura 4.12 – Personagem na cena *ForestQuest*. Os dados sobre o personagem são visíveis apenas para os outros usuários.



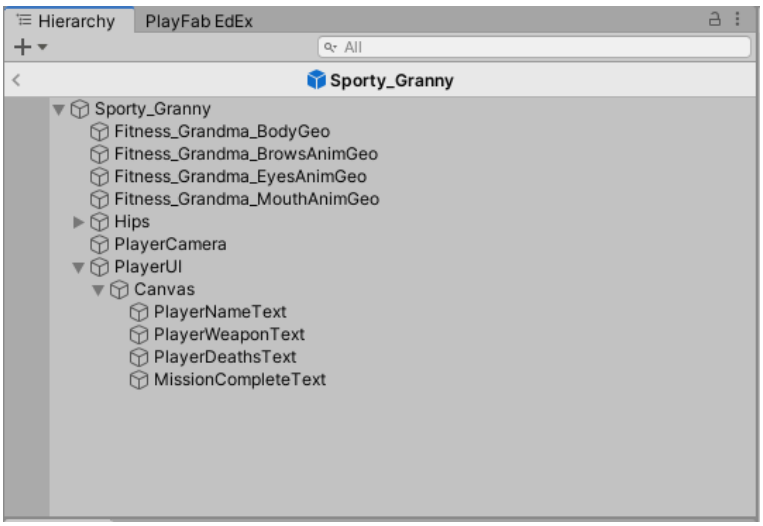
Fonte: Própria

4.4.1 Prefab

O personagem a ser controlado por cada jogador é instanciado a partir do prefab *Sporty_Granny*. O prefab é um objeto que faz parte da cena *ForestQuest*, já contendo animações, scripts de movimentação, colisores e rigidbody (componente do Unity que permite ao objeto ser influenciado pela física do cenário).

Contudo, algumas alterações foram realizadas. Uma câmera, uma interface para mostrar os dados de usuário e os componentes relacionados ao Photon foram adicionados. Na Figura 4.13 podem ser vistos os *game objects* que compõem *Sporty_Granny* na *hierarchy* do Unity. O visual do personagem é o apresentado na Figura 4.12

Figura 4.13 – Objetos que compõem o prefab *Sporty_Granny*.



Fonte: Própria

4.4.2 Instanciação de Jogador

Quando a cena *ForestQuest* é carregada, é necessária a instanciação de cada jogador dentro da *game room*. O objeto da cena responsável por essa tarefa é o *GameSceneManager*. O método de instanciação é chamado em cada cliente assim que a cena é carregada e recebe como parâmetros:

- O nome do prefab a ser instanciado;
- Um *Vector3* correspondente a uma posição em três dimensões;
- Um *Quaternion* que corresponde à rotação do personagem.

O prefab utilizado foi apresentado na seção 4.4.1. A posição em três dimensões é aleatória, limitada para que os personagens sejam instanciados dentro de uma certa região do mapa. É passado também um valor para que o personagem não tenha rotação inicialmente: *Quaternion.Identity*. O trecho do código responsável pela instanciação de personagem pode ser visto na Figura 4.14.

Figura 4.14 – Trecho de código correspondente à instanciação de cada jogador.

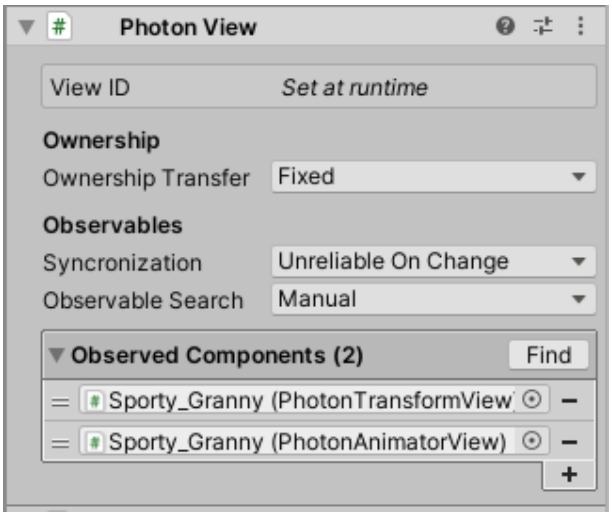
```
PhotonNetwork.Instantiate(playerPrefab.name, new Vector3(randomX, y, randomZ), Quaternion.identity);
```

Fonte: Própria

4.4.3 Componente Photon View

A sincronização dos objetos na cena é feita através do componente *Photon View*, adicionado ao prefab *Sporty_Granny*. Esse componente permite que alguns outros componentes do prefab sejam observáveis por outros jogadores, bastando arrastá-los para *Observed Components*, dentro de *Photon View*. O componente também é responsável por identificar um objeto através da rede, atribuindo um *viewID* para o mesmo, em tempo de execução. Na Figura 4.15 pode ser visto o componente citado.

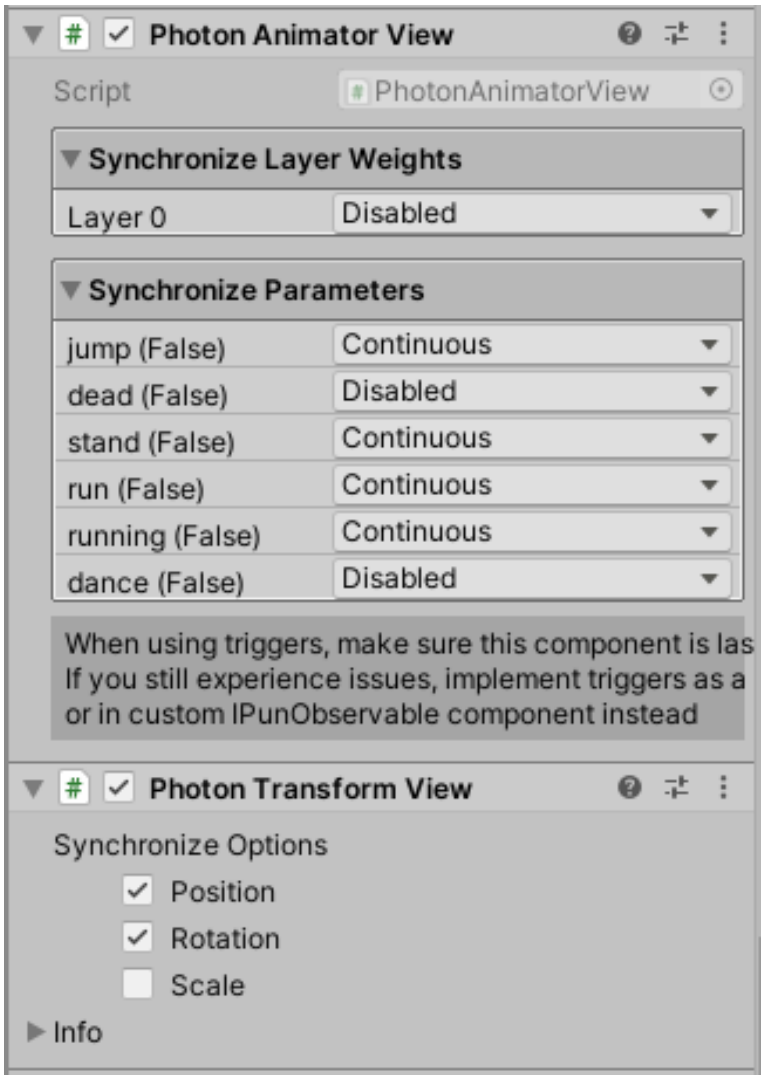
Figura 4.15 – Componente *Photon View* no prefab *Sporty_Granny*. É possível notar que há dois *Observed Components*. O *View ID* está assinalado como *Set at runtime* (Definido em tempo de execução).



Fonte: Própria

Para garantir a funcionalidade do jogo, é necessário o compartilhamento das animações e da posição e rotação de cada jogador. Uma vez que os componentes *Animator*, responsável pela animação, e *Transform*, responsável pela posição, rotação e escala, são arrastados para *Observed Components*, dois novos componentes são criados: *Photon Animator View* e *Photon Transform View*. A Figura 4.16 apresenta ambos os componentes.

Figura 4.16 – *Photon Animator View* e *Photon Transform View*.



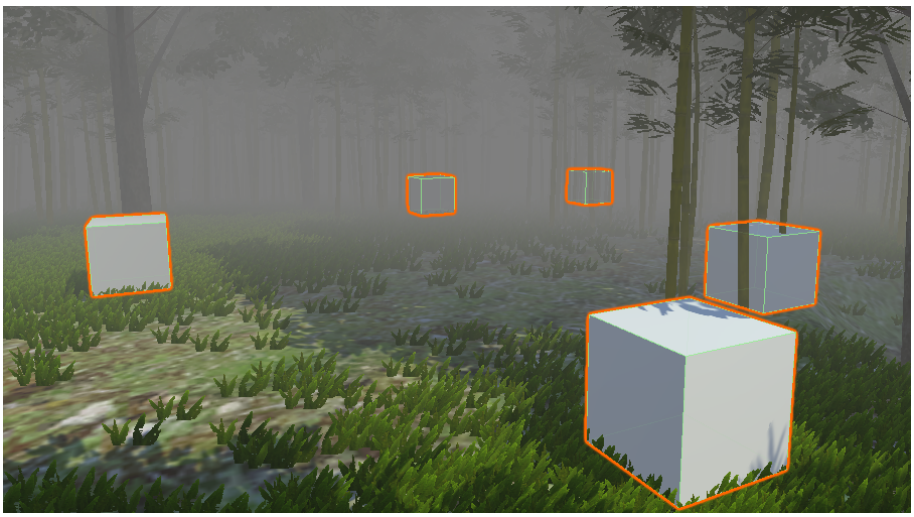
Fonte: Própria

Através do componente *Photon Animator View* são transmitidos alguns parâmetros, os quais correspondem às animações observáveis pela rede: *jump* (pular), *stand* (ficar de pé), *run* (correr) e *running* (correndo). Por meio de *Photon Transform View* são transmitidas as opções *Position* (Posição) e *Rotation* (Rotação). *Scale* (Escala) não é transmitida, pois mudanças na escala não ocorrem durante a execução do jogo.

4.5 Quest

A *quest* realizada pelo jogador consiste em uma série de objetivos. O usuário deve controlar seu personagem e alcançar uma sequência de cubos espalhados pela cena. É possível visualizar os cubos que ficam espalhados pelo mapa na Figura 4.17.

Figura 4.17 – Cubos espalhados pela cena *ForestQuest*. O objetivo da missão é alcançar cada um deles.



Fonte: Própria

Uma vez que o jogador ainda não tenha completado a *quest*, a interface do usuário é apresentada da seguinte maneira:

- Na região inferior direita há uma Lista de Tarefas (*Task List*) com a sequência de objetivos a serem cumpridos;
- Na região superior da interface há uma bússola que aponta na direção do próximo objetivo;
- Na região superior esquerda são mostrados alguns dos dados persistentes armazenados no PlayFab (nome de usuário, item de inventário e número de mortes).

Na Figura 4.18 pode ser visto o estado da interface de usuário antes da missão ser finalizada.

Quando a missão é completada, isto é, no momento em que as tarefas são cumpridas, há uma mudança na interface de usuário. A bússola é desativada e o sinalizador *Quest Done!* é apresentado ao jogador. A Figura 4.19 apresenta a interface de usuário após a conclusão da missão.

Há ainda uma mecânica de morte. Caso o jogador caia em um buraco, ele renasce em uma região aleatória do mapa após 7 segundos. O número de mortes e o estado do jogo (missão completa ou missão em andamento) são informações as quais também são armazenadas no PlayFab: correspondentes a cada usuário e persistentes entre as partidas.

Figura 4.18 – Interface do usuário com a missão em andamento.

Fonte: Própria

Na Figura 4.20 pode ser vista a tela de morte. A mensagem *You are dead. Respawning soon* (Você morreu. Renascendo em breve) é apresentada na interface do usuário por 7 segundos.

4.5.1 Remote Procedure Calls no Protótipo

As *Remote Procedure Calls* (Chamadas de Procedimento Remoto) são chamadas pelo usuário em clientes remotos os quais estão em uma mesma sala. Elas são realizadas a partir do componente *PhotonView* do objeto. No caso do protótipo, elas são utilizadas para *desativar o modelo do personagem* (prefab *Sporty_Granny*) em caso de morte.

Caso não houvesse uma RPC, o modelo seria desativado apenas no cliente do jogador que caiu no buraco. Utilizando RPC, o modelo pode ser desativado em todos os clientes da mesma sala. Para isso, são passados três parâmetros:

- A assinatura do método a ser chamado em clientes remotos. É importante ressaltar que o método precisa de uma flag (*[PunRPC]*) para poder ser passado como atributo em uma Chamada de Procedimento Remoto, conforme na Figura 4.21;
- Os alvos da RPC;
- Os parâmetros do método a ser chamado remotamente.

Figura 4.19 – Interface do usuário com missão completa. O sinalizador *Quest Done!* pode ser visto na região superior central.

Fonte: Própria

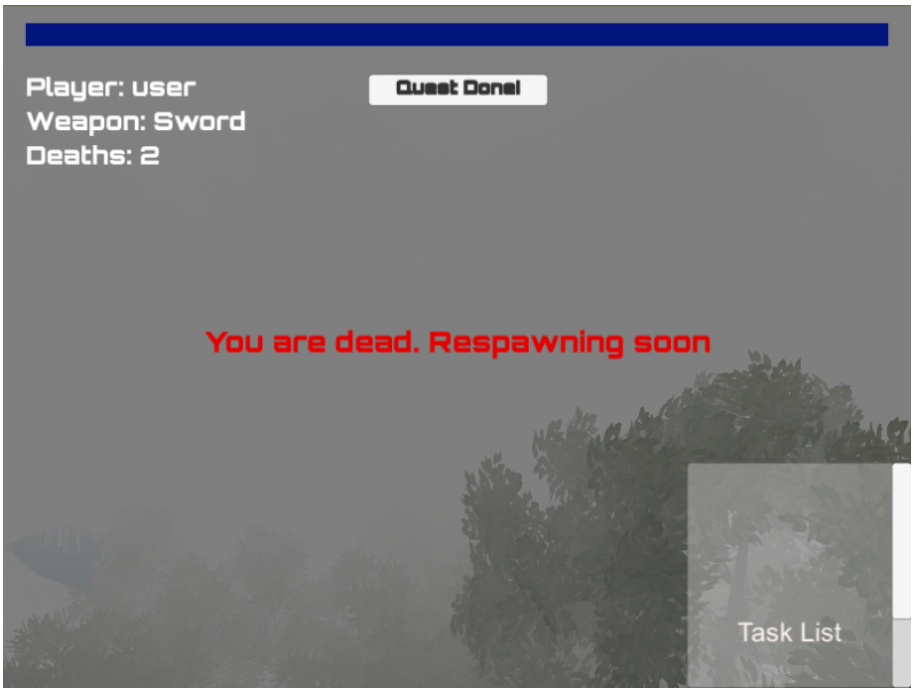
Na RPC chamada em *ForestQuest*, a assinatura do método para ativar e desativar o modelo é *"ActivateDeactivatePlayerModel"*. O alvo é *RpcTarget.AllBuffered*, visando enviar a RPC para todos os usuários na sala e executá-la imediatamente. Finalmente, o parâmetro do método a ser chamado remotamente é *false* (desativando o modelo). Na Figura 4.22 pode ser visto o trecho do código responsável pela *Remote Procedure Call* descrita.

4.5.2 Envio e Recebimento de Dados

No protótipo existem informações que são enviadas e recebidas diretamente do PlayFab, e outras que precisam ser armazenadas temporariamente como *Custom Properties* (Propriedades Customizadas descritas na seção 3.3.1) atribuídas a cada jogador. No momento que o usuário faz login no jogo, são solicitadas informações ao PlayFab, e em caso de sucesso na obtenção das mesmas, elas são armazenadas temporariamente. Na Figura 4.23; *receivedWeaponData*, *receivedMissionCompleteData* e *receivedDeathsData* são informações recebidas do PlayFab, armazenadas em um *hashtable* e configuradas como Propriedades Customizadas.

Os dados precisam ser configurados na forma de *Custom Properties* porque eles são compartilhados entre os jogadores. Quando o personagem de certo jogador é instanciado,

Figura 4.20 – Tela de morte.



Fonte: Própria

Figura 4.21 – Método para ativar ou desativar modelo, assinalado com a flag *[PunRPC]*.

```
[PunRPC]
void ActivateDeactivatePlayerModel(bool option)
{
    playerModel.SetActive(option);
    playerUI.SetActive(option);
}
```

Fonte: Própria

as Propriedades Customizadas são acessadas e mostradas na interface; elas são visíveis, sobre o personagem, para todos os jogadores da cena. Na ocorrência de atualizações nesses dados, por exemplo, na ocasião do personagem controlado por um jogador cair no buraco e o número de mortes aumentar em *1*, a Propriedade Customizada é atualizada instantaneamente.

Figura 4.22 – Código responsável por realizar a RPC em todos os jogadores da sala. É necessário acessar o componente *PhotonView* do objeto para executá-la.

```
gameObject.GetComponent<PhotonView>().RPC("ActivateDeactivatePlayerModel", RpcTarget.AllBuffered, false);
```

Fonte: Própria

Figura 4.23 – O armazenamento é feito através da *hashtables*. Contudo, é utilizada aqui uma classe específica do *Photon* para tabelas de dispersão: *ExitGames.Client.Photon.Hashtable*.

```
ExitGames.Client.Photon.Hashtable hash = new ExitGames.Client.Photon.Hashtable();

hash.Add("Weapon", receivedWeaponData);
hash.Add("MissionComplete", receivedMissionCompleteData);
hash.Add("Deaths", receivedDeathsData);

PhotonNetwork.SetPlayerCustomProperties(hash);
```

Fonte: Própria

Para o envio de dados é feito o processo contrário: a Propriedade Customizada é acessada e enviada através de uma requisição para a nuvem do PlayFab.

4.5.3 Configurando os Estados do Jogo

Dentro da cena *ForestQuest* há um objeto denominado *QuestManager*, responsável por toda a lógica da missão. Para saber se o jogador concluiu ou não a missão, assim que a cena é carregada, é feita uma verificação nas Propriedades Customizadas do jogador local, as quais foram recebidas do PlayFab. Na situação em que o jogador concluiu a missão, são realizadas as devidas alterações na cena do jogo, conforme explicado na seção 4.5.

5 Experimentação e Resultados

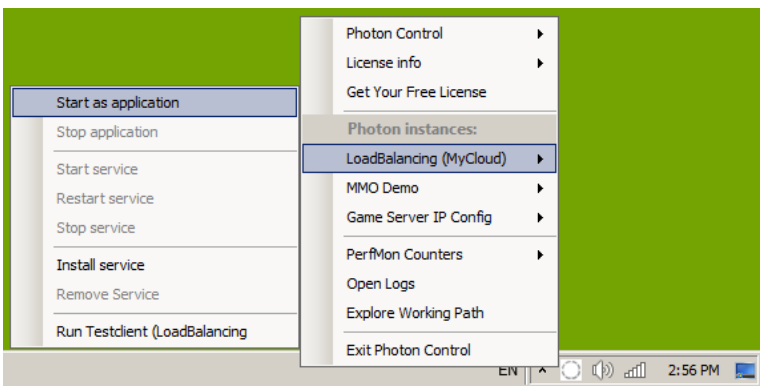
Este capítulo é composto por uma descrição dos experimentos realizados, e respectivos resultados, para validar o funcionamento do protótipo, disponível neste link. Primeiro é mostrado como é realizada a configuração do acesso ao Photon Server. Em seguida, são efetuados três experimentos: criação de contas e autenticação, teste do sistema de criação e entrada nas salas, e, por fim, teste de funcionalidades básicas de jogos online na cena *ForestQuest*.

5.1 Configurando o Acesso ao Photon Server

Para a realização dos experimentos descritos no presente capítulo, o servidor foi configurado como *Set Local IP: 192.168.1.4*, ou seja, apenas clientes na mesma rede local podem o acessar. Para configurar o servidor como público e permitir o acesso através da internet, é necessário alterar a configuração do servidor para *Set Public IP*.

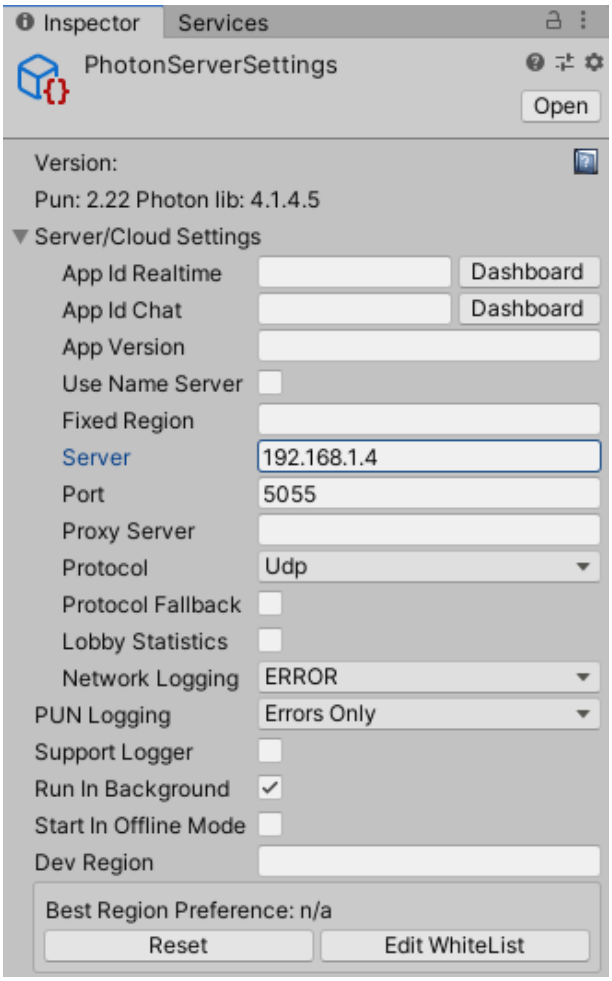
Também faz-se necessário realizar as configurações dentro do projeto. Primeiramente, *PhotonServerSettings* é acessado de dentro do projeto desenvolvido no motor de jogos Unity. Nesse componente são configurados o IP do servidor a ser acessado, a porta que será utilizada e o protocolo. Na Figura 5.2 é possível visualizar as configurações dentro do projeto: *Server: 192.168.1.4* (Servidor), *Port: 5055* (Porta) e *Protocol: Udp* (Protocolo). A porta 5055 é a porta padrão utilizada na conexão dos clientes ao *Master Server* através de protocolo UDP, para a aplicação *LoadBalancing*. Com o Photon Server instalado na máquina, basta executar a aplicação para deixar o servidor disponível para acesso, conforme pode ser visto na Figura 5.1.

Figura 5.1 – Inicialização da aplicação *LoadBalancing*.



Fonte: (PHOTON, 2020f)

Figura 5.2 – Configurações de rede dentro do projeto desenvolvido. *Server: 192.168.1.4* (Servidor), *Port: 5055* (Porta) e *Protocol: Udp* (Protocolo)



Fonte: Própria

5.2 Experimento 1: Cadastro e Autenticação

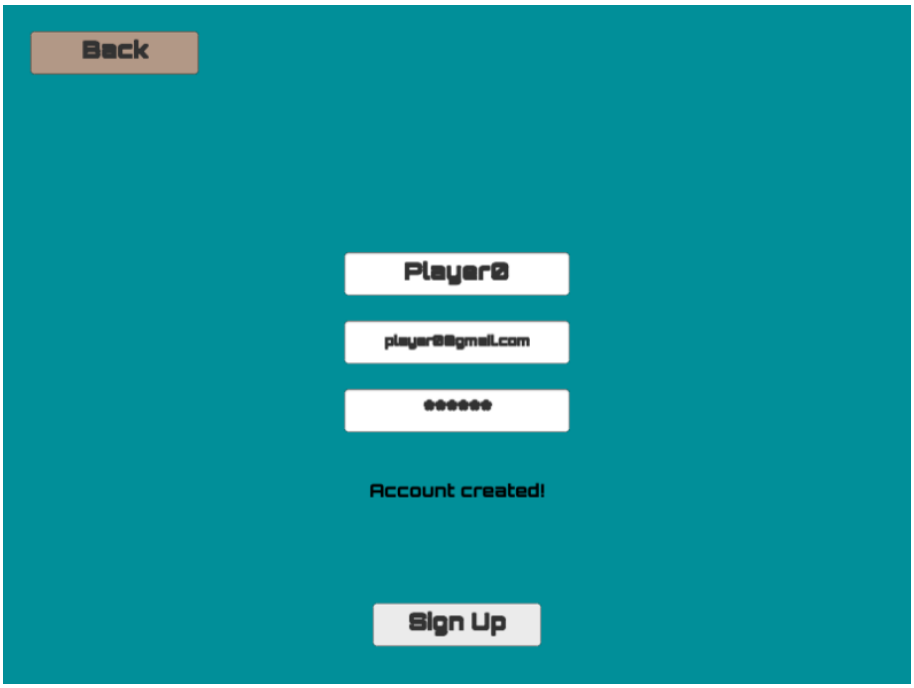
Para a realização desse teste, foram criadas 10 contas de usuários fictícios através de uma *build* do protótipo. Em todas elas tinha um nome de usuário (*username*), e-mail, senha (*password*) e uma variável *Weapon* associada ao inventário, inicialmente vazia. Em todas as contas havia também o número de mortes (*Deaths*) e uma variável para sinalizar se o usuário completou ou não a missão (*MissionComplete*). A Tabela 5.1 mostra os dados iniciais das 10 contas criadas para teste.

A criação das contas e a autenticação são realizadas a partir dos painéis apresentados na seção 4.2. Em caso de sucesso, a mensagem *Account created!* (Conta criada!) é apresentada ao usuário. Na Figura 5.3 pode ser visualizado o estado do painel em caso de sucesso no cadastro do *Player0*.

Tabela 5.1 – Dados correspondentes às contas cadastradas no PlayFab.

Username	E-mail	Password	Weapon	Deaths	MissionComplete
Player0	player0@gmail.com	pl0123		0	NO
Player1	player1@gmail.com	pl1123		0	NO
Player2	player2@gmail.com	pl2123		0	NO
Player3	player3@gmail.com	pl3123		0	NO
Player4	player4@gmail.com	pl4123		0	NO
Player5	player5@gmail.com	pl5123		0	NO
Player6	player6@gmail.com	pl6123		0	NO
Player7	player7@gmail.com	pl7123		0	NO
Player8	player8@gmail.com	pl8123		0	NO
Player9	player9@gmail.com	pl9123		0	NO

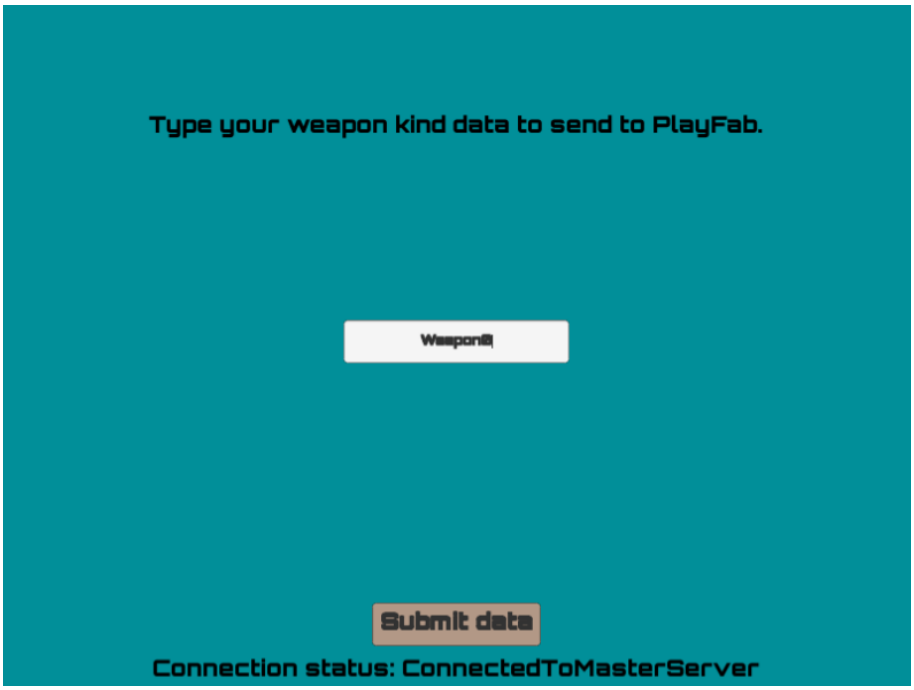
Figura 5.3 – Conta do *Player0* criada com sucesso. A mensagem pop-up *Account Created!* é apresentada ao usuário.



Fonte: Própria

Com a conta criada, é possível realizar a autenticação por meio de e-mail e senha. Em caso de sucesso na autenticação, o jogador é redirecionado para o painel da Figura 5.4, onde pode escolher um item de inventário (*Weapon*). Para *Player0* será *Weapon0*. Ao clicar em *Submit Data*, a informação é armazenada no PlayFab.

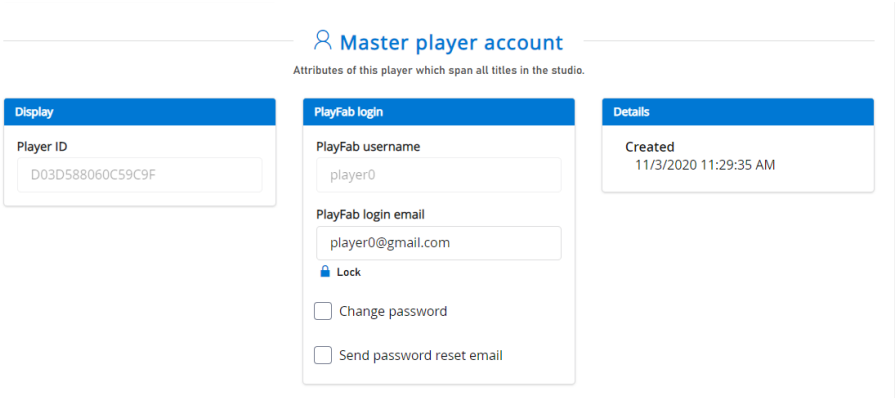
Figura 5.4 – Painel onde o jogador submete o item de inventário para o *PlayFab*.



Fonte: Própria

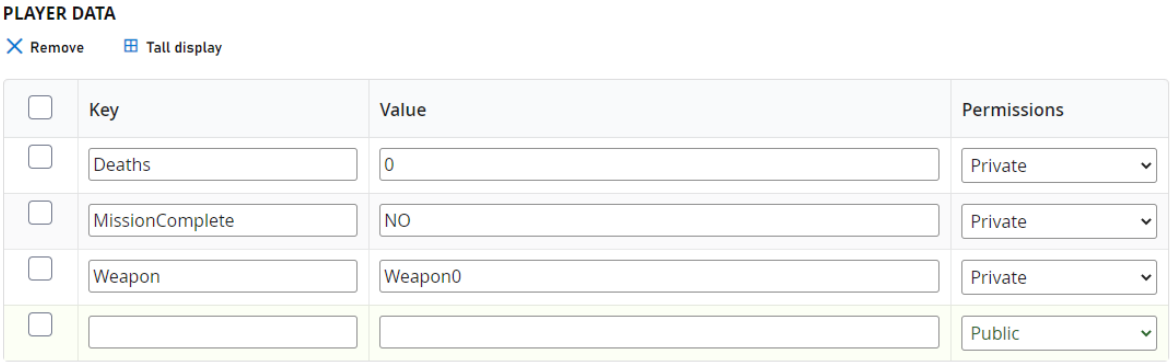
Além disso, diversas informações associadas ao usuário ficam disponíveis no gerenciador online do PlayFab. Na Figura 5.5 podem ser vistas algumas dessas informações: *PlayerID*, *username*, e-mail, data e horário de criação da conta. Na Figura 5.6 podem ser vistos os dados já citados do *Player0*.

Figura 5.5 – Informações associadas ao usuário. Disponíveis no gerenciador online do PlayFab.



Fonte: Própria

Figura 5.6 – Variáveis criadas para armazenar dados específicos do *Player0*. Também disponíveis no gerenciador online do PlayFab



Fonte: Própria

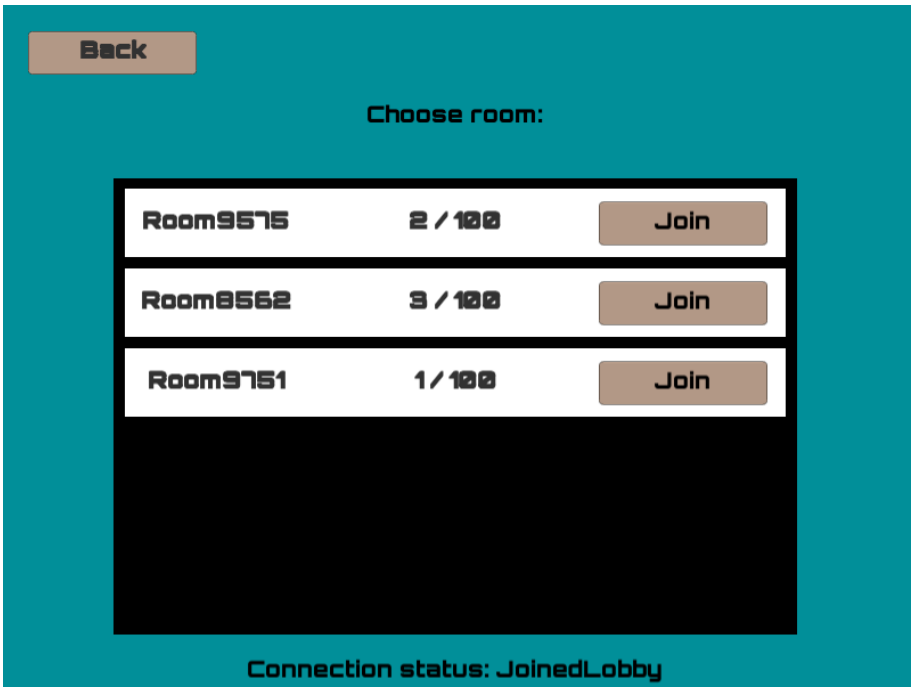
O processo descrito nesta seção foi realizado para cada um dos 10 jogadores. Do *Player0* ao *Player9* foram cadastradas respectivamente a *Weapon0* à *Weapon9*. Todas as contas foram criadas com sucesso.

5.3 Experimento 2: Game Rooms

Este experimento visa validar se a criação de *salas de jogo* está funcionando devidamente. Para isso, é realizado o login em 6 contas e a criação de 3 *game rooms*. O jogador cujo o *username* é *Player0* tenta então acessar uma delas. A criação e o acesso às salas é feito como descrito na seção 4.3.

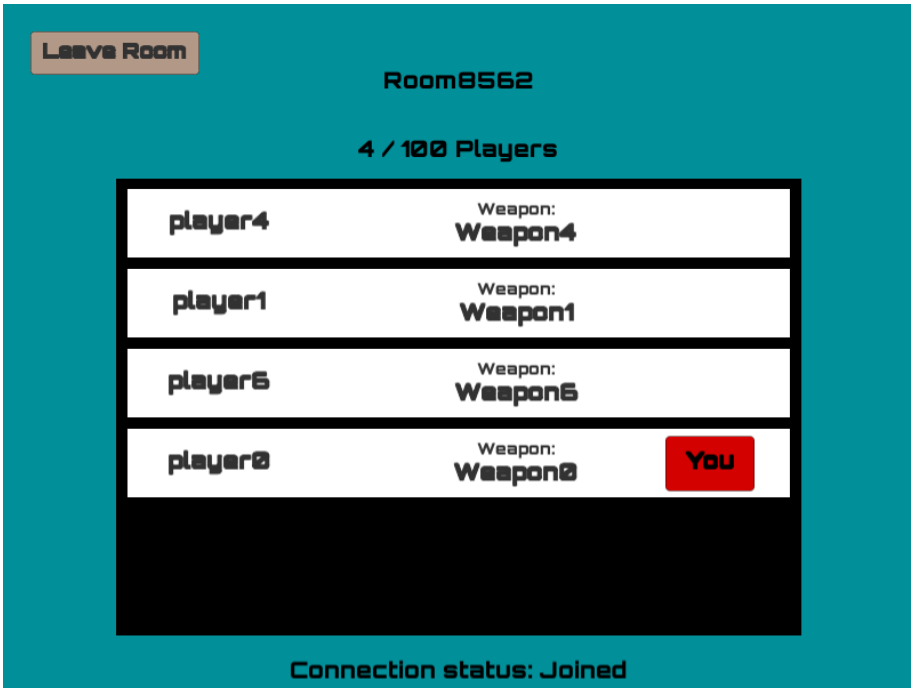
Na Figura 5.7 pode ser visualizado o painel do lobby, no qual é mostrado ao *Player0* as salas disponíveis. O jogador então acessa a sala denominada *Room8562*. A visão do jogador dentro da *Room8562* é mostrada na figura 5.8. A sala é composta por 4 jogadores: *Player0*, *Player1*, *Player4* e *Player6*. Os itens de inventário (*Weapon*) e os nomes de usuário de cada um são puxados do PlayFab e ficam disponíveis para a visualização de outros jogadores.

Figura 5.7 – Visão do jogador ao acessar o lobby no experimento. Há três salas disponíveis: *Room9575*, *Room8562* e *Room9751*.



Fonte: Própria

Figura 5.8 – Visão do jogador dentro da *Room8562*. Neste painel podem ser visualizados os *usernames* e itens de inventário de cada jogador.



Fonte: Própria

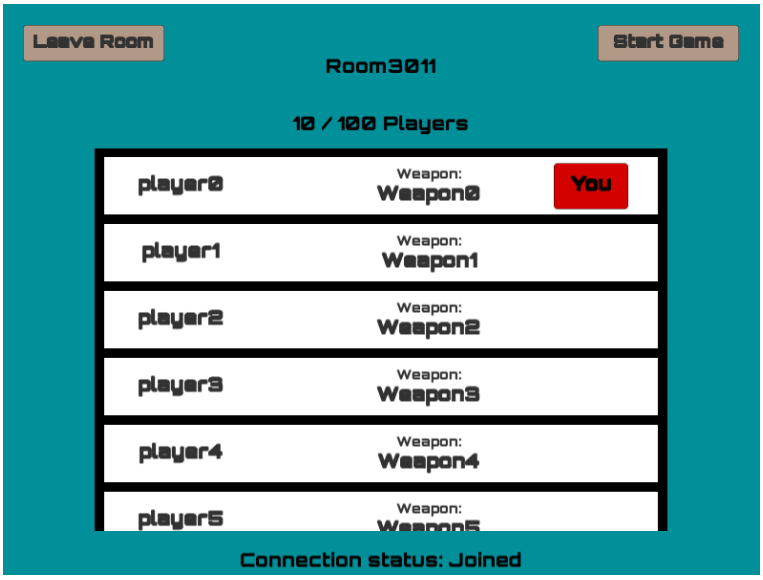
5.4 Experimento 3: Mundo Virtual Compartilhado

Para o Experimento 3, é criada uma sala com todos os 10 jogadores da Tabela 5.1, e então o jogador que criou a sala inicia a partida. Dentro da partida, são realizados testes para comprovar que todos os jogadores são instanciados no mesmo mapa, e que os dados estão sendo compartilhados entre eles e devidamente armazenados no PlayFab.

5.4.1 Criação de Sala

Após a autenticação dos 10 jogadores, o *Player0* cria uma sala, a qual é acessada por todos os outros 9 jogadores. O criador da sala inicia a partida e a cena *ForestQuest* é carregada para todos os usuários na sala. Na Figura 5.9 pode ser vista a sala com os 10 jogadores.

Figura 5.9 – Visão do criador da sala. Para iniciar a partida, basta clicar em *Start Game* na região superior direita do painel.



Fonte: Própria

5.4.2 Cena *ForestQuest*

No momento em que o mestre da sala inicia a partida, a cena *ForestQuest* é carregada para todos os jogadores. Os personagens controláveis são instanciados em locais aleatórios do mapa. É possível observar na Figura 5.10 que todos os personagens estão agrupados dentro da *camera view* do *Player0*. Neste momento, nenhum jogador interagiu ainda com o mundo virtual. Portanto, ninguém completou a quest e o contador de mortes de todos ainda está zerado.

Figura 5.10 – Todos os jogadores agrupados dentro da *CameraView* do *Player0*.

Fonte: Própria

Após uma série de interações com o mundo virtual, alguns jogadores concluíram a *quest* e realizaram a mecânica de morte, incrementando o contador. Na Figura 5.11 pode ser visualizado que os dados sobre os diversos personagens têm valores diferentes. Os jogadores que concluíram a missão agora apresentam o sinalizador *Quest Done!*.

Figura 5.11 – Todos os jogadores agrupados dentro da *CameraView* do *Player0* após uma série de interações com o mundo virtual

Fonte: Própria

Tomando o *Player5* como exemplo, na interface de usuário, sobre o personagem, outros jogadores podem ver quatro informações:

- O nome de usuário: *player5*;
- O item de inventário do usuário: *Weapon5*;
- A quantidade de mortes: *2*;
- O sinalizador de missão completa: *Quest Done!*.

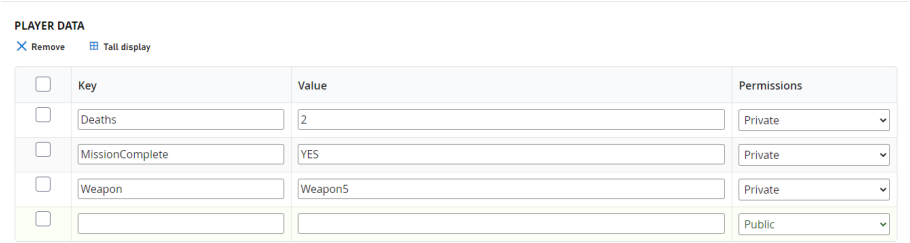
Na Figura 5.12 pode ser vista a interface descrita sob a visão do jogador *Player0*. Essas informações do jogador ficam armazenadas no gerenciador online do PlayFab. Na Figura 5.13 podem ser vistos os dados do *Player5* armazenados nos servidores do PlayFab.

Figura 5.12 – Interface do *Player5* sob a visão do *Player0*.

Fonte: Própria

Para todos os jogadores da sala, as interações que ocorreram dentro do mundo virtual e foram apresentadas nas interfaces de usuário, foram refletidas na nuvem do PlayFab. Os dados se mantiveram sincronizados, de forma que mesmo com o jogador saindo do jogo, ao realizar autenticação novamente, seus recursos foram mantidos. Na Tabela 5.2 podem ser vistos os dados de cada usuário após o Experimento 3.

Figura 5.13 – Informações persistentes do *Player5*. Armazenadas nos servidores do PlayFab.



Fonte: Própria

Tabela 5.2 – Dados correspondentes às contas cadastradas no PlayFab após interações no mundo virtual.

Username	E-mail	Password	Weapon	Deaths	MissionComplete
Player0	player0@gmail.com	pl0123	Weapon0	0	NO
Player1	player1@gmail.com	pl1123	Weapon1	1	NO
Player2	player2@gmail.com	pl2123	Weapon2	0	NO
Player3	player3@gmail.com	pl3123	Weapon3	0	YES
Player4	player4@gmail.com	pl4123	Weapon4	0	NO
Player5	player5@gmail.com	pl5123	Weapon5	2	YES
Player6	player6@gmail.com	pl6123	Weapon6	0	NO
Player7	player7@gmail.com	pl7123	Weapon7	0	YES
Player8	player8@gmail.com	pl8123	Weapon8	3	NO
Player9	player9@gmail.com	pl9123	Weapon9	0	YES

6 Conclusões

Este trabalho foi constituído, primeiramente, por um estudo comparativo entre diferentes soluções para algumas das principais problemáticas envolvendo *jogos online com grande quantidade de usuários*. Foram analisados diversos fatores, tanto para as soluções de rede quanto para as de dados, tais como: confiabilidade, fácil integração e suporte. Em seguida, foi desenvolvido um protótipo utilizando as tecnologias selecionadas. No protótipo foram então realizados três testes para a validação do mesmo como solução: cadastro e autenticação de usuários, criação de salas de jogo e instanciação de diversos usuários em uma mesma sala.

Após a realização de diferentes experimentos, um resultado positivo foi alcançado, o modelo desenvolvido funcionou conforme o esperado. Através do *Photon Unity Networking* e do *Photon Server* foi possível aos usuários acessarem um servidor auto-hospedado, no qual a aplicação *LoadBalancing* permitiu a criação de salas de jogo e o compartilhamento de um mesmo ambiente virtual, onde os usuários puderam interagir entre si e com o mapa. Por meio das funcionalidades do PlayFab, foi viável criar diferentes contas, realizar autenticação e armazenar as informações correspondentes a cada usuário.

O protótipo desenvolvido no motor de jogos *Unity* contém, desta forma, uma parcela das funções essenciais para jogos massivos online. As tecnologias utilizadas se mostraram como ferramentas poderosas e de uso acessível para desenvolvedores.

As ferramentas utilizadas possibilitam ainda a realização de testes mais complexos com o mesmo protótipo e a criação de um modelo mais robusto, com mais funcionalidades. O presente documento pode servir de referência para uso das supracitadas tecnologias.

Os objetivos do trabalho, portanto, foram alcançados. A viabilidade da solução de rede e dados para jogos massivos online foi verificada através de *estudos comparativos*, *prototipagem* e *experimentação*.

6.1 Trabalhos Futuros

A partir do projeto desenvolvido, existem diversas possibilidades de experimentos e implementações.

Como possíveis experimentos, pode-se apontar:

- Realizar testes com a criação de mais contas de usuário;
- Colocar mais jogadores dentro do mesmo ambiente virtual;

- Configurar o Photon Server com um IP público, para que os jogadores possam acessá-lo através da internet.

Como possíveis implementações, pode-se apontar:

- Adição de novas mecânicas de gameplay dentro do protótipo;
- Uso de outras funcionalidades backend do PlayFab, como placares de líderes, serviços de comércio, análise de jogadores em tempo real, entre outras;
- Implementação de funcionalidades de *chat*, para que os jogadores possam interagir através de texto ou voz;
- Aplicação das funcionalidades do protótipo em um jogo online de entretenimento, sério ou educacional.

Referências

ARMITAGE, G.; CLAYPOOL, M.; BRANCH, P. *Networking and Online Games: Understanding and Engineering Multiplayer Internet Games*. 1. ed. Wiley, 2006. ISBN 9780470018576,0470018577. Disponível em: <http://gen.lib.rus.ec/book/index.php?md5=3db17a0970666e78a03b017544d44a24>. Citado 2 vezes nas páginas 16 e 17.

ATAVISM. *WHY ATAVISM ONLINE?* 2020. Disponível em: <https://www.photonengine.com/en-US/Quantum>. Acesso em: 03 junho 2020. Citado na página 28.

AZURE PLAYFAB. *Everything you need to build and operate a live game*. 2020. Disponível em: <https://playfab.com/>. Acesso em: 03 junho 2020. Citado na página 30.

BARRI, I.; GINÉ, F.; ROIG, C. A scalable hybrid p2p system for mmofps. *18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, p. 1, 2010. Citado na página 23.

CHAMBERS, C.; FENG, W. chang; FENG, W. chi. Towards public server mmos. *The 5th Workshop on Network System Support for Games*, p. 1–7, 2017. Citado 2 vezes nas páginas 18 e 20.

CHILLI CONNECT. *LIVE GAME MANAGEMENT*. 2020. Disponível em: <https://www.chilliconnect.com/>. Acesso em: 03 junho 2020. Citado na página 30.

CHOI, G.; KIM, M. Battle royale game : In search of a new game genre. *International Journal of Culture Technology (IJCT) – Volume-2, Issue-2, Jun., 2018*, p. 5–10, 2018. Citado na página 23.

GAME SPARKS. *The Easiest Way to Build, Deploy and Scale Game Backends*. 2020. Disponível em: <https://gamesparks.com/>. Acesso em: 03 junho 2020. Citado na página 31.

JETSONEN, T. *Development of online game prototype with Unity engine*. 9 p. Dissertação (Tese de Bacharelado (Bacharelado em Engenharia de Software)) — Jyväskylä University of Applied Sciences, Finlândia, 2016. Citado na página 25.

JUNIOR, O. H.; OLIVEIRA, B. B. de; ALECRIM, R. M. de. Jogo fps educacional metafórico para o ensino da matemática do ensino médio. *SBGames 2013*, p. 1, 2013. Citado na página 23.

LIMA, E. B. da S. *Quest Design Canvas: Um modelo de criação de quests para jogos digitais de gênero RPG*. 34 p. Dissertação (Dissertação de mestrado (Mestrado em Engenharia de Software)) — Universidade Federal do Rio Grande do Norte, Natal, 2018. Citado na página 21.

MILLENIUM. *LoL revenue tops $1.5 billion in 2019*. 2020. Disponível em: <https://www.millenium.us.org/news/10506.html>. Acesso em: 12 maio 2020. Citado na página 24.

MMO POPULATIONS. *TOP MMOS IN 2020*. 2020. Disponível em: <https://mmo-population.com/top/2020>. Acesso em: 12 maio 2020. Citado na página 21.

MMORPG. *Fortnite Generated $1.8 Billion Revenue in 2019, More Than Any Other Game for Second Straight Year*. 2020. Disponível em: <https://www.mmorpg.com/>. Acesso em: 12 maio 2020. Citado na página 23.

MOREIRA, A. V. M.; OTHERS. A data mining solution for detection of potential buyers on mmorpgs. *XVI SBGames*, p. 1, 2017. Citado na página 21.

MULTIPLAY. *Multiplay*. 2020. Disponível em: <https://multiplay.com/>. Acesso em: 03 junho 2020. Citado na página 31.

ODIERNA, B. A.; SILVEIRA, I. F. Player game data mining for player classification. *XVII SBGames*, p. 1–3, 2018. Citado na página 21.

PHOTON. *BOLT*. 2020. Disponível em: <https://www.photonengine.com/en-US/BOLT>. Acesso em: 03 junho 2020. Citado na página 27.

PHOTON. *Photon Cloud or Photon Server?* 2020. Disponível em: <https://doc.photonengine.com/en-us/server/current/getting-started/onpremises-or-saas>. Acesso em: 03 junho 2020. Citado na página 26.

PHOTON. *QUANTUM*. 2020. Disponível em: <https://www.photonengine.com/en-US/Quantum>. Acesso em: 03 junho 2020. Citado na página 28.

PHOTON. *REALTIME*. 2020. Disponível em: <www.photonengine.com/en-us/Realtime>. Acesso em: 03 junho 2020. Citado 5 vezes nas páginas 26, 27, 35, 36 e 37.

PHOTON. *SERVER*. 2020. Disponível em: <https://www.photonengine.com/en-US/Server>. Acesso em: 03 junho 2020. Citado na página 26.

PHOTON. *Starting Photon in 5 Minutes*. 2020. Disponível em: <https://doc.photonengine.com/en-US/server/current/getting-started/photon-server-in-5min>. Acesso em: 13 dezembro 2020. Citado na página 57.

POLANCEC, D.; MEKTEROVIC, I. Developing moba games using the unity game engine. *MIPRO 2017*, p. 1–3, 2017. Citado 2 vezes nas páginas 24 e 25.

SANTOS, V. H. dos. *Experimentos com Aprendizado por Reforço em Cenário de Combate de StarCraft*. 20 p. Dissertação (Monografia (Graduação em Ciência da Computação)) — Universidade Federal do Rio Grande do Norte, Natal, 2017. Citado na página 22.

THE HISTORY OF VIDEO GAMES. *TENNIS FOR TWO REVIEW-1958*. 2020. Disponível em: <https://videogamehistorydevelopment.weebly.com/tennis-for-two.html>. Acesso em: 09 junho 2020. Citado na página 16.

TSIPIS, A.; KOMIANOS, V.; OIKONOMOU, K. A cloud gaming architecture leveraging fog for dynamic load balancing in cluster-based mmos. p. 1–2, 2019. Citado 2 vezes nas páginas 17 e 18.

UNITY. *Evolving multiplayer games beyond UNet*. 2020. Disponível em: <https://blogs.unity3d.com/2018/08/02/evolving-multiplayer-games-beyond-unet/?_ga=2.196459631.2110835785.1591191102-34374465.1590610136>. Acesso em: 03 junho 2020. Citado na página 25.

UNITY. *Multiplayer Overview*. 2020. Disponível em: <https://docs.unity3d.com/Manual/UNetOverview.html>. Acesso em: 03 junho 2020. Citado na página 25.