UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE MATEMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

CARLOS SERGIO DE PAIVA ARAUJO
LUAN CERQUEIRA MARTINS

Técnicas de Compensação de Latência em Jogos Multijogadores, Conectados e Síncronos

RIO DE JANEIRO
2019

CARLOS SERGIO DE PAIVA ARAUJO
LUAN CERQUEIRA MARTINS

Técnicas de Compensação de Latência em Jogos Multijogadores, Conectados e Síncronos

Trabalho de conclusão de curso de graduação apresentado ao Departamento de Ciência da Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Vinícius Gusmão Pereira de Sá

RIO DE JANEIRO
2019

Araújo, Carlos Sérgio de Paiva
A663t Técnicas de compensação de latência em jogos multijogadores, conectados e síncronos / Carlos Sérgio de Paiva Araújo, Luan Cerqueira Martins. – Rio de Janeiro, 2019.

37 f.

Orientador: Vinícius Gusmão Pereira de Sá.
Trabalho de Conclusão de Curso (graduação) - Universidade Federal do Rio de Janeiro, Instituto de Matemática, Bacharel em Ciência da Computação, 2019.

1. Jogos. 2. Latência. 3. Sistemas distribuídos. I. Martins, Luan Cerqueira. II. Sá, Vinícius Gusmão Pereira de. III. Universidade Federal do Rio de Janeiro, Instituto de Matemática. IV. Título.

CARLOS SERGIO DE PAIVA ARAUJO
LUAN CERQUEIRA MARTINS

Técnicas de Compensação de Latência em Jogos Multijogadores, Conectados e Síncronos

Trabalho de conclusão de curso de graduação apresentado ao Departamento de Ciência da Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em ___ de _____ de _____

BANCA EXAMINADORA:

Vinícius Gusmão Pereira de Sá
D.Sc.

Claudson Ferreira Bornstein
PhD

Valeria Menezes Bastos
D.Sc.

AGRADECIMENTOS

Agradeço muito à UFRJ por todas as experiências que passei, aprendizados que tive e as pessoas que conheci durante esses anos que estive lá. Eu não seria quem eu sou hoje se não fosse pela UFRJ.

Queria agradecer a todos amigos e colegas do meu período que me ajudaram chegar até aqui, aos membros da GDP que me ensinaram muito (muito mais que eu consigo escrever aqui), ao Luan meu par que está nesse projeto comigo desde o começo e a todas as demais pessoas que me apoiaram até a entrega desse trabalho.

Em especial, gostaria de agradecer às professoras e os professores que confiaram em mim e me apoiaram durante essa jornada, Prof. Carla Amor Divino Delgado, Prof. João Carlos Pereira da Silva, Prof. Rodrigo Penteado Ribeiro de Toledo, Prof. Juliana Vianna Valério, Prof. Valeria Bastos e Prof. Vinícius Gusmão Pereira de Sá.

Por fim, não poderia deixar de mencionar a minha família e meu namorado Héctor que não só me aturaram todo esse tempo, mas que sem o apoio e esforço de vocês eu não estaria escrevendo esse trabalho.

Carlos Sergio de Paiva Araujo

AGRADECIMENTOS

Começo agradecendo a oportunidade oferecida pela UFRJ que me mostrou uma paixão por programação que até então eu não conhecia, graças a incríveis professores que eu gostaria de ter demonstrado melhor minha gratidão durante esse tempo. Posso citar Prof. Rodrigo Penteado Ribeiro de Toledo, Prof. Vinícius Gusmão Pereira de Sá, Prof. Valeria Bastos, Prof. Juliana Vianna Valério, Prof. Adriano Joaquim de Oliveira Cruz (Eu não poderia ter um primeiro contato melhor com programação) e Prof. Paulo Henrique Aguiar Rodrigues

Sou muito grato ao GDP - projeto que participei desde meu primeiro período na faculdade - que me mostrou o desafio incrível de desenvolver jogos, aprendi desde coisas técnicas até aprendizados de design e até mesmo como apreciar a multidisciplinaridade na construção de um jogo.

Impossível não demonstrar minha gratidão a Deus por ter sido minha pedra angular sempre, à minha família, especialmente Ademar e Rosângela que sempre me apoiaram e educaram, à minha namorada Rebeca que me motivou como ninguém e ao Carlos que, além de aprender junto comigo no GDP e em diversas classes, me acompanhou nesse projeto.

Luan Cerqueira Martins

RESUMO

O nível de qualidade da experiência conectada em jogos multijogador cresce continuamente na indústria de jogos. Uma má partida devido a conexão fraca pode ser a diferença entre mais uma compra ou instalação de um jogo. Com isso em mente, pessoas desenvolvedoras de jogos se esforçam para fazer a experiência a melhor possível independente da conexão. Esse nível de exigência está presente em todos os possíveis gêneros e estilos de jogos. Se tem um modo multijogador conectado, quanto mais parecer que não se está jogando pela internet, melhor. Há várias técnicas e modelos arquiteturais que permitem o alcance desse objetivo.

O principal objetivo é "diminuir a distância" entre jogadores, diminuindo o impacto da latência no jogo, também conhecida como *lag*. Latência descreve o tempo para que uma mensagem entre uma pessoa jogando chegue a outra (ou a um servidor), normalmente medida em milissegundos.

Desenvolvemos um jogo básico que utiliza algumas dessas ferramentas para ilustrar os princípios por trás de técnicas de compensação da latência e como essas afetam a percepção de um jogo por quem está segurando o controle.

Palavras-chave: Jogos. Latência. Sistemas distribuídos.

ABSTRACT

The experience while playing multiplayer online games has continuously increased in the gaming industry. One bad match caused by bad connectivity can be the difference between a game's purchase or installation. With that in mind, game developers strive to make playing online the best experience posible regardless of the quality of the player's connection.
This level of exigency is present across all genres of games. If it has an online multiplayer mode, it needs to look like it isn't being played online. There are several architecture models and techniques that can be used to achieve that.
The main objective of those are to "decrease the distance" between players by decreasing the impact of latency in the game, better known as lag. Latency describes the time it takes for a message to travel from one player to another (or to a server) through the internet, it's normally measured in milliseconds.
We developed a simple game that uses some lag compensation techniques to illustrate the principles behind them and how they can affect the game experience.

Keywords: Games. Latency. Distributed system.

LISTA DE ILUSTRAÇÕES

LISTA DE CÓDIGOS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

lag	Latency at Game

SUMÁRIO

1 MOTIVAÇÃO, CONTEXTO E OBJETIVO

1.1 MOTIVAÇÃO

Tivemos três principais motivadores para a escolha e execução desse projeto. A primeira foi a falta de material acadêmico relacionado ao assunto, durante nossas pesquisas encontramos muitos vídeos, textos em páginas pessoais e guias, porém a quantidade de artigos encontrados foi muito baixa em comparação.

O diferente paradigma de desenvolvimento também foi atrativo, um jogo multijogador funciona como um sistema distribuído que sofre com o tempo de rede, e nós tínhamos pouca ou quase nenhuma experiência com esse tipo de desenvolvimento.

Por fim, nossa experiência com o Game Development Project (GDP), o grupo de extensão de desenvolvimento de jogos na Universidade Federal do Rio de Janeiro (UFRJ), nos guiou para esse projeto. Participando do grupo tivemos a oportunidade de aprender muito sobre desenvolvimento de jogos e isso nos possibilitou seguir com esse projeto hoje.

1.2 CONTEXTO E OBJETIVO

1.2.1 Jogos Online

Jogos com modos multijogador conectados pela rede não são novidade, datando de 1973 (GLAZER,). Com o passar dos anos e o maior acesso à Internet, jogos online se tornaram imensamente populares. A empresa Riot Games, por exemplo, confirmou que seu jogo League of Legends (LEAGUE..., 2009) tem 100 milhões de jogadores únicos mensais (TASSI, 2016) e inclusive enche estádios com seus campeonatos, como na foto 1.

Figura 1 – League of Legends 2016 World Championship

No entanto, mesmo com a poderosa estrutura de conexão e banda larga que podemos ver (SANTOS, 2016), a latência da conexão continua sendo um dos maiores desafios a se contornar no desenvolvimento de jogos. Jogadores com uma latência muito alta normalmente tem desvantagem nas partidas em relação aos outros, além de sua experiência jogando ser consideravelmente pior.

Quando nos referimos a latência queremos falar do tempo necessário para que uma informação possa chegar de um cliente do jogo para outro (DONGEN, 2017). Um cliente representa uma máquina com o jogo executando, seja um celular ou um computador.

A definição do que é considerado uma latência alta ou baixa varia muito com o tipo de jogo em questão (AXISROB,), por exemplo, para alguns jogos acima de 100 milissegundos já é considerado inaceitável, para outros não. Contudo, uma máxima para jogos online é que quanto mais próximo de 0, melhor, uma prova disso são os campeonatos, em que a maioria das partidas são feitas em rede LAN, para que não exista lag entre os jogadores.

Tratar da latência é importante devido as dimensões do mercado (NEWZOO,), a indústria de jogos se tornou altamente competitiva e existem milhares de empresas todos os dias brigando para que você instale ou compre o jogo dela. Isso juntamente com o surgimento das grandes lojas online e seus sistemas de avaliação, que permitem que jogadores pontuem um jogo publicamente, para que um jogo faça sucesso, precisa ser feito com muita qualidade.

Além disso, vemos um crescimento enorme da presença de jogos conectados no mercado (WEPC,), ou seja, a nova tendência entra de encontro diretamente com os problemas de latência.

Um jogo multijogador pode ser classificado de acordo com a maneira com que lida com as ações dos jogadores, podendo ser categorizado como assíncrono ou síncrono.

1.2.1.1 Jogos Assíncronos

São jogos que não se passam em tempo real, o que significa que uma pessoa não necessita tomar uma ação ao mesmo tempo ou imediatamente depois que outras a fazem.

Esse gênero de jogo se tornou muito popular com o crescimento de jogos dentro de redes sociais e em celulares, como por exemplo: Draw Something (DRAW..., 2012) (ícone na figura 2) da OMGPOP, que tem entre 50 e 100 milhões de instalações no Android (DRAW...,), e Clash of Clans (CLASH..., 2012) (ícone na figura 3) da Supercell, que tem entre 100 e 500 milhões de instalações, também no Android (CLASH...,).

Em jogos assíncronos um jogador não precisa ver o que outro está fazendo instantaneamente, diminuindo o impacto que a latência causa para a experiência de jogo.

Figura 2 – Ícone do Draw Something Classic no Android

Figura 3 – Ícone do Clash of Clans no Android

1.2.1.2 Jogos Síncronos

Jogos síncronos são aqueles em que todos os jogadores atuam ao mesmo tempo e a diferença de tempo entre a ação e reação das pessoas jogando importa.

Trata-se do gênero mais famoso de jogos multijogador para computador e consoles e recentemente tem criado grande espaço em celulares também. Podemos citar: Minecraft (MINECRAFT, 2009) (imagem na figura 4) da Microsoft, com mais de 100 milhões de cópias vendidas (MINECRAFT. . . ,), e Clash Royale (CLASH. . . , 2016) (ícone na figura 5) da Supercell, que tem entre 100 e 500 milhões de instalações no Android (CLASH. . . ,).

Jogos síncronos sofrem bastante com problemas de latência, principalmente os que tem uma interação mais acelerada. A necessidade de se observar o que outros jogadores fazem instantaneamente gera diversos desafios na implementação de um jogo.

Figura 4 – Logo de Minecraft com figuras do jogo

Figura 5 – Ícone do Clash Royale no Android

1.2.2 Objetivo do Projeto

O objetivo do projeto é demonstrar quais técnicas podem ser utilizadas para mitigar os efeitos da latência e como elas afetam um jogo. Para alcançarmos esse objetivo, desenvolvemos um jogo multijogador online síncrono chamado Bounty Hunters (figura 6) e aplicamos essas técnicas durante seu desenvolvimento.

O jogo como um todo foi pensado para que as técnicas de mitigação pudessem ser aplicadas, inclusive as próprias funcionalidades do jogo foram planejadas de tal forma que aumentassem ou deixassem visível algum desafio relacionado a latência. As técnicas foram aplicadas na implementação das seguintes funcionalidades:

1. Seleção de personagens automática sem duplicidade
2. Movimentação das naves de jogadores conectados

Figura 6 – Foto de uma partida com 4 personagens

3. Captura de pontos

4. Representação do uso do turbo de jogadores conectados

Bounty Hunters é inspirado no clássico PacMan (PáGINA. . . ,), porém com algumas diferenças. No jogo você controla uma nave espacial e seu objetivo é coletar o maior número de pontos possível até que todos os pontos do cenário sejam capturados, vence quem tiver a maior quantidade de pontos.

São permitidos até 4 jogadores simultaneamente, cada jogador tem um personagem único para a partida e os pontos são capturados pela primeira pessoa a encostar nele e só por ela. Os controles do jogo são as setas direcionais para movimentação e a tecla A para ativar o turbo que, enquanto ativo, adiciona um indicador visual na nave e aumenta a sua velocidade.

Algumas problemáticas que aparecem com frequência em jogos multijogadores são: sincronização de informações entre jogadores, diminuição da latência aparente e controle do volume de dados consumidos pelo jogo. Vamos analisar mais em detalhe cada tipo e apontar em quais funcionalidades de Bounty Hunters vemos esses desafios.

1.2.2.1 Sincronização da Informação

Manter a sincronização das informações em um jogo pode se tornar um grande desafio ao colocarmos a latência em consideração, um exemplo comum são os jogos baseados em tempo, como por exemplo em Super Smash Bros (figura 7).

Nesses casos é necessário que o contador em todos os clientes chegue a zero ao mesmo tempo. A existência de latência na comunicação entre os clientes deverá ser levada em conta para que a informação do tempo seja atualizada para todos no momento correto.

As funcionalidades do jogo relacionadas a esse problema são:

Figura 7 – Partida de Super Smash Bros. for Wii U com contador de tempo

1. Definir os personagens para os jogadores evitando duplicidade.

2. Garantir que o ponto é entregue apenas ao primeiro que capturá-lo.

1.2.2.2 Latência Aparente

Levando em consideração que latência sempre existirá, a grande preocupação dos desenvolvedores é a "esconder" para que não seja perceptível.

No jogo Halo: Reach (HALO..., 2010) da Bungie, para ativar certas habilidades, era necessária uma validação de um servidor, o que causava lag aparente. Isso ocorre pois o jogador tem que esperar até que a resposta do servidor seja recebida para que enfim sua habilidade seja ativada.

A solução encontrada para que não se percebesse essa latência foi simplesmente acrescentar animações (ALDRIDGE, 2011). Enquanto a pessoa jogando vê essa animação, por baixo dos panos a troca de mensagens com o servidor pode acontecer sem causar frustração.

As funcionalidades do jogo relacionadas a esse problema são:

1. Movimentar as naves dos jogadores conectados de forma fluída e contínua.

2. Validar se um ponto pego deveria ser recebido sem que a pessoa jogando perceba.

1.2.2.3 Volume de Banda

Volume de banda é uma preocupação transversal a todos os jogos conectados a Internet. Afim de tornar o tráfego mais rápido e menos custoso é importante ser consciente quanto à quantidade e tamanho das mensagens enviadas.

As funcionalidades do jogo relacionadas a esse problema são:

1. Demonstrar os efeitos do turbo dos jogadores conectados sem necessitar de informações extras.

1.2.3 Tecnologia

Para um foco maior no problema em questão - a latência em jogos online multijogador - foi utilizado uma ferramenta para desenvolvimento de jogos, ou "engine" para jogos, chamada Unity3D (UNITY,), é possível ver o ambiente de desenvolvimento na figura 8.

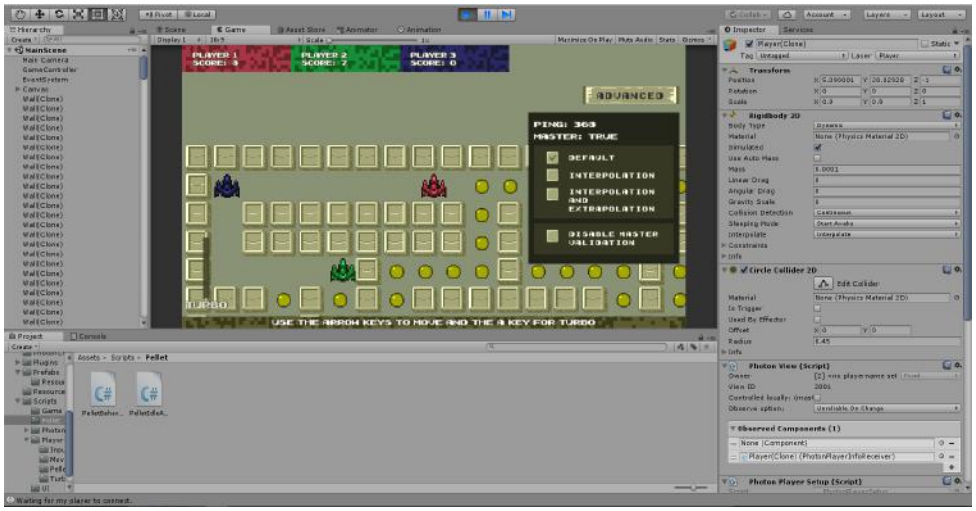

Figura 8 – Editor da Unity com o jogo executando

A Unity3D encapsula muitos dos desafios no desenvolvimento de jogos, como por exemplo a parte gráfica, interface com periféricos e até mesmo distribuição do produto final.

A linguagem de programação escolhida foi C# (MICROSOFT,), a ferramenta conta com uma versão da linguagem para ser utilizada dentro dela.

A biblioteca Photon Unity Networking (PHOTON,) foi utilizada para gerenciamento de rede. A maior vantagem de seu uso é a abstração da lógica de baixo nível de conexão e envio de pacotes na rede.

1.2.4 Implementações Adicionais

A construção do jogo demandou desenvolvimento adicional para se ter uma aplicação completa. O primeiro ponto que podemos destacar é a interface gráfica e menus de navegação.

A biblioteca Photon Unity Networking requereu configuração e a implementação de lógica para inicialização de uma sala virtual do jogo, conexão a essa sala e sincronização de dados dos objetos conectados.

O protocolo de comunicação configurado foi UDP (HOFFMAN, 2017), pois durante o jogo são enviados muitos pacotes e o custo de tempo adicional de se utilizar o protocolo

TCP poderia influenciar na latência. Além de que se um pacote tiver algum problema durante o envio, é preferível enviar outra informação mais atualizada do que tentar recuperar esse pacote.

Além disso, para facilitar os testes do jogo, foram desenvolvidas funcionalidades para a depuração e análise em tempo de execução, como controle das regras de tratamento de latência, indicativo da latência atual na tela e simulação de latência.

2 ESTRUTURAS DE COMUNICAÇÃO

2.1 TIPOS BÁSICOS DE ESTRUTURA

Existem diversas formas de se organizar a comunicação e validação de informações na arquitetura de um jogo. A estrutura escolhida afeta fortemente o impacto da latência para cada ação dentro do jogo, podendo assim dificultar ou facilitar um determinado problema (DONGEN, 2014).

Vamos explorar algumas estruturas básicas para comunicação. É importante ressaltar que cada estrutura tem suas vantagens e desvantagens e, dependendo do contexto, algumas podem ser mais adequadas que outras.

Vamos usar um cenário de duas personagens: A e B. As personagens se movem somente 1 unidade de distância por comando e sua animação demora 200 milissegundos para terminar, só depois é possível se movimentar novamente. O objetivo é chegar até uma moeda a 2 unidades de distância de ambas.

Para os exemplos abaixo não estamos considerando o esforço ou o custo para o desenvolvimento das estruturas de comunicação, mas somente o tempo de tráfego na rede dos comandos do jogo.

2.1.1 Cliente-Servidor com Servidor Dedicado

No modelo Cliente-Servidor (figura 9) a comunicação entre clientes passa primeiro por um servidor (SILVEIRA,), neste caso um servidor dedicado mantido pela equipe de desenvolvimento. Este é responsável pela manutenção do estado do jogo, validações e sincronização de informação. Muitos jogos são feitos dessa forma, podemos citar o Counter-Strike: Global Offensive (ENTERTAINMENT,), nesse caso o processamento dos ataques dos jogadores é feito em um servidor dedicado.

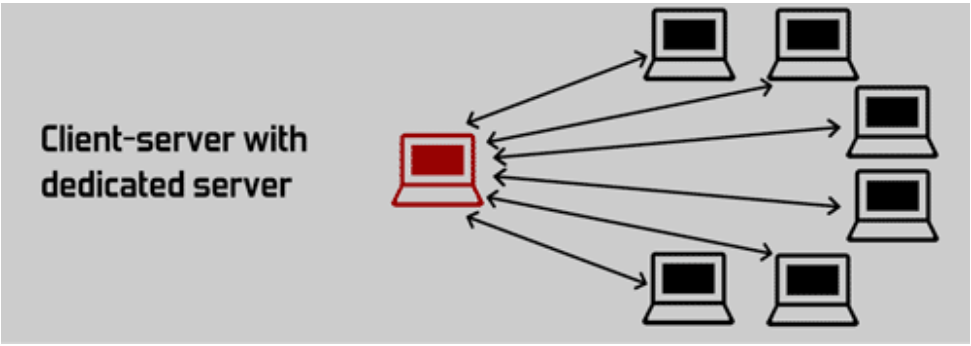

Figura 9 – Esquema Cliente-Servidor

Uma das principais vantagens de se manter um estado verdadeiro único em um servidor controlado é uma maior facilidade na prevenção de alterações indesejadas. Caso alguma

mensagem alterada seja enviada, o servidor tem a possibilidade de identificar e impedir a ação de ser executada (SILVEIRA,), podendo inclusive expulsar do jogo a pessoa responsável.

Outra vantagem do modelo é a facilidade na resolução de conflitos e problemas de sincronia. Como existe um estado verdadeiro é possível usá-lo como referência para sincronizar os estados de clientes.

Imaginando o cenário das personagens A e B, seus clientes terão que se comunicar com um servidor para que consigam obter a moeda. Para qualquer cliente (C), servidor (S) e latência entre os dois (L_{CS}), a latência total (L_{TOTAL}) para que a ação de movimento seja computada é:

$$L_{TOTAL} = L_{CS}$$

Com latência entre o cliente de A e S 100 milissegundos e entre o cliente de B e S 200 milissegundos, a comunicação se dará conforme o diagrama 10.

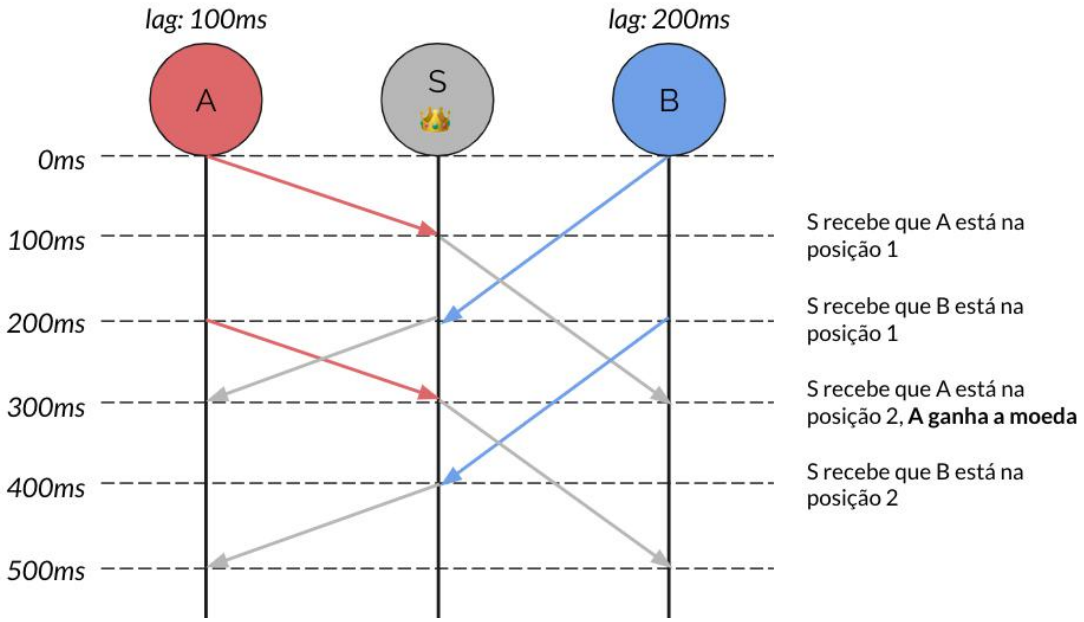

Figura 10 – A e B no modelo Cliente-Servidor

Nesse caso, A chega primeiro na posição 2 e, como o servidor tem o verdadeiro estado do jogo, A conseguirá a moeda. Qualquer inconsistência de estado poderá ser facilmente revertida sabendo que existe um único estado correto.

Contudo, exige que as mensagens passem por um ponto para serem encaminhadas para os outros jogadores, o que adiciona latência nos processamentos do jogo.

Outro ponto de atenção é a necessidade de manutenção do servidor, podendo representar um custo muito alto para as pessoas desenvolvedoras do jogo dependendo do contexto da equipe.

2.1.2 Cliente-Servidor com Servidor no Cliente

Muito similar ao modelo Cliente-Servidor com servidor dedicado, a diferença aqui é que quem representa o servidor é um dos clientes conectados e seu estado é tratado como o verdadeiro durante o jogo (figura 11). Um jogo muito conhecido com essa arquitetura para seu modo multijogador é Diablo II (VALVE,), em que os jogadores que criam as sessões do jogo assumem o papel de servidor.

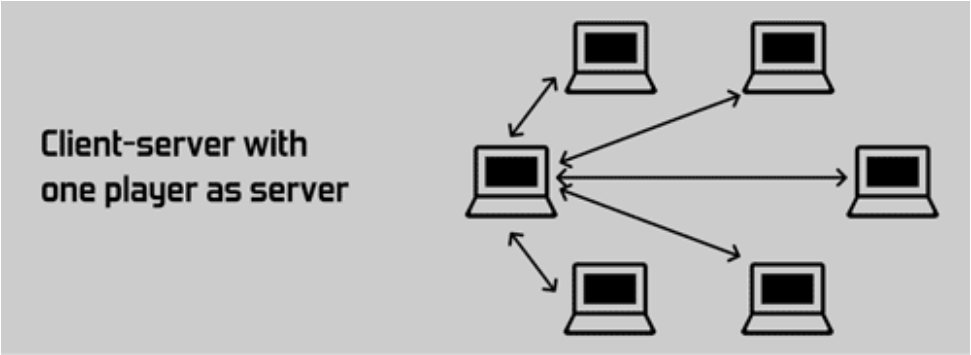

Figura 11 – Esquema Cliente-Servidor com servidor em um cliente

Não ter custo de manutenção e seguir com um estado único verdadeiro são algumas das vantagens de um de termos um "cliente servidor", no entanto, a facilidade de se prevenir trapaças diminui já que as validações são feitas numa máquina fora do controle do time de desenvolvimento.

Como toda comunicação trafega pelo cliente servidor algumas preocupações a mais surgem para o desenvolvimento, principalmente o consumo de banda e o impacto que a velocidade e qualidade de conexão desse cliente tem para os outros.

Em casos em que o próprio cliente servidor necessite de alguma validação, o mesmo não envia mensagens na rede, ou seja não existe latência. Imaginando o cenário das personagens A e B, em que A é o cliente servidor ($L_{AA} = 0$) e B tem uma latência 100 milissegundos até A ($L_{BA} = 100$), a troca de mensagens acontecerá de acordo com o diagrama 12.

Nesse cenário A consegue a moeda, pois não existe latência para se comunicar com ele mesmo, enquanto B não a alcança. Isso demonstra uma das problemáticas desse modelo, o cliente que funciona como servidor pode ter uma grande vantagem em comparação com os outros.

2.1.3 Peer to Peer

No modelo Peer to Peer (figura 13), não existe um único servidor ou cliente que é considerado o verdadeiro, nesse modelo todos clientes se comunicam entre si e cada um considera a sua versão do jogo como verdadeira (SILVEIRA,). Como exemplo pode-

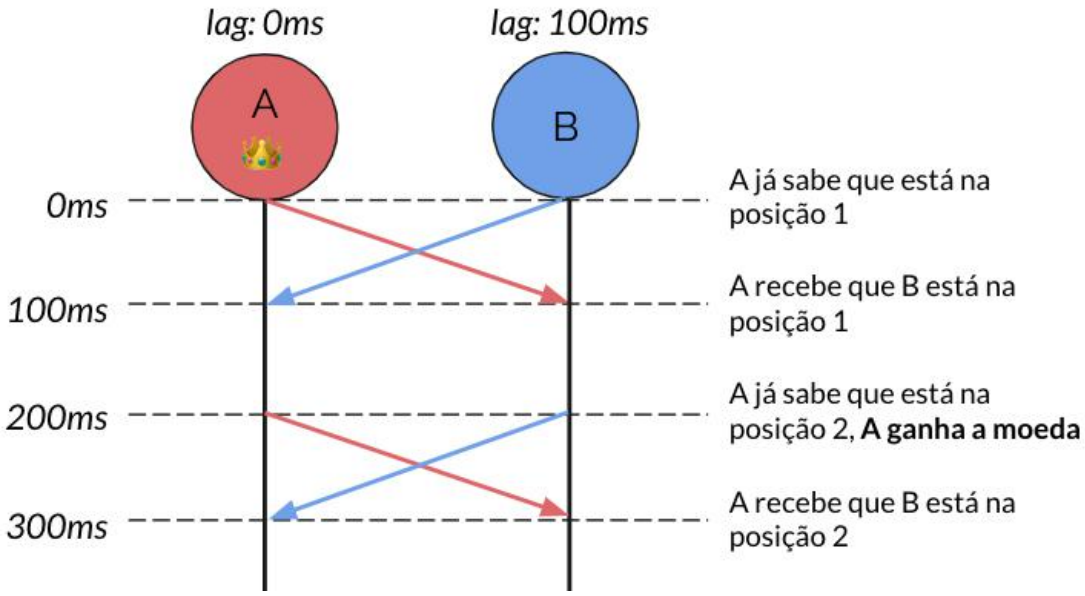

Figura 12 – A e B no modelo Cliente-Servidor com validação no cliente

mos citar Awesomenauts (GAMES,), em que todas as validações são feitas nos próprios clientes.

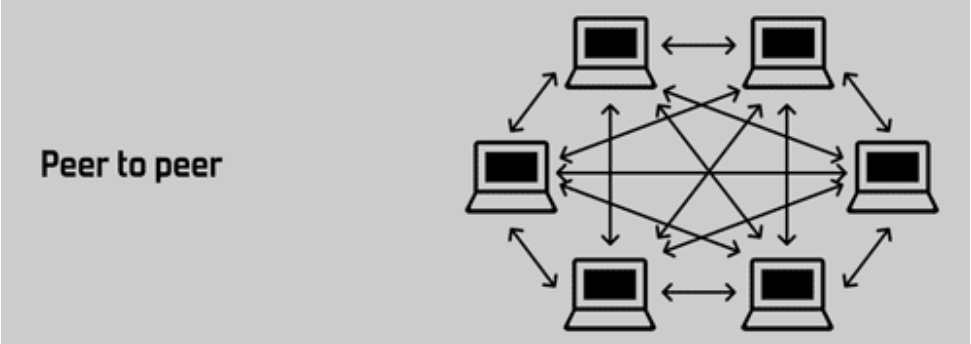

Figura 13 – Esquema Peer to Peer

A principal vantagem desse modelo é a ausência do efeito da latência, como as validações são feitas no próprio cliente não existe latência entre ele e o servidor. Outra é não necessitar de servidores dedicados, diminuindo eventuais custos de projeto.

Como todos os clientes consideram seu próprio estado como verdadeiro, a sincronização se torna um problema complexo, um exemplo disso será ilustrado com o cenário das personagens A e B.

Além disso, há a necessidade de propagar os comandos de todos para todos na ausência de um centralizador de informações, representando um aumento do uso de banda para todas as pessoas jogando.

No caso das personagens A e B, vamos adicionar mais um cliente C ao jogo, porém somente como observador. Nesse momento é importante ressaltar que dado dois clientes,

a latência entre eles é a mesma a partir de qualquer um dos clientes, em nosso exemplo teremos que $L_{AB} = L_{BA}$, $L_{AC} = L_{CA}$ e $L_{BC} = L_{CB}$. O diagrama 14 assume $L_{AB} = 100$, $L_{AC} = 200$ e $L_{BC} = 300$.

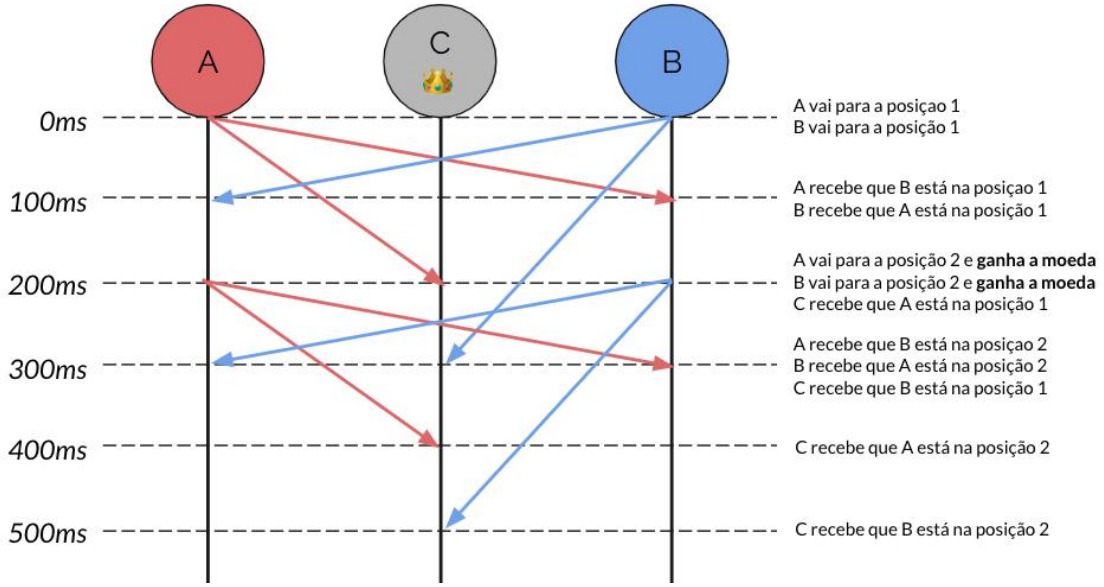

Figura 14 – A e B no modelo Peer to Peer

Observando o diagrama, em 0,2s B consegue pegar a moeda, já que só descobre que A estava na mesma posição em 0,3s, como esse fenômeno acontece com os dois clientes, ambos pegarão a moeda. Para o cliente C o comportamento é mais curioso, nesse caso ele conseguirá ver que A alcança a moeda antes de B, porém os dois receberão pontos por isso.

É fácil perceber as possíveis complexidades que se adicionam ao trabalhar com a estrutura Peer to Peer, porém a vantagem de não precisar se preocupar com a latência para um servidor de jogo é bastante valiosa.

2.1.4 Estruturas Híbridas

Uma abordagem interessante é usar de estruturas diferentes para ações e momentos diferentes dentro de um jogo. Em um jogo com conexão Peer to Peer é possível ter comandos que necessitem de validação de um outro cliente que funciona como servidor enquanto outros podem ser validados na própria máquina cliente.

O importante é entender as vantagens e desvantagens de cada modelo, pensar em qual se aplica mais ao contexto do jogo sendo desenvolvido, e escolher a opção que faça mais sentido.

2.2 EM BOUNTY HUNTERS

No jogo optamos por utilizar a estrutura Peer to Peer, não ser necessário manter um servidor dedicado foi o principal motivo para essa escolha. A comunicação entre clientes se dá através de um servidor de mensagens que conecta os clientes entre si, isso é algo da própria biblioteca escolhida e não altera como a comunicação se dá, porém aumenta a latência total da troca de mensagens.

Apesar da estrutura principal ser Peer to Peer, em alguns momentos durante o fluxo do jogo consideramos um determinado cliente, o primeiro que entrou na partida, como o cliente servidor. Isso foi feito para garantir que a captura de pontos e a seleção das personagens funcionasse corretamente.

No próximo capítulo vamos entrar em mais detalhes sobre as técnicas implementadas para resolver os problemas encontrados no desenvolvimento das funcionalidades do jogo.

3 TÉCNICAS DE TRATAMENTO DE LATÊNCIA

3.1 TÉCNICAS APLICADAS

3.1.1 Personagens sem Duplicidade

A seleção de personagens é feita de forma automática para simplificar o jogo, caso contrário seria necessário implementar mais funcionalidades que não tem relação com latência, por exemplo construir uma tela de escolha de personagens. O objetivo dessa funcionalidade era impedir que duas pessoas jogando estivessem com a mesma personagem ao mesmo tempo na mesma partida.

Uma alternativa para implementação era usar o estado do próprio cliente para determinar qual personagem deveria ser selecionado. Isso requer que cada cliente tenha conhecimento de todas as personagens do jogo e de quais estão selecionadas no momento.

A lista de personagens disponíveis no jogo é um dado imutável durante a execução e todo cliente tem acesso; contudo, para saber quais personagens estão selecionadas, é necessária a comunicação com outros clientes. Como cada um seleciona sua própria personagem, deve-se enviar essa informação para os outros clientes para que as tenham atualizada.

Por essa dependência, a seleção de personagens fica sensível aos efeitos da latência. Vamos imaginar dois novos clientes, A e B, se conectando a uma partida com um cliente servidor S. Considerando $L_{AS} = 100$, $L_{AB} = 300$ e $L_{BS} = 200$, temos o diagrama 15.

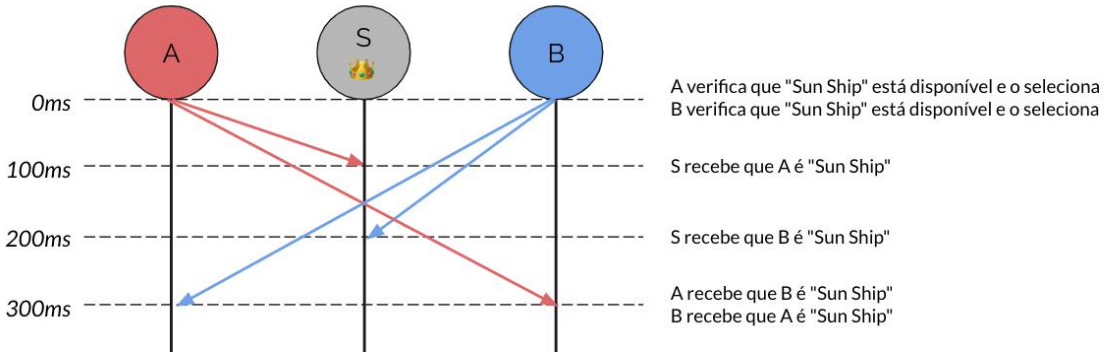

Figura 15 – Diagrama de comunicação entre clientes

Sabendo que A e B se conectam ao mesmo tempo e tem as mesmas informações a disposição, os dois selecionam a mesma personagem. Isso significa que não se pode confiar no estado do próprio cliente, pois esse pode estar desatualizado.

Para solucionar esse problema foi utilizada uma abordagem de validação no formato cliente-servidor. Agora, quando um novo cliente se conecta, é solicitada a seleção de uma

personagem para o cliente servidor, considerando as mesmas latências temos o diagrama 16.

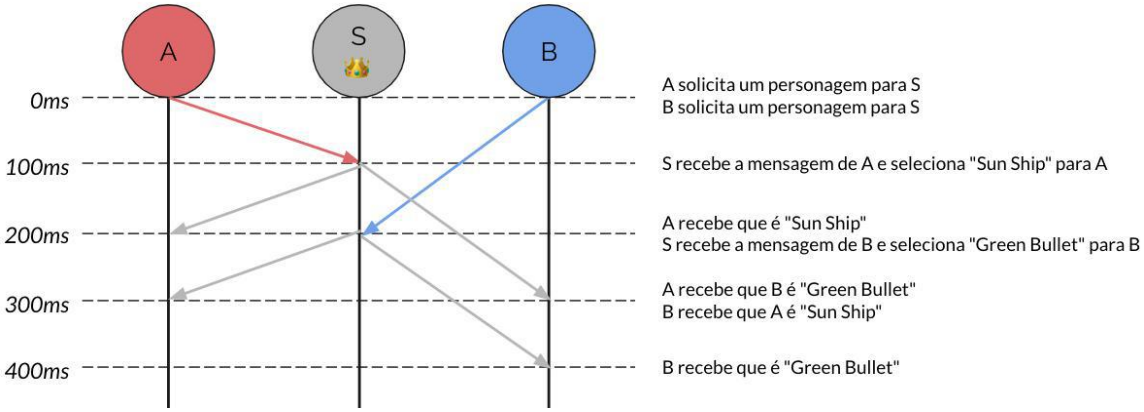

Figura 16 – Diagrama de comunicação entre clientes

Com o controle da seleção sendo feito em um único cliente, conseguimos garantir que cada jogador estará com uma personagem diferente. No momento em que S recebe a solicitação de A, "Sun Ship" está disponível e S a seleciona para A, quando a mensagem de B chega a nave não está mais disponível e outra é selecionada.

O lado negativo aqui é o acréscimo de tempo de espera para quem está jogando entre a conexão e o início da partida. Isso ocorre pois a partida só começa depois que a personagem de todos jogadores foi selecionada. Dado um cliente C e servidor S, o tempo total para que o jogo comece pode ser expressado por:

$$L_{TOTAL} = 2 * LCS$$

Esse aumento de tempo pode gerar frustração aos jogadores, por isso, visando melhorar a experiência, foi adicionada uma tela de espera enquanto esse processo acontece.

3.1.2 Movimentação

Movimentação é uma das partes mais importantes para a experiência de jogar Bounty Hunters. Tratamos aqui especificamente da movimentação das personagens conectadas, ou seja, da visualização das atualizações de posição de jogadores em um cliente.

De tempos em tempos um cliente recebe atualizações da posição dos outros jogadores e deve exibir isso a quem está jogando. Saber a exata posição de seus adversários é importante, erros como visualizar a personagem adversária numa direção incorreta ou "saltando" - se movimentando sem fluidez - atrapalham e podem influenciar o resultado de uma partida.

Pensando nesses objetivos, foram construídas três diferentes formas de se sincronizar esses dados, a fim de verificar qual teria o melhor resultado, essas são: atualização direta, interpolação e extrapolação.

3.1.2.1 Atualização Direta

Atualizar diretamente significa que, assim que uma atualização de um outro jogador é recebida, a posição da personagem é alterada imediatamente.

Primeiramente, foi necessário criar um serializador e um de-serializador de dados específicos para movimentação. Esses componentes utilizam a biblioteca Photon Unity Networking para transmitir e receber as informações das pessoas jogando.

Nesse momento só trabalhamos com o envio da posição atual do jogador, representada por um vetor de dois valores, o ponto no eixo horizontal, ou "x", e o ponto no eixo vertical, ou "y".

O primeiro problema observado foi que a movimentação da personagem conectada não estava fluída. Isso ocorre pois existe um tempo "T" entre o envio de uma atualização de posição e a próxima, fazendo com que a personagem pareça que está "saltando" de um ponto ao outro.

Foi identificado também que a posição de um jogador em um cliente sempre estava atrasada em relação ao cliente original. Como existe latência entre a comunicação dos clientes, a mensagem só chega um tempo "L" depois que foi enviada e, enquanto isso, o jogador continua se movendo no cliente original, gerando essa inconsistência.

Imaginando dois clientes A e B, podemos ver no diagrama abaixo a representação dos efeitos dos tempos "T" e "L" (17) na sincronização da posição do jogador B (1 e 2).

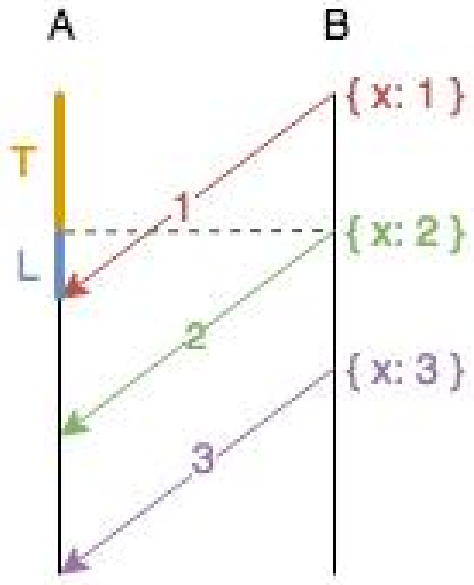

Figura 17 – Diagrama representando as atualizações de posição entre clientes A e B

Tabela 1 – Descrição das ações da imagem 3.5

Ação	Descrição
1	B envia a A que está na posição {x: 1}
2	B envia a A que está na posição {x: 2}
3	B envia a A que está na posição {x: 3}

Tabela 2 – Descrição por tempo da imagem 3.5

Tempo	Estado em A	Estado em B
0,0s	x: 0	x: 1
T + L	x: 1	x: 2
2*T + L	x: 2	x: 3
3*T + L	x: 3	x: 3

Observando os diagramas, é perceptível a influência da latência e do tempo entre atualizações na dessincronização da posição. Vale ressaltar que, enquanto para A o valor recebido é alterado bruscamente, de 1 para 2 por exemplo, no cliente B essa mudança pode ser gradual no tempo

3.1.2.2 Interpolação

O objetivo da interpolação (GAMBETTA,) é resolver o primeiro problema observado na atualização direta. O algoritmo consiste em alterar a posição da personagem conectada gradualmente pelo tempo enquanto uma nova mensagem com a posição mais atualizada não é recebida.

Seguindo essa estratégia a experiência de jogo melhora consideravelmente, já que os jogadores poderão ver seus adversários se movendo de maneira fluída e consistente. No entanto, as posições no cliente original e nos outros ainda não estarão sincronizadas.

Para essa implementação foram necessárias cinco variáveis muito importantes: a posição atualizada da personagem conectada (d'), a posição atual da personagem nesse cliente (d), o tempo entre uma atualização e outra (delta), um contador de tempo (t) e o momento da última atualização (ultima).

Antes de seguir, precisamos entrar em alguns detalhes sobre o funcionamento da Unity. Cada componente da ferramenta pode implementar um método chamado Update que é executado de tempos em tempos automaticamente, parte da lógica de nosso algoritmo reside nesse método. O restante está em um método invocado pela Photon Unity Networking quando recebemos uma mensagem com a posição atualizada.

Também utilizamos de três funcionalidades que a Unity provê: o momento no tempo atual, o tempo entre execuções do método Update e uma função que, baseada em uma porcentagem, retorna um valor dentro de um intervalo.

Abaixo podemos ver um trecho de pseudocódigo (3.1) que ilustra o funcionamento do algoritmo de interpolação. Os nomes das dependências foram alterados para melhor

Código 3.1 – Algoritmo de interpolação em pseudocódigo

```
var d;
var d';
var t;
var delta;
var ultima;

// Metodo executado periodicamente
function atualizar() {
    t += tempoPassado();
    // tempoPassado nos retorna o tempo entre uma chamada
    // ao metodo atualizar e outra

    posicao = gradual(d, d', t/delta)
    // gradual retorna o ponto entre d e d' referente
    // a porcentagem t/delta atualizamos a posicao
    // da personagem com esse valor
}

// Metodo executado quando uma atualizacao chega
// Recebe a nova posicao da personagem
function posicaoRecebida(novaPosicao) {
    d = d';
    d' = novaPosicao;

    delta = agora() - ultima;
    ultima = agora();
    // agora nos retorna o tempo atual no momento da execucao
    t = 0;
}
```

entendimento.

Agora com isso conseguimos atualizar a posição da personagem em um cliente de forma gradual, utilizando o tempo real entre uma mensagem e outra para regular a velocidade do movimento.

3.1.2.3 Extrapolação

O segundo desafio relacionado a movimentação é o da consistência das posições entre clientes distintos. Com a interpolação ganhamos suavidade mas, devido à latência, a posição de uma pessoa jogando em seu cliente sempre estará a frente da posição em outro cliente, e uma forma de se amenizar isso é com uso de extrapolação.

Para fazermos extrapolação utilizamos das mesmas variáveis da interpolação, porém com uma alteração em como é calculado d' (3.2). Ao invés de somente armazenar a

posição recebida, utilizamos também da informação da velocidade para estimar a posição atual do jogador.

Código 3.2 – Algoritmo de extrapolação em pseudocódigo

```
// Recebe a nova posicao e a velocidade da personagem
function posicaoVelocidadeRecebida(novaPosicao, velocidade) {
    delta = agora() - ultima;
    ultima = agora();
    t = 0;

    d = d';
    d' = novaPosicao + velocidade*delta;
}
```

Com essa alteração na lógica estimamos o quanto a personagem se moveu no cliente original multiplicando a velocidade que ela estava pelo tempo que se passou até a atualização ser recebida.

Extrapolação muitas vezes pode causar inconsistências entre os clientes, então é muito importante ter cuidado ao fazer a estimativa, para que o efeito não seja o oposto do desejado e amplifique as discrepâncias entre os clientes.

3.1.3 Captura de Pontos

Muito similar ao problema de personagens duplicados. Na captura de pontos, se levarmos em consideração o estado do próprio cliente para validar se um ponto pode ser capturado ou não, é possível que dois jogadores peguem o mesmo ponto.

Seguindo a mesma solução que foi usada para o outro problema, foi adicionado um fluxo de validação com o cliente servidor. Ao encostar em um ponto é enviada uma mensagem ao cliente servidor para que se confirme se o ponto foi capturado ou não, caso a resposta seja sim, mais mensagens são enviadas para que a pontuação seja atualizada, caso seja não, nada acontece.

Vamos simular um cenário de dois clientes A e B com $L_{AB} = 200$, onde ambos passam pelo mesmo ponto no segundo 0,1s. Como podemos ver na figura 18, quando não existe validação ambos os clientes capturam seus pontos e enviam confirmações para outros clientes.

Com a intervenção de um cliente servidor S, esse cenário não acontece. Considerando $L_{AS} = L_{BS} = 200$ temos o diagrama 19.

Contudo, o tempo para que a pontuação seja atualizada aumenta consideravelmente. No primeiro cenário que imaginamos o tempo inicial para atualização de pontos era próximo ao valor da latência, com a necessidade de validação esse tempo dobra. Isso tem um

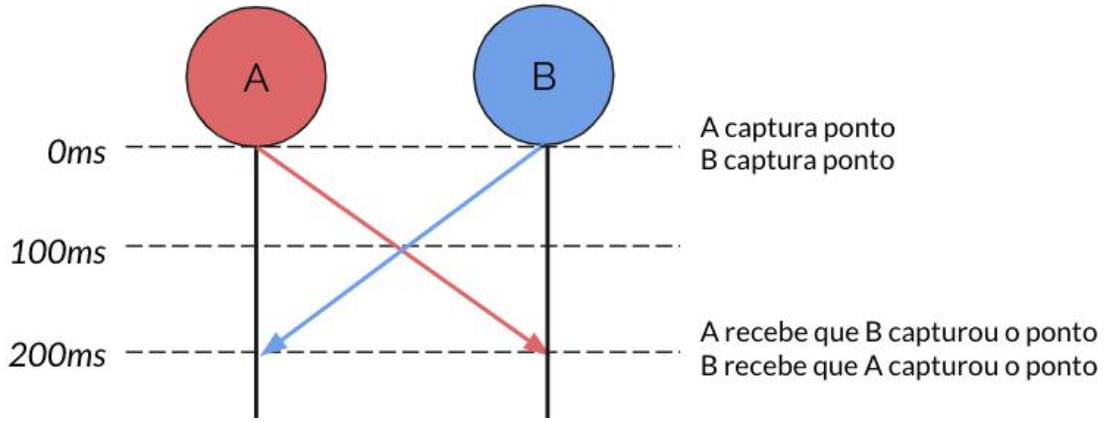

Figura 18 – Cenário sem intervenção de cliente servidor

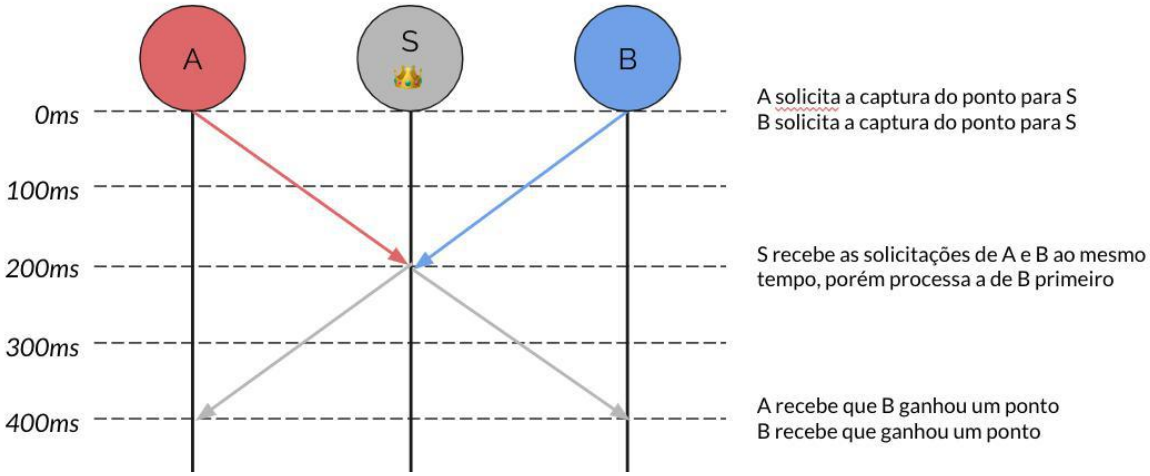

Figura 19 – Cenário com intervenção de cliente servidor

impacto muito grande na experiência do jogo, pois ao encostar em um ponto é esperado que ele desapareça imediatamente.

Para solucionar esse problema, aprendemos com a experiência dos desenvolvedores de Halo: Reach (HALO..., 2010) já citada anteriormente e adicionamos uma animação ao objeto de ponto no momento em que uma personagem passa por ele. O objetivo disso foi adicionar um "tempo" ao processo de captura de ponto para que as mensagens de validação pudessem ser trocadas, camuflando os efeitos da latência.

3.1.4 Representação do Turbo

A representação do turbo significa que, se um jogador em seu cliente ativa o turbo, todos os outros clientes devem conseguir ver que a habilidade está sendo usada por esse jogador.

A primeira solução pensada para essa questão foi o envio de uma nova informação para

indicar se estava ativado ou não, porém isso adiciona mais um dado às mensagens entre clientes. Cada mensagem de atualização de posição é composta por dois vetores de 8 bytes representando a posição e velocidade, totalizando 16 bytes de tamanho. Ao adicionar um valor booleano, de 1 byte, representando o uso do turbo, teríamos um aumento de 6.25%.

É sempre interessante tentar limitar ou diminuir a quantidade de dados trafegados dentro de um jogo para que o seu consumo de Internet não seja grande, como a própria documentação da framework (Photon Engine) usada no projeto recomenda (DOCUMENTATION,).

Analisando os dados que já estavam sendo enviados nas mensagens, concluímos que era possível utilizar a velocidade atual da personagem para inferir o uso do turbo.

Como os valores das velocidades, com e sem turbo, são conhecidas por todos os clientes, comparamos o valor da velocidade recebida na atualização da posição com a que se tem no cliente. Se os valores forem próximos, podemos considerar que o turbo está ativado, pois de nenhuma outra forma a personagem alcançaria essa velocidade.

4 CONCLUSÃO

4.1 RELEVÂNCIA PARA AS APLICAÇÕES

Utilizando Bounty Hunter como objeto de análise podemos observar como as técnicas para mitigar os efeitos da latência são necessárias de diversas maneiras e em diferentes partes do jogo.

Sem o uso dessas técnicas o jogo teria problemas de consistência, não seria possível a implementação de algumas funcionalidades e a experiência do jogador seria frustrante. Considerando que, para empresas de jogos, uma nova instalação ou nova compra são vitais, esses pontos não podem ser desconsiderados.

Vale lembrar que o jogo não para de funcionar quando a latência é muito alta, mas a jogabilidade pode se tornar inviável, principalmente para jogos síncronos onde o reflexo e tempo de reação são importantes.

Quando a latência é muito pequena (menor que 25ms) os tratamentos de latência não são tão necessários porque a troca de informações é rápida o suficiente para manter o jogo fluido, mas se torna cada vez mais necessário em latências mais altas. O problema quando dois jogadores tentam capturar o mesmo ponto em um curto espaço de tempo é muito fácil de reproduzir em uma latência muito alta, por exemplo 500ms, pois é apenas necessário que ambos os jogadores capturem o ponto antes de receber a informação de que o outro capturou.

Por ultimo é importante lembrar que, com a ascensão de jogos multijogadores conectados para celular, essa necessidade se torna cada vez mais presente. Nesse cenário temos que trabalhar com processamento e qualidade de rede inferiores aos que vemos em computadores e consoles e ainda assim prover a melhor experiência possível para os jogadores.

4.2 PRÓXIMOS PASSOS

Ainda com Bounty Hunters como exemplo, é possível ver as distintas formas que as técnicas de mitigação podem assumir, indo de adicionar animações até alterar como a comunicação entre clientes ocorre. O importante é ter em mente o objetivo final de aplicá-las, melhorar a experiência de um jogo.

Um aprendizado importante foi de sempre trabalhar nessas soluções de forma iterativa. Inicialmente as melhoras podem ser poucas ou não resolvem o problema por completo, porém, como podemos ver no caso da movimentação, ao iterar sobre a solução obtemos grandes resultados.

Um problema conhecido para ser corrigido é a perda de informação quando o cliente

servidor se desconecta da partida. Quando isso ocorre algumas informações da pontuação se perdem enquanto um novo cliente servidor é selecionado.

Quanto a melhorias, uma dívida técnica no projeto é o isolamento de todos os comportamentos da biblioteca Photon Unity Networking. Uma boa prática conhecida é isolar comportamentos de bibliotecas em objetos específicos para que, se for um desejo mudar a biblioteca usada, sejam necessários menos passos, além de deixar mais fácil de identificar as lógicas específicas do jogo.

Outro ponto importante a ser considerado é a avaliação de otimizações na estratégia para movimentação, por exemplo, ao invés de mensagens serem enviadas a partir de mudanças na posição, poderiam ser a partir de mudanças na velocidade? A ideia seria medir quanto essa mudança impacta nas inconsistências entre clientes e na taxa de envio de mensagens.

Mais um estudo possível é a comparação da performance e experiência do jogo se alterarmos o protocolo de comunicação escolhido de UDP para TCP. A ideia seria entender como as diferenças entre os protocolos afetam ou poderiam afetar o jogo em diferentes cenários.

REFERÊNCIAS

ALDRIDGE, D. **I Shot You First: Networking the Gameplay of HALO: REACH**. 2011. <https://www.gdcvault.com/play/1014345/I-Shot-You-First-Networking/>.

AXISROB. **Discussão sobre valores aceitáveis de latência em jogos**. <https://forum.unity.com/threads/question-about-acceptable-levels-of-latency-in-online-gaming.261271/>.

CLASH of Clans. 2012. <https://supercell.com/en/games/clashofclans/>.

CLASH of Clans Android Page. <https://play.google.com/store/apps/details?id=com.supercell.clashofclans>.

CLASH Royale. 2016. <https://clashroyale.com/>.

CLASH Royale Android Page. <https://play.google.com/store/apps/details?id=com.supercell.clashroyale&hl=pt_BR>.

DOCUMENTATION, P. E. **Performance Tips - Photon Engine**. <https://doc.photonengine.com/en-us/realtime/current/reference/performance-tips>.

DONGEN, J. van. **Core network structures for games**. 2014. <http://joostdevblog.blogspot.com/2014/09/core-network-structures-for-games.html>.

DONGEN, J. van. **The many meanings of the confusing word 'lag'**. 2017. <http://joostdevblog.blogspot.com/2017/01/the-many-meanings-of-confusing-word-lag.html>.

DRAW Something. 2012. <https://www.zynga.com/games/draw-something>.

DRAW Something Android Page. <https://play.google.com/store/apps/details?id=com.omgpop.dstfree&hl=pt_BR>.

ENTERTAINMENT, B. **Página de Diablo II**. <http://us.blizzard.com/pt-br/games/d2/>.

GAMBETTA, G. **Fast-Paced Multiplayer (Part III): Entity Interpolation**. <http://www.gabrielgambetta.com/entity-interpolation.html>.

GAMES, R. **Página de Awesomenauts**. <https://www.awesomenauts.com/>.

GLAZER, J. **Multiplayer Game Programming: An Overview of Networked Games**. <http://www.informit.com/articles/article.aspx?p=2461064>.

HALO Reach. 2010. <https://halo.bungie.net/projects/reach/>.

HOFFMAN, C. **What's the Difference Between TCP and UDP?** 2017. <goo.gl/MhnM2B>.

LEAGUE of Legends. 2009. <https://play.br.leagueoflegends.com/pt_BR>.

MICROSOFT. **Página da documentação do C#**. <https://docs.microsoft.com/pt-br/dotnet/csharp/>.

MINECRAFT. 2009. <https://minecraft.net/pt-br/>.

MINECRAFT number of sales worldwide. <https://www.statista.com/statistics/680124/minecraft-unit-sales-worldwide/>.

NEWZOO. **Mercado de jogos global chega a 137.9 bilhões de dólares**. <https://newzoo.com/insights/articles/global-games-market-reaches-137-9-billion-in-2018-mobile-games-take-half/>.

PHOTON. **Página da biblioteca Photon Unity Networking**. <https://www.photonengine.com/pun>.

PáGINA do jogo Pac Man na Wikipédia. <https://pt.wikipedia.org/wiki/Pac-Man>.

SANTOS, B. F. **Apesar de expansão, acesso à internet no Brasil ainda é baixo**. 2016. <https://exame.abril.com.br/brasil/apesar-de-expansao-acesso-a-internet-no-brasil-ainda-e-baixo/>.

SILVEIRA, R. **Getting Started - Multiplayer Game Programming**. <https://hub.packtpub.com/getting-started-multiplayer-game-programming/>.

TASSI, P. **Riot Games Reveals 'League of Legends' Has 100 Million Monthly Players**. 2016. <https://www.forbes.com/sites/insertcoin/2016/09/13/riot-games-reveals-league-of-legends-has-100-million-monthly-players/#6cfa41b25aa8>.

UNITY. **Página da Unity3D**. <https://unity3d.com/pt>.

VALVE. **Página de CS: GO**. <https://blog.counter-strike.net/>.

WEPC. **2019 Video Game Statistics**. <https://www.wepc.com/news/video-game-statistics/#online-gaming>.