MARCELO TETSUHIRO SADAIKE

**MELHORIA DO TEMPO DE RESPOSTA PARA EXECUÇÃO DE JOGOS EM UM
SISTEMA EM CLOUD GAMING COM IMPLEMENTAÇÃO DE CAMADAS E
PREDIÇÃO DE MOVIMENTO**

São Paulo

2017

MARCELO TETSUHIRO SADAIKE

**MELHORIA DO TEMPO DE RESPOSTA PARA EXECUÇÃO DE JOGOS EM UM
SISTEMA EM CLOUD GAMING COM IMPLEMENTAÇÃO DE CAMADAS E
PREDIÇÃO DE MOVIMENTO**

Dissertação apresentada à Escola
Politécnica da Universidade de São Paulo
para obtenção do Título de Mestre em
Ciências.

São Paulo

2017

MARCELO TETSUHIRO SADAIKE

**MELHORIA DO TEMPO DE RESPOSTA PARA EXECUÇÃO DE JOGOS EM UM
SISTEMA EM CLOUD GAMING COM IMPLEMENTAÇÃO DE CAMADAS E
PREDIÇÃO DE MOVIMENTO**

Dissertação apresentada à Escola
Politécnica da Universidade de São Paulo
para obtenção do Título de Mestre em
Ciências.

Área de Concentração:
Engenharia de Computação

Orientadora:
Prof^a. Dra. Graça Bressan

São Paulo

2017

Este exemplar foi revisado e alterado em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, 29 de Agosto de 2017

Assinatura do autor

Assinatura do orientador

FICHA CATALOGRÁFICA

Sadaike, Marcelo Tetsuhiro

Melhoria do tempo de resposta para execução de jogos em um sistema em Cloud Gaming com implementação de camadas e predição de movimento/ M. T. Sadaike -- versão corr. -- São Paulo, 2017.

87 p.

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.COMPUTAÇÃO EM NUVEM 2.JOGOS DE COMPUTADOR 3.FRAMEWORKS I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t.

AGRADECIMENTOS

Agradeço ao meu pai Sérgio, que sempre esteve presente, me auxiliando e contribuindo no avanço dos meus estudos. Pela compreensão e apoio nos momentos mais difíceis. Por ser um exemplo de pessoa no qual eu me inspiro muito.

Agradeço à minha mãe Célia, que sempre me ouviu e me deu conselhos. Pelo incentivo e apoio durante toda a realização desse trabalho. Educou-me durante toda a minha vida e sou a pessoa que sou, graças a ela.

Agradeço à minha irmã Letícia, que me apoiou durante todo o meu mestrado, me ajudando sempre que eu precisava.

Agradeço à minha namorada Estela, pelo suporte constante e incansável. Pela compreensão e pela cooperação ao longo de todo o mestrado.

Agradeço especialmente à minha orientadora Graça, que me auxiliou durante todo o meu mestrado, abrindo novas oportunidades e me orientando sempre. Considero um modelo de profissional e pessoa no qual eu respeito muito.

Agradeço aos meus colegas Rodolfo e José Carlos que fizeram do LARC uma segunda casa para mim, conversamos, rimos e falamos muito sobre Cuba.

Agradeço aos meus familiares e amigos que sempre se interessaram pelo meu trabalho e me deram sugestões e opiniões.

RESUMO

Com o crescimento da indústria dos jogos eletrônicos, surgem novos mercados e tecnologias. Os jogos eletrônicos da última geração exigem cada vez mais processamento e placas de vídeo mais poderosas. Uma solução que vem ganhando cada vez mais destaque é o Cloud Gaming, no qual o jogador realiza um comando e a informação é enviada e processada remotamente em uma nuvem, localizada na Internet, e que retorna com as imagens como uma sequência de vídeo para o jogador. Para melhorar a qualidade de experiência (QoE) é proposto um modelo que diminui o tempo de resposta entre o jogador e a nuvem, através de um arcabouço chamado Cloud Manager que utiliza a técnica de implementação de camadas, na camada do plano de fundo e predição de movimentos, utilizando uma matriz de predição, na camada do personagem. Para validar os resultados é utilizado um jogo de ação com ponto de vista onipresente dentro do sistema de Cloud Gaming Uniquitous.

Palavras-Chave: Computação em nuvem; Jogos; Pré-processamento; Implementação por Camadas; Predição de Movimento; Matriz de Predição.

ABSTRACT

With the growing video games industry, new markets and technologies are emerging. Electronic games of the new generation are increasingly requiring more processing and powerful video cards. The solution that is gaining more prominence is Cloud Gaming, which the player performs a command, the information is sent and processed remotely on a cloud, then the images return as a video stream back to the player using the Internet. To improve the Quality of Experience (QoE), it is proposed a model that reduces the response time between the player command and the stream of the resulting game scenes through a framework called Cloud Manager that use layer caching techniques, in the background, and future state prediction using a prediction matrix, in the character layer. To validate the results, a action game with god-view as point of view is used in a Cloud Gaming system called Uniquitous.

Keywords: Cloud Computing; Gaming; Preprocessing; Layer Caching; Movement Prediction; Prediction Matrix.

SUMÁRIO

RESUMO.....	6
ABSTRACT	7
SUMÁRIO.....	8
LISTA DE ILUSTRAÇÕES	12
LISTA DE ABREVIATURAS E SIGLAS	10
1 INTRODUÇÃO	12
1.1 MOTIVAÇÃO.....	12
1.2 DESAFIOS.....	13
1.3 OBJETIVOS	14
1.3.1 Objetivo Geral.....	14
1.3.2 Objetivos Específicos	14
1.4 METODOLOGIA	14
1.5 ORGANIZAÇÃO.....	15
2 CONCEITOS.....	16
2.1 QUALIDADE DE EXPERIÊNCIA (QoE).....	16
2.2 QUALIDADE DE SERVIÇO (QoS).....	16
2.3 JOGOS ELETRÔNICOS.....	17
2.3.1 Gêneros de Jogos eletrônicos	18
2.3.1.1 Simulação	18
2.3.1.2 Estratégia.....	19
2.3.1.3 Ação.....	20
2.3.1.4 Aventura.....	20
2.3.1.5 RPGs	20
2.3.1.6 Quebra-cabeça	21

2.3.2	Pontos de vista do jogador	21
2.3.3	Classificação pela quantidade de jogadores	23
2.3.4	Comandos do jogador	23
2.4	PREDIÇÃO DE MOVIMENTO	23
2.5	COMPUTAÇÃO EM NUVEM	25
2.5.1	Modelos de serviços de Computação em nuvem	26
2.5.2	Modelos de implementação de Computação em nuvem	27
2.5.3	Segurança na nuvem.....	27
2.5.4	Escalonamento	28
2.6	CLOUD GAMING	28
2.6.1	Vídeo Streaming e Codificação na nuvem.....	30
2.6.2	Tempo de Resposta	31
2.6.3	Efeito da latência e do jitter na performance dos jogadores	32
2.7	MOBILE CLOUD COMPUTING (MCC).....	34
3	TRABALHOS RELACIONADOS	36
4	SISTEMAS DE CLOUD GAMING	40
4.1	SISTEMAS DE CLOUD GAMING	40
4.2	INICIALIZAÇÃO DO SISTEMA DE CLOUD GAMING	43
4.3	ARQUITETURA DO UNIQUITOUS.....	43
4.3.1	Visão Geral de Alto Nível do Sistema.....	44
4.3.2	Projeto Unity	45
4.3.2.1	Jogo Unity	45
4.3.2.2	Janela do Jogo.....	45
4.3.2.3	Fonte de Áudio.....	45
4.3.3	Servidor Uniquitous	45
4.3.3.1	Captura de Áudio	45

4.3.3.2	Codificação e Transmissão de Áudio	46
4.3.3.3	Captura de Tela do Unity	46
4.3.3.4	Codificação de Imagem.....	47
4.3.3.5	Transmissão de Imagem.....	48
4.3.3.6	Recepção da Entrada de Dados	49
4.3.4	Cliente Uniquitous.....	49
4.3.4.1	Reprodução de Áudio	49
4.3.4.2	Recepção e Decodificação de Áudio	49
4.3.4.3	Visualização de Imagem	49
4.3.4.4	Decodificação de Imagem.....	50
4.3.4.5	Recepção de Imagens	50
4.3.4.6	Transmissão dos Dados de Entrada	50
4.3.5	Fluxo de dados	51
5	ARQUITETURA DO ARCABOUÇO E PROPOSTA DO MECANISMO DE PREDIÇÃO DE MOVIMENTO.....	52
5.1	Implementação por camadas	52
5.2	Criação da Matriz de predição	54
5.3	Predição de movimento utilizando matriz de predição	57
5.3.1	Movimento do personagem	57
5.3.2	Predição de movimento	60
5.4	Melhorias na predição de movimento.....	66
5.4.1	Condições para o pré-processamento.....	66
5.4.2	Matriz de Treinamento.....	67
6	VALIDAÇÃO E RESULTADOS	70
6.1	CONFIGURAÇÃO	71
6.2	PROJETO UNITY.....	71

6.3	JANELA DE JOGO.....	71
6.4	CAPTURE DE TELA DO UNITY	72
6.5	CODIFICAÇÃO DE IMAGEM.....	72
6.6	TRANSMISSÃO DE IMAGEM.....	73
6.7	RECEPÇÃO DA ENTRADA DE DADOS	74
6.8	ANGRY BOTS.....	74
7	CONCLUSÕES	79
	REFERÊNCIAS.....	80

LISTA DE ILUSTRAÇÕES

Figura 1 - Jogo de tiro com visão em primeira pessoa.....	22
Figura 2 - Jogo de corrida com visão em terceira pessoa.....	22
Figura 3 - Jogo de MOBA com visão onipresente.....	22
Figura 4 - Jogos em nuvem padrão: o processamento do jogo dependerá da latência de rede. Adaptado de (LEE et al., 2015).	24
Figura 5 - Pré-processamento utilizando predição de movimento. (LEE et al., 2015).	24
Figura 6 - Diferença entre o Cloud Gaming e jogos online. Adaptado de (CHEN et al., 2014).	29
Figura 7 - Performance do Jogador vs latência para categorias de jogo. A faixa cinzenta horizontal representa o limiar de tolerância jogador para a latência (CLAYPOOL, 2006).....	33
Figura 8 - Arquitetura do Uniquitous com o Cloud Manager.	44
Figura 9 - Co-rotina da codificação de uma imagem JPEG.	48
Figura 10 - Arquitetura do Cloud Manager.	53
Figura 11 - Mapa do jogo com altura G e largura J.	54
Figura 12 - Representação do personagem no círculo cinza central e a distância Δ quando ele se locomove uma distância unitária em uma das quatro direções.	55
Figura 13 - Representação da matriz de predição T	56
Figura 14 - Vetor $V_{i,j}$ na coordenada cartesiana (x, y).	56
Figura 15- Inicialização da matriz T com todas as células valendo (0,0).	57
Figura 16 - Exemplo de movimento do personagem. (a) Personagem inicia no ponto (1,1). (b) Personagem moveu para direita. (c) Personagem moveu para diagonal inferior esquerda. (d) Personagem moveu para cima. (e) Personagem moveu para direita. (f) Personagem moveu para diagonal inferior direita.	59
Figura 17 – As flechas pretas representam os possíveis movimentos e as linhas pontilhadas representam a divisória para classificar o ângulo mais próximo.	62

Figura 18 - Exemplo do algoritmo de predição de movimento sendo utilizado. (a) Posição inicial do personagem. (b) Personagem moveu para direita. (c) Personagem moveu para diagonal inferior direita. (d) Personagem moveu para direita.	64
Figura 19 – União entre a matriz de predição T e a matriz de treinamento U. (a) Caminho utilizado para o treinamento. (b) Matriz de treinamento U. (c) União entre a Matriz de predição T com a Matriz de treinamento U.	68
Figura 20 - Uniquitous com o Cloud Manager.	70
Figura 21 - Imagem do jogo “Angry Bots” do gênero ação com visão onipresente. ...	75
Figura 22 - Caminho pré-definido marcado em azul, que o personagem percorre dentro do mapa.	75

LISTA DE ABREVIATURAS E SIGLAS

2D	Duas dimensões
3D	Três dimensões
API	Application Programming Interface
CPU	Unidade Central de Processamento
DLL	Dynamic-link library
GUI	Graphical User Interface
IAAS	Infraestrutura como Serviço
JPEG	Joint Photographics Experts Group
NIST	National Institute of Standards and Technology
P2P	Peer-to-peer
PAAS	Plataforma como Serviço
QoE	Qualidade de Experiência
QoS	Qualidade de Serviço
RPC	Chamada de procedimento remoto
RPG	Role-Playing Game
RTS	Estratégia em tempo real
SAAS	Software como Serviço
TA	Tempo da rede com comando
TBS	Estratégia por turnos

TCP	Protocolo de Controle de Transmissão
TJ	Tempo do jogo
TP	Tempo de processamento
TR	Tempo de resposta
TRE	Tempo de reprodução
TVS	Tempo de rede com vídeo e som
UDP	User Datagram Protocol
URL	Uniform Resource Locator

1 INTRODUÇÃO

Ao longo dos últimos quarenta anos, a indústria de jogos eletrônicos cresceu de forma constante, aumentando seu público em todo o mundo, ampliando sua demografia e adicionando novas plataformas (consoles, computadores, consoles portáteis, celulares, tablets, etc.). A oferta e a demanda mudaram de acordo com as evoluções tecnológicas das interfaces, dispositivos e redes. Em 2013, havia 1,2 bilhão de jogadores ativos em todo o mundo, com 880 jogos ativos e com as receitas previstas de US\$ 86,1 bilhões em 2016, uma taxa de crescimento anual prevista de 6,7% (WARMAN, 2013). Em outras palavras, a indústria de jogos é uma forma estabelecida de entretenimento e uma parte da cultura moderna em diversas regiões do mundo.

Os jogos modernos, no entanto, têm novas exigências de desempenho gráfico e CPU que só os computadores de ponta e consoles de última geração podem suprir. Outros dispositivos como os dispositivos móveis geralmente não têm a capacidade de executar um jogo com saída gráfica de alta qualidade.

1.1 MOTIVAÇÃO

A computação em nuvem centraliza os recursos de computação e serviços em *datacenters* espalhados pelo mundo que permitem o acesso sob demanda (ARMBRUST et al., 2010).

Desenvolvedores podem desenvolver, implementar e executar a aplicação na nuvem sem ter que se preocupar com o custo e a gestão do hardware e software necessários. As vantagens da computação em nuvem incluem redução de custos, alta disponibilidade e escalabilidade (SHIRAZ et al., 2013).

O termo Cloud Gaming tem sido definido como uma forma de serviço, no qual um jogo eletrônico, ao ser hospedado em uma nuvem, permite aos usuários usufruírem de um jogo em dispositivos heterogêneos, desde que haja acesso à Internet (HUANG et al., 2013).

Em Cloud Gaming, o processamento é realizado remotamente na nuvem e retorna imagens como uma sequência de vídeo em tempo real de volta para o jogador utilizando a Internet. O sistema deve recolher as ações de um jogador, transmiti-las

para a nuvem, processar as ações, renderizar os resultados, codificar/comprimir as alterações do jogo e transmitir o vídeo (cenas do jogo) de volta para o jogador.

Acrescenta-se também que os custos envolvidos com hardware e software para o jogador são reduzidos, comparado aos mesmos custos de jogos tradicionais, fora da nuvem. Isso significa que não é preciso ter uma cópia física do game, ou mesmo baixá-lo e instalá-lo (HUANG et al., 2013).

Ao colocar os jogos na nuvem, problemas de incompatibilidade entre hardwares e softwares são parcialmente resolvidos, não havendo a necessidade de atualizar os computadores/celulares do usuário, além de apoiar dispositivos heterogêneos, dificulta a pirataria de jogos. Assim, sistemas de jogos em nuvem podem mudar a maneira como os jogos de computador são distribuídos e jogados.

1.2 DESAFIOS

Os ambientes em nuvem apresentam grande variabilidade e imprevisibilidade ao longo do tempo, devido às diferentes cargas em que é submetida (ARMBRUST et al., 2009)(REDDY, 2011).

Devido ao fato de que a nuvem possui o conceito de elasticidade e atendimento a diversas demandas simultaneamente, diferentes resultados podem ser obtidos a partir da mesma tarefa executada em tempos distintos (ARMBRUST et al., 2009)(PHAN et al., 2011).

Há desafios ao fornecer jogos pela nuvem. Como descrito por Huang et al. (HUANG et al., 2013) usuários esperam alta qualidade de vídeos, assim como, baixo tempo de resposta que são requisitos fundamentais para uma boa QoE. A alta qualidade de vídeo e as altas taxas de transmissões de vídeo requerem uma maior largura de banda, enquanto que a experiência de jogo é altamente sensível à latência de rede.

Oferecer serviços de jogos em nuvem de uma forma comercialmente viável se mostrou uma tarefa desafiadora como demonstrado pela OnLive que sofreu dificuldades financeiras (ONLIVE, 2015). O principal desafio para os provedores de jogos em nuvem é encontrar o equilíbrio entre dois objetivos contraditórios: reduzir o investimento em *hardware* e aumentar a QoE. Para que o cliente tenha uma boa QoE, é necessário um *hardware* de última geração, e isso pode causar um gasto financeiro

enorme. Já o uso de *hardware* menos potente leva a um QoE menos agradável, e isso que pode fazer com que os jogadores optem por outras opções além dos serviços de jogos em nuvem. Além disso, diferentes gêneros de jogo exigem requisitos diferentes de *hardware*, o que pode resultar em recursos de *hardware* insuficientes ou desperdiçados se os recursos do servidor não estiverem bem planejados. Por exemplo, os servidores configurados para jogos de tiro em primeira pessoa em 3D em primeira pessoa podem ser um exagero para jogos casuais 2D. Essa diversidade faz com que encontrar o melhor balanço entre lucro e QoE fique ainda mais difícil.

1.3 OBJETIVOS

A seguir são definidos os objetivos geral e específicos deste trabalho.

1.3.1 Objetivo Geral

Esta dissertação tem como objetivo geral melhorar o tempo de resposta para execução de jogos em um sistema em Cloud Gaming. A solução se aplica a jogos do gênero plataforma com ponto de vista onipresente e devem permitir acesso ao código fonte, uma vez que o jogo deve ser desenvolvido em camadas ou ser adaptado para ser dividido em camadas.

1.3.2 Objetivos Específicos

Neste trabalho, a fim de melhorar o tempo de resposta para execução de jogos em um sistema em Cloud Gaming é proposto um arcabouço que gerencia os jogos na nuvem composto de duas principais funções: implementação de camadas e predição de movimento, utilizando uma matriz de predição.

1.4 METODOLOGIA

Como método, partiu-se primeiramente da pesquisa e posterior análise do estado da arte no tema, reunindo artigos específicos de computação em nuvem e de jogos na nuvem.

Dado este contexto como foco desta pesquisa, são adotados os seguintes passos:

- Mapeamento do estado da arte de computação em nuvem e de jogos em nuvem.
- Revisão das aplicações existentes que fornecem o serviço para jogos em nuvem.
- Mapeamento dos requisitos de um sistema em nuvem projetado para atender a jogos em nuvem.
- Projeto de um arcabouço e sua implementação de forma a atender ao requisito de melhorar o tempo de resposta para execução de jogos em um sistema em Cloud Gaming.
- Validação e análise de desempenho da proposta implementada.

1.5 ORGANIZAÇÃO

Esta dissertação apresenta no Capítulo 2 os principais conceitos relacionados ao Cloud Gaming, além do seu funcionamento e arquitetura. Posteriormente são mencionados os mecanismos fundamentais de QoE e QoS para jogos em nuvem existentes. No Capítulo 3 são apresentados os trabalhos relacionados com esta pesquisa, ressaltando a contribuição da dissertação no meio acadêmico. No Capítulo 4 é descrito os sistemas de Cloud Gaming. O Capítulo 5 detalha a arquitetura do sistema e a proposta do mecanismo de predição de movimento. No Capítulo 6 são apresentados os experimentos realizados e os resultados. As conclusões são apresentadas no Capítulo 7.

2 CONCEITOS

2.1 QUALIDADE DE EXPERIÊNCIA (QoE)

QoE é um conceito que representa a percepção que o usuário tem sobre a qualidade de um determinado serviço. Embora a QoE não possa muitas vezes ser quantificada numericamente, é um dos indicadores mais importantes na avaliação de experiência do usuário. Para as empresas de jogos bem como os fornecedores de serviço IP, a QoE deve ser a melhor possível. Para os clientes, a experiência com o produto difere de pessoa para pessoa. Isso faz com que a QoE seja um conceito bastante subjetivo porque há clientes mais fáceis de agradar do que outros e com comportamentos/expectativas distintas. Contudo, apesar da subjetividade inerente, o QoE é um dos parâmetros mais importantes para avaliar a qualidade dos serviços prestados (CARVALHO et al., 2010).

2.2 QUALIDADE DE SERVIÇO (QoS)

QoS é genericamente definida como o nível de desempenho e de garantia que a rede deve satisfazer aos requisitos do tráfego e do serviço. Implementar QoS numa rede implica controlar o tráfego na rede, definindo prioridades e limites de QoS aos contratos de serviço negociados com os clientes. A QoS, ao contrário da QoE, define numericamente os parâmetros específicos de qualidade de um serviço de rede, ou de um determinado grupo de aplicações específicas (CARVALHO et al., 2010). Resumidamente, podem-se especificar alguns dos parâmetros usados na avaliação da QoS nos jogos em nuvem (MATHY et al., 1999):

- Latência ou tempo de resposta: mais vulgarmente chamado pelos jogadores como *lag* (*latency at game*) ou *ping*, é o tempo, geralmente em milissegundos, entre envio do comando do jogador e o retorno com o vídeo do jogo correspondente sendo exibido na tela do usuário.
- Taxa de transferência: é a quantidade de dados transferidos entre cliente e servidor num determinado intervalo de tempo.
- Perda de pacotes: é a quantidade de pacotes que foram perdidos durante a transmissão entre o cliente e o servidor devido a degradação do sinal na rede, congestão na rede, entre outros.

- Estrangulamento: acontece quando um emissor tenta enviar uma quantidade de dados que o receptor, por diversos motivos, não consegue receber.
- Estabilidade de ligação: é a capacidade da rede manter a comunicação de dados, sem quebra de serviço.
- Largura de banda: é a capacidade de transmissão da rede ou da ligação.

2.3 JOGOS ELETRÔNICOS

Os jogos eletrônicos podem ser classificados em relação aos requisitos de recursos, sistemas: *offline/online*, número de jogadores e arquiteturas. Para alcançar uma boa jogabilidade, os jogos eletrônicos exigem requisitos na precisão de jogo, capacidade de resposta e equidade entre todos os jogadores, exigindo mais processamento da arquitetura de sistema de jogo (MISHRA et al., 2014).

Jogos eletrônicos online possuem quatro arquiteturas principais: cliente-servidor, em que um servidor centralizado gerencia o mundo dentro do jogo; *peer-to-peer* (P2P), em que os pares compartilham a carga de processamento; híbrido, em que um servidor centralizado lida com as tarefas mais importantes e de computação intensiva, enquanto os pares, lidam com as atualizações de tela e as técnicas de otimização; e Cloud Gaming, uma arquitetura de jogo dentro da nuvem, que aproveita os recursos oferecidos pela computação em nuvem para garantir melhores técnicas de renderização, tempos de resposta mais curtos, melhor precisão e maior equidade em relação às outras arquiteturas apresentadas (MISHRA et al., 2014).

Existem também desafios para cada uma dessas quatro arquiteturas. Arquiteturas cliente-servidor sofrem com a baixa escalabilidade, inferior tolerância a falhas e alto custo de processamento, enquanto arquiteturas P2P sofrem com o controle consistência, maior potencial de trapaça, e maior complexidade de implementação. Arquiteturas híbridas lidam parcialmente com os pontos fracos de cada abordagem, mas não resolvem totalmente estes problemas (MISHRA et al., 2014). Já os ambientes em nuvem não apresentam garantias de execução de aplicações que possuam requisitos específicos de tempo real, como latência, variação de latência e atendimento de determinadas tarefas dentro de determinado prazo (PHAN et al., 2011).

2.3.1 Gêneros de Jogos eletrônicos

Os jogos eletrônicos são organizados em diferentes gêneros com base em sua interação com o jogador. Um gênero de jogo eletrônico é definido pela sua jogabilidade e são classificados independentemente do conteúdo (APPERLEY, 2006).

Nomes descritivos de gêneros levam em conta os objetivos do jogo, o protagonista e até mesmo a perspectiva oferecida ao jogador. O termo "subgênero" pode ser usado para se referir a uma categoria dentro de um gênero para especificar ainda mais o gênero do jogo em discussão. Considerando que "jogo de tiro" é um nome de gênero, "tiro em primeira pessoa" e "tiro em terceira pessoa" são subgêneros comuns do gênero de jogos de tiros (LECKY-THOMPSON, 2008). Alguns exemplos de gêneros de jogos eletrônicos são apresentados nos itens 2.3.1.1 até 2.3.1.6:

2.3.1.1 Simulação

Os jogos de simulação são geralmente desenvolvidos para simular o mundo real. Myers (MYERS, 2003) define que a maioria dos jogos de simulação segue mais as regras do mundo "real", como às leis da física, do que os outros gêneros de jogos. O gênero de simulação é geralmente dividido em alguns subgêneros: simulação de construção, simulação do cotidiano, simulação de veículos e jogos de esporte.

Os jogos de simulação de construção focam no gerenciamento e manutenção dos recursos, combinado à construção de algo, seja uma empresa ou uma cidade. Muitos destes jogos são pequenas representações que simulam sistemas do mundo real e oferecem ao jogador a possibilidade de gerenciar seu próprio negócio virtual. Um de suas características principais é o foco na economia do jogo e os sistemas de comércio e trocas. Nestes jogos as escolhas dos jogadores devem ser cuidadosas, pois podem comprometer o sistema inteiro (SATO; CARDOSO, 2008).

Na simulação do cotidiano, há a reprodução idêntica de todas ou quase todas as características, reações, variáveis e situações encontradas na realidade (universo cotidiano). A narrativa destes jogos é muito flexível. O jogador escolhe e estabelece seus próprios objetivos, podendo ficar no jogo sem nunca finalizá-lo. O jogador tem caminhos diferentes para participar desta narrativa, possibilitando rumos distintos de seu personagem. O jogador pode focar-se mais, em determinado momento, na

evolução de seus atributos e em outro momento, nos desafios propostos pelo jogo (SATO; CARDOSO, 2008).

Os jogos de simulação de veículos podem ser simulações do mundo real ou serem fantasiados. Os jogos de corrida possuem em comum o fato do avatar do jogador estar correndo e ele estar no controle. Isto é, as ações e escolhas são completamente controladas pelo jogador. É característico deste gênero também que os pontos de vista das câmeras sejam escolhidos com a finalidade de criar a ilusão de velocidade e controle (SATO; CARDOSO, 2008).

Os jogos de esporte são simulações simplificadas dos esportes. Muitos destes jogos baseiam-se nas regras e nas características estéticas dos esportes da vida real. Muitos destes jogos envolvem times, competições e torneios (SATO; CARDOSO, 2008).

Alguns exemplos de jogos de simulação são: SimCity 4, Fifa 2017 e Euro Truck Simulator 2.

2.3.1.2 Estratégia

Os jogos de estratégia caracterizam-se pelo foco nas táticas e planejamento, tanto quanto no gerenciamento de unidades e recursos. Abrangem temas como conquistas, exploração e comércio (SATO; CARDOSO, 2008).

O gênero de estratégia é geralmente dividido em dois subgêneros: estratégia em tempo real (RTS) e estratégia por turnos (TBS). A diferença entre os subgêneros está no fator tempo, no caso do RTS, o tempo dentro do jogo é contínuo, ou seja, as ações de todos os jogadores ocorrem simultaneamente durante a partida. Já no caso do TBS, cada jogador tem o seu turno para realizar as ações possíveis, enquanto os outros jogadores esperam.

Em ambos os casos, cada informação influencia nas possíveis jogadas futuras. À medida que o jogo progride, o jogador descobrirá que os inimigos têm diferentes pontos fracos e, além disso, novas armas, estilos de ataque e fatores ambientais no jogo podem ser explorados para eliminar o inimigo com maior eficiência. Coletar e analisar as informações para então poder escolher a melhor resposta para uma dada situação é uma característica que define os jogos de estratégia (APPERLEY, 2006).

Alguns exemplos de jogos de estratégia são: Age of Empires, Civilization V e Advance Wars.

2.3.1.3 Ação

Jogos de ação oferecem ação intensiva como a sua principal característica, no qual o reflexo rápido e a coordenação são essenciais nesse gênero de jogo. Para os autores, os jogos de ação são experiências em tempo real com ênfase na restrição de tempo para as respostas de tarefas físicas do jogador. Os jogos de ação mais comuns são jogos de tiro e de espionagem (GRACE, 2005).

O gênero de ação é geralmente dividido em dois subgêneros: jogo em primeira pessoa e jogo em terceira pessoa. A diferença entre os subgêneros está nos pontos de vista do jogador que é explicado no item 2.1.2.

Em ambos os casos, o avatar no jogo atua como uma representação virtual que conecta o jogador com o ambiente virtual. (APPERLEY, 2006).

Alguns exemplos de jogos de ação são: CS Go, Far Cry e GTA.

2.3.1.4 Aventura

Jogos de aventura oferecem exploração e resolução de quebra-cabeças como a sua principal característica (GRACE, 2005).

Raciocínio, criatividade e curiosidade são as habilidades mais comuns de um jogador de jogo de aventura. Quando o ambiente do jogo de aventura é em duas dimensões, o jogo é classificado como jogo de plataforma.

Alguns exemplos de jogos de aventura são: The Legend of Zelda, Tomb Raider e Monkey Island.

2.3.1.5 RPGs

Os jogos de RPGs baseiam-se na criação e desenvolvimento de personagens. Os RPGs são jogos que possibilitam ao jogador, ao mesmo tempo em que desenvolve o seu personagem, controlar seu inventário, explorar mundos e acumular riqueza, status e experiência (SATO; CARDOSO, 2008).

Alguns exemplos de jogos de RPGs são: Final Fantasy, Dragon Quest e Dragon Age.

2.3.1.6 Quebra-cabeça

Os jogos de quebra-cabeça (ou jogos de enigmas e charadas. O termo *puzzle* em inglês abrange todas estas possibilidades) são jogos que podem enfatizar o enredo ou a ação e podem incluir elementos de estratégia, no entanto, seu foco está na solução de problemas de um modo geral (SATO; CARDOSO, 2008).

Alguns exemplos de jogos de quebra-cabeça são: Candy Crush Saga, Tetris e Limbo.

2.3.2 Pontos de vista do jogador

A perspectiva óptica no jogo apresenta, ao jogador, as características do ambiente no domínio espacial do jogo em relação ao personagem. A perspectiva define as maneiras pelas quais as ações o jogador pode executar dentro do espaço no jogo. As diferentes perspectivas ópticas no jogo são definidas como pontos de vista do jogador (TAYLOR, 2002). Os pontos de vista mais comuns são visão em primeira pessoa, visão em terceira pessoa e visão onipresente.

Em jogos eletrônicos, a visão em primeira pessoa refere-se a uma perspectiva gráfica a partir do ponto de vista do personagem do jogador, mostrada na Figura 1. A visão em terceira pessoa é semelhante à primeira pessoa, mas a vista é a partir da parte de trás do personagem do jogador, o que significa que o jogador pode ver o seu avatar na tela, mostrada na Figura 2. Já os jogos com visão onipresente sempre têm um ângulo de vista fixo; os jogadores assistem toda a cena como um pássaro, para que eles possam facilmente visualizar o ambiente ao redor de seus avatares, como mostra a Figura 3.

Figura 1 - Jogo de tiro com visão em primeira pessoa.

Figura 2 - Jogo de corrida com visão em terceira pessoa.

Figura 3 - Jogo de MOBA com visão onipresente.

2.3.3 Classificação pela quantidade de jogadores

Todos os jogos consistem de jogadores. A quantidade de jogadores interagindo com o jogo define se o jogo é classificado como *single-player* ou *multiplayer*. Com a opção *multiplayer*, mais de um indivíduo interage com o ambiente do jogo, enquanto com a opção de *single-player* apenas um indivíduo interage com o ambiente do jogo (HARTEVELD, 2011).

2.3.4 Comandos do jogador

Os comandos que o usuário pode realizar têm campos importantes como tempo, ângulo de visão, movimento e os botões. O campo de tempo corresponde a quantos milissegundos de simulação o comando corresponde. O campo de ângulo de visão é um vetor que representa a direção em que o jogador está olhando. O campo de movimento é determinado pela ação do teclado, mouse e joystick para ver se qualquer um dos botões de movimento foi apertado. O campo de botões é apenas um campo de bits com um ou mais bits definidos para cada botão que está sendo pressionado (BERNIER, 2001).

2.4 PREDIÇÃO DE MOVIMENTO

Existe um grande número de modos como os jogadores pensam e a constante mudança de ações em tempo real torna difícil de prever a próxima ação.

Predição de movimento implica prever os movimentos futuros em intervalos de tempo discreto (LEE et al., 2015).

No processo padrão de processamento de um jogo na nuvem, como mostra a Figura 4, o cliente envia o comando i_5 no tempo t_5 , a nuvem recebe o comando no tempo t_7 , o jogo termina de processar as informações em t_8 e o cliente recebe os quadros do jogo f_5 no tempo t_{10} .

Na abordagem com predição de movimento, como mostra a Figura 5, para prever o que o usuário irá enviar de comando no tempo t_5 , a análise já começa em t_0 . Em t_0 , o cliente envia a entrada i_0 , a nuvem recebe i_0 em t_2 e especula-se os possíveis movimentos $i'_1, i'_2, \dots, i'_5$, gerando o quadro de saída f'_5 utilizando a predição de movimento. O cliente recebe o quadro f'_5 e se o comando i_5 corresponder ao comando

especulado pela nuvem, o quadro f'_5 poderia ser mostrado, caso contrário, o processo seguia conforme o processo padrão.

Figura 4 - Jogos em nuvem padrão: o processamento do jogo dependerá da latência de rede.
Adaptado de (LEE et al., 2015).

Figura 5 - Pré-processamento utilizando previsão de movimento. (LEE et al., 2015).

Embora os fundamentos da especulação de movimento sejam bem compreendidos, o dinamismo e a sensibilidade das alterações gráficas dificultam a especulação. Se houver um rápido aumento na quantidade de estados de espaço e sensibilidade faz com que os usuários sejam capazes de, visualmente, identificar estados incorretos (LEE et al., 2015).

Para prever ou antecipar uma situação futura, as técnicas de aprendizagem como Cadeias de Markov, Modelos de *Hidden Markov*, Redes Bayesianas ou Redes Neurais são candidatas promissoras. O desafio é converter esses algoritmos para serem usados dentro do contexto certo (GELLERT; VINTAN, 2006). Para melhorar a precisão das previsões de movimento, podem-se utilizar os dados previamente coletados do mapa do jogo (COUÉ et al., 2006).

2.5 COMPUTAÇÃO EM NUVEM

A computação em nuvem é a centralização de recursos de computação e serviços em poderosos *datacenters* que permitem o acesso sob demanda (ARMBRUST et al., 2010).

Computação em nuvem permite criar estruturas de serviços online que podem ser usadas por milhares de computadores para compartilhar dados, aplicações e capacidade de computação para alcançar um resultado desejável de forma transparente para o usuário final que idealmente interage com uma única entidade (HEY; TREFETHEN, 2005).

Um modelo de Computação em nuvem deve possuir algumas características essenciais para a sua implementação. O NIST define essas características como (MELL; GRANCE, 2011):

- Serviço sob demanda: O usuário obtém recursos conforme desejado, sem a necessidade de interação humana com o provedor de serviços.
- Acesso via rede: Os recursos e serviços são disponibilizados via rede e acessíveis a partir de plataformas heterogêneas (computadores, notebooks, smartphones) através de interfaces padronizadas (navegadores Web, aplicações de linha de comando).
- Compartilhamento: O conjunto de recursos da nuvem é compartilhado através de um modelo capaz de suportar diversos usuários sob uma infraestrutura comum (modelo *multi-tenant*). A localização exata dos recursos não é necessariamente conhecida ou mesmo relevante para o usuário.
- Elasticidade: A quantidade de recursos oferecida pode ser aumentada ou diminuída de maneira automática e imediata, de modo que para os usuários

os recursos aparentam ser infinitos, sempre disponíveis e na quantidade adequada às suas necessidades.

- Métricas: Diversos mecanismos de medição são implementados na nuvem para oferecer funcionalidades de balanceamento de carga (de modo que os sistemas sejam capazes de determinar as quantidades ideais de recursos a serem oferecidas para cada usuário) e monitoração (analisando a utilização dos recursos e gerando relatórios de cobrança com base no uso dos serviços).

2.5.1 Modelos de serviços de Computação em nuvem

Existem três modelos de serviços de Computação em nuvem que são Infraestrutura como Serviço, Plataforma como Serviço e Software como Serviço (MELL; GRANCE, 2011):

- Software como Serviço (SaaS): Aplicações executadas na infraestrutura da nuvem e distribuídas para os usuários sob o modelo *multi-tenant*, isto é, vários usuários utilizando a mesma instância do serviço. As aplicações normalmente são acessíveis através de um cliente *thin*, como um navegador Web, ou um programa dedicado. O usuário não administra ou controla a infraestrutura da nuvem como o hardware, o sistema operacional, as ferramentas e serviços que suportam a aplicação, estando, portanto restrito a realizar modificações apenas dentro do serviço utilizado (parâmetros de configuração e outros aspectos relacionados).
- Plataforma como Serviço (PaaS): Serviços que permitem ao usuário implementar e distribuir aplicações criadas pelo mesmo ou por terceiros, utilizando linguagens de programação, bibliotecas e outras ferramentas suportadas pelo provedor do serviço. O usuário não possui controle sobre a infraestrutura da nuvem, mas pode realizar configurações relacionadas à plataforma de distribuição de aplicações.
- Infraestrutura como Serviço (IaaS): Recursos infraestruturais oferecidos através da nuvem, como armazenamento, processamento, virtualização e outros elementos computacionais. O usuário pode executar quaisquer

programas nesta infraestrutura, incluindo sistemas operacionais e outras aplicações, embora ainda não possua controle total sobre a mesma. No entanto, algumas configurações de rede (relacionadas à firewalls, por exemplo) e de armazenamento podem estar disponíveis para modificação.

2.5.2 Modelos de implementação de Computação em nuvem

Existem quatro modelos de implementações, nas quais é possível projetar e configurar uma determinada nuvem. Esses quatro modelos são (MELL; GRANCE, 2011):

- **Nuvem Privada:** a infraestrutura da nuvem é operada somente pela organização cliente. Pode ser gerenciada pela própria empresa ou via terceirização. Os serviços são disponibilizados para serem consumidos pela própria organização e a sua localização pode ser dentro da organização ou fora dela.
- **Nuvem Pública:** a infraestrutura da nuvem está disponível para o público em geral sob o modelo “pague pelo uso”. É oferecida por entidades que possuem grande capacidade de processamento e armazenamento.
- **Nuvem Comunitária:** a infraestrutura da nuvem é compartilhada por várias organizações que possuem interesses e objetivos comuns. Pode ser gerenciada pelas organizações ou por terceiros e a sua localização pode ser dentro das organizações ou fora delas.
- **Nuvem Híbrida:** a infraestrutura da nuvem é formada por duas ou mais nuvens, sejam elas privadas, públicas ou comunitárias, que permanecem únicas. Elas são padronizadas e unidas por tecnologias que permitem a portabilidade de dados e aplicações.

2.5.3 Segurança na nuvem

Com relação à segurança, técnicas de replicação de dados, criptografia, isolamento de clientes e recursos devem ser desenvolvidos e demonstrados de modo que os clientes possam ter o nível de segurança esperado. (SOOD, 2012) apresenta uma visão holística sobre o tema de segurança em ambientes em nuvem. Entretanto, não

se espera que Nuvens públicas possam conter informações críticas e sensíveis dos usuários. Nuvens privadas podem ser utilizadas para armazenamento de informações sensíveis, mas precisam demonstrar primeiramente que atendem efetivamente aos requisitos de segurança acima listados.

2.5.4 Escalonamento

Nos ambientes em nuvem existe a necessidade de atender diversas demandas simultaneamente que requerem o suporte a distintos mecanismos e algoritmos de escalonamento para ambientes distribuídos, de modo que possa ser realizado um controle de qual tarefa deve ser executada em qual momento e por qual recurso. Os algoritmos de escalonamento mais utilizados em ambientes de nuvem são baseados em vazão e máxima utilização de recursos, sem características específicas de cumprimento de tarefas em determinado intervalo de tempo, principalmente em sistemas de tempo real (PHAN et al., 2011).

2.6 CLOUD GAMING

Em Cloud Gaming, o processamento é realizado remotamente na nuvem e retorna imagens como uma sequência de vídeo em tempo real de volta para o jogador utilizando a Internet. O sistema deve recolher as ações de um jogador, transmiti-las para a nuvem, processar as ações, renderizar os resultados, codificar/comprimir as alterações do jogo e transmitir o vídeo (cenas do jogo) de volta para o jogador.

Os maiores desafios em Cloud Gaming (WEI; VICTOR; MIN, 2013) são:

- Satisfazer as expectativas dos jogadores em QoE durante jogos on-line fornecendo QoS de forma flexível em ambientes heterogêneos de rede.
- Suprir o aumento na demanda de recursos de computação.
- Processar e transmitir dados de dispositivos móveis para a nuvem através de redes sem fio.
- Para dispositivos móveis, o custo energético para enviar as informações e receber as imagens pela rede sem fio.

Para avaliar o QoE do cliente algumas características são consideradas como:

- Características de tráfego: a largura de banda da rede e o consumo dentro de uma sessão de jogo.
- Latência ou tempo de resposta: a diferença de tempo entre o início de um evento e o momento em que seus efeitos se tornam perceptíveis em cada etapa.
- Qualidade gráfica: a fidelidade do vídeo transmitido para o cliente.

A Figura 6 mostra a diferença entre jogos online e Cloud Gaming.

Figura 6 - Diferença entre o Cloud Gaming e jogos online. Adaptado de (CHEN et al., 2014).

Os parâmetros de QoS em jogos tais como tempo de resposta, *jitter* e perda de pacotes são conhecidos por influenciar a experiência do usuário ou a jogabilidade.

Como Cloud Game é um serviço de nuvem, este não pode ser iniciado sem acesso à rede e os jogos não podem ser reproduzidos uma vez que a Internet está desconectada.

Os dados dos jogos ficam em uma nuvem, o que evita a perda de dados devido a algum dano causado no dispositivo do usuário. Implica também que em qualquer momento ou em qualquer lugar, os jogadores podem se conectar ao jogo e o conteúdo e estado do jogo ficam salvos na nuvem. Portanto, os jogadores podem ter uma experiência de jogo contínua através de múltiplas redes e dispositivos.

Os usuários frequentemente interagem através de vários dispositivos (smartphone, tablet, laptop). Os jogadores estão interessados nos jogos, sem se preocupar com a plataforma. Eles estão dispostos a jogar o mesmo jogo em diferentes dispositivos, se o conteúdo for entregue sem problemas. Portanto, proporcionar uma boa experiência para jogos em tempo real com características multiplataforma é crítico. Suspender e retomar um jogo deve ser simples e rápido, de modo que o jogador não perceba a transição.

Um dos maiores desafios referente à possibilidade de se jogar a partir de qualquer dispositivo é a inexistência de um mapeamento direto do teclado e mouse com eventos de toque, impedindo o fornecimento de uma interface natural para o usuário em dispositivos móveis (GUAN et al., 2015).

2.6.1 Vídeo Streaming e Codificação na nuvem

Requisitos dos streaming de vídeo de jogos na nuvem são semelhantes aos de outras aplicações clássicas de mídia ao vivo. Ambas as soluções de jogos em nuvem e de streaming de mídia ao vivo devem rapidamente codificar / comprimir os vídeos vindos da nuvem e distribuí-los aos usuários finais (CHEN et al., 2014). Em ambos, não há acesso a quadros futuros antes de eles serem produzidos, o que implica que a codificação deve ser feita com muito poucos quadros. No entanto, streaming de vídeo ao vivo e jogos em nuvem também têm diferenças importantes. Em primeiro lugar, jogos em nuvem tradicionais não têm a capacidade de armazenar quadros de vídeo no lado do cliente. Quando um jogador realiza um comando, o comando tem

que atravessar a Internet para uma nuvem, ser processado pelo servidor do jogo, representado pela unidade de processamento, comprimido pelo codificador de vídeo, e transmiti-lo de volta para o leitor. Por outro lado, um streaming de mídia ao vivo, pode carregar no *buffer* alguns segundos de vídeo com perda muito pequena na QoE (CLAYPOOL, 2006).

2.6.2 Tempo de Resposta

O tempo de resposta (TR) é a métrica de QoS mais crítica percebida pelo usuário para jogos em nuvem. É definido o tempo de resposta como o tempo entre envio do comando do jogador e o retorno com o vídeo do jogo correspondente sendo exibido na tela do usuário.

Para medir o TR em um sistema de jogos em nuvem, o TR é dividido em cinco componentes:

- Tempo da rede com comando (TA): TA representa o tempo necessário para transmitir comandos de um usuário ao servidor.
- Tempo de rede com vídeo e som (TVS): TVS representa o tempo necessário para transmitir uma tela de jogo com áudio do servidor para o cliente.
- Tempo de processamento (TP): TP representa o tempo necessário para o servidor de jogos em nuvem para receber e processar o comando de um usuário, codificar e empacotar o quadro do jogo correspondente para o cliente.
- Tempo do jogo (TJ): TJ representa o tempo necessário para o software de jogos para processar um comando do usuário e formar o próximo quadro do jogo que contém as respostas do comando.
- Tempo de reprodução (TRE): TRE representa o tempo para o cliente para receber, decodificar e exibir um quadro do jogo.

O tempo entre mandar o comando (TA) para a nuvem e o de mandar o vídeo do servidor para o usuário (TVS) podem não ser o mesmo, pois os dados enviados e recebidos pela nuvem têm um perfil diferente, são recebidos pela nuvem os comandos e enviados o vídeo e o áudio para o cliente. Os canais entre o cliente e a nuvem também apresentam assimetria na largura de banda entre o *upload* e *download*.

A equação para o tempo de resposta (TR) corresponde a:

$$TR = TA + TP + TVS + TJ + TRE$$

2.6.3 Efeito da latência e do *jitter* na performance dos jogadores

Duas propriedades são importantes para medir a performance dos jogadores: a precisão necessária para concluir a ação e o prazo da ação para ser concluída.

Ações com maior precisão e prazos curtos são sensíveis a latências até mesmo modestas, enquanto as ações com menor precisão e maiores prazos não são influenciados de maneira significativa com as latências da Internet. Os efeitos da latência em jogos dependem do gênero do jogo, as interações de jogos com uma perspectiva em primeira pessoa são mais sensíveis à latência, enquanto os jogos com uma perspectiva em terceira pessoa ou perspectiva onipresente são menos sensíveis à latência (CLAYPOOL, 2006).

A Figura 7 mostra um gráfico que relaciona a latência com a performance do jogador em 3 gêneros de jogos distintos: onipresente, primeira pessoa e terceira pessoa (CLAYPOOL, 2006).

Figura 7 - Performance do Jogador vs latência para categorias de jogo. A faixa cinzenta horizontal representa o limiar de tolerância jogador para a latência (CLAYPOOL, 2006).

A Tabela 1 mostra qual o limiar de tolerância que cada gênero possui sem afetar a métrica de QoE do jogador.

Tabela 1 - Limiar de tolerância para cada gênero de jogo em um ambiente de Cloud Gaming (CLAYPOOL, 2006).

Modelo	Perspectiva	Exemplos de gêneros	Sensibilidade	Limiar de tolerância (ms)
Personagem	Primeira pessoa	FPS, Corrida	Alta	100
	Terceira pessoa	Esporte, RPG	Média	500
Onipresente	Varia	RTS, SIM	Baixa	1000

Uma concepção comum entre os jogadores é que latências de rede inferiores a 100 milissegundos são necessárias para um bom QoE, independentemente do gênero de jogo (CLAYPOOL, 2006).

Outro fator que influencia o QoE do usuário é o *jitter*, que consiste na variação de latência durante o jogo. Normoyle (NORMOYLE, 2014) define que o *jitter* pode influenciar de maneira mais negativa o QoE do que uma latência alta.

2.7 MOBILE CLOUD COMPUTING (MCC)

Com o crescente aumento de pessoas utilizando dispositivos móveis, como smartphones e tablets, o mercado de jogos eletrônicos procura cada vez mais atender esse público.

A escassez de recursos computacionais é um dos principais problemas enfrentados pelos usuários e desenvolvedores de aplicações para dispositivos móveis (SATYANARAYANAN, 2009).

MCC é uma tecnologia de computação móvel que, através da Internet, disponibiliza recursos de variadas nuvens e tecnologias de redes com funcionalidade irrestrita, armazenamento e mobilidade para servirem uma infinidade de dispositivos móveis em qualquer lugar, a qualquer hora independentemente do ambiente e da plataforma com base no sistema “pague pelo que usar” (pay-as-you-use) (ABOLFAZLI et al., 2013).

Como os dispositivos móveis são alimentados por bateria, a conservação de energia é importante. A maior parte do consumo de energia para aplicações em MCC deve-se à transmissão de dados sem fio, já que para transmitir dados através de uma rede sem fio consome quantidades significativas de energia. Kumar e Lu (KUMAR, LU; 2010) apresentam um estudo comparando a economia de energia quando o processamento é feito na nuvem em relação ao processamento feito no próprio dispositivo móvel. O trabalho mostra que há benefícios ao utilizar a nuvem, quando a quantidade de processamento é alta e necessita poucas comunicações com a nuvem.

Ao projetar um aplicativo que exige grande poder computacional, adequado apenas para dispositivos mais poderosos, grande parte dos usuários com smartphones mais limitados e com recursos mais simples não conseguem utilizar a aplicação. Além

disso, como a tecnologia das baterias não evoluiu no mesmo ritmo que outros componentes de hardware (CUERVO et al., 2010), as aplicações mais intensivas, quando alocadas na nuvem permite que dispositivos móveis possam utilizar as aplicações, além de reduzir o consumo de bateria.

3 TRABALHOS RELACIONADOS

Esta seção lista vários experimentos, estudos de casos e trabalhos de pesquisa relevantes para a melhoria na QoE de sistemas de jogos em nuvem. Isso fornece uma visão geral da arquitetura do sistema e insights em áreas de foco para avaliar o desempenho do trabalho proposto, incluindo o tempo de processamento, resolução de tela do jogo, qualidade de imagem do jogo, taxa de quadros do jogo e fatores críticos para o desempenho dos jogadores.

Barboza et al. (BARBOZA et al., 2015) propõem dividir a codificação da imagem do jogo em várias camadas e armazená-las em cache. Cada camada armazenará uma imagem em cache e quando um novo quadro é gerado, as imagens que estão armazenadas em cache são comparadas com o novo quadro de imagem gerado e se não há modificações, a imagem em cache é enviada em vez de codificar e transmitir o quadro. Com isso, menos partes da imagem são codificadas em cada quadro, portanto, um tempo de codificação médio mais baixo será alcançado, o que consequentemente reduz a carga do servidor. Ao mesmo tempo, enviar apenas camadas modificadas para o cliente reduz os requisitos de largura de banda de transmissão.

Noimark e Cohen-or (NOIMARK; COHEN-OR, 2003) apresentam um sistema que utiliza as informações do vídeo para segmentar os objetos do plano de fundo e do plano frontal e variar o nível de quantização individualmente para cada camada. Este sistema também se baseia em informações conhecidas sobre a cena para acelerar a escolha de padrões de codificação e a estimativa de movimento para reduzir o tempo e o poder de processamento necessários para realizar uma codificação em tempo real. Normalmente, um codificador usa um padrão pré-definido para determinar quais quadros devem ser codificados de forma independente e quais deverão utilizar a estimativa de movimento. Para mostrar o fundo, Noimark e Cohen-or (NOIMARK; COHEN-OR, 2003) utilizam um recurso de MPEG-4 que apresenta a imagem como um mosaico que é reconstruído no cliente apenas com as partes relevantes para o quadro atual. Partes de fundo modificadas são atualizadas no cliente usando informações de compensação de movimento global enviadas pelo servidor e quando as operações da câmera não podem ser representadas com essas transformações, toda a imagem é atualizada.

Para evitar o alto custo durante as cenas de movimento utilizando o MPEG-4, o trabalho de Cheng (CHENG et al., 2004) propõe um algoritmo que utiliza os dados em 3D para calcular os vetores de movimento diretamente a partir da cena. Estes vetores de movimento são adquiridos através de uma projeção de um pixel 2D de volta para o espaço 3D usando a posição do pixel, o buffer de profundidade e a matriz de projeção da câmera. Em seguida, o ponto 3D é novamente projetado no espaço 2D utilizando as configurações da câmera para o quadro anterior. Esta estratégia permite a aquisição de um vetor que descreve o movimento do pixel entre os dois quadros. Mesmo que essa técnica permita a aquisição de um vetor de movimento rápido, que só falha em alguns casos específicos, o algoritmo é aplicável somente para cenas estáticas, interagindo apenas com o movimento da câmera. Isto tem grande uso em aplicações de visualização, mas não funciona bem com jogos que têm objetos dinâmicos e que estão constantemente mudando as propriedades, formas, posição e rotação.

Eisert e Fechteler (EISERT; FECHTELER, 2008) propõem que os comandos gráficos sejam interceptados antes de serem enviados para a placa de vídeo no servidor e, em seguida, codificados e transmitidos para o cliente, que irá processá-los localmente. Isto evita a alta demanda de codificação de vídeo no servidor e permite que os comandos possam ser transmitidos logo que eles são emitidos, em vez de esperar por um quadro de vídeo inteiro ficar pronto. Para um servidor, esta técnica facilita o fornecimento de jogos para vários clientes devido à redução na demanda de processamento, e também é possível aplicar em jogos existentes sem modificá-los, uma vez que os comandos gráficos são interceptados no nível da API. Algumas desvantagens são que o jogo exige mais do hardware, uma vez que ainda é necessária a capacidade para interpretar e processar os comandos gráficos recebidos e pode ser alcançada uma alta taxa de bits quando há um envio de comandos gráficos, de objetos e de texturas de jogo. Já os comandos gráficos que acessam o atual estado da Unidade de Processamento Gráfico (GPU) geram um tempo de resposta entre o pedido no servidor, a consulta no cliente e o retorno e uso dos dados novamente no servidor.

A fim de resolver esses problemas, Eisert e Fechteler (EISERT; FECHTELER, 2008) propõem utilizar um cache de memória e simular todos os possíveis quadros futuros no servidor. Todo acesso ao cache de memória de vídeo pode ser feita

localmente no servidor, eliminando o tempo de resposta do comando. Um quadro independente é completamente codificado e os quadros seguintes são codificados apenas com alterações relacionadas a ele.

Claypool et al. (CLAYPOOL et al., 2009) usou um jogo personalizado com níveis que combinavam diferentes ações e perspectivas para medir o desempenho do usuário com diferentes configurações de exibição. A análise das experiências mostrou que a taxa de quadros tem uma influência muito maior sobre o desempenho do usuário do que a resolução de quadros.

Chang et al. (CHANG et al., 2011) propuseram uma metodologia para quantificar o desempenho de clientes dentro dos jogos. Dos resultados de estudo de caso com três clientes, LogMeIn, TeamViewer e UltraVNC, eles demonstraram que a taxa de quadros da tela e a degradação da qualidade de quadros dos clientes são críticos para o desempenho do jogo e que a taxa de quadros tem um impacto maior no desempenho do jogo do que a qualidade do quadro.

Hemmati et al. (HEMMATI et al., 2011) propõe um método de adaptação de conteúdo com base na codificação seletiva de objetos para reduzir os requisitos de banda e a carga no servidor para streaming de vídeo na nuvem. Objetos menos importantes para a atividade do jogador são selecionados e retirados do quadro codificado, reduzindo o tempo de codificação e a taxa de bits. Seleção de objeto e seu impacto sobre a experiência do jogador são discutidos por Rahimi, Shirehjini e Shirmohammadi et al. (RAHIMI; SHIREHJINI; SHIRMOHAMMADI, 2011). Enquanto essa técnica apresenta redução na taxa de bits e no tempo de processamento, especialmente quando os resultados são extrapolados para servidores de nuvem e o conteúdo é entregue para um grande número de usuários simultaneamente, a desvantagem é a remoção completa de objetos menos importantes do cenário. Este fato leva a resultados visualmente estranhos para o usuário, que veria objetos “saltando” na tela durante o jogo.

Huang et al. (HUANG et al., 2013) selecionaram três jogos de diferentes gêneros para serem executados em sistemas de jogos em nuvem e mediram o atraso do sistema do GamingAnywhere. Eles dividiram o atraso do sistema em um número de componentes menores para analisar melhor cada subcomponente de atraso de forma independente.

Trabalhos como o de Hemmati, Shirmohammadi, Eisert e Fechteler, Noimark e Cohen-or tinham como foco diminuir o tempo de resposta fazendo com que o quadro da tela toda não fosse renderizado constantemente. Para isso, o cliente deveria ter perdas na qualidade da imagem, ou parte do processamento deverá ser feito no lado do cliente.

O arcabouço Cloud Manager tem como objetivo diminuir o tempo de resposta, mas diferentemente dos outros sistemas presentes na literatura, o Cloud Manager fará com que a camada do personagem e a do plano de fundo sejam tratadas de maneiras independentes. Na camada do personagem é aplicada a predição de movimento.

Para a validação dos resultados, foi utilizado a mesma técnica proposta por Huang, no qual para medir o atraso do sistema é analisado cada subcomponente de forma independente.

4 SISTEMAS DE CLOUD GAMING

4.1 SISTEMAS DE CLOUD GAMING

Sistemas de Cloud Gaming oferecem jogos na nuvem como um serviço. A forma como o serviço é distribuído ao cliente varia de acordo com o sistema. Alguns serviços só podem ser acessados pelo computador, enquanto outros são acessados por *set-top boxes* na Tv. O próprio jogo pode ser adaptado para ser fornecido na nuvem. Ele pode ser (CHEN et al., 2011):

- modificado para rodar no servidor na nuvem;
- codificado no servidor e decodificado no cliente;
- adaptado para rodar parte do jogo no cliente e outras partes no servidor;

Na Tabela 2 são mostrados alguns exemplos de sistemas de Cloud Gaming disponíveis no mercado.

Tabela 2 - Sistemas de Cloud Gaming disponíveis no mercado.

Nome	Status de acesso	Status de Desenvolvimento	Código fonte disponível?	Licença	Preço
GamingAnywhere	Ativo	Ativo	Sim	Código Aberto	Grátis
Uniquitous	Ativo	Ativo	Sim	Código Aberto	Grátis
Ubitus GameCloud	Ativo	Ativo	Não	Proprietário	Não disponível
Steam In-Home Streaming	Ativo	Ativo	Não	Proprietário	Freeware
Remote Play	Ativo	Ativo	Não	Proprietário	Freeware
Cloud Gaming eXtreme	Ativo	Ativo	Não	Proprietário	\$0.30 por hora na Amazon Web Services

Em 2013, o primeiro sistema completo de jogos em nuvem de código aberto, o GamingAnywhere, foi lançado. Como um sistema aberto, o controle de streaming de vídeo pode ser substituído por outro componente implementando um diferente padrão, algoritmo ou protocolo (HUANG et al., 2013). Com sua extensibilidade e reconfigurabilidade, o GamingAnywhere é utilizado pelos pesquisadores que desejam observar e medir diferentes aspectos de jogos em nuvem com configurações personalizadas.

No entanto, o GamingAnywhere só pode executar jogos que já estejam prontos, pois o jogo no servidor é executado independentemente e não há controle sobre o conteúdo do jogo em si. Os pesquisadores/desenvolvedores não têm acesso ao código-fonte do jogo, pois ele interage com o aplicativo do jogo através das API

específicas do sistema operacional. Os usuários não são capazes de estudar o efeito de um conteúdo específico do jogo, como um personagem do jogo, um objeto do jogo ou um áudio do ambiente no jogo, uma vez que não é possível modificar o jogo em si se não tiver acesso ao código-fonte do jogo.

O sistema de Cloud Gaming utilizado neste trabalho é o Uniquitous, pois entre os sistemas disponíveis no mercado, é um dos únicos gratuitos e cujo código fonte está disponível para modificações e estudo. O Uniquitous (LUO; CLAYPOOL, 2015) fornece uma plataforma flexível e de fácil acesso para pesquisadores de jogos em nuvem e desenvolvedores de jogos, pois é implementado usando o Unity, que é um sistema de criação de jogos multiplataforma, incluindo um motor de jogo e um ambiente de desenvolvimento integrado (UNITY, 2016).

Uma vez que o Unity permite a criação de projetos de jogos e o Uniquitous pode ser integrado em qualquer projeto de desenvolvimento Unity, isso faz com que o Uniquitous tenha um controle de plataforma sobre o conteúdo do jogo e o sistema de jogo em um ambiente de jogo baseado em nuvem. Isso é conveniente tanto para os desenvolvedores Unity quanto para o Uniquitous.

O Uniquitous permite modificações em suas estruturas internas, configurações em seus parâmetros de sistema, como qualidade de imagem, resolução de imagem e qualidade de áudio, de modo a atender aos diferentes requisitos cliente-servidor. Mais importante ainda, os ajustes de conteúdo do jogo podem ser feitos no Uniquitous para vários fins. Por exemplo, objetos do jogo podem ser removidos ou adicionados para que a complexidade de cena no jogo possa ser ajustada influenciando assim o desempenho de jogos na nuvem. Ou, a fim de ajustar a taxa de quadros para jogos na nuvem, as configurações da câmera podem ser ajustadas ou novas câmeras podem ser adicionadas à cena do jogo (LUO; CLAYPOOL, 2015).

O Uniquitous é composto por duas entidades, o servidor Uniquitous e o cliente Uniquitous. A arquitetura permite que os desenvolvedores de jogos Unity possam executar seus projetos Unity no servidor com algumas pequenas modificações para interagir remotamente com seus projetos no cliente. Em Uniquitous, os jogadores inserem o comando no cliente, que em seguida, é enviado para o servidor. Ao receber o comando do cliente, os projetos Unity em execução no servidor atualizam seus estados de jogo e o mecanismo Unity processa novos conteúdos com base nos estados atualizados do jogo. O Uniquitous captura telas de jogos renderizadas e áudio

dos jogos, compacta as imagens e dados de áudio e transmite ambos para o cliente. O cliente recebe continuamente os dados de imagem e dados de áudio e os descomprime para exibição e reprodução (LUO; CLAYPOOL, 2015).

4.2 INICIALIZAÇÃO DO SISTEMA DE CLOUD GAMING

Um usuário se autentica pela primeira vez no sistema através de um portal do servidor, que fornece uma lista de jogos disponíveis. O usuário, então, seleciona um jogo e envia uma solicitação para jogar. Após o recebimento do pedido, o portal do servidor encontra um servidor de jogos disponível, carrega o jogo selecionado no servidor, e retorna a URL do servidor do jogo para o usuário. Finalmente, o usuário se conecta ao servidor do jogo e começa a jogar. O portal do servidor é um serviço Web fornecendo uma interface de autenticação e de seleção de jogos para o usuário.

São definidos dois tipos de fluxos de rede na arquitetura, o fluxo de dados e o fluxo de controle. O fluxo de controle envia as ações do usuário a partir do cliente para o servidor, enquanto que o fluxo de dados transmite áudio e vídeo a partir do servidor para o cliente. O jogo selecionado por um usuário é executado em um servidor de jogo. Há um agente que funciona juntamente com o jogo escolhido no mesmo servidor. O agente pode ser um processo stand-alone ou um módulo (sob a forma de um objeto compartilhado ou uma DLL compartilhada) inserido no jogo selecionado. A escolha depende do tipo de jogo e como o jogo é implementado. O agente tem duas tarefas principais. A primeira tarefa é a de capturar os quadros de áudio e vídeo do jogo, codificar os quadros usando os *codecs* escolhidos, e em seguida entregar os quadros codificados para o cliente, através do fluxo de dados. A segunda tarefa é a de o agente interagir com o jogo. O agente recebe as ações do usuário, e ele deve se comportar como o usuário e jogar junto com o jogo repetindo os eventos do teclado, do mouse, dos joysticks e até mesmo dos gestos recebidos.

4.3 ARQUITETURA DO UNIQUITOUS

Nesta seção, é apresentada a arquitetura do sistema de Cloud Gaming Uniquitous. Primeiro, é dada uma visão geral de alto nível da arquitetura baseada em fluxos de dados do sistema. Em seguida, é descrito a implementação de cada componente no sistema. A arquitetura do sistema é mostrada na Figura 8.

Figura 8 - Arquitetura do Uniquitous com o Cloud Manager.

4.3.1 Visão Geral de Alto Nível do Sistema

De acordo com a arquitetura da Figura 8, existem três entidades, que são Projeto Unity, Servidor Uniquitous e Cliente Uniquitous. O Servidor Uniquitous e o Cliente

Uniquitous são configurados em dois computadores diferentes. O Projeto Unity é executado no Servidor Uniquitous e ambos funcionam dentro do programa Unity.

4.3.2 Projeto Unity

4.3.2.1 Jogo Unity

O Jogo Unity refere-se a todos os scripts do jogo que são afetados pela entrada do usuário, como movimentos de personagens do jogo e interações com o sistema GUI. Todos os comandos enviados pelo cliente para o jogo Unity no servidor são do componente Recepção da Entrada de Dados.

4.3.2.2 Janela do Jogo

A Janela do Jogo faz parte do Unity e funciona com a câmera do Unity para capturar e exibir o conteúdo do jogo para o jogador. A câmera do Unity é um componente essencial para renderizar a cena do jogo no Unity.

4.3.2.3 Fonte de Áudio

A Fonte de Áudio é fornecida pelo componente Áudio. O Áudio é um componente essencial do Unity que recebe todo o áudio do jogo na cena e reproduz sons através dos alto-falantes do computador.

4.3.3 Servidor Uniquitous

4.3.3.1 Captura de Áudio

O componente Captura de Áudio é usado para capturar dados de áudio do componente Áudio. Foi implementada a Captura de Áudio usando a função *OnAudioFilterRead*, que obtém um pedaço de dados de áudio aproximadamente a cada 20ms. Cada pedaço de dados de áudio é um array de 2048 floats com valores que variam de -1 a 1. O array de 2048 floats é o tamanho padrão do buffer de dados que são passados para o *OnAudioFilterRead*. Embora seja possível alterar o tamanho do buffer, não é recomendado de acordo com a documentação do Unity, pois o desempenho pode piorar ao coletar dados de áudio (AUDIO BUFFER SIZE IN UNITY,

2016). Toda vez que o *OnAudioFilterRead* passa um pedaço de dados de áudio, este é convertido em um array de 8192 bytes e escrito no fluxo de rede. Os dados são então enviados para o componente Codificação e Transmissão de Áudio pelo socket TCP usando o endereço IP do localhost.

4.3.3.2 Codificação e Transmissão de Áudio

O componente Codificação e Transmissão de Áudio é implementado usando FFMPEG. FFMPEG é uma solução multi-plataforma para gravar, converter e transmitir áudio e vídeo (FFMPEG, 2016). Quando o streaming de áudio é ativado no Uniquitous, o FFMPEG é iniciado como um processo separado chamando *process.Start*, definindo as opções das linhas de comando do FFMPEG que são passadas para o processo iniciado. As opções incluem configurações do fluxo de entrada de áudio e da saída de áudio, como o endereço IP, taxa de amostragem de áudio, taxa de bits de codificação, número de canais e codec de áudio usado. FFMPEG obtém o fluxo de áudio do endereço de IP do localhost, que é enviado a partir do componente Captura de Áudio por um socket TCP. Em seguida, os dados de áudio são compactados com o codificador MP3 fornecido pelo FFMPEG e transmitidos por UDP para o componente Recepção e Decodificação de Áudio no cliente. O endereço IP de destino do fluxo de saída do FFMPEG deve ser o endereço IP da máquina em que o cliente está sendo executado. O processo FFMPEG é executado em segundo plano, de modo que a janela do terminal não seja exibida.

4.3.3.3 Captura de Tela do Unity

O componente Captura de Tela do Unity é usado para capturar a tela do jogo na Janela do Jogo. O método *Texture2D.ReadPixels* é chamado em uma co-rotina para ler os pixels da tela da Janela de Jogo e os dados são salvos no formato *Texture2D* de 24 bits. A taxa de captura é sincronizada automaticamente com a taxa de processamento da compressão de imagem, uma vez que o mecanismo de co-rotina garante que ele não irá capturar um novo quadro até que o codificador de imagem termine de codificar o último quadro. Um timer é definido antes de chamar o *Texture2D.ReadPixels* para certificar-se de que ele captura todo o conteúdo no final de cada quadro.

O componente Captura de Tela do Unity também possui os métodos *Texture2D.BackgroundReadPixels* e *Texture2D.CharacterReadPixels* que permitem ler os pixels do plano de fundo e os pixels do avatar, respectivamente.

4.3.3.4 Codificação de Imagem

Foi usado um codificador JPEG para fazer codificação de imagem. O JPEGEncoder usado pelo Uniquitous é um script escrito em C# por Andreas Broager e publicado na Comunidade Unity. Ele está localizado em um *namespace* para que ele possa ser usado por outros scripts importando o *namespace*. O construtor JPEGEncoder aceita como entrada: o objeto Unity Texture2D, o fator de qualidade e a opção de bloqueio. O fator de qualidade pode ser qualquer valor entre 1 e 100 e um fator de qualidade maior indica que uma menor compressão ou uma maior qualidade após compressão. Quando a codificação de imagens com opção de bloqueio está definida como verdadeira, a *thread* principal do Unity será bloqueada até que a codificação seja feita. Como o JPEGEncoder inicia uma nova *thread* para a codificação de imagem e esta *thread* leva um tempo maior para concluir a tarefa de codificação para texturas com um tamanho grande, o Uniquitous define a opção de bloqueio para falso. Isso permite que o segmento de codificação em JPEG seja executado em uma co-rotina para que a *thread* principal do Unity ainda possa ser executada normalmente enquanto o segmento de codificação JPEG está sendo executado.

A Figura 9 ilustra como funciona a co-rotina, onde uma tarefa de codificação de imagem é dividida em T1, T2 ... Tn e cada um deles representa o tempo que leva para concluir uma parte da tarefa. A frequência com que a co-rotina é chamada depende da taxa de quadros que o jogo está sendo executado.

Figura 9 - Co-rotina da codificação de uma imagem JPEG.

JPEGEncoder pega os dados do pixel no formato Texture2D do componente Captura de Tela do Unity e comprime os dados em um array de bytes.

4.3.3.5 Transmissão de Imagem

Chamadas de Procedimento Remoto (RPC) fornecidas pelo sistema interno do Unity não funcionam bem ao enviar uma grande quantidade de dados, como uma imagem, uma vez que causa uma quantidade crescente de atraso ao receber os dados no cliente, por isso, o Uniquitous usa um pacote de rede *third-party* chamado uLink (ULINK, 2014) para fazer as conexões entre o servidor e o cliente. Na transmissão de imagens, *uLink.NetworkView.UnreliableRPC* é chamado para obter os arrays de bytes do componente Codificação de Imagem e transmitir os dados através de um método RPC *ReceiveImages* no componente Recepção de Imagem no cliente. Os objetos do jogo no script das RPC que estão anexados no cliente e no servidor precisam ter o mesmo ID de Unity para se comunicarem com êxito. *ReceiveImages* no cliente é chamado somente quando os dados de um quadro completo estão disponíveis no componente Codificação de Imagem.

4.3.3.6 Recepção da Entrada de Dados

O componente Recepção da Entrada de Dados no servidor é responsável por receber todos os dados de entrada do cliente e passar os dados para o jogo Unity em execução no servidor. A Recepção da Entrada de Dados é implementada usando o método RPC *ReceiveInputs*.

4.3.4 Cliente Uniquitous

4.3.4.1 Reprodução de Áudio

O componente de Reprodução de Áudio é implementado usando FFPLAY. Os dados de áudio descomprimidos do componente Recepção e Decodificação de Áudio são reproduzidos na janela SDL. Quando o FFPLAY reproduz apenas áudio, a janela SDL fica desativada para que não distraia os usuários do jogo Unity.

4.3.4.2 Recepção e Decodificação de Áudio

O componente Recepção e Decodificação de Áudio é implementado usando FFPLAY. FFPLAY é um reprodutor multimídia portátil que usa as bibliotecas FFMPEG e a biblioteca SDL (SDL, 2016). Quando a opção de recepção de áudio é ativada no cliente Uniquitous, FFPLAY é iniciado como um processo separado, chamando pelo *process.Start*, definindo as opções das linhas de comando que são passadas para o processo iniciado. FFPLAY é usado para receber a transmissão de áudio enviada pelo componente Codificação e Transmissão de Áudio no servidor Uniquitous. Em seguida, o fluxo de áudio é decodificado para ser reproduzido.

4.3.4.3 Visualização de Imagem

O componente de Visualização de Imagem é implementado usando o método *GUI.DrawTexture* chamado na função de evento *OnGUI*. Ele obtém a textura passada pelo componente Decodificação de Imagem e desenha-o na janela do jogo do cliente. O tamanho da textura e a sua posição na tela da janela podem ser ajustados pelos usuários. A taxa de exibição de imagem é sempre mais rápida do que a taxa de

recepção de imagem porque a função de evento *OnGUI* é chamada automaticamente pelo Unity várias vezes em cada quadro.

4.3.4.4 Decodificação de Imagem

Todos os dados de imagem recebidos pelo componente de Decodificação de Imagem são processados no quadro principal. Foi usado o *Texture2D.LoadImage* para carregar o *array* de bytes que contém os dados de cada quadro do jogo. *Texture2D.LoadImage* pode carregar uma imagem JPG ou PNG de um *array* de byte brutos. O *array* bytes é decodificado e convertido de volta para o formato Texture2D de 24 bits para ser usado.

4.3.4.5 Recepção de Imagens

Recepção de Imagem é implementada pelo método RPC *ReceiveImages* no cliente. Ele pega o array de bytes enviados a partir do componente Transmissão de Imagem no servidor. A taxa de recebimento é a mesma que a taxa de invocação do método *uLink.NetworkView.UnreliableRPC* no servidor.

4.3.4.6 Transmissão dos Dados de Entrada

Similar ao componente Transmissão de Imagem, o componente Transmissão de Dados de Entrada é implementado pelas RPC fornecidas pelo uLink. O método *uLink.NetworkView.UnreliableRPC* é invocado para enviar os dados de entrada para o método RPC *ReceiveInputs* no componente Recepção da Entrada de Dados do servidor. Existem três tipos de dados de entrada que podem ser enviados: posições do mouse, cliques do mouse e teclas apertadas no teclado. O Unity dá acesso à leitura de todos os dados de entrada dos dispositivos de entrada. Os movimentos do mouse são traduzidos para as coordenadas (tipo Vector3) do cursor do mouse na tela do cliente. Os cliques do mouse são traduzidos para o valor inteiro 1 ou 0. As setas do teclado, quando apertadas, são convertidas em valores entre -1 e 1 ao longo dos eixos horizontal e vertical. Outras teclas pressionadas no teclado são traduzidas para os seus valores exatos das teclas pressionadas.

4.3.5 Fluxo de dados

Três tipos de fluxos de dados são definidos na arquitetura:

Fluxo de dados de imagem:

Janela do jogo → Cloud Manager → Captura de Tela do Unity → Codificação de Imagem → Transmissão de Imagem → Cloud Manager → Recepção de Imagem → Decodificação de Imagem → Visualização de Imagem.

Fluxo de dados de áudio:

Áudio → Cloud Manager → Captura Áudio → Codificação e Transmissão de Áudio → Cloud Manager → Recepção e Decodificação de Áudio → Reprodução de Áudio.

Fluxo de dados de entrada

Entrada da Transmissão → Entrada Recepção → Cloud Manager → Jogo Unity.

O fluxo de dados de imagem transporta os dados com os quadros do jogo.

O fluxo de dados de áudio transporta dados do áudio do jogo.

O fluxo de dados de entrada transporta dados da entrada do usuário.

O fluxo interno representa as comunicações dentro do Unity e o fluxo externo representa as comunicações entre o servidor e o cliente na rede.

5 ARQUITETURA DO ARCABOUÇO E PROPOSTA DO MECANISMO DE PREDIÇÃO DE MOVIMENTO

Uma vez que a codificação é uma abordagem centralizada no interior do sistema em Cloud Gaming, é fácil recolher informações de todos os jogadores e fornecer a melhor solução para minimizar a taxa de transmissão do servidor.

Um desafio para todos os sistemas de Cloud Gaming é que em cada quadro gerado no servidor é preciso renderizar, codificar, enviar e decodificar, e ainda manter um baixo tempo de atraso no sistema.

O tempo entre mandar o comando pelo cliente e receber o streaming do vídeo do jogo é crucial para uma boa QoE. Para diminuir esse tempo de resposta, foi proposto o arcabouço Cloud Manager que utiliza dois algoritmos: implementação por camadas e predição de movimentos.

O Cloud Manager proposto neste trabalho foi projetado para jogos *single-player* e de visão onipresente do gênero de ação ou aventura.

5.1 Implementação por camadas

O jogo deve ser adaptado ou criado especificamente em duas camadas no Unity, para que o sistema em Cloud Gaming possa codificar as diferentes camadas do jogo separadamente. As duas camadas são a camada do personagem e a camada do plano de fundo. O programador deve ter acesso a cada camada do quadro.

Com a implementação por camadas, foi analisado separadamente o plano de fundo e o plano do personagem, conforme mostra a Figura 10.

Figura 10 - Arquitetura do Cloud Manager.

No plano de fundo estão presentes todos os componentes do cenário e dos personagens não controlados pelo jogador. Quando o jogador realiza um comando, um quadro da imagem do jogo deve ser gerado, e se esse quadro não for diferente do quadro do plano de fundo anterior, o sistema em nuvem não precisa codificar o comando novamente, enviando assim o quadro armazenado no servidor da nuvem.

No plano do personagem, é realizado o pré-processamento do quadro do personagem baseado na predição de movimento. Para retornar o conteúdo pré-processado do jogo para o usuário, o usuário deve executar um comando, e se esse comando corresponder à predição calculada, o Cloud Manager envia o conteúdo já pré-processado, caso contrário, o Cloud Manager não envia o conteúdo pré-processado e ao invés disso, o Sistema em Cloud realiza o processo correspondente ao comando realizado pelo cliente.

A última informação de cada camada é armazenada no servidor da nuvem, dessa forma, o sistema precisa combinar os dados pré-codificados com dados codificados em tempo real.

5.2 Criação da Matriz de predição

A matriz de predição é construída a partir do ambiente do jogo, neste trabalho, a matriz de predição será denominada matriz T.

Dentro do mapa do jogo, os valores dos parâmetros da altura G e comprimento J são obtidos a partir das coordenadas dos componentes que estão nas extremidades de cima, de baixo, da esquerda e da direita do mapa do jogo. O parâmetro altura G é a distância entre o componente que está na extremidade superior e o componente que está na extremidade inferior. O parâmetro comprimento J é a distância entre o componente que está na extremidade esquerda e o componente que está na extremidade direita, como mostra a Figura 11.

Figura 11 - Mapa do jogo com altura G e largura J.

O valor do parâmetro Δ é o deslocamento unitário do personagem em uma das quatro direções: direita, esquerda, cima ou embaixo, como mostra a Figura 12.

Figura 12 - Representação do personagem no círculo cinza central e a distância Δ quando ele se locomove uma distância unitária em uma das quatro direções.

Com os valores de G , J e Δ , pode-se criar a matriz de predição T . A quantidade de colunas N e a quantidade de linhas M , podem ser calculadas através das seguintes fórmulas:

$$M = \left\lceil \frac{G}{\Delta} \right\rceil$$

$$N = \left\lceil \frac{J}{\Delta} \right\rceil$$

Em toda matriz, a quantidade de linhas e colunas deve ser um número inteiro, por isso, os resultados de M e N foram arredondados para cima.

A quantidade total de células que a matriz T possui é dada pela fórmula:

$$T = M * N$$

A matriz T é mostrada na Figura 13, onde $n = N-1$ e $m = M-1$.

Figura 13 - Representação da matriz de predição T

Para realizar a predição de movimento, é atribuído um vetor $\vec{V}_{i,j}$, para cada célula da matriz T, que representa o possível próximo movimento do personagem. O valor inicial de $\vec{V}_{i,j}$ depende se o Cloud Manager utiliza um treinamento prévio, caso contrário os valores das células da matriz T são inicializados como (0,0).

O vetor $\vec{V}_{i,j}$, na célula (i, j), pode ser representado por $v_{i,j} = (x_{i,j}, y_{i,j})$ de acordo com a Figura 14.

Figura 14 - Vetor $\vec{V}_{i,j}$ na coordenada cartesiana (x, y).

5.3 Predição de movimento utilizando matriz de predição

No plano do personagem, foi utilizada a predição de movimento utilizando uma matriz de predição que armazena o possível próximo movimento do personagem.

5.3.1 Movimento do personagem

Quando a matriz T é criada, todas as células terão como valor padrão $v_{i,j} = (0,0)$, como mostra a Figura 15.

	0	1	2	...	n
0	(0,0)	(0,0)	(0,0)	...	(0,0)
1	(0,0)	(0,0)	(0,0)	...	(0,0)
2	(0,0)	(0,0)	(0,0)	...	(0,0)
⋮	⋮	⋮	⋮	⋱	(0,0)
m	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)

Figura 15- Inicialização da matriz T com todas as células valendo (0,0).

Toda vez que o personagem se locomove, o valor do vetor $v_{i,j} = (x_{i,j}, y_{i,j})$ deverá ser atualizado. Dado que o personagem está na posição $(i1, j1)$ com o valor de vetor $v_{i1,j1} = (x_{i1,j1}, y_{i1,j1})$ e vá para a célula $(i2, j2)$, o novo valor do $v'_{i1,j1} = (x'_{i1,j1}, y'_{i1,j1})$ será calculado por:

$$x'_{i1,j1} = x_{i1,j1} + (i2 - i1)$$

$$y'_{i1,j1} = y_{i1,j1} + (j2 - j1)$$

Como a quantidade de movimentos é limitada, pode-se resumir o cálculo de $x'_{i1,j1}$ e $y'_{i1,j1}$ para oito casos (cima, baixo, esquerda, direita e as diagonais), como mostra a Tabela 3.

Tabela 3 - Valores de $x'_{i1,j1}$ e $y'_{i1,j1}$ baseado no movimento realizado pelo personagem.

Movimento	Novo valor de $x'_{i1,j1}$ e $y'_{i1,j1}$
Para direita	$x'_{i1,j1} = x_{i1,j1} + 1$ $y'_{i1,j1} = y_{i1,j1} + 0$
Para diagonal superior direita	$x'_{i1,j1} = x_{i1,j1} + 1$ $y'_{i1,j1} = y_{i1,j1} + 1$
Para cima	$x'_{i1,j1} = x_{i1,j1} + 0$ $y'_{i1,j1} = y_{i1,j1} + 1$
Para diagonal superior esquerda	$x'_{i1,j1} = x_{i1,j1} - 1$ $y'_{i1,j1} = y_{i1,j1} + 1$
Para esquerda	$x'_{i1,j1} = x_{i1,j1} - 1$ $y'_{i1,j1} = y_{i1,j1} + 0$
Para diagonal inferior esquerda	$x'_{i1,j1} = x_{i1,j1} - 1$ $y'_{i1,j1} = y_{i1,j1} - 1$
Para baixo	$x'_{i1,j1} = x_{i1,j1} + 0$ $y'_{i1,j1} = y_{i1,j1} - 1$
Para diagonal inferior direita	$x'_{i1,j1} = x_{i1,j1} + 1$ $y'_{i1,j1} = y_{i1,j1} - 1$

Um exemplo do personagem se movimentando no jogo, partindo da célula (1,1), é mostrado na Figura 16.

Figura 16 - Exemplo de movimento do personagem. (a) Personagem inicia no ponto (1,1). (b) Personagem moveu para direita. (c) Personagem moveu para diagonal inferior esquerda. (d) Personagem moveu para cima. (e) Personagem moveu para direita. (f) Personagem moveu para diagonal inferior direita.

No início do exemplo mostrado na Figura 16(a), a matriz T é inicializada com todas as células valendo $(0,0)$ e o personagem na posição inicial $(1,1)$, ou seja, o valor de $v_{1,1} = (0,0)$.

Na Figura 16(b), o personagem anda para direita, saindo da célula $(1,1)$ para a célula $(2,1)$, com isso, o valor na célula $(1,1)$ muda de $(0,0)$ para $(1,0)$, de acordo com a Tabela 3, ou seja, o novo valor de $v_{1,1}$ será $v_{1,1} = (1,0)$.

Na Figura 16(c), o personagem anda para diagonal inferior esquerda, saindo da célula $(1,2)$ para a célula $(2,1)$, com isso, o valor na célula $(1,2)$ muda de $(0,0)$ para $(-1,-1)$, de acordo com a Tabela 3, ou seja, o novo valor de $v_{1,2}$ será $v_{1,2} = (-1,-1)$.

Na Figura 16(d), o personagem anda para cima, saindo da célula $(2,1)$ para a célula $(1,1)$, com isso, o valor na célula $(2,1)$ muda de $(0,0)$ para $(0,1)$, de acordo com a Tabela 3, ou seja, o novo valor de $v_{2,1}$ será $v_{2,1} = (0,1)$.

Na Figura 16(e), o personagem anda para direita, saindo da célula $(1,1)$ para a célula $(2,1)$, com isso, o valor na célula $(1,1)$ muda de $(1,0)$ para $(2,0)$, de acordo com a Tabela 3, ou seja, o novo valor de $v_{1,1}$ será $v_{1,1} = (2,0)$.

Na Figura 16(f), o personagem anda para diagonal inferior direita, saindo da célula $(1,2)$ para a célula $(2,3)$, com isso, o valor na célula $(1,2)$ muda de $(-1,-1)$ para $(0,-2)$, de acordo com a Tabela 3, ou seja, o novo valor de $v_{1,2}$ será $v_{1,2} = (0,-2)$.

Notar que nos casos Figura 16 (e) e Figura 16 (f), o valor nas células não valem $(0,0)$ e que os novos valores das células dependem dos valores antigos contidos na mesma.

5.3.2 Predição de movimento

Para realizar a predição de movimento, é analisada a célula (i,j) em que o personagem está presente. Considerando o valor da célula como $v_{i,j} = (x_{i,j}, y_{i,j})$, será calculado o valor do comprimento r e do ângulo θ , a predição de movimento será realizada se o valor r for diferente de 0, caso contrário, se r valer 0, não será feita a predição de movimento.

Para calcular o valor do comprimento r e do ângulo θ são usadas as seguintes fórmulas:

$$r = \sqrt{x_{i,j}^2 + y_{i,j}^2}$$

$$\theta = \arctg\left(\left|\left(\frac{y_{i,j}}{x_{i,j}}\right)\right|\right) \quad , \text{ se } x_{i,j} > 0 \text{ e } y_{i,j} \geq 0$$

$$\theta = 360^\circ - \arctg\left(\left|\left(\frac{y_{i,j}}{x_{i,j}}\right)\right|\right) \quad , \text{ se } x_{i,j} > 0 \text{ e } y_{i,j} < 0$$

$$\theta = 180^\circ - \arctg\left(\left|\left(\frac{y_{i,j}}{x_{i,j}}\right)\right|\right) \quad , \text{ se } x_{i,j} < 0 \text{ e } y_{i,j} \geq 0$$

$$\theta = 180^\circ + \arctg\left(\left|\left(\frac{y_{i,j}}{x_{i,j}}\right)\right|\right) \quad , \text{ se } x_{i,j} < 0 \text{ e } y_{i,j} < 0$$

$$\theta = 90^\circ \quad , \text{ se } x_{i,j} = 0 \text{ e } y_{i,j} > 0$$

$$\theta = 270^\circ \quad , \text{ se } x_{i,j} = 0 \text{ e } y_{i,j} < 0$$

O comprimento r representa o nível de confiança do vetor $\overrightarrow{V_{i,j}}$, uma vez que, quanto maior o valor de r menor será a influência da soma de um novo vetor.

O ângulo θ é usado para fazer a predição do próximo movimento. Dado que o personagem pode se locomover para uma quantidade limitada de direções, sendo estes os oito movimentos (cima, baixo, esquerda, direita e as diagonais), a predição do movimento é definida pelo ângulo de movimento mais próximo ao ângulo θ do vetor $\overrightarrow{V_{i,j}}$, como mostra a Figura 17 e a Tabela 4.

Figura 17 – As flechas pretas representam os possíveis movimentos e as linhas pontilhadas representam a divisória para classificar o ângulo mais próximo.

Esses ângulos das linhas pontilhadas presentes na Figura 17, foram escolhidas, pois são exatamente a metade entre dois movimentos próximos.

Por exemplo, dado que o ângulo θ vale 78° que está entre os movimentos para cima de 90° e para diagonal superior direita de 45° , este ângulo está mais próximo do ângulo de 90° do que do ângulo de 45° , uma vez que, está no intervalo entre $67,5^\circ$ e $112,5^\circ$, portanto a predição de movimento é para cima.

Tabela 4 - Predição de movimento baseado no ângulo θ do vetor $\vec{V}_{i,j}$.

Ângulo θ do vetor $\vec{V}_{i,j}$	Predição de movimento
$0^\circ \leq \theta < 22,5^\circ$	Para direita
$22,5^\circ \leq \theta < 67,5^\circ$	Para diagonal superior direita
$67,5^\circ \leq \theta < 112,5^\circ$	Para cima
$112,5^\circ \leq \theta < 157,5^\circ$	Para diagonal superior esquerda
$157,5^\circ \leq \theta < 202,5^\circ$	Para esquerda
$202,5^\circ \leq \theta < 247,5^\circ$	Para diagonal inferior esquerda
$247,5^\circ \leq \theta < 292,5^\circ$	Para baixo
$292,5^\circ \leq \theta < 337,5^\circ$	Para diagonal inferior direita
$337,5^\circ \leq \theta < 360^\circ$	Para direita

Para exemplificar o jogo usando a predição de movimento, foi feito um exemplo que é demonstrado na Figura 18.

	0	1	2	3	4	5	6	... n
0	(2,-9)	(1,-5)	(3,-4)	(-5,-8)	(0,0)	(0,0)	(5,-2)	...
1	(4,-8)	(5,-5)	(-1,9)	(-2,-5)	(0,0)	(0,0)	(23,3)	...
2	(9,-2)	(23,-2)	(30,-2)	(34,-4)	(34,0)	(34,0)	(32,2)	...
3	(11,3)	(13,-5)	(4,-20)	(-3,-21)	(0,0)	(0,0)	(3,-9)	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
m								

(a)

Posição do personagem: (1,1)

Valor do vetor: (5,-5)

Ângulo θ : 315°

Predição de movimento:

● Para diagonal inferior direita

	0	1	2	3	4	5	6	... n
0	(2,-9)	(1,-5)	(3,-4)	(-5,-8)	(0,0)	(0,0)	(5,-2)	...
1	(4,-8)	(6,-5)	(-1,9)	(-2,-5)	(0,0)	(0,0)	(23,3)	...
2	(9,-2)	(23,-2)	(30,-2)	(34,-4)	(34,0)	(34,0)	(32,2)	...
3	(11,3)	(13,-5)	(4,-20)	(-3,-21)	(0,0)	(0,0)	(3,-9)	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
m								

(b)

Posição do personagem: (1,2)

Valor do vetor: (-1,9)

Ângulo θ : $96,34^\circ$

Predição de movimento:

● Para cima

	0	1	2	3	4	5	6	... n
0	(2,-9)	(1,-5)	(3,-4)	(-5,-8)	(0,0)	(0,0)	(5,-2)	...
1	(4,-8)	(5,-5)	(0,8)	(-2,-5)	(0,0)	(0,0)	(23,3)	...
2	(9,-2)	(23,-2)	(30,-2)	(34,-4)	(34,0)	(34,0)	(32,2)	...
3	(11,3)	(13,-5)	(4,-20)	(-3,-21)	(0,0)	(0,0)	(3,-9)	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
m								

(c)

Posição do personagem: (2,3)

Valor do vetor: (34,-4)

Ângulo θ : $6,71^\circ$

Predição de movimento:

● Para direita

	0	1	2	3	4	5	6	... n
0	(2,-9)	(1,-5)	(3,-4)	(-5,-8)	(0,0)	(0,0)	(5,-2)	...
1	(4,-8)	(5,-5)	(-1,9)	(-2,-5)	(0,0)	(0,0)	(23,3)	...
2	(9,-2)	(23,-2)	(30,-2)	(35,-4)	(34,0)	(34,0)	(32,2)	...
3	(11,3)	(13,-5)	(4,-20)	(-3,-21)	(0,0)	(0,0)	(3,-9)	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
m								

(d)

Posição do personagem: (2,4)

Valor do vetor: (34,0)

Ângulo θ : 0°

Predição de movimento:

● Para direita

Figura 18 - Exemplo do algoritmo de predição de movimento sendo utilizado. (a) Posição inicial do personagem. (b) Personagem moveu para direita. (c) Personagem moveu para diagonal inferior direita. (d) Personagem moveu para direita.

Na Figura 18, um exemplo de predição de movimento é apresentado, utilizando a matriz de predição T . Neste caso, a matriz T já está preenchida com valores, mostrando um estado após vários movimentos feito pelo personagem no jogo.

Na Figura 18(a), o personagem começa na posição inicial na célula (1, 1), com valor de (5, -5). O valor do ângulo θ calculado vale 315° e conferindo a Tabela 4, a predição de movimento é para diagonal inferior direita.

Na Figura 18(b), mostra que o personagem se locomoveu para a direita, resultando na diferença entre a predição com o movimento realizado, com isso, o quadro pré-processado é descartado e o processamento do jogo deve ser feito normalmente. O novo valor da célula (1, 1) mudará de (5, 5) para (6, 5), de acordo com a Tabela 3. O personagem move para a célula (1, 2) com o valor de (-1, 9), o valor do ângulo θ calculado vale $96,34^\circ$ e conferindo a Tabela 4, a predição de movimento é para cima.

Na Figura 18(c), mostra que o personagem se locomoveu para a diagonal inferior direita, resultando na diferença entre a predição com o movimento realizado, com isso, o quadro pré-processado é descartado e o processamento do jogo deve ser feito normalmente. O novo valor da célula (1, 2) mudará de (-1, 9) para (0, 8), de acordo com a Tabela 3. O personagem move para a célula (2, 3) com o valor de (34, -4), o valor do ângulo θ calculado vale $6,71^\circ$ e conferindo a Tabela 4, a predição de movimento é para direita.

Na Figura 18(d), mostra que o personagem se locomoveu para a direita, como o movimento foi mesmo que o movimento predito, o quadro pré-processado é enviado para o cliente. O novo valor da célula (2, 3) mudará de (34, -4) para (35, -4), de acordo com a Tabela 3. O personagem move para a célula (2, 4) com o valor de (34, 0), o valor do ângulo θ calculado vale 0° e conferindo a Tabela 4, a predição de movimento é para direita.

Notar que cada célula tem o seu próprio valor de vetor e elas são independentes entre si, fazendo com que a predição de movimento dependa apenas da célula em que o personagem está presente.

Quanto mais o personagem se locomover pelo jogo, mais valores serão adicionados a matriz T e consequentemente melhor será a precisão da predição de movimento.

5.4 Melhorias na predição de movimento

A fim de melhorar o algoritmo de predição de movimento, duas propostas foram apresentadas.

5.4.1 Condições para o pré-processamento

Considerar o seguinte caso, o personagem realiza dez movimentos e o algoritmo de predição prevê corretamente o 2º, 3º, 5º, 7º e 9º movimento, realizando o pré-processamento desses quadros. Enquanto que o tempo médio de resposta é menor, ao usar o algoritmo de predição, a variação do tempo de resposta pode causar o *jitter* gerando uma sensação de desconforto para o usuário, como descrito na seção Efeito da latência e do *jitter* na performance dos jogadores.

Para tentar diminuir essa variação no tempo de resposta são propostos dois processos que realizam o pré-processamento em condições diferentes para cada caso:

- Realizar o pré-processamento após três predições corretas consecutivas;
- Realizar o pré-processamento após cinco predições corretas consecutivas;

Os valores de três e cinco predições foram escolhidos empiricamente.

Antes do jogo ser executado, apenas uma das três opções pode ser escolhida e não pode ser trocada ao longo do jogo: realizar diretamente, realizar após três predições corretas consecutivas ou realizar após cinco predições consecutivas.

Realizar o pré-processamento após três predições corretas consecutivas significa que o pré-processamento será realizado para as próximas predições apenas se três movimentos consecutivos tiverem seus movimentos preditos corretamente e caso uma predição for incorreta, o pré-processamento só passará a ser realizado novamente quando forem feitas três predições corretas consecutivas novamente. A mesma lógica vale para realizar o pré-processamento após cinco predições corretas consecutivas.

Começar a realizar o pré-processamento após acertar três ou cinco predições consecutivas implica em um tempo médio de resposta maior, uma vez que são realizados menos quadros de pré-processamento.

5.4.2 Matriz de Treinamento

Uma forma de melhorar o algoritmo de predição de movimento é incorporando a matriz de treinamento com a matriz de predição. A matriz de treinamento é representada, nesta dissertação, por uma matriz U com dimensões iguais à matriz T . Para cada célula é atribuído o valor de um vetor $\overrightarrow{P_{i,j}}$.

A matriz de treinamento pode ser criada de várias formas distintas que geram resultados distintos:

- O caminho mais percorrido por todos os jogadores que terminaram o percurso.
- A soma de todas as matrizes T geradas pelos jogadores que terminaram o percurso.
- O caminho com o menor número de movimentos do ponto de partida até a chegada.

Após a criação da matriz de treinamento U , é necessário incorporar a matriz U junto à matriz de predição T .

A Figura 19 mostra um exemplo da criação de uma matriz U e da união de uma matriz de treinamento U com uma matriz de predição T .

	0	1	2	3	4	5	6	7	8	9	10	... n
0								→	→	↓		...
1		→	→	→	↓			↑		↓		...
2					→	→	→	↑		↓		...
3										●		...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
m												⋮

(a)

	0	1	2	3	4	5	6	7	8	9	10	... n
0	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(5,0)	(5,0)	(0,-5)	(0,0)	...
1	(0,0)	(5,0)	(5,0)	(5,0)	(0,-5)	(0,0)	(0,0)	(0,5)	(0,0)	(0,-5)	(0,0)	...
2	(0,0)	(0,0)	(0,0)	(0,0)	(5,0)	(5,0)	(5,0)	(0,5)	(0,0)	(0,-5)	(0,0)	...
3	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
m												⋮

(b)

	0	1	2	3	4	5	6	7	8	9	10	... n
0	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(5,0)	(5,0)	(0,-5)	(0,0)	...
1	(0,0)	(5,0)	(5,0)	(5,0)	(0,-5)	(0,0)	(0,0)	(0,5)	(0,0)	(0,-5)	(0,0)	...
2	(0,0)	(0,0)	(0,0)	(0,0)	(5,0)	(5,0)	(5,0)	(0,5)	(0,0)	(0,-5)	(0,0)	...
3	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
m												⋮

(c)

Figura 19 – União entre a matriz de predição T e a matriz de treinamento U. (a) Caminho utilizado para o treinamento. (b) Matriz de treinamento U. (c) União entre a Matriz de predição T com a Matriz de treinamento U.

Na Figura 19(a), é definido o caminho para realizar o treinamento, que tem o ponto de partida no ponto (1,1) e o ponto de chegada no ponto (3,9).

Na Figura 19(b), é criada a matriz de treinamento U. Neste exemplo, foi definido de maneira empírica que o comprimento para cada vetor é de cinco unidades.

Na Figura 19(c), foi realizada a soma das matrizes T e U, como os valores das células de T iniciais são todas (0,0), pode-se assumir que os novos valores das células de T serão os mesmos que os valores das células de U.

6 VALIDAÇÃO E RESULTADOS

Para medir o desempenho do Cloud Manager, são avaliados os componentes do sistema Uniquitous individualmente (LUO, CLAYPOOL; 2015). Foram medidas as taxas de quadros no servidor e no cliente em diferentes configurações do sistema. A Figura 20 mostra onde é implementado o Cloud Manager dentro do servidor do Uniquitous.

Figura 20 - Uniquitous com o Cloud Manager.

Para a validação dos resultados, foram realizados 140 testes com o jogo “Angry Bots” do ponto de início até o ponto de chegada de formas diferentes para cada teste. Nos testes foram gravados os números de quadros processados pelo codificador JPEG e o tempo de processamento de cada quadro. O tamanho de janela do jogo foi de 640x480 em todos os testes.

6.1 CONFIGURAÇÃO

A máquina que foi utilizada como servidor possui o processador AMD FX-6300, 3.5GHz, 8mb, Six-Core e placas gráficas Radeon Sapphire R9 270X Dual-X OC 2GB GDDR5 256Bits, executando uma edição de 64 bits do Windows 7 Enterprise com 16 GB de RAM.

A máquina do cliente possui processador Intel 3.4GHz i5-2410 e placas gráficas integrada HD Graphics Family 2000, executando uma edição de 64 bits do Windows 7 Home Premium com 4 GB de RAM.

O projeto de jogo Unity usado para testes é fornecido pela Unity Technologies, listado abaixo:

- O projeto “Angry Bots” (Versão 4.0).

6.2 PROJETO UNITY

O Projeto Unity é o componente que contém o jogo Unity, a janela do jogo e o áudio. Sem configurar o Uniquitous, o projeto do jogo na IDE do Unity foi executado para um número fixo de quadros e foi observado o desempenho da CPU usando a unidade Profiler no editor. O tempo de CPU é proveniente das atividades nos *scripts* Unity IDE e Unity. De acordo com as estatísticas do Profiler, as atividades incluem renderização, *scripts*, física, coleta de lixo (*garbage collection*), *VSync* e outros. O tempo total de CPU dessas atividades é definido como tempo de CPU não-Uniquitous.

6.3 JANELA DE JOGO

A Janela do Jogo é uma interface que apresenta todo o conteúdo processado pela câmera do Unity. A câmera decide qual tipo de renderização a cena do jogo usa. O

tempo de processamento da CPU também pode ser capturado pelo Unity Profiler. Neste trabalho, todos os testes foram feitos com o mesmo tamanho de tela, para evitar diferentes tempos de renderização e poder avaliar de maneira objetiva a influência do Cloud Manager no sistema.

6.4 CAPTURA DE TELA DO UNITY

Para capturar a tela que é enviada para o cliente, o componente Captura de Tela do Unity pode utilizar os seguintes métodos: o método *Texture2D.BackgroundReadPixels* que lê os pixels do plano de fundo da janela do jogo do editor Unity junto com o método *Texture2D.CharacterReadPixels* que lê os pixels do plano do avatar da janela do jogo do editor Unity. O outro método é o *Texture2D.ReadPixels* que lê os pixels da tela na janela do jogo do editor Unity quando a técnica de implementação por camadas não é utilizada. Em todos os casos, os dados são salvos no formato *Texture2D* de 24 bits para uso posterior no estágio de compressão. O tempo de captura da tela é definido a partir do momento em que o *Texture2D.ReadPixels*, *Texture2D.BackgroundReadPixels* ou *Texture2D.CharacterReadPixels* começa a ser executado até a hora em que os dados de pixel no formato *Texture2D* estão disponíveis. A propriedade *Time.realTimeSinceStartup* é utilizada para coletar o registro da data/hora. O registro Tb e o registro Te são colocados antes e depois do processo de captura de tela do Unity, respectivamente, o tempo que leva para capturar a tela é dado por Te-Tb.

Tabela 5 - Tempo médio de captura de tela do Unity e erro padrão no projeto “Angry Bots”.

Resolução	Tempo médio (ms)	Erro padrão
640x480	9.32	0.019

6.5 CODIFICAÇÃO DE IMAGEM

O tempo de processamento do codificador de imagem limita a velocidade com que o servidor Uniquitous processa cada quadro e, portanto, controla a taxa de quadros

de envio. A taxa de quadros do jogo para os jogadores no cliente é igual à taxa de envio no servidor.

Foi avaliado o tempo de codificação JPEG adicionando registros de tempo antes e depois da codificação JPEG. Quando a codificação é iniciada, o método *Datetime.Now* é chamado para gravar o tempo de início T_b e o tempo de término T_e que foi gravado assim que o segmento de codificação sinalizou que o processo terminou. Foi definido que diferença entre T_e e T_b é o tempo que leva para fazer a codificação da imagem.

No nível do aplicativo, o tempo de codificação JPEG depende da quantidade dos dados de pixel armazenados na imagem. Existem duas variáveis independentes que afetam o tamanho da imagem: o fator de qualidade (Q) e o tamanho de janela do jogo (R). O fator de qualidade foi definido como 10 e o tamanho de janela como 640x480.

6.6 TRANSMISSÃO DE IMAGEM

O tempo de transmissão da imagem é o tempo que leva a função RPC para processar os dados compactados da imagem do jogo e enviar os dados. Uma vez que a transmissão de imagens lida apenas com os dados comprimidos das imagens em diferentes tamanhos, os resultados da avaliação podem ser aplicados em todos os projetos testados.

Depois que a imagem do jogo é codificada pelo codificador JPEG, os dados de pixel armazenados no array de bytes de saída do codificador são enviados através da RPC não-confiável fornecida pelo uLink. Foi executado um script inicializando um array de bytes de dados e logo após foi chamado a RPC que leu e transmitiu o array de bytes no servidor Uniquitous. O tamanho do array de bytes em cada rodada variou entre 0 e 65480 bytes. Enquanto isso, o Unity Profiler foi usado para medir o tempo de CPU da RPC durante a execução na função Update por vários segundos em cada rodada. RPC é chamada uma vez por quadro e foi coletado 20 tempos de processamento RPC dos 20 quadros em cada rodada. A quantidade máxima de dados transmitidos não pode exceder 65480 bytes porque o comprimento do datagrama UDP é um inteiro de 16 bits e deve incluir o cabeçalho UDP.

6.7 RECEPÇÃO DA ENTRADA DE DADOS

O tempo de processamento do componente Recepção da Entrada de Dados é o tempo que leva a função RPC para receber e processar os dados de controle de entrada. A quantidade de dados recebidos por RPC não-confiáveis depende das interações dos jogadores com o jogo no Unity e varia de acordo com os diferentes tipos de gêneros de jogo. No entanto, as entradas de controle mais comuns são posições do mouse, cliques do mouse, teclas apertadas no teclado e conteúdo de texto. Eles são representados por strings, inteiros e valores booleanos no Unity e podem ser convertidos para bytes por RPC para transmissão. O comprimento dos dados dessas entradas é menor do que 1000 bytes. Os dados de entrada de controle demoram menos do que 0,1 milissegundos para processar, por isso ele não é considerado no tempo de processamento para receber entradas de controle de jogo no servidor Uniquitous.

6.8 ANGRY BOTS

Para validar a proposta, foi utilizado o jogo “Angry Bots”, ilustrado na Figura 21. O jogo “Angry Bots” baseia-se numa demonstração técnica e é fornecido livremente com o Unity. O jogo é do gênero ação com ponto de vista onipresente.

O código do jogo presente no Unity é separado em camadas de plano de fundo e de personagem, facilitando a implementação por camadas.

Dentro do jogo, os inimigos foram retirados do jogo, permitindo analisar apenas o plano do personagem e o plano de fundo com o cenário estático.

A ação de disparar do personagem foi desabilitada para realizar a validação considerando apenas a movimentação do personagem.

A Figura 22 mostra o mapa do jogo “Angry Bots” através do modo de edição do Unity. Uma possível trajetória que o personagem pode percorrer no mapa é mostrada pela linha azul e que é usada para o treinamento. O personagem partirá do ponto verde e o teste terminará quando o personagem chega no ponto laranja.

Figura 21 - Imagem do jogo “Angry Bots” do gênero ação com visão onipresente.

Figura 22 - Caminho pré-definido marcado em azul, que o personagem percorre dentro do mapa.

A matriz de predição gerada para os testes possui largura J igual a 158,0171, com altura G igual a 102,1862 e o valor de Δ é 1,2311. Com isso, o valor de N é 129 e o de M é 83.

Para cada rodada executada, o personagem principal se locomoveu do ponto de partida até o ponto de chegada e foram gravados os números de quadros processados pelo codificador JPEG e o tempo de processamento de cada quadro. Foram

calculadas as médias dos tempos de resposta em milissegundos para cada caso, conforme mostrado na Tabela 6, e calculada a relação entre a quantidade de predições de movimentos e o acerto das previsões em cada caso, como mostra a Tabela 7.

Para esta dissertação, assim como na Figura 19, o treinamento foi feito a partir de um conjunto de movimentos pré-determinados do ponto de partida até o ponto de chegada, mostrado na Figura 22. O comprimento dos vetores na matriz de treinamento foi definido de maneira empírica como cinco unidades. A quantidade de movimentos realizados no treinamento foi 234, os quais foram realizados de acordo com o caminho pré-definido pela linha azul na Figura 22.

Foram adotados os seguintes casos nos testes:

- Caso 1: Sem o Cloud Manager.
- Caso 2: Utilizando o pré-processamento com a predição de movimento.
- Caso 3: Utilizando o pré-processamento após acertar três predições de movimento consecutivas.
- Caso 4: Utilizando o pré-processamento após acertar cinco predições de movimento consecutivas.
- Caso 5: Utilizando um treinamento prévio e o pré-processamento com a predição de movimento.
- Caso 6: Utilizando um treinamento prévio e o pré-processamento após acertar três predições de movimento consecutivas.
- Caso 7: Utilizando um treinamento prévio e o pré-processamento após acertar cinco predições de movimento consecutivas.

Para avaliar o desempenho do algoritmo de predição de movimento foi calculada a precisão da predição de movimentos. A precisão denota a proporção entre os casos positivos corretos e os casos positivos previstos (POWERS, 2011).

Para medir a precisão da predição realizada foi utilizada a seguinte fórmula:

$$Precisão = \frac{\text{Quantidade de movimentos com predição correta}}{\text{Quantidade de quadros pré-processados realizados}}$$

Tabela 6 - Valor médio do tempo de resposta.

Caso	Valor médio do tempo de resposta (ms)	Desvio padrão	Melhora percentual do tempo de resposta em relação ao Caso 1(%)
Caso 1	67,98	1,98	0
Caso 2	61,81	1,87	9,08
Caso 3	63,98	1,88	5,88
Caso 4	65,22	1,99	4,06
Caso 5	46,98	1,89	30,89
Caso 6	53,07	1,90	21,93
Caso 7	65,10	1,95	4,24

Tabela 7 – Precisão do algoritmo de predição de movimento.

Caso	Quantidade média de movimentos por teste	Quantidade de quadros pré-processados realizados	Quantidade de movimentos com predição correta	Precisão da predição de movimento
Caso 2	276	35	30	0,8571
Caso 3	259	19	15	0,7895
Caso 4	267	8	5	0,625
Caso 5	280	80	76	0,95
Caso 6	264	66	59	0,8939
Caso 7	259	35	33	0,9429

O tempo total de resposta apresentou um valor menor quando utilizado o Cloud Manager, pois ao realizar a predição de movimento apenas para o personagem, há o pré-processamento da imagem do próximo quadro, não necessitando ter que fazer o processamento do quadro inteiro.

Os resultados foram de acordo com o esperado, com os casos 3 e 4 terem tempo de resposta médio maior que o caso 2, pois só começam a fazer o pré-processamento quando atingem a condição de acertos consecutivos. O mesmo vale para os casos 6 e 7 em relação ao caso 5.

Ao utilizar o Cloud Manager sem treinamento, o pré-processamento quase não é utilizado, uma vez que o algoritmo de predição de movimento não é executado quando o jogador passa pela primeira vez naquela localização e o jogador raramente passa pelo mesmo ponto e quando o faz, normalmente é apenas uma vez.

O Cloud Manager com o treinamento, no melhor caso, apresentou melhora no tempo de resposta em relação aos outros testes, com uma diminuição de 30,89% no tempo resposta em relação ao tempo de resposta sem o Cloud Manager.

A precisão da predição de movimento teve uma precisão média de 0,8403 que pareceu ser influenciada pelo treinamento realizado neste trabalho.

Apesar do tempo de resposta ser maior no caso da pré-processamento depois de três e cinco acertos, a variação de tempo de resposta em curtos espaços de tempo foi menor o que pode sugerir que houve menos *jitter* nesses casos, ou seja uma melhor QoE.

7 CONCLUSÕES

Nesta dissertação, foi proposto um arcabouço Cloud Manager que melhora o tempo de resposta para execução de jogos em um sistema em Cloud Gaming utilizando dois algoritmos: implementação de camadas e predição de movimento. Para validar a proposta, foram utilizados o sistema Uniquitous de Cloud Gaming e o jogo “Angry Bots” que é um jogo do gênero tiro e que possui ponto de vista onipresente, o cenário tem pouca variação e a quantidade de movimentos que o personagem do jogo pode realizar são limitados.

O melhor resultado obtido teve uma redução de 30,89% no tempo resposta com o arcabouço Cloud Manager em relação ao jogo sendo executado diretamente no Uniquitous.

Uma das limitações da proposta é que, dependendo do gênero e do ponto de vista do jogo, esta solução pode não ser viável ou ter resultados insatisfatórios, visto que em jogos que possuem muita variação de movimentos o pré-processamento não é realizado.

Outra questão que deve ser considerada é se a solução do processamento em dispositivos móveis é viável, uma vez que o consumo de bateria para manter o celular conectado a uma rede sem fio é alto. Se o dispositivo tiver os requisitos de *hardware* para executar o jogo, pode ocasionar de gastar menos processamento e bateria para executar o jogo no próprio dispositivo do que enviar os comandos pela rede sem fio.

A matriz de predição, apresentada nessa dissertação, faz uma análise do comportamento do usuário e busca apresentar uma predição baseada nas ações realizadas anteriormente, tendo em vista isso, a proposta tem potencial para ser utilizada em outras aplicações e áreas, por exemplo, no mercado financeiro como a bolsa de valores.

Para trabalhos futuros, será criada uma base maior de movimentos para realizar outros tipos de arquivos de treinamento, podendo gerar resultados melhores.

Outra pesquisa futura será sobre a influência no QoE nos casos em que o pré-processamento é realizado após três ou cinco acertos consecutivos da predição.

Outra área de estudo será tentar implementar o Cloud Manager em jogos de outros gêneros e pontos de vista e medir a sua eficiência.

REFERÊNCIAS

- Abolfazli S., Sanaei Z., Ahmed E., Gani A., Buyya R.; **Cloud-based augmentation for mobile devices**: motivation, taxonomies, and open challenges. In: IEEE Communication Surveys and Tutorials, 2013.
- Apperley T. H.; **Genre and game studies**: Toward a critical approach to video game genres. *Simulation & Gaming*, 37(1), p. 6-23, 2006.
- Armbrust M., Fox A., Griffith R., Joseph A.D., Katz R.H., Konwinski A., Lee G., Patterson D.A., Rabkin A., Stoica I., Zaharia M.; **Above the Clouds**: A Berkeley View of Cloud Computing. Tech. Rep. UCB/EECS- 2009-28, EECS Department, University of California, Berkeley, 2009.
- Armbrust M., Fox A., Griffith R., Joseph A.D., Katz R.H., Konwinski A., Lee G., Patterson D.A., Rabkin A., Stoica I., Zaharia M.; **A view of cloud computing**. *Communications of the ACM*, p. 50-58, 2010.
- Audio buffer size in Unity:
<http://docs.unity3d.com/ScriptReference/AudioSettings.SetDSPBufferSize.html>,
 Acesso em: 10 novembro 2016.
- Barboza D. C., Muchaluat-Saade D. C., Clua E. W. G.; **A real-time game streaming optimization technique based on layer caching**. In *Consumer Communications and Networking Conference (CCNC)*, 2015 12th Annual IEEE (pp. 714-719). IEEE. 2015.
- Bernier Y. W.; **Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization**. *Game Developers Conference*, 2001.
- Broager A.; **JPEG Encoder Source for Unity in C#**: <https://code.google.com/p/unity-jpeg-encoder/source/browse/tags/Version1/UnityProject/Assets/?r=21>, Acesso em: 05 novembro 2016.
- Carvalho J., Oliveira J., Carvalho P.; **Jogos on-line**: Estudo sobre qoe e qos. In *CRC 2010 – 10ª Conferência sobre Redes de Computadores*, p. 83-88, Universidade do Minho, Braga, Portugal, 2010.
- Chang Y., Tseng P., Chen K., Lei C.; **Understanding the Performance of Thin-Client Gaming**. *Proceedings of IEEE CQR*, 2011.
- Chen K. T., Chang Y. C., Hsu H., Chen D., Huang C. Y., Hsu C.; **On the Quality of Service of Cloud Gaming Systems**. *IEEE Transactions on Multimedia*, vol. 16, nº 2, 2014.
- Chen K. T., Chang Y. C., Tseng P. H., Huang C. Y., Lei C.L.; **Measuring the latency of cloud gaming systems**. In *Proceedings of ACM Multimedia 2011*, 2011.
- Cheng L., Bhushan A., Pajarola R., Zarki M.; **Real-Time 3D Graphics Streaming using MPEG-4**. *IEEE/ACM Workshop on Broadband Wireless Services and Applications*, ICS-UCI, 2004.

Claypool M., Claypool K.; **Latency and Player Actions in Online Games**. Communications of the ACM, vol. 49, no. 11, 2006.

Claypool M.; **Motion and scene complexity for streaming video games**. Proc. 4th ACM Int. Conf. Foundations of Digital Games, p. 34-41, 2009.

Coué C., Pradalier C., Laugier C., Fraichard T., Bessière P.; **Bayesian occupancy filtering for multitarget tracking: an automotive application**. The International Journal of Robotics Research, 25(1), 19-30, 2006.

Cuervo E., Balasubramanian A., Cho D., Wolman A., Saroiu S., Chandra R., Bahl P.; **MAUI: making smartphones last longer with code offload**. Proc. ACM 8th Annual International Conference on Mobile Systems, Applications and Services (MobiSys'10), p. 49-62, 2010.

Eisert P., Fechteler P.; **Low delay streaming of computer graphics**. 15th IEEE International Conference on Image Processing, 2008.

FFMPEG Project: <http://www.ffmpeg.org/about.html>, Acesso em: 05 dezembro 2016.

Gellert A., Vintan, L.; **Person movement prediction using hidden Markov models**. Studies in Informatics and control, p. 15(1), 17. 2006.

Grace L.; Game type and game genre. Disponível em: http://www.lgrace.com/documents/Game_types_and_genres.pdf, 2005.

Guan H., Yao J., Qi Z., Wang R.; **Energy-efficient SLA guarantees for virtualized GPU in cloud gaming**. IEEE Transactions on Parallel and Distributed Systems, 26(9), 2434-2443. 2015.

Harteveld C., Bekebrede G.; **Learning in single-versus multiplayer games: The more the merrier?**. Simulation & Gaming, p. 42(1), 43-63, 2011.

Hemmati M., Javadtalab A., Shirehjini A., Shirmohammadi S., Arici T.; **Game as video: bit rate reduction through adaptive object encoding**. 23rd ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '13). ACM, New York, NY, USA, p. 7-12, 2013.

Hey A.J., Trefethen A.E.; **Cyberinfrastructure for e-Science**. Science 308, p. 817-821, 2005.

Huang C., Hsu C., Chang Y., Chen K.; **Gaminganywhere: An open cloud gaming system**. In: Proceedings of ACM Multimedia Systems. p. 36-47., 2.6, 2013.

Kumar K., Lu Y. H.; **Cloud computing for mobile users: Can offloading computation save energy?**. Computer, 43(4), 51-56. 2010.

Lecky-Thompson G. W.; **Video Game Design Revealed**. Cengage Learning. p. 23, 2008.

Lee K., Chu D., Cuervo E., Degtyarev Y., Grizan S., Kopf J., Wolman A., Flinn J.; **Outatime**: Using Speculation to Enable Low-Latency Continuous Interaction for Mobile Cloud Gaming. In Proc. of MobiSys, 2015.

Luo M., Claypool M.; **Uniquitous**: Implementation and evaluation of a cloud-based game system in unity. Proc. of IEEE GEM, 2015.

Mathy L., Edwards C., Hutchison D.; **Principles of QoS in group communications**. Telecommunication Systems, p. 11(1-2), p. 59–84, 1999.

Myers D.; **The nature of computer games**: Play as semiosis. New York: Peter Lang, 2003.

Mell P., Grance T.; **The nist definition of cloud computing**. In: National Institute of Standards and Technology Special Publication, 2011.

Mishra D., Zarki M. E., Erbad A., Hsu C., Venkatasubramanian N.; **Clouds + games**: A multifaceted approach. IEEE Internet Comput., vol.18, no.3, p. 20-27, 2014.

Noimark Y., Cohen-Or D.; **Streaming scenes to MPEG-4 videoenabled devices**. Computer Graphics and Applications, IEEE, vol.23, no.1, p. 58,64, 2003.

Normoyle A., Guerrero G., Jörg S.; **Player perception of delays and jitter in character responsiveness**. In Proceedings of the ACM Symposium on Applied Perception, ACM, p. 117-124, 2014.

OnLive launches new company to avoid bankruptcy; <http://techland.time.com/2012/08/20/onlive>. Acesso em: 21 maio 2015.

Phan L. T. X., Zhang Z., Zheng Q., Loo B. T., Lee I.; **An empirical analysis of scheduling techniques for real-time cloud-based data processing**. Proceedings of the 2011 IEEE International Conference on Service-Oriented Computing and Applications, 2011.

Powers D. M., **Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation**. Journal of Machine Learning Technologies, 2011.

Rahimi H., Shirehjini A., Shirmohammadi S.; **Context-aware prioritized game streaming**. Multimedia and Expo (ICME). 2011 IEEE International Conference, p. 1,6, 11-15, 2011.

Reddy V.K., Thirumala R.B., Reddy L.S.S., Kiran S.; **Research Issues in Cloud Computing**. Global Journal of Computer Science and Technology, 2011.

Sato A. K. O., Cardoso M. V.; **Além do gênero**: uma possibilidade para a classificação de jogos. VII SBGames - Simpósio Brasileiro de Games e Entretenimento Digital, Belo Horizonte, 2008.

Satyanarayanan M., Bahl P., Caceres R., Davies N.; **The case for vm-based cloudlets in mobile computing.** IEEE Pervasive Comput. 8, p. 14-23, 2009.

SDL; Simple Directmedia Layer; <http://www.libsdl.org/>, Acesso em: 05 novembro 2016

Shiraz M., Gani A., Khokhar R. H., Buyya R.. **A Review on Distributed Application Processing Frameworks in Smart Mobile Devices for Mobile Cloud Computing.** IEEE Communications Surveys & Tutorials, p. 294-313, 2013.

Sood S.K.; **A combined approach to ensure data security in cloud computing.** Journal of Network and Computer Application, 35(6), p. 1831-1838, 2012.

Taylor L. N.; **Video games:** Perspective, point-of-view, and immersion. University of Florida, 2002.

ULink: <http://developer.muchdifferent.com/unitypark/uLink/uLink>, Acesso em: 15 agosto 2016

Unity3d official website: <http://unity3d.com/>, Acesso em: 11 agosto 2016

Warman P.; **Global Monetization of Games: Emerging Markets as Drivers of Growth.** Casual Connect San Francisco, 2013.

Wei C., Victor C. M. L., Min C.; **Next Generation Mobile Cloud Gaming.** 2013 IEEE Seventh International Symposium on Service-Oriented System Engineering, 2013.