UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

FELIPE LANGE SEVERINO

Protegendo a economia virtual de MMOGs através da detecção de cheating

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Cláudio R. Geyer
Orientador

Porto Alegre, Junho de 2012

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Severino, Felipe Lange

Protegendo a economia virtual de MMOGs através da detecção de cheating / Felipe Lange Severino. – Porto Alegre: PPGC da UFRGS, 2012.

76 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2012. Orientador: Cláudio R. Geyer.

1. Simulação distribuída. 2. Jogos Online Massivamente Multijogadores. 3. MMOG. 4. Economia virtual. 5. Cheating. I. Geyer, Cláudio R.. II. Título.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

AC/DC	Algorithm for Cheating Detection by Cheating
CCP	Cheat Controlled Protocol
DACA	Dynamic Anti-Cheating Architecture
DaCaP	Dynamic anti-CheAting Peer to peer
DVE	Distributed Virtual Environment
FPS	First Person Shooter
GPPD	Grupo de Processamento Paralelo e Distribuído
MMOFPS	Massively Multiplayer Online First Person Shooter
MMOG	Massively Multiplayer Online Game
MMORPG	Massively Multiplayer Online Role Playing Game
MMORTS	Massively Multiplayer Online Real Time Strategy
MUD	Multi User Dungeon
NVE	Network Virtual Environment
P2P	Peer-to-peer
PPGC	Programa de Pós-Graduação em Computação
RPG	Role Playing Game
RTS	Real Time Strategy
SOI	Sphere of Influence
UFRGS	Universidade Federal do Rio Grande do Sul
WRD	Weighted Responsible Division

LISTA DE FIGURAS

RESUMO

Nos últimos anos Jogos Online Massivamente Multijogadores (MMOG) têm se expandido em popularidade e investimento, influenciado, especialmente, pela evolução da conexão residencial (conexões mais rápidas a preços mais baixos). Com o crescimento dessa demanda, surgem problemas na utilização da arquitetura cliente-servidor, normalmente utilizada em jogos comerciais. Entre as arquiteturas alternativas de suporte a MMOGs estão as arquiteturas *peer-to-peer*. Porém essas arquiteturas apresentam problemas relativos a segurança, problemas esses que possuem, muitas vezes, soluções de baixo desempenho, sendo impraticáveis em jogos reais. Entre os problemas de segurança mais significativos para MMOGs encontra-se o *cheating*, ou a ação que um ou mais jogador toma para burlar as regras em favor próprio. A preocupação com cheating agrava-se quando o efeito desse *cheating* pode causar danos irreversíveis à economia virtual e, potencialmente, afetar todos os jogadores. O presente trabalho faz uso de uma divisão celular do mundo virtual para restringir o impacto de um dado *cheating* a uma única célula, evitando que este se propague. Para tanto é realizada uma classificação do estado do jogador e utiliza-se uma técnica de detecção de *cheating* para cada uma das classificações. Foram realizados experimentos através de simulação para testes de aplicabilidade do modelo e análise de desempenho e acuracidade. Os testes indicam que o modelo proposto consegue, de forma eficaz, realizar a proteção da economia virtual, impedindo que a ocorrência de um *cheating* atinja todos os jogadores.

Palavras-chave: Simulação distribuída, Jogos Online Massivamente Multijogadores, MMOG, Economia virtual, Cheating.

Protecting the virtual economy in MMOGs by cheat detection

ABSTRACT

In the past few years, Massively Multiplayer Online Games (MMOG) grew in both popularity and investment. This growth has been influenced by the evolution of residential connection (faster and cheaper connections). With the demand, some limitations imposed by the client-server architecture becomes more significant. Peer-to-peer architectures aim to solve those problems by distributing the game among several computers. However, those solutions usually lack security, or presents low performance. Among the problems, cheating can be considered the most significant to MMOGs. Cheating can be defined as the action taken by a player when this action is against the rules. This may be aggravated when this action can cause irreversible damage to the virtual economy and, potentially, affect all players in the virtual world. This work's goal is to restrict the cheating impact using a cellular world division. The proposal is to restrict the cheating in a limited virtual space, preventing the propagation. A state classification is presented, and different cheating detection techniques are presented to each element of this classification. Simulation is used to make the experiments aiming to test the performance and accuracy of the proposal. Results indicate that the proposed solution can efficiently protect the virtual economy, restraining the effects of a cheating occurrence to a small portion of the virtual world.

Keywords: Distributed simulation, Massively Multiplayer Online Games, MMOG, Virtual economy, Cheating.

1 INTRODUÇÃO

1.1 Temática e motivação

Jogos Online Massivamente Multijogadores (*Massively Multiplayer Online Games*), ou MMOGs, tiveram sua origem no final da década de 70 (FUJIMOTO, 1999). Entretanto, sua popularidade aumentou significativamente apenas no final da década de 90, aproveitando a popularização dos computadores pessoais e a redução do custo da conexão residencial de alta capacidade (banda larga). Atualmente, MMOGs contam com uma fatia significativa do mercado de jogos, alcançando bilhões de dólares em movimentação financeira. Jogos como World of Warcraft (BLIZZARD, 2004), EVE Online (CCP, 2003), Lineage II (NCSOFT, 2003) e Guild Wars (ARENANET, 2010) possuem milhões de jogadores em todo o mundo, estando milhares deles online simultaneamente.

Semelhante a diversos sistemas distribuídos atuais, MMOGs têm sua origem em arquiteturas cliente-servidor. No entanto, este modelo de arquitetura prova-se ineficiente e custoso devido ao grande número de usuários. Dessa forma, surgem soluções alternativas, que utilizam arquiteturas distribuídas como *peer-to-peer*. Soluções *peer-to-peer* buscam resolver os problemas apresentados pela arquitetura centralizada através da distribuição da carga computacional entre diversas máquinas, muitas vezes pertencendo aos próprios jogadores. Porém, essas soluções apresentam problemas de segurança ou desempenho, frequentemente requerendo um custo computacional (processamento ou, mais comumente, comunicação) muito alto para garantir certo nível de segurança e, portanto, manter um nível aceitável de satisfação dos jogadores.

A utilização de arquiteturas distribuídas para MMOGs levanta diversas questões relevantes à pesquisa científica. Uma lista, não exaustiva, de problemas ainda sem uma solução eficiente inclui (DARLAGIANNIS; HECKMANN; STEINMETZ, 2006; SCHIELE et al., 2007):

- Distribuição do estado de jogo;
- Manutenção da comunicação eficiente entre diferentes recursos (jogadores ou servidores);
- Distribuição de carga entre os recursos computacionais utilizados;
- Segurança.

A distribuição de carga e as questões de segurança têm sido as temáticas mais abordadas nos últimos anos, existindo diversos trabalhos publicados nesses temas. Levando em consideração a questão de segurança, um dos problemas mais comuns e específicos para

jogos, é denominado cheating, que pode ser definido como o comportamento assumido por um dado jogador para obter uma vantagem ilegal sobre os demais jogadores ou sobre o jogo em si.

O tratamento de cheating torna-se essencial na presença de jogos online, tais como MMOGs, devido à geração de insatisfação de outros jogadores, podendo ocasionar a desistência de muitos deles e, portanto, prejuízo para os desenvolvedores. Devido à existência de uma economia virtual global, as formas de cheating que possam afetar esta economia podem, potencialmente, causar danos irreparáveis ao equilibrio do jogo, gerando insatisfação e desistência de jogadores. Desta forma, assume-se que é essencial a existência de um mecanismo de proteção contra esse tipo de ataque.

1.2 Trabalhos desenvolvidos no PPGC

Diversos trabalhos foram desenvolvidos na área de jogos no Programa de Pós-Graduação em Computação do Instituto de Informática da UFRGS nos últimos anos. Entre eles encontram-se as dissertações de (ALEGRETTI, 2006) e de (GASPARETO, 2008). O primeiro utiliza algoritmos genéticos para implementar um sistema multi-agente para o jogo Quake II. O segundo trabalho faz uso de redes neurais artificiais para detecção de *speed cheating* em MMOGs.

Dentro do Grupo de Processamento Paralelo e Distribuído (GPPD) os trabalhos envolvendo jogos tiveram início em meados de 2004. Entre os trabalhos mais relevantes, encontra-se a proposta de Cecin et al. para criação de um modelo híbrido para suporte a MMOGs (CECIN et al., 2004), culminando na dissertação de mestrado do mesmo autor. Esse trabalho teve continuidade na tese de doutorado defendida pelo mesmo autor em 2009 (CECIN, 2009).

Outros trabalhos desenvolvidos pelo grupo incluem a criação de um modelo P2P para MMOGs (VILANOVA et al., 2008) e, mais recentemente, a dissertação apresentada por Bezerra (2009), que possui como foco o balanceamento de carga em MMOGs utilizando uma arquitetura distribuída.

1.3 Trabalho desenvolvido

Tendo em vista a motivação para proteção de MMOGs contra cheating e seus efeitos na economia virtual, o presente trabalho apresenta uma solução para restrição do efeito de cheating em um espaço virtual denominado célula. Essa restrição é realizada através da detecção do cheating à medida que os jogadores se deslocam no mundo virtual.

A contribuição apresenta-se de duas formas. A primeira é definida como uma solução genérica, passível de aplicação aos mais diversos tipos de jogos existentes (dado um conjunto de características comuns a todos eles). A segunda etapa realiza a aplicação desta solução genérica em soluções específicas para um jogo modelo, ou seja, um jogo que possua características semelhantes a outros MMOGs.

A validação é feita através da simulação do comportamento dos jogadores e dos cheaters, considerando uma escala massiva. Os resultados mostram que a solução apresentada defende de forma eficiente a economia virtual obtendo índices de detecção com níveis semelhantes àqueles apresentados pelo estado da arte.

1.4 Estrutura do texto

O presente trabalho é dividido em seis capítulos, sendo o primeiro essa introdução. O restante possui temática definida abaixo:

- Capítulo 2 - Conceitos: realiza a definição dos principais conceitos utilizados no restante do trabalho, como definição de MMOGs, classificação de arquiteturas de suporte e definição de cheating. Esse capítulo é considerado essencial para a compreensão do trabalho desenvolvido;

- Capítulo 3 - Estado da arte: neste capítulo busca-se dar continuidade a motivação do desenvolvimento do trabalho através da descrição do estado da arte em cheating e MMOGs. Dois principais aspectos são abordados: classificação de cheating e trabalhos que buscam resolver essa problemática;

- Capítulo 4 - Proposta: descreve a contribuição do trabalho. A solução é dividida em duas grandes partes: uma solução genérica, que atinja diversos tipos de jogos, e uma solução específica, provando a aplicabilidade e a eficiência da solução;

- Capítulo 5 - Testes e resultados: exibe os resultados obtidos através da simulação do comportamento dos jogadores. Também é discutido qual o posicionamento do trabalho desenvolvido em relação ao estado da arte;

- Capítulo 6 - Conclusão: sumariza as principais contribuições apresentadas pelo trabalho desenvolvido.

2 CONCEITOS

Neste capítulo apresentam-se os conceitos essenciais para compreensão do trabalho desenvolvido. São apresentados conceitos relacionados a MMOGs e suas especificações relativas à solução desenvolvida.

O capítulo é dividido em três seções: a Seção 2.1 apresenta a definição e caracterização de MMOGs; a Seção 2.2 discute as diferentes arquiteturas de suporte utilizadas em jogos; e, por fim, a Seção 2.4 aborda a definição e características de cheating em jogos, um dos principais aspectos discutidos nesse trabalho.

2.1 MMOG

A presente seção descreve Jogos Online Massivamente Multijogadores (*Massively Multiplayer Online Games*), ou MMOGs. São apresentados a definição de MMOGs como simuladores distribuídos, suas características, um breve histórico e uma classificação dos tipos existentes. Também é realizada a definição do conceito de economia virtual, conceito esse que está diretamente ligado aos jogos atuais.

2.1.1 MMOGs como simuladores

Segundo Fujimoto, simulação computacional é uma computação que modela o comportamento de algum sistema real ou imaginário em função do tempo (FUJIMOTO, 1999). Seguindo este raciocínio, uma simulação computacional distribuída é um sistema computacional que simula um comportamento em múltiplas máquinas. Os principais objetivos para distribuição ou paralelismo de uma simulação são:

- Redução do tempo de execução;
- Integração de simuladores que são executados em máquinas de diferentes fabricantes;
- Tolerância a falhas;
- Distribuição geográfica.

A utilização de tolerância a falhas tem como objetivo garantir a execução da simulação em caso de falhas, como problemas nos componentes computacionais. A distribuição geográfica refere-se à utilização de múltiplos participantes na simulação, interagindo a partir de diferentes locais físicos.

Ainda considerando o tema de simulações, pode-se classificar uma simulação em dois grupos: simulação analítica e ambientes virtuais (FUJIMOTO, 1999). Simulações analíticas buscam fazer uma análise quantitativa do comportamento do ambiente simulado. Esta

simulação, normalmente, é executada buscando-se o menor tempo de execução possível. Simulações analíticas também são caracterizadas pela interação humana mínima ou inexistente. Exemplos para este tipo de simulação são os mais diversos, podendo-se citar a análise de um escalonamento de tarefas em um sistema distribuído ou análise de tráfego em um aeroporto.

Ambientes virtuais, entretanto, são simulações que possuem como objetivo recriar realisticamente um ambiente. Neste tipo de simulação, as interações humanas são, frequentemente, consideradas como parte essencial da simulação. Ao contrário das simulações analíticas, que almejam minimizar o tempo de execução, ambientes virtuais buscam realizar a simulação em tempo real. Este tipo de simulação é usado com frequência para treinamento, como no caso de exércitos ou operação de aeronaves.

Nos últimos anos, ambientes virtuais têm sido utilizados como meio de entretenimento, onde um ou mais participantes interagem em um ambiente virtual fictício, imaginado pelo desenvolvedor ou a partir de alguma obra de ficção conhecida. Estes ambientes permitem a interação de participantes geograficamente distribuídos, sendo estes tipos de simulações conhecidas como *Distributed Virtual Environments* (DVE). No contexto deste trabalho considera-se Jogos Online Massivamente Multijogadores (MMOGs) como um tipo de DVE. Na Seção 2.1.2 são definidas as principais características de MMOGs e as diferenças deste estilo de jogo para jogos online multijogadores.

2.1.2 Caracterização de MMOGs

O conceito de Jogos Online Massivamente Multijogadores, ou *Massively Multiplayer Online Games* (MMOG), ainda não possui uma caracterização definitiva. Entretanto, algumas características são utilizadas para diferenciar MMOGs de jogos multijogadores e de jogos online, a saber:

- Número de jogadores;
- Distribuição geográfica dos jogadores;
- Presença de economia virtual persistente.

Uma das características mais marcantes em MMOGs diz respeito ao número de jogadores. Enquanto a maioria dos jogos multijogadores convencionais se restringe a menos de uma centena de jogadores, nos MMOGs atuais os números quebram a barreira dos milhares de jogadores, muitos deles simultaneamente online.

Além do número de jogadores, outra diferença é a distribuição geográfica destes jogadores. Enquanto muitos jogos multijogadores são restritos a pequenos grupos, normalmente próximos geograficamente (utilizando *lan houses* por exemplo), no caso de MMOGs é possível encontrar-se uma grande distribuição geográfica.

Por fim, a presença de uma economia virtual que persista mesmo quando o jogador não está conectado tornou-se uma característica marcante dos MMOGs nos últimos anos. Na seção 2.1.5 serão discutidos a caracterização e os efeitos da economia virtual em MMOGs.

Apesar de auxiliar na classificação e na definição de um MMOG, as características listadas acima não são exclusivas ou definitivas. Há, por exemplo, a ocorrência de jogos multijogadores que permitem jogadores amplamente distribuídos geograficamente. Além disso, muitos desses conceitos são mutáveis de acordo com o nível tecnológico disponibilizado, como é discutido no breve histórico dos MMOGs, apresentado na Seção 2.1.3.

Outro aspecto que diferencia MMOGs de outros tipos de jogos é o foco em desempenho de rede. Jogos multijogadores possuem uma limitação severa em relação à quantidade de participantes simultâneos, em sua maioria não passando de dezenas de jogadores, sendo possível desenvolver um mecanismo de comunicação menos complexo, porém eficaz. Além disso, considerando um número menor de jogadores é possível obter melhor qualidade de comunicação, ou seja, menor latência na troca de mensagens entre os jogadores e o ambiente. Em MMOGs é necessário um mecanismo complexo de comunicação, que leve em consideração a presença de milhares de jogadores, levando em consideração a necessidade de comunicação de baixa latência.

O conceito de latência de rede é muito utilizado tanto em jogos online quanto em MMOGs. Trata-se do tempo levado entre o envio de uma mensagem até o seu recebimento. A importância da latência em jogos deve-se aos diferentes requisitos de cada jogo ou tipo de jogo, ou seja, alguns tipos de jogos toleram latências maiores sem afetar a percepção do usuário. A Seção 2.1.4 descreve os diferentes requisitos necessários a cada tipo de jogo.

No presente trabalho serão de maior relevância o número de jogadores e a presença de economia virtual, uma vez que o distanciamento geográfico entre jogadores impacta apenas na qualidade do serviço (aumento de latência).

2.1.3 Histórico

Os MMOGs tiveram origem no final da década de 70 com o jogo *Multi-user Dungeon* (MUD) (FUJIMOTO, 1999; WIKIPEDIA, 2010a). Este jogo foi fortemente baseado no jogo *Adventure*, e possui diversas referências ao clássico jogo de interpretação *Dungeons and Dragons* (D&D). Em *Adventure*, o jogador controla um personagem num ambiente virtual através de comandos em um terminal texto como *walk north* e as consequências dessas ações eram narradas ao jogador, também utilizando o modo texto. MUD, por sua vez, atingiu uma alta popularidade entre alunos de graduação que podiam adquirir um computador pessoal (FUJIMOTO, 1999). Em suas últimas versões, MUD estava presente em servidores da Universidade de Essex (Reino Unido) até meados de 1987. O termo MUD, anos mais tarde, passou a ser utilizado como referência a qualquer jogo online multiplayer baseado em texto.

MMOGs gráficos, entretanto, não existiam até 1986, quando foi lançado Air Warrior (WIKIPEDIA, 2010b), um simulador de combate aéreo ambientado na segunda guerra mundial. Disponibilizado exclusivamente para Macs, *Air Warrior* possuía um custo entre 10 a 12 dólares para cada hora de jogo.

Cinco anos mais tarde, a AOL Games (AOL, 2010) lança *Neverwinter Nights*, jogo considerado o primeiro MMORPG gráfico (ver Seção 2.1.4 para descrição de MMORPGs), podendo gerenciar até 50 jogadores simultâneos custando 6 dólares a hora de jogo. Em 1995, devido à sua popularidade, o limite máximo de jogadores simultâneos foi aumentado para comportar até 500 jogadores.

Com a evolução da conexão residencial, bem como a qualidade do hardware disponível para uso pessoal, a partir do final da década de 90, a quantidade e variedade de MMOGs tiveram um considerável crescimento. Atualmente é possível encontrar jogos de diversos tipos, com milhões de jogadores e milhares destes online simultaneamente. Exemplos de MMOGs populares são World of Warcraft (BLIZZARD, 2004) e Eve Online (CCP, 2003), que atualmente divulgam marcas como 11 milhões de inscrições e 54 mil jogadores simultâneos, respectivamente.

2.1.4 Tipos de MMOGs

Esta seção procura descrever os diversos tipos de MMOGs disponíveis atualmente. Tipos de jogos remete à forma de interação dos jogadores entre si e com o ambiente virtual, bem como no estilo de jogo em si (ambiente virtual). O tipo de jogo influencia tecnicamente no desenvolvimento de um jogo. Fatores que são afetados são, por exemplo, a latência máxima necessária para cada jogador, e a quantidade de atualizações de tela (*frames por segundo*).

Os principais tipos de MMOGs são:

- Role Playing Game (RPG);
- First Person Shooter (FPS);
- Real Time Strategy (RTS);
- Estratégia baseado em turnos;
- Simulação;
- Esportes;
- Corrida;
- Casual;
- Música;
- Jogos em Redes Sociais.

Um dos tipos mais populares de MMOGs são os *Massively Multiplayer Online Role Playing Games* (MMORPGs). Esse tipo de jogo possui, em sua maioria, ambientação de fantasia medieval, como os populares World of Warcraft (BLIZZARD, 2004) e Lineage 2 (NCSOFT, 2003), porém há ocorrência de outros gêneros, como super-heróis (e.g. City of Heroes (NCSOFT, 2011)) ou ficção científica (e.g. Star Trek Online (ATARI, 2011)).

Entre os fatores que impulsionam o desenvolvimento deste estilo de jogo pode-se destacar os requisitos técnicos. Em MMORPGs o jogador, normalmente, controla um único personagem (também chamado de avatar) e envia comandos para interação deste personagem com o ambiente virtual. O resultado desta interação é percebido, geralmente, após um intervalo de tempo na grandeza de segundos. Desta forma, os requisitos de rede podem ser relaxados, podendo ser tolerado uma latência maior do que jogos do tipo FPS e requerendo uma largura de banda menor do que jogos do tipo RTS.

Jogos do tipo *First Person Shooter* (FPS) são mais conhecidos por ter como base simulações de guerra como, por exemplo o jogo PlanetSide (Sony Entertainment, 2010). Jogos massivos de FPS (MMOFPS) não são tão comuns devido aos seus requisitos serem mais restritos. O tempo de resposta (latência) deve ser mínimo, para que o jogador tenha a impressão de que suas ações possuem repercussão imediata no mundo virtual, assim como problemas de ordenamento de comandos dos jogadores (quando dois jogadores enviam dois comandos quase simultaneamente, como saber, por exemplo, se o jogador A atirou primeiro ou o jogador B se moveu primeiro).

Finalizando os tipos mais comuns de MMOGs, encontram-se os *Real Time Strategy* (RTS), ou jogos de estratégia em tempo real, ou *Massively Multiplayer Online Real Time*

Strategy (MMORTS). Jogos do estilo RTS consistem em cada jogador controlar múltiplas unidades (frequentemente exércitos), podendo designar uma função diferente a cada uma das unidade. Estes jogos se tornaram muito populares no início da década de 90 com o surgimento do jogo *Age of Empires* (Microsoft Games, 2010), que permitia até 8 jogadores simultâneos. Um exemplo de MMORTS é *Mankind* (O2 Online Entertainment, 2011), lançado em 1998.

De uma forma geral, jogos do estilo RTS toleram latências mais altas do que os demais. Entre os motivos para esta tolerância pode-se incluir o tempo necessário para o jogador tomar uma decisão de ação e enviar seus comandos. Entretanto, devido ao número de mensagens enviadas por cada jogador ser muito mais alto que nos demais tipos de jogos (normalmente uma mensagem para cada unidade controlada), pode-se alcançar centenas de mensagens por turno de jogo, o consumo da largura de banda passa a ser preocupante. Se for considerado a existência de centenas ou milhares de jogadores simultâneos o consumo pode atingir escalas impraticáveis.

Os demais tipos de jogos existem em menores proporções, com destaque para os jogos em redes sociais, que podem ser classificados como um tipo de MMOG, que vêm ganhando uma popularidade considerável nos últimos anos. A Figura 2.1 (WOODCOCK, 2009) ilustra a porcentagem de mercado para diferentes tipos de MMOGs, considerando os gêneros mais populares (fantasia e ficção científica). Pode-se perceber claramente que a maioria de mercado pertence a jogos do tipo MMORPGs.

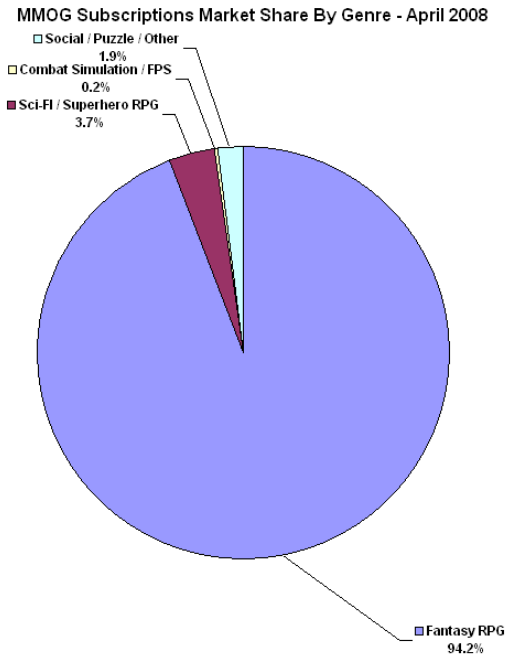

Figura 2.1: *Fatia de mercado de MMOGs por tipo.*

2.1.5 Economia virtual

Com a evolução dos MMOGs surge o conceito de economia virtual. A economia virtual em um MMOG baseia-se nas aquisições e nas transações financeiras feitas pelo jogador. Estas aquisições podem ser realizadas dentro do jogo, com dinheiro virtual, ou utilizando dinheiro real para aquisição de bens virtuais (FAIRFIELD, 2005).

Para que um MMOG possa ser considerado persistente, um dos aspectos a permanecer em constante evolução deve ser a economia virtual. Em muitos casos, como no jogo World of Warcraft por exemplo, o sistema de compra e venda de bens permanece em funcionamento mesmo enquanto o jogador não está presente. Essa forma de funcionamento dá ao jogador uma ilusão de maior realismo na economia do jogo, uma vez que o mundo permanece evoluindo mesmo durante sua ausência.

Com a existência de uma economia virtual há o surgimento de crimes virtuais, que buscam tirar proveito das falhas desse sistema econômico. Um exemplo são os grupos clandestinos que realizam a venda de unidades monetárias virtuais (comumente chamados de moedas de ouro ou algo similar). Esses grupos, em geral, são compostos por dezenas de empregados marginalmente pagos para minerar os recursos que serão vendidos a jogadores (WIKIPEDIA, 2010c). Outro problema são os cheatings, que podem afetar de forma irreparável a economia do jogo. Esse problema em específico (cheating) será detalhado na Seção 2.4.

2.2 Arquiteturas de suporte

Para oferecer suporte à execução da simulação distribuída e ao gerenciamento dos jogadores e do estado do jogo, pode-se utilizar uma gama de diferentes arquiteturas de suporte. Esta seção apresenta uma classificação apresentada por (FUJIMOTO, 1999), que divide as arquiteturas de suporte em cliente-servidor, multiservidor e *peer-to-peer* (P2P), representadas na Figura 2.2 (FUJIMOTO, 1999). A essa classificação é adicionada uma quarta categoria, denominada híbrida, que faz uso de características de arquiteturas cliente-servidor e P2P.

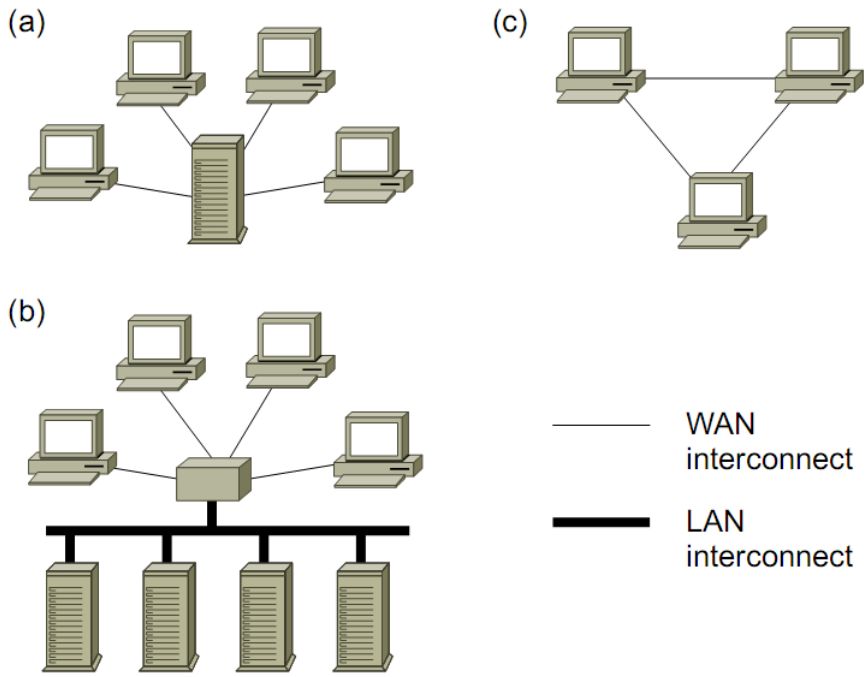

Figura 2.2: *Arquiteturas de suporte a MMOGs. (a) Cliente-servidor. (b) Multiservidor. (c) Peer-to-peer.*

2.2.1 Cliente-servidor

Arquiteturas cliente-servidor, representadas na Figura 2.2 (a), foram as primeiras soluções apresentadas, sendo a solução mais trivial para implementação. Nesse tipo de arquitetura utiliza-se um único elemento central confiável, normalmente da própria empresa desenvolvedora do jogo, onde é guardado o estado do jogo.

Além de ser confiável, a utilização de um elemento centralizador facilita o processo de gerenciamento de controle do jogo. Entretanto, esse tipo de arquitetura apresenta um ponto único de falha, podendo comprometer a satisfação dos jogadores caso haja falta de serviço por um período muito longo de tempo.

Escalabilidade é outro problema apresentado nessa solução, devido à existência de apenas um ponto central para comunicação entre todos os jogadores. A falta de escalabilidade pode se tornar um problema ainda mais grave quando a popularidade de um dado jogo aumenta rapidamente, sendo inviável a realização de melhorias para comportar o aumento do número de usuários.

Quando considera-se a escalabilidade do sistema, também é necessário avaliar o custo financeiro para obtenção de uma máquina potente o suficiente para gerenciar todos os jogadores. O número de jogadores simultâneos atualmente pode chegar à escala de dezenas de milhares, sendo impraticável o investimento em um único servidor (CCP, 2003).

2.2.2 Multiservidor

Buscando tratar alguns dos problemas apresentados pela arquitetura cliente-servidor, as arquiteturas multiservidor, representadas na Figura 2.2 (b), utilizam diversos servidores para distribuição da carga de simulação. Nesse tipo de arquitetura podem ser utilizados desde clusters de alta capacidade até servidores geograficamente distribuídos. Apesar de não existir abundância de informações oficiais, cogita-se que essa seja a arquitetura utilizada por empresas desenvolvedoras dos MMOGs mais populares atualmente.

Entre as melhorias em relação à arquitetura cliente-servidor convencional estão o aumento na tolerância a falhas e o aumento de escalabilidade. Entretanto, o aumento de escalabilidade ainda é limitado pela capacidade combinada dos servidores, sendo necessário um gerenciamento para balanceamento de carga eficiente, ou seja, com uma comunicação entre servidores sendo mantida ao mínimo. Ainda que sejam adotadas soluções eficientes, um aumento repentino na popularidade do jogo ainda põe em risco a estabilidade do sistema, cujo desempenho pode ser prejudicado, afastando jogadores.

Em relação ao custo financeiro, arquiteturas multiservidor podem apresentar um custo menor quando comparadas a arquiteturas cliente-servidor convencionais, dada a utilização de clusters invés de uma única máquina de alto desempenho. Entretanto, o custo ainda pode ser proibitivo para empresas com pouco capital.

2.2.3 Peer-to-Peer

Com o objetivo de oferecer uma solução financeiramente econômica e escalável, as arquiteturas *peer-to-peer* (P2P), representadas na Figura 2.2 (c), utilizam os computadores pertencentes aos próprios jogadores para manter o estado do jogo, sem a utilização de nenhum elemento central (FUJIMOTO, 1999). Uma forma de utilização desta arquitetura é a divisão do estado de jogo entre os jogadores pertencentes ao grupo, de forma que cada jogador mantenha uma porção do mundo virtual e jogadores interessados em atualizações da uma dada área contatem o *peer* responsável por ela.

As vantagens apresentadas por soluções P2P são a óbvia redução de custo, uma vez

que o investimento com um servidor central é inexistente, e o aumento da escalabilidade. Entretanto, o gerenciamento da divisão do mundo virtual gera um custo adicional de comunicação entre os elementos, podendo prejudicar o desempenho do sistema e gerar insatisfação dos jogadores.

Outro problema ocasionado pelo uso de arquiteturas P2P é o gerenciamento global do sistema (gerenciamento de jogadores e atualização de estados) e segurança. Esse último problema possui importância considerável, uma vez que os jogadores terão, em muitos casos, acesso físico à máquina que armazena o estado do jogo. De uma forma geral, soluções que utilizam arquiteturas P2P e possuem preocupação com segurança formam uma rede totalmente conectada, causando sobrecarga no sistema e afetando o desempenho.

Soluções utilizando arquiteturas P2P são aplicadas em diversos jogos *multiplayer* comerciais. Um exemplo tradicional deste tipo de arquitetura é Age of Empires (Microsoft Games, 2010). Porém, soluções P2P para MMOGs ainda estão no campo acadêmico, em busca de soluções que as tornem compatíveis com produtos comerciais.

2.2.4 Híbrida

Com os diversos problemas de segurança apresentados por arquiteturas P2P, e buscando reduzir o custo e aumentar a escalabilidade de arquiteturas multiservidor, há o surgimento de arquiteturas híbridas. Essas arquiteturas fazem uso de elementos centrais confiáveis e buscam reduzir a carga do servidor, ou dos servidores, atrvés da utilização das máquinas dos jogadores.

Normalmente são utilizados servidores centrais e confiáveis para operações de autenticação, formando grupos P2P entre os jogadores para realização da simulação em si. Apesar de reduzir os problemas apresentados por arquiteturas P2P, problemas de gerenciamento e segurança continuam existindo nos grupos formados pelos jogadores e, apesar de reduzir a carga do servidor, ainda podem existir problemas de escalabilidade e custo, além da possibilidade de haver um ponto central de falhas.

Assim como no caso das arquiteturas P2P, não há informações sobre esse tipo de arquitetura ser comumente utilizada em MMOGs mantendo-se, atualmente, como uma solução acadêmica. O Capítulo 3 trás algumas soluções de segurança envolvendo arquiteturas híbridas de relevância para este trabalho.

2.3 Divisão do mundo virtual

Nas seções anteriores foi citada o problema da divisão do mundo virtual em diferentes regiões, sendo essas regiões atribuídas a diferentes servidores ou espalhadas entre os *peers* que compõem a rede. Essa seção busca demonstrar como esta divisão pode ser realizada utilizando-se o conceito de células e regiões para divisão do mundo virtual. Esta seção descreve o cenário de divisão do mundo virtual utilizado no restante deste trabalho.

Considera-se que um dado espaço contíguo do mundo virtual (e.g. um quadrado) é denominado célula. Normalmente, e para este trabalho, considera-se que todas as células possuem o mesmo tamanho e formato. O conceito de célula é imperceptível ao usuário, tratando-se de uma divisão lógica do mundo virtual.

O conjunto de uma ou mais células contíguas denomina-se de região. A Figura 2.3 ilustra a divisão do mundo virtual em células e o agregamento de células em regiões. O conceito de região não implica algo perceptível ao usuário, embora este o possa perceber através de diferentes localizações no mundo virtual (como, por exemplo, diferentes países poderiam ser diferentes regiões). Regiões também podem ser utilizadas de forma

lógica e transparente para o usuário, para fins de gerenciamento do mundo virtual (e.g. balanceamento de carga).

Utilizando esses conceitos é possível realizar-se diferentes trabalhos, como balanceamento de carga, onde uma quantidade maior de células ou células mais populares sejam atribuídas a um servidor de maior capacidade. Também é possível a atribuição de mais de uma região a um único servidor, embora normalmente seja realizada a atribuição de uma única região a um servidor, conforme representado na Figura 2.4. No contexto deste trabalho este conceito será utilizado para restringir o efeito de cheating em uma única célula.

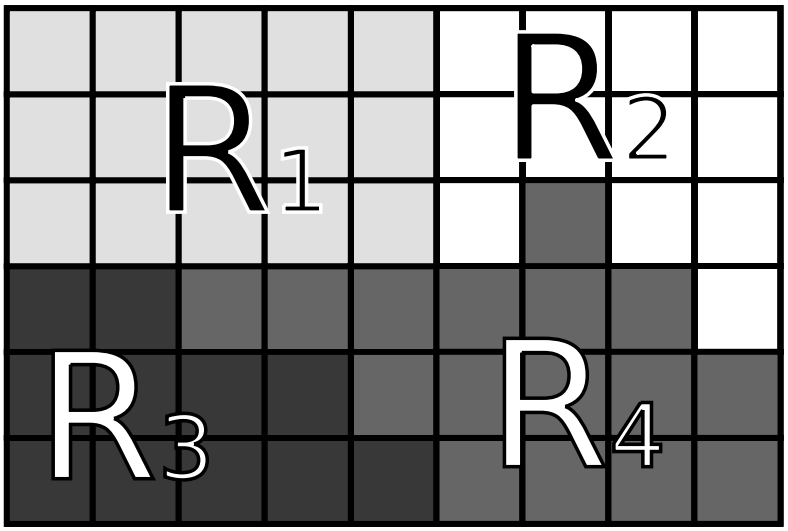

Figura 2.3: *Exemplo de divisão do mundo virtual em células e regiões.*

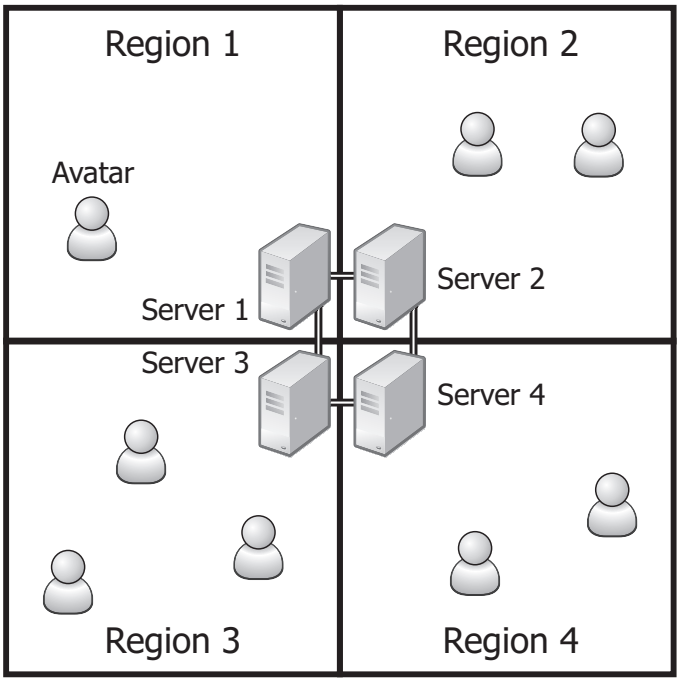

Figura 2.4: *Exemplo de atribuição de diferentes regiões a servidores.*

2.4 Cheating

Esta seção apresenta a conceituação de cheating, que pode ser definido como roubo ou trapaça em jogos. São apresentados sua definição, suas formas de ocorrência e uma breve discussão entre detecção e prevenção de cheating.

2.4.1 Definição

Os MMOGs, assim como diversos outros sistemas distribuídos, estão suscetíveis a ataques. Entretanto, no caso específico de jogos, há uma preocupação em relação a ataques específicos provenientes de jogadores. Esses tipos de ataques são denominados *cheating*[1].

No contexto desse trabalho adotaremos a caracterização apresentada por (YAN; RANDELL, 2005). Segundo os autores cheating é definido como sendo qualquer comportamento que um jogador adote para obter uma determinada vantagem, ou atingir uma determinada meta se, de acordo com as regras do jogo, ou a critério do operador do jogo (provedor do serviço, não necessariamente o desenvolvedor), esta vantagem ou meta não deveria ser obtida.

Além dos problemas convencionais causados por ataques em sistemas distribuídos, cheating em jogos possuem agravantes, especialmente quando consideramos MMOGs com economias virtuais persistentes. Em caso de ocorrência de cheating, é comum jogadores desanimarem com o jogo honesto e, muitas vezes, passam a utilizar cheating ou acabam desistindo do jogo.

2.4.2 Tipos de cheating

No trabalho proposto por Yan et. al (2005) são listados 15 diferentes formas de realizar cheating. Estas formas são detalhadas abaixo.

1. exploração de confiança indevida;
2. comprometimento de servidores de jogo;
3. modificação de infraestrutura do cliente;
4. negação de serviço;
5. comprometimento de senha;
6. *time cheating*
7. exploração de falta de sigilo;
8. exploração de falta de autenticação;
9. exploração de *bug* ou de *loophole*;
10. exploração de inteligência artificial;
11. abuso dos procedimentos de jogo;
12. relacionado a posses virtuais;

[1] Apesar de *cheating* tratar-se de uma expressão proveniente da língua inglesa, considera-se o termo popular e de conhecimento comum na área de jogos. Desta forma, julga-se que as traduções convencionais (e.g. trapaça, roubo) não são suficientes para exprimir o sentido da expressão.

13. conluio;
14. relacionado a mau uso interno;
15. engenharia social.

Confiança indevida refere-se à alteração do código do jogo ou arquivos de configuração presentes no cliente. Normalmente utiliza-se alguma forma de criptografia nos dados do cliente para proteção contra esse tipo de cheating. De forma similar, comprometimento de servidores de jogo são alterações realizadas no lado servidor da aplicação.

A modificação da infraestrutura do cliente acontece nos casos em que o cheater realiza alterações na infraestrutura em si e não no cliente. Um exemplo é a alteração de *drivers* de vídeo para renderizar paredes como se fossem transparentes. Cheatings por negação de serviço e comprometimento de senha são formas de ataque muito comuns em outras formas de sistemas distribuídos, não sendo necessário nenhum detalhamento adicional.

Time Cheating considera que um dado jogador consiga realizar um atraso no envio de sua ação até que seja definida a ação dos demais jogadores. Tendo recebido a ação dos demais jogadores, o jogador cheater toma uma decisão com base em um conhecimento indevido (as ações dos demais jogadores) e realiza a alteração do *timestamp* (tempo de envio) de sua ação, para que o sistema receba essa ação atrasada e considere válida, julgando que ocorreu algum atraso na sua entrega (problema de comunicação).

A exploração da falta de sigilo e a falta de autenticação estão relacionados ao gerenciamento das mensagens trocadas entre o jogador e o sistema. No primeiro caso, as mensagens trocadas entre jogador e sistema não são protegidas (criptografadas) podendo ser alteradas pelo próprio jogador ou por um outro jogador malicioso. O segundo caso diz respeito a problemas no mecanismo de autenticação de usuário, o que possibilita a um único jogador assumir diversas identidades dentro do jogo.

Exploração de *bugs* ou de *loopholes* são referentes à exploração contínua de uma falha, como a venda de um item por um preço maior do que o valor de sua aquisição.

Algumas formas de cheating, entretanto, não possuem uma raiz tão técnica, utilizando elementos externos ao jogo, ou até mesmo externos à computação, como são os casos da exploração de inteligência artificial, abuso dos procedimentos de jogo, cheating relacionado a posses virtuais, conluio, cheating relacionado ao mau uso interno e uso de engenharia social.

No caso de exploração de inteligência artificial utiliza-se técnicas de IA para escolha da melhor ação a ser tomada (como, por exemplo, o melhor próximo movimento em um jogo de xadrez).

Como exemplo de cheating por abuso dos procedimento de jogo pode-se citar alguns jogos onde um dos jogadores percebe a derrota iminente e desconecta-se do jogo para evitar a penalidade da derrota. Cheating relacionado a posses virtuais faz referência a casos onde as posses virtuais são trocadas ou vendidas por dinheiro real, prática proibida em muitos MMOGs.

Cheating por conluio ocorre quando há a cooperação de dois ou mais jogadores para obtenção de uma dada vantagem. Esta cooperação não acontece necessariamente dentro do jogo (podendo ser utilizado, por exemplo, telefones), tornando complexa a prevenção ou detecção deste tipo de cheating. Mau uso interno envolve administradores ou operadores de jogo, que utilizam seus poderes para beneficiar um ou mais jogadores. Por fim, engenharia social é a utilização de golpes por cheaters para obtenção de senhas ou outros dados de jogadores honestos, por exemplo quando um jogador passa-se por administrador de jogo e solicita que um outro jogador informe sua senha.

Além dessas formas de cheating, Cecin (CECIN, 2009) identifica uma forma chamada *state cheating*, ou cheating por alteração de estado. Nesse tipo de cheating, o jogador cheater realiza alterações no estado do jogo (por exemplo, alterando valores de variáveis, como quantidade de ouro possuída pelo personagem) de forma a favorece-lo.

A lista descrita acima é considerada novamente no Capítulo 3, onde são avaliadas diferentes formas de classificação de cheating e é realizada uma seleção das formas de cheating mais relevantes para o presente trabalho.

2.4.3 Detecção vs Prevenção

Neste trabalho consideramos a premissa adotada por (LAURENS et al., 2007), onde é defendido que a disputa entre o desenvolvedor de jogos online contra um desenvolvedor de cheating favorece ao desenvolvedor de cheating. Isso se deve pelo ciclo de vida de um jogo assumir que a criação de um novo método de exploração a um cheating requerir apenas algumas semanas, enquanto a correção deste problema pode necessitar de meses, como é mostrado na Figura 2.5 (LAURENS et al., 2007).

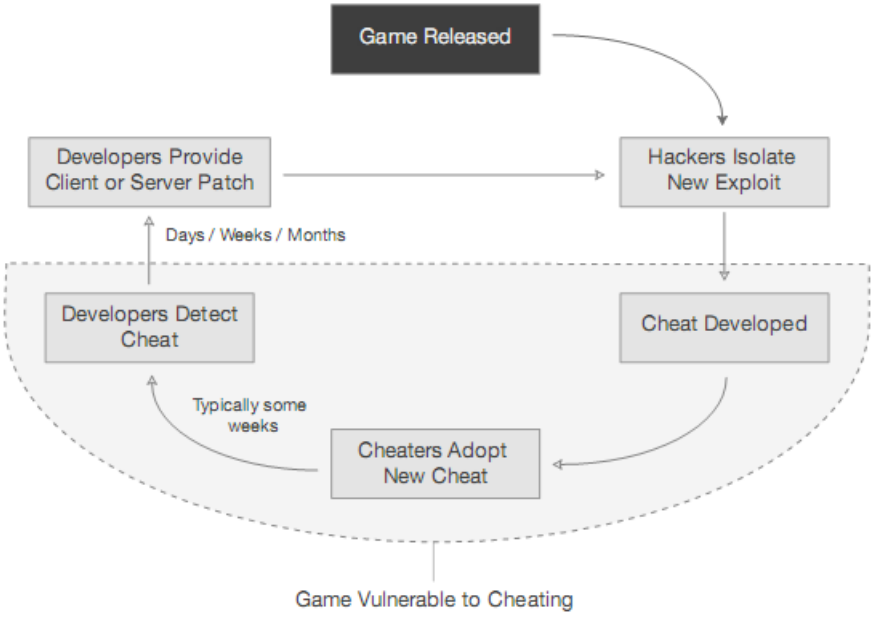

Figura 2.5: *Ciclo de vida de cheat.*

Como pode ser notado no ciclo de vida ilustrado pela figura, enquanto os desenvolvedores do jogo criam um *patch* para solucionar o problema, os desenvolvedores de cheating podem estar em etapa de desenvolvimento de um novo mecanismo para explorar uma falha recém-descoberta. Dessa forma, alguns sistemas aguardam dias ou até semanas antes de punir um *cheater*, defendendo que com este atraso na divulgação da descoberta do cheating é possível punir mais cheaters antes que eles troquem o mecanismo.

3 ESTADO DA ARTE

O presente capítulo apresenta trabalhos que consideram o tratamento de cheating em jogos online e MMOGs. Os trabalhos são classificados em dois grupos: classificação de cheating e soluções anti-cheating. O primeiro grupo é apresentado na Seção 3.1, e busca classificar as diferentes formas de cheating, de forma a auxiliar o desenvolvimento de mecanismos de proteção tal como o presente trabalho. O segundo grupo é apresentado na Seção 3.2, e descreve diferentes mecanismos para detectar ou prevenir cheating em jogos online.

Ao final de cada uma das seções é realizada uma breve discussão sobre os trabalhos apresentados, tendo como guia o objetivo de desenvolver um mecanismo para proteção da economia virtual em MMOGs utilizando uma arquitetura composta por elementos não-confiáveis.

3.1 Classificação de cheating

Atualmente não há uma classificação formal definitiva para cheating em MMOGs sendo, usualmente, utilizada uma lista exaustiva de formas conhecidas de cheating. Contudo, existem trabalhos que buscam uma definição formal e classificação de cheating de acordo com características comuns.

Nessa seção serão apresentadas algumas formas de classificação e taxonomia de cheating em MMOGs, resultado dos trabalhos de (JHA; KATZENBEISSER; VEITH, 2007), (WEBB; SOH, 2007), (GAUTHIERDICKEY et al., 2004) e (YAN; RANDELL, 2005).

3.1.1 Classificação de ataques contra semântica

Jha et al. (2007) definem que jogos são um tipo de *Networked Virtual Environment* (NVE), e consideram cheating como um ataque a integridade semântica da comunidade online, ou seja, a tentativa de um usuário malicioso de compromoter as regras lógicas que governam a simulação. Não há uma forma de prevenir que o programa cliente seja modificado por jogadores maliciosos, modificações essas que podem incluir alterações para exposição de informações ocultas ou modificações em dano de armas. Dessa forma, qualquer método que busque garantir segurança em uma NVE deve assumir que todos os clientes são não confiáveis.

De acordo com os autores, o *gap* semantico é o principal meio pelo qual clientes maliciosos atacam a integridade semântica. *Gap* Semântico é a diferença entre a representação do ambiente virtual entre o servidor e o cliente. Enquanto o cliente possui uma representação concreta da simulação (envolvendo cálculos físicos) o servidor possui apenas uma representação abstrata da simulação (por exemplo, em um jogo de corrida, ele possui a

informação de que o jogador está em um carro, em um certo lugar, se movendo a uma certa velocidade).

Levando em consideração as definições acima, é dada uma classificação para ataques semânticos (desconsiderando ataques a sistemas de segurança ou meta-estratégias (como conluio)). Os ataques semânticos são divididos em: Violação de Integridade Semântica e Amplificação do Cliente.

Violação de Integridade Semântica compreende os ataques que violam as leis lógicas e físicas do NVE. Todos os ataques dessa classe envolvem o uso de software maliciosamente modificado. Podem haver duas formas de violação da integridade semântica: Corrupção de Regra e Alteração de Causalidade. Corrupção de Regra é a tentativa de modificação da simulação, por parte do cliente, de forma que a simulação assuma um estado aceitável pelo servidor do sistema, mas inaceitável pelas regras estabelecidas. Um exemplo seria a alteração na física de um veículo em um jogo de corrida, permitindo que este alcance altas velocidades sem consequências negativas. Alteração de causalidade compreende a alteração de estados anteriores para obtenção de vantagens. Um exemplo é a alteração de posição para evitar o dano causado por uma explosão.

Amplificação do Cliente é a utilização de softwares modificados para exploração não aceitável de possibilidades na NVE. Esta classe também se divide em duas categorias: *Sniffing* e Agentes. *Sniffing* é a exposição de informações transferidas por motivos técnicos mas que não deveriam estar disponíveis imediatamente, por exemplo, um jogador poderia renderizar um muro opaco como transparente. Agentessão utilizados para aumentar a capacidade de um jogador humano como, por exemplo, a utilização de uma mira automática um jogo de tiro.

3.1.2 Taxonomia de cheating em jogos online

Yan e Rendell (2005) realizaram um trabalho de análise dos tipos existentes de cheating com o objetivo de criar uma taxonomia clara e abrangente, podendo ser usada por especialistas, administradores e até jogadores. Os autores definem que cheating é qualquer comportamento que um jogador use para ganhar vantagem sobre outros jogadores ou atingir um objetivo se, de acordo com as regras do jogo, esta vantagem ou objetivo não poderia ser alcançado.

Também é defendido pelos autores que grande parte dos cheats são devido à falta ou ausência de segurança nos jogos online, e que as investigações sobre cheating, de uma forma geral, são restritas a estudos caso-a-caso. Assim, Yan e Rendell propõem uma taxonomia tridimensional, classificando cheating de acordo com três características: vulnerabilidade (o que é explorado), consequência (que tipo de falha pode ser alcançada) e responsável (quem está trapaceando).

Para definir uma taxonomia completa, os autores desenvolvem uma lista exaustiva com os tipos de cheating conhecidos. A lista, sumarizada, é apresentada na seção 2.4.2.

Esta lista exaustiva, em conjunto com as três categorias apresentadas anteriormente (vulnerabilidades, possiveis falhas e responsável), é utilizada para gerar a classificação apresentada na Figura 3.1 (YAN; RANDELL, 2005).

As vulnerabilidades podem ser divididas em inadequação no design do sistema e vulnerabilidades envolvendo pessoas. O primeiro ainda se divide entre falhas no jogo ou em outros sistemas. As vulnerabilidade envolvendo pessoas subdividem-se em falhas envolvendo jogadores ou operadores. As possíveis falhas compreendem o roubo de informações ou posses, negação de serviço, violação de integridade, mascaramento e violação de igualdade. Os responsáveis se subdividem em cooperativo, podendo envolver opera-

Classification of the various types of cheating	Vulnerability (-ies)				Possible Failure(s)					Exploiter(s)			
		System Design Inadequacy	People							Independent		Cooperative	
	In Underlying Systems	In Game System	Player	Game Operator	Fairness Violation	Masquerade	Integrity Violation	Service Denial	Theft of Information or Possessions	Single Player	Game Operator	Multiple players	Operator and Player
A) Cheating by Exploiting Misplaced Trust		●					●		●	●			
B) Cheating by Collusion		●			●				●			●	
C) Cheating by Abusing the Game Procedure		●			●					●			
D) Cheating Related to Virtual Assets			●		●					●			
E) Cheating by Exploiting Machine Intelligence		●			●					●			
F) Cheating by Modifying Client Infrastructure	●						●			●			
G) Cheating by Denying Service to Peer Players	●	●						●		●			
H) Timing Cheating		●			●		●			●			
I) Cheating by Compromising Passwords			●						●	●			
J) Cheating by Exploiting Lack of Secrecy		●					●		●	●			
K) Cheating by Exploiting Lack of Authentication		●				●				●			
L) Cheating by Exploiting a Bug or Design Loophole		●			●					●			
M) Cheating by Compromising Game Servers	●						●			●			
N) Cheating Related to Internal Misuse				●			●				●		●
O) Cheating by Social Engineering			●						●	●			

Figura 3.1: *Taxonomia de cheating em jogos online.*

dores e jogadores ou apenas múltiplos jogadores, e independente, envolvendo apenas um operador ou apenas um jogador.

Os autores ainda dividem as formas de cheating em específicos e genéricos, ou seja, aqueles que são específicos para jogos ou comuns a outras formas de aplicações de rede. As formas de cheating com especial relevância para jogos são as listadas de 1 a 7, sendo as demais classificadas como genéricas.

3.1.3 Classificação por níveis de ocorrência

GauthierDickey, Zappala e Lo (2004) propõem uma taxonomia distinta para classificação das formas comuns de cheating em jogos multijogadores. As formas de cheating são classificadas de acordo com o nível em que ocorrem, podendo ser: jogo, aplicação, protocolo e rede. O resultado dessa classificação pode ser visto na Figura 3.2 (GAUTHIERDICKEY et al., 2004).

A TAXONOMY OF CHEATING

Cheat	Level	Distributed		Client/Server
		P2P	Multicast	
Denial of Service	Network	✓	✓	✓
Fixed Delay	Protocol	✓	✓	⋆
Timestamp	Protocol	✓	✓	⋆
Suppressed Update	Protocol	✓		
Inconsistency	Protocol	✓	✓	
Collusion	Protocol	✓	✓	✓
Secret revealing	Application	✓	✓	⋆
Bots/reflex enhancers	Application	✓	✓	✓
Breaking game rules	Game	✓	✓	✓

Figura 3.2: *Tabela de classificação por níveis de ocorrência.*

Cheats que ocorram em nível de jogo são resultados da manipulação das regras de jogo para resultados não esperados. A nível de aplicação o código da aplicação é modificado para, injustamente, dar vantagem ao jogador. Em nível de protocolo, é possível alterar os pacotes ou protocolos de rede. Por fim, cheats em nível de rede ocorrem através de falhas em segurança de rede, como negação de serviço, por exemplo. Na 3.2 também é possível perceber a divisão das formas de cheating nas diferentes arquiteturas: *Peer-to-peer*, *Multicast* e Cliente-servidor.

3.1.4 Revisão da classificação por níveis de ocorrência

Webb e Soh (2007) modificaram o esquema de classificação proposto em (GAUTHIERDICKEY et al., 2004) para: jogo, aplicação, protocolo e infraestrutura. Cheats envolvendo infraestrutura involvem a modificação ou manipulação de partes da infraestrutura utilizadas pelo jogo como, por exemplo, *drivers*, bibliotecas, hardware e rede.

Em seu trabalho, Webb e Soh realizam um estudo sobre alguns métodos anti-cheating baseados na classificação apresentada. O resultado é apresentado pela Figura 3.3 citewebb2007cnc, que ilustra diversos tipos de cheating e possíveis soluções.

Cheat	C/S	PB/VAC2	AS	NEO/SEA	RACS
Game Level					
Bug	✓	×	✓	✓	✓
RMT	✓	×	×	×	✓
Application Level					
Information Exposure, Invalid Commands	✓	×	×	×	✓
Bots/reflex enhancers	×	✓	×	×	×
Protocol Level					
Suppressed update, Timestamp Fixed delay, Inconsistency	✓	×	✓	✓	✓
Collusion	☒	☒	☒	☒	☒
Spoofing, Replay	✓	×	×	✓	✓
Undo	N/A	×	✓	×	N/A
Blind Opponent	N/A	×	N/A	N/A	✓
Infrastructure Level					
Information Exposure	✓	✓	×	×	✓
Proxy/Reflex Enhancers	☒	☒	☒	☒	☒

✓ - solvable × - not yet solved

☒ - not solvable N/A - not applicable

Figura 3.3: *Revisão da classificação por níveis de ocorrência.*

3.1.5 Classificação por nível de impacto

O trabalho desenvolvido por (CECIN, 2009) introduz o conceito de nível de dano de cheating em jogos online. Neste trabalho é defendido que não é possível projetar um jogo online que possua qualidade de serviço, seja escalável e imune a todos os tipos de cheating ao mesmo tempo. Tendo em vista essa assertiva, deve ser utilizada uma forma de seleção para determinar qual tipo de cheating tratar, sendo preferível um jogo onde o impacto sofrido pela ocorrência de cheating seja o mínimo possível.

Na classificação apresentada neste trabalho são utilizados três grupos para representar o dano causado pela ocorrência de cheating em um jogo:

- Dano imperceptível;
- Dano limitado;
- Dano ilimitado.

Dano imperceptível é aquele que não é realizado na presença de outros jogadores, e busca melhorar o personagem do jogador para encontros futuros. Frequentemente outros jogadores não conseguem distinguir um jogador cheater de um jogador que tenha passado muito tempo em jogo, portanto a ocorrência de cheating causa um mínimo de frustração em outros jogadores. Um exemplo deste grupo é o uso de automação para execução de certas ações, também conhecido como uso de *bots*. Cheating de dano imperceptível pode ser classificado como um caso especial de dano limitado.

No caso de cheatings de dano limitado, a execução do cheating é realizada na presença de outros jogadores, concedendo vantagem direta sobre eles. Cheating presentes neste grupo causam insatisfação em jogadores alvo, porém os resultados ainda são limitados

pelas regras do jogo. Exemplos de cheating limitados são mira automática ou reflexos extraordinários em jogos de tiro (dano infligido pela bala ainda é limitado pelas regras do jogo), teletransporte de um personagem para chegar primeiro a um baú de tesouro (tesouro obtido ainda é limitado pelo jogo).

Por fim, cheatings de danos ilimitados são aqueles que, uma vez realizados, conferem a vantagem desejada pelo cheater. Normalmente este tipo de cheating requer acesso ao estado do jogo, sendo arquiteturas que contenham um agente centralizador confiável normalmente imunes a este grupo. Um exemplo deste tipo de cheating é a criação de uma quantidade significativa ouro (ou qualquer tipo de unidade monetária virtual) em um jogo aventura medieval. Outra característica deste grupo é a potencialidade para afetar a economia virtual de um MMOG, sendo possível afetar uma grande parcela, ou até mesmo todos, os jogadores participantes daquele mundo virtual.

3.1.6 Considerações

Entre as classificações relacionadas, e levando em consideração o objetivo proposto pelo presente trabalho, utiliza-se a classificação por nível de impacto apresentada por (CECIN, 2009), e a lista exaustiva de tipos de cheating, apresentada por (YAN; RANDELL, 2005), sendo esta lista explorada na Seção 2.4.

Pelo fato de ser considerado como objetivo primário deste trabalho a proteção da economia virtual contra jogadores maliciosos, é necessário a realização de uma análise para verificação de qual forma de cheating pode possuir maior impacto no mundo virtual. Tendo em mente este objetivo a classificação proposta por (CECIN, 2009) pode ser utilizado para classificar as formas de cheating e, desta forma, melhor avaliar a eficácia do mecanismo proposto.

Como consideração final pode-se concluir que a classificação de cheating analisada, aliada com a lista de cheating completa explorada na Seção 2.4, leva à conclusão de que a forma de cheating mais relevante para os objetivos desse trabalho é *state cheating*. Assim, uma solução que proponha a restrição do impacto de cheating no mundo virtual deve realizar uma abordagem considerando os efeitos deste tipo de cheating.

3.2 Soluções anti-cheating

Essa seção apresenta técnicas que buscam solucionar a problemática de cheating em jogos online. São apresentadas soluções desenvolvidas por (LIU; LO, 2008), (LIU; TANG, 2009), (FERRETTI; ROCCETTI, 2006), (BAUGHMAN; LIBERATORE; LEVINE, 2007), (CHEN; MAHESWARAN, 2004), (LAURENS et al., 2007) e (IZAIKU et al., 2006).

3.2.1 DaCaP

Liu e Lo (2008) definem uma arquitetura distribuída com foco em distribuição de carga e anti-cheating denominada *Dynamic anti-CheAting Peer to peer*, ou DaCAP. A arquitetura proposta pode ser classificada como híbrida, uma vez que mescla características de arquiteturas *peer-to-peer* e cliente-servidor.

DaCAP classifica as áreas virtuais em *Hotspot*, *Raid* e áreas não-*hotspot*, sendo que as duas primeiras fazem uso de arquitetura cliente-servidor. *Hotspot* são áreas cuja concentração de jogadores é muito alta, como cidades virtuais, por exemplo. Nesses pontos também é muito comum a ocorrência frequente de migração de jogadores (jogadores entram e saem da área com muita frequência), o que poderia se tornar um problema em uma

arquitetura *peer-to-peer*. *Raids* são aglomerados independente de jogadores em busca de um objetivo comum (uma mesma *quest* por exemplo). Nestes grupos é muito comum o compartilhamento de informações, como *chat* ou atributos de personagens, sendo o controle por servidores uma forma eficiente de controlar estes grupos.

As demais áreas, não-*hotspots*, adotam a arquitetura P2P. Jogadores que estejam nessas áreas são organizados em grupos em uma arquitetura P2P *overlay* totalmente conectada, ou seja, cada membro do grupo é conectado a todos os demais membros.

O mecanismo anti-cheating em áreas *Hotspot* e *raid* é controlado pelo servidor. Nas redes P2P o mecanismo é baseado em observação mútua por parte dos integrantes do grupo. Caso um jogador cometa alguma infração em uma rede P2P será denunciado por seus pares. Caso um jogador informe dados errôneos sobre si mesmo ele será marcado como falsário. Caso um jogador informe dados errados sobre outro jogador, este jogador será marcado como informante. Para evitar falsificação colaborativa, o sistema escolhe aleatoriamente um número de jogadores de outras áreas para se juntar ao grupo. Além disso, para evitar que um jogador faça denúncias falsas sobre outros jogadores, será armazenado o número de denuncias feitas por cada jogador. Caso um jogador exceda um limite de denúncias este será marcado como informante malicioso.

Para testar o desempenho apresentado pelo DaCAP foram efetuadas simulações e seus resultados comparados com arquitetura cliente-servidor tradicional. Foi simulado um espaço virtual de 300x600 com diferentes niveis de concentração, sendo os jogadores localizados em *hotspots* ou não-*hotspots* com diferentes probabilidades. O mundo virtual foi dividido em 50 áreas de tamanho 60x60, sendo cinco delas aleatoriamente selecionadas como *hotspots*. A velocidade dos jogadores foi estabelecida como um pixel por segundo, com oito possíveis direções de movimento. A movimentação de cada jogador é aleatória, seguindo as seguintes probabilidades: 50% de probabilidade de movimento inicial (50% de chance de não se movimentar); enquanto movendo, 80% de chance de permanecer em movimento; jogadores parados têm 70% de chance de permanecer na posição inicial.

Foram realizados testes com variação no número de jogadores (entre 1000 e 30000) e variando a quantidade de cheaters no sistema (10 a 90%). Também foram realizados testes para análise de carga de processamento e análise de tráfego de rede. Resultados mostram significativa redução de carga no servidor, tanto em consumo de rede quanto CPU. Também é possível constatar uma taxa de detecção de cheating decrescente em relação à variação de cheaters presentes no sistema (quanto maior a proporção de cheaters em relação a jogadores honestos, pior o desempenho do sistema). Entretanto, foi possível alcançar 90% de detecção com uma taxa de 90% de jogadores cheaters (considerando grupos de 90 jogadores).

3.2.2 DACA

Dynamic Anti-Cheating Architecture (DACA) é um modelo de arquitetura apresentado por (LIU; TANG, 2009) que, assim como DaCAP, objetiva a otimização no balanceamento de carga em redes P2P e criar uma arquitetura eficientemente *anti-cheating*. Seguindo uma arquitetura híbrida, os jogadores formam um rede *overlay* para interagir entre si e com o jogo. Em cada área, um jogador é apontado como sendo o *data holder* para supervisionar as operações do grupo P2P. Existem três tipos de servidores: autenticação, *database* e jogo, sendo este último responsável pela divisão e união das áreas, participar da simulação em áreas suspeitas e controle de jogadores maliciosos.

Para prover balanceamento de carga, é proposta uma forma de divisão do mundo virtual em regiões de acordo com a popularidade de uma dada região, denominada *Weighted*

Responsible Division (WRD). Na WRD, o mundo virtual é dividido em regiões quadradas desiguais centradas em *hotspots*. A Figura 3.4 (LIU; TANG, 2009) ilustra um exemplo de divisão do mundo virtual.

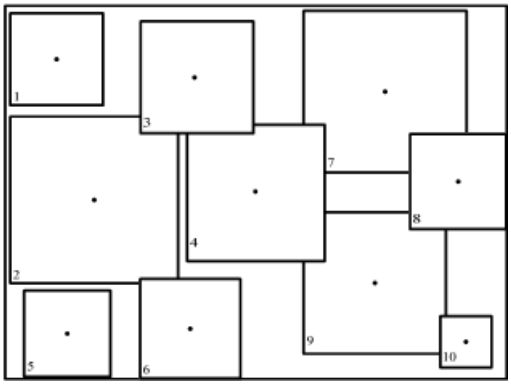

Figura 3.4: *Divisão do mundo virtual utilizando WRD.*

Para balancear a carga, um *hotspot* popular é cercado por regiões menores, sendo os pontos em comum controlados pela menor região. O servidor fica encarregado de verificar uma movimentação do centro do *hotspot*, que pode mudar de acordo com a mudança de popularidade da região. Para a divisão ou união dinâmica, é estabelecido um limite superior, para o qual a região deve ser dividida em regiões menores, e um limite inferior, para o qual a região deve ser unida a outra região.

O mecanismo *anti-cheating* é fornecido na forma de monitoramento recíproco. Para prover segurança são utilizados três integrantes: *data holder*, *data verifier* e *computation verifier*. *Data holder* é um jogador responsável pelo controle de informação de uma área. Para evitar cheating, o *data holder* é escolhido em um grupo diferente de jogadores. *Data verifier* é um jogador aleatoriamente selecionado pelo servidor para realizar a verificação de consistência entre a base de dados e o *data holder*, evitando dessa forma, que o *data holder* forneça informações falsas aos jogadores. Por fim, o *computation verifier* é um jogador pertencente a outro grupo aleatoriamente selecionado para participar da computação, buscando evitar *group cheating* (conluio).

Durante um jogo, cada jogador envia suas ações para os outros membros de seu grupo. A partir disso existem três situações possíveis:

Não há cheating: em uma área sem cheating o *data holder* receberá apenas dados consistentes, que serão enviados ao servidor e demais membros do grupo para atualização;

Existência de uma minoria cheating: onde os dados recebidos pelo *data holder* serão inconsistentes e os dados da maioria dos jogadores (que serão consistentes) serão enviados. Se houver algum informante o servidor irá participar da computação;

Existência de uma maioria cheating: a maioria dos dados coletados pelo *data holder* serão consistentes e enviados, porém serão dados como incorretos e informados pela minoria ou, ao menos, um jogador aleatoriamente escolhido por DACA. Neste caso ocorrerá um *rollback* e o servidor irá participar da computação, onde quaisquer resultados que discordem daqueles encontrados pelo servidor serão considerados cheating e punidos de acordo.

Para avaliar o desempenho da arquitetura foram realizadas simulações utilizando um ambiente quadrado de 1000 x 1000. Foram simuladas três formas de divisão. A primeira é em formato de grid estático denominado Grid, a segunda utiliza *hotspots* para criação de diagramas Voronoi (divisões desiguais centradas em *hotspots*) chamada HVOR e, por fim, a terceira forma de divisão utiliza WRD. Fora utilizados duas diferentes quantias de *hotspots*: 50 e 100, aleatoriamente distribuídos na área de jogo. O limite inferior (menor quantidade de jogadores em uma área sem que a área una-se a outra) é 100, e o limite superior (maior quantidade de jogadores em uma área sem que esta área divida-se) é de 250. O número de jogadores varia entre 1000 e 30000, sendo que cada jogador realiza 4 ações por segundo.

Considerando os objetivos do presente trabalho, os resultados de maior interesse referem-se à carga e à detecção de cheating. A carga do servidor foi similar em todos os métodos de divisão, sendo em torno de 68% mais eficaz em relação a uma arquitetura completamente centralizada. A solução *anti-cheating* não sofre influência em relação à forma de divisão das regiões. Os resultados encontrados foram semelhantes entre todos os métodos, sendo decrescente em relação à proporção de cheaters presente, e próximo a 76% de detecção em áreas onde 90% dos jogadores são cheaters.

3.2.3 AC/DC

Ferretti e Roccetti (2006) focam seu trabalho no desenvolvimento de uma forma de detecção de *time cheating* através do de um algoritmo chamado AC/DC (*Algorithm for Cheating Detection by Cheating*). De acordo com os autores, *time cheating* é a alteração da informação de tempo de eventos de jogo. Os autores também defendem que a importância deste tipo de cheating reside no fato de que os jogadores estão geograficamente distribuídos e, tipicamente, utilizam diferentes sistemas ou tecnologias de rede.

Nesse trabalho os autores focaram-se em jogos de tempo real, considerando uma forma específica de *time cheating* conhecida como *lookahead cheats*. Nesse tipo de cheating, um jogador aguarda a geração de ações de outros jogadores antes de gerar sua própria ação, informando ao sistema que esta ação teria sido produzida antes de outras.

A forma de funcionamento do AC/DC é simples, consistindo da inserção de um atraso no envio de mensagens a um jogador quando houver a suspeita de cheating. Este processo de atraso no envio de mensagens deve ser iniciado por um *peer* confiável a todos, e somente em caso de suspeita de cheating. Caso o jogador suspeito continue a atrasar o envio de suas mensagens além de um limite aceitável, esse jogador é dado como cheater.

3.2.4 Lockstep e Asynchronous Synchronization

Baughman, Liberatore e Levine (2007) defendem que o uso de *dead reckoning* pode fragilizar a segurança do jogo, permitindo cheating. *Dead reckoning* é uma técnica utilizada como forma de compensação por variações de latência de comunicação e perda de conexão através da suposição do estado de um jogador, cuja atualização não foi recebida, baseando-se no último estado conhecido. Segundos os autores, o uso de um simples protocolo *stop-and-wait* impede algumas formas de cheating, mas não evitam *lookahead cheating*, pois um jogador ainda pode aguardar a ação de todos os demais *peers* antes de enviar sua ação.

Desta forma, para evitar *lookahead cheating*, é apresentado o *Lockstep Protocol*, um protocolo *stop-and-wait* com uma etapa de comprometimento. Quando o *frame t* (ou rodada) é completo, cada jogador faz sua decisão para o *frame t+1* e anuncia um *hash* de sua decisão como forma de comprometimento. Uma vez que todos os jogadores tenham

informado os *hashes* de suas ações, todos comunicam suas ações em texto simples. A validação da ação tomada pode ser feita através do comparativo dos *hashes* da nova ação com a ação enviada anteriormente. Este protocolo, porém, insere uma penalidade em questões de desempenho. Mesmo que a corretude do jogo seja garantida, o jogo, e todos os jogadores, deverão ser executados na velocidade do jogador mais lento.

Em busca de melhorar o desempenho do *lockstep protcol*, os autores apresentam o protocolo *Asynchronous Synchronization*. Neste novo protcolo cada cliente avança no tempo de forma assíncrona em relação aos demais clientes, mas entra em um modo *lockstep* quando for necessária alguma interação, garatindo a corretude do jogo.

Utilizando o protocolo *Asynchronous Synchronization*, cada cliente deve possuir dados sobre o avanço dos demais jogadores, podendo analisar a área em que suas ações possam influenciar o jogador. A essa área é dado o nome de *sphere of influence* (SOI). Dois jogadores cujas SOI não se intersecionem terão seus eventos decididos sem afetar um ao outro. Cada SOI é dividida em duas partes: SOI base é a área máxima onde o jogador pode influenciar ou ser influenciado por outros em qualquer turno e SOI delta, cuja influência pode ocorrer em turnos subsequentes.

Em um dado turno, um jogador toma sua decisão (passo 1), anuncia o comprometimento de sua decisão aos demais jogadores (passo 2), aceita comprometimentos de um turno anterior ao último revelado (passo 3) e por fim, antes de revelar seu comprometimento, o jogador local deve decidir quais outros jogadores remotos ele deve esperar pelo envio do comprometimento. Um jogador remoto não está no estado de espera apenas se não há intersecção com a SOI dilatada da última ação do jogador remoto ou se um comprometimento foi aceito pelo jogador local. As SOI de cada jogador remoto são calculadas baseando-se na última posição conhecida e adicionando um valor delta para cada *frame* que o jogador local esteja à frente do jogador remoto em seu último *frame* conhecido. Se o cliente local estiver em um futuro relativo a outro jogador, então a influência potencial do outro jogador é dilatada para o jogador local no próximo *frame*. Se o jogador local não estiver no futuro de um jogador remoto, nenhuma dilatação é realizada. A Figura 3.5 (BAUGHMAN; LIBERATORE; LEVINE, 2007) ilustra um jogador local entrando na área de influência de um jogador remoto.

A análise de desempenho realizada pelos autores buscou comparar o protocolo *Asynchronous Synchronization* com o protocolo *Lockstep* utilizando o jogo XPilot, um jogo bi-dimensional multijogador. XPilot foi configurado para rodar, aproximadamente, 4000 *frames* de jogo com diversos jogadores automatizados em um mapa 300x300, e o jogo foi modificado para armazenar as coordenadas x e y em um arquivo de log. A simulação utilizou os arquivos de log como entrada, considerando cada coordenada x e y como sendo uma decisão de cada jogador em um turno. Cada unidade de tempo do simulador é representado como 10 ms de tempo real. Em cada unidade de tempo do simulador um jogador realiza a leitura do arquivo de log para o próximo turno e envia aquele turno para outros jogadores.

Foi assumida uma topologia do tipo estrela, com o servidor no meio e os jogadores nas pontas da estrela, devido ao uso deste tipo de topologia em jogos online do XPilot. Cada jogador possui um atraso de conexão em relação ao servidor baseado em uma distribuição exponencial com média de cinco unidades de tempo do simulador. Também foi definido que jogadores não poderiam ter um turno em tempo inferior a quatro unidades de tempo do simulador, devido ao tempo de reação humana, além de não poderem avançar de turno mais de uma vez a cada 10 unidades de tempo do simulador, buscando simular um jogo rodando a 10 *frames* por segundo.

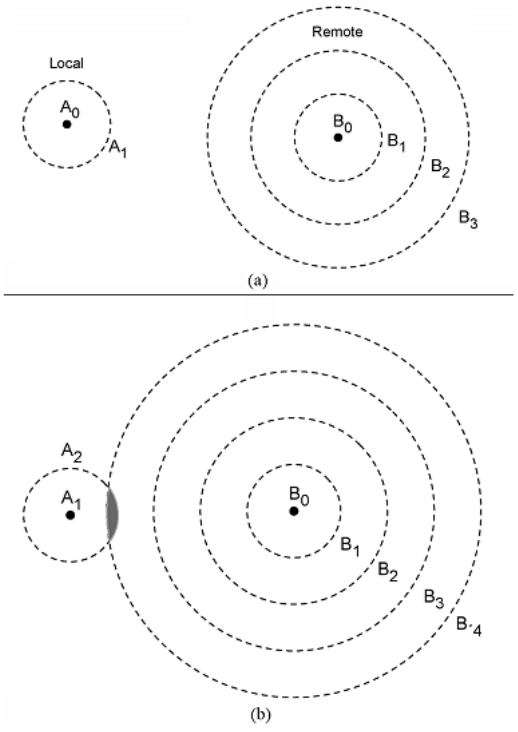

Fig. 2. (a) Dilation to time = 3. (b) Dilation to and intersection at time = 4.

Figura 3.5: *Utilização de SOI.*

Foram utilizados *traces* de 10, 25, 50 e 75 jogadores. Para cada trace foram utilizados quatro diferentes tamanhos de SOI. O menor SOI foi configurado para possuir o tamanho do maior movimento possível em um turno (SOI de tamanho 1). O maior foi configurado com um tamanho infinito, correspondendo ao protocolo *Lockstep*. Para propósitos de comparação também foram simulados SOI com o dobro do tamanho mínimo (SOI de tamanho 2) e quádruplo do tamanho mínimo (SOI de tamanho 4).

Nos resultados apresentados, o protocolo *Asynchronous Synchronization* possui um desempenho consideravelmente superior ao *lockstep*. Isso ocorre porque a utilização do protocolo *lockstep* exige que todos os jogadores aguardem pelo jogador mais lento, enquanto que, com o *Asynchronous Synchronization*, entre 30 e 40% dos turnos podem ser resolvidos sem atraso.

3.2.5 Cheat Controlled Protocol

Chen e Maheswaran (2004) apresentam o protocolo *Cheat Controlled Protocol* (CCP), desenvolvido com o objetivo de detectar *time cheating* em arquiteturas centralizadas. O objetivo é evitar que um dado jogador possua mais tempo do que os demais para reagir a um dado evento do mundo virtual, sendo isso possível através do atraso no envio de suas ações até o recebimento das ações dos demais jogadores.

O funcionamento do *Cheat Controlled Protocol* é baseado no tempo de resposta de um dado jogador ao envio de uma mensagem de atualização. A diferença entre o tempo que o servidor envia uma mensagem de atualização e o jogador envia seu comando como resposta a esta mensagem inclui o tempo de comunicação (*Actual Round Trip Time*), tempo de processamento desta mensagem (*Actual Update Message Processing Time*) e o tempo de reação do jogador. Obviamente, o tempo de envio de uma mensagem varia conforme

as condições da rede. O servidor estima o tempo atual de envio de uma mensagem a um jogador em um dado momento (chamado de *Estimated Current Round Trip Time*). Além disso, o servidor tolera alguma variação neste tempo estimado, chamado de *Round Trip Time Tolerance*. Da mesma forma, o tempo de processamento também é estimado pelo servidor (*Estimated Update Message Processing Time*) e possui um determinada tolerância (*Update Message Processing Time Tolerance*).

Baseando-se nos valores determinados anteriormente, pode ser calculado o tempo esperado de recebimento (*Proposed Arrival Time*). Comparando-se o tempo esperado com o tempo real de recebimento temos que, caso o tempo real seja menor ou igual ao esperado, o jogador é honesto e sua mensagem é aceita pelo servidor. Caso o tempo real seja maior do que o tempo esperado, podem haver diversas causas para o atraso como congestionamento na conexão entre o cliente e o servidor, congestão de computação na máquina do cliente, ou uma tentativa de cheating por parte do jogador. Para distinguir entre os dois primeiros casos e a tentativa de cheating, o *Cheat Controlled Protocol* verifica se há algum congestionamento de rede observado pelo servidor, e define que uma mensagem atrasada é uma mensagem ilegal e será penalizada.

A medida da latência de rede é realizada através da verificação da porcentagem do recebimento de mensagens atrasadas. Quando esta porcentagem ultrapassar o limite dado pelo *Ping Threshold* e não há um *ping* efetuado para o cliente, um *ping* é realizado. A última operação de ping realizada é utilizada para atualizar o tempo estimado de comunicação de uma mensagem do servidor para o jogador. Caso o tempo desta última operação não seja acima do esperado, o servidor reduz o tempo estimado de comunicação em um valor fixo, diminuindo o tempo esperado de recebimento.

A latência computacional depende puramente da máquina do cliente, não sendo possível uma medida dos recursos devido à possibilidade de interferência, por parte do jogador, na medida. Desta forma, o *Cheat Controlled Protocol* provê um tempo limite máximo de processamento na máquina cliente, sendo que é esperado que o jogador use sua máquina de forma que cumpra este tempo limite, chamado de *Estimated General Update Message Processing Time*. Com o estabelecimento deste tempo, as ações de um jogador malicioso ficam limitadas ao tempo dado, coibindo o uso de cheating.

Para experimentos e análise de desempenho foram utilizadas simulações utilizando o jogo de tiro (*shooting game*) XPilot, com jogadores automatizados e servidor executados no PlanetLab. Assim como (BAUGHMAN; LIBERATORE; LEVINE, 2007) o jogo foi modificado para que a saída do programa fosse em arquivo, então o servidor e os jogadores automatizados executam os *traces* emulando interações reais. Os resultados foram baseados nos traços de 56 jogadores e 1 servidor, onde os jogadores foram separados em dois tipos: cheating e não-cheating.

Baseando-se na latência da rede e no poder de processamento, os jogadores foram divididos em três grupos para análise individual de desempenho. O primeiro grupo foi composto por 45 latências de rede e poder de processamento estáveis, onde 35 dos jogadores eram não-cheaters e 10 eram cheaters. O segundo grupo foi composto por 5 latências de rede estáveis mas poder de processamento instável, 3 dos quais eram não-cheaters. Por fim, o terceiro grupo foi composto de 6 latências de rede flutuantes porém com poder de processamento estável, sendo 2 desses não-cheaters.

Os resultados encontrados relativos à detecção de cheating foram considerados satisfatórios pelos autores, atingindo cerca de 93% de detecção em testes com o terceiro grupo de jogadores.

3.2.6 Detecção automatizada

Laurens et al. (2007) apresentam um trabalho buscando explorar os padrões criados por jogadores cheaters através do uso de detecção automatizada. A solução apresentada no trabalho baseia-se na definição de que jogadores cheaters apresentam um padrão de comportamento diferentes de jogadores honestos, independentemente da forma de cheating que estejam utilizando.

Utilizando uma detecção de padrões de comportamento é possível detectar qualquer tipo de cheating, sendo este conhecido ou não pelos desenvolvedores do jogo, permitindo um maior tempo para o desenvolvimento de um *patch* corretivo. Além disso, caso o jogador cheater conheça o sistema de análise de comportamento, isso poderá forçar o jogador a diminuir a vantagem ganha através do cheating. Entretanto, a solução considera apenas o uso de uma arquitetura cliente-servidor e, essencialmente, avalia a utilização de *wallhacking*, ou seja, a alteração do cliente de jogo para visualização através de paredes.

Os testes foram realizados utilizando-se três jogadores com experiência no jogo *CounterStrike Source*. Os resultados foram apresentados na forma de *cheat-scores* onde, quanto maior a pontuação de um jogador, mais provavelmente o sistema acusará o jogador de cheating. A variação na pontuação variou largamente com valores entre 1,4 vezes e 2000 vezes maior entre jogadores cheaters e não cheaters, possuindo um valor médio de 22 vezes. Ou seja, um mesmo jogador, em um mesmo mapa, apresentou em média uma pontuação 22 vezes mais alta quando utilizou técnicas de cheating do que quando jogou normalmente. Com estes valores foi possível atingir mais de 70% de detecção de jogadores cheaters.

3.2.7 Comparação de eventos

Entre os trabalhos listados, o que mais se aproxima da proposta apresentada pelo presente trabalho foi o trabalho proposto por Izaiku et al. (2006). Neste trabalho é considerada uma arquitetura *peer-to-peer*, onde o mundo virtual é dividido em subáreas e, para cada subárea, é atribuído um servidor nodo responsável, que deverá enviar atualizações do estado do jogo para os demais jogadores presentes ou interessados nesta área. Para detecção de cheating é utilizado o conceito de nodo monitor, sendo que cada jogador envia eventos tanto para nodos monitores quanto para o nodo responsável. Para evitar conlúio, devem ser utilizados diversos nodos monitores, que devem entrar em um consenso utilizado um sistema de votação, sendo a decisão realizada por maioria simples.

O trabalho faz uso de *timeslots*, onde um *timeslot* é o tempo que todos os jogadores possuem para enviar um evento. Caso o evento enviado por um jogador seja recebido pelos monitores em um *timeslot* diferente, porém dentro de um tempo de tolerância, o nodo responsável deve determinar a qual *timeslot* aquele evento pertence. O tempo de tolerância é utilizado para garantir que jogadores não enviem seus eventos com atraso proposital.

O objetivo desse trabalho é realizar a detecção do envio de lista de eventos (ou ações) falsificadas por jogadores ou pelo próprio nodo responsável. Para realizar a detecção da ocorrência de cheating, os nodos monitores realizam um comparativo entre os estados enviados, presentes no nodo, com o evento enviado pelo jogador.

Um jogador pode tentar realizar um evento falso, como por exemplo utilizar um item que ele não possua. Este caso é prevenido em um nodo responsável, ou monitor, definindo-se que o próximo estado de jogo, utilizando uma lista de eventos com eventos impossíveis, é idêntica ao próximo estado utilizando o evento impossível.

Um nodo responsável pode tentar falsificar a lista de eventos de duas formas: alterando a lista de eventos e ignorando eventos enviados por algum jogador. Para tratar o primeiro caso é utilizado um mecanismo de assinatura digital, onde cada jogador adiciona uma assinatura à mensagem de evento. Os nodos responsável e monitores fazem o comparativo entre eventos recebidos por jogadores e a confirmação de autenticidade é feita conferindo-se a assinatura digital. Se os nodos responsável e o monitores possuírem a mesma assinatura digital para todos os eventos, apesar de terem diferentes estados de jogo, algum jogador é considerado como tendo enviado diferentes eventos no mesmo *timeslot*. Caso contrário, ou um nodo monitor ou o nodo responsável é acusado de cometer cheating. Este caso é detectado através do envio do hash da lista de eventos recebida no *timeslot* anterior à mensagem de evento enviada. Para detectar que um nodo responsável está ignorando eventos enviados por jogadores é utilizado um mecanismo de votação entre o nodo responsável e os nodos monitores.

Jogadores podem tentar enviar diferentes estados para os nodos responsável e monitores. Para detecção deste caso é realizada a comparação dos eventos com assinaturas recebidos de cada jogador.

Para análise do custo da solução (*overhead*) foi considerado que cada mensagem enviada pelo jogador (pacote contendo o evento) possui 64 bytes, que cada jogador deverá realizar o envio de um evento (tamanho do *timeslot*) a cada 200 ms e que o número máximo de jogadores em uma determinada área é 100. Considerando estes valores, o custo do mecanismo para cada jogador considerando 2 nodos monitores é de 5.12 Kbps. Cada nodo monitor terá um custo de 256 Kbps para recebimento das mensagens enviadas pelos jogadores. Mensagens utilizadas para o mecanismo de votação foram consideradas negligíveis por possivelmente possuirem um máximo de 16 bytes e não serem enviadas a cada *timeslot*. Considerando a ocorrência de cheating e a necessidade de *rollback* (regresso do estado de jogo para o último estado válido), o custo em nodos monitores pode ser de até 512 Kbps para recebimento dos eventos enviados por jogadores e envio de pacotes para o *rollback*.

Testes práticos foram realizados utilizando o PlanetLab considerando o uso de 1 nodo responsável, 2 nodos monitores e 50 jogadores em uma subárea, utilizando latências variadas. Resultados mostram que o tempo médio para detecção de cheating foi de 1024 ms, e o tempo médio para execução de *rollback* foi de 8023 ms.

3.2.8 Considerações

Entre os trabalhos relacionados é possível perceber a ausência de tratamento para cheatings que possam causar danos ilimitados ao mundo virtual. Diversas soluções propostas utilizam mecanismos híbridos, havendo a existência de um agente central confiável para tratamento de casos onde haja a suspeita de cheating. Também é possível perceber a problemática em relação a desempenho em muitas das soluções, sendo possível diminuir a carga do servidor central em alguns casos, porém havendo um possível aumento de carga em *peers*, podendo tornar a solução impraticável.

Especificamente no caso do trabalho apresentado por (IZAIKU et al., 2006) é possível utilizar o mecanismo proposto para realizar proteção da economia virtual. Entretanto o mecanismo utiliza uma variável de tempo (*timeslot*) para realizar a detecção, sendo possível que o efeito do cheating espalhe-se antes de que haja a detecção.

A Tabela 3.1 demonstra um comparativo entre os trabalhos utilizando características relevantes para este trabalho.

A coluna arquitetura indica a arquitetura alvo (ou requisito), sendo consideradas as

Tabela 3.1: *Tabela comparativa entre soluções anti-cheating.*

Trabalho	Arquitetura	Requer elemento confiável	Tipo de cheating detectado	Proteção à Economia Virtual
DaCaP	Híbrida	Sim	Diversos	Vulnerável em grupos de jogadores
DaCa	Híbrida	Sim	Diversos	Não
AC/DC	P2P	Sim	Time cheating	Não
Lockstep	C/S	Sim	Time cheating	Não
AS	C/S	Sim	Time cheating	Não
CCP	C/S	Sim	Time cheating	Não
Detecção automatizada	C/S	Sim	Diversos	Pode ser aplicável (com modificações)
Comparação de eventos	P2P	Não	Diversos	Aplicável (detectando-se em períodos de tempo)

arquiteturas apresentadas na Seção 2.2: *peer-to-peer* (P2P), cliente-servidor (C/S) e híbrida. *Requer elemento confiável* é referente à necessidade de um elemento no qual o sistema pode confiar plenamente a qualquer momento durante a execução do sistema. Os tipos de cheating são os listados na Seção 2.4, porém é possível encontrar a classificação *diversos*, sendo esta classificação dada a soluções que consideram diversos tipos de cheating. Por fim a coluna *Proteção à economia virtual* descreve a possibilidade de utilização da solução para proteção contra ataques que afetem a economia virtual do jogo.

Como pode ser percebido através das considerações apresentadas por esta seção, não há uma solução eficaz que contemple o uso de uma arquitetura composta por elementos não confiáveis e que consiga proteger o mundo virtual contra ataques que possam, potencialmente, afetar todos os jogadores presentes. Desta forma faz-se necessário o desenvolvimento de uma solução que consiga restringir o impacto destes tipos de cheating. A solução apresentada pelo presente trabalho considera a contenção do efeito de cheating dentro de um espaço do jogo, sendo possível limitar este impacto ao espaço delimitado. O Capítulo 4 descreve este mecanismo.

4 PROPOSTA

Esse capítulo apresenta uma proposta de solução para proteção da economia virtual de um MMOG. A solução proposta é dividida em duas etapas distintas: modelo e simulador. A Seção 4.1, onde o modelo é apresentado, descreve a proposta para restrição do impacto da ocorrência de cheating a um espaço virtual limitado, apresentando também a arquitetura considerada e uma classificação para o estado do jogador. A Seção 4.2, denominada simulador, apresenta a aplicação deste modelo com características práticas, ilustrando como o modelo (abstrato) pode ser aplicado em jogos reais através da utilização de simulação sobre um jogo modelo.

4.1 Modelo

Essa seção apresenta a primeira parte da solução proposta. Nela é descrita a forma genérica da solução, ou modelo, onde são considerados características genéricas de MMOGs para elaboração de um modelo de proteção à economia global.

4.1.1 Introdução

A presente capítulo descreve uma solução para proteção da economia virtual para MMOGs. Cabe aqui relembrar o objetivo principal da proposta apresentada: proteção da economia virtual contra jogadores maliciosos, uma vez que estes podem cometer cheating que, potencialmente, afetam todos os demais jogadores.

Através da utilização de uma divisão do mundo virtual em pequenos espaços chamados células, e seu agrupamento em regiões, busca-se delimitar a área de efeito de um cheating a uma célula. Para realizar essa restrição a um espaço virtual limitado, utiliza-se um mecanismo de detecção de cheating que realiza a verificação do estado do personagem do jogador (representação virtual do jogador) conforme este se move no mundo virtual.

Utilizando uma restrição limitada a uma célula, objetiva-se evitar que o efeito do cheating espalhe-se pelo mundo virtual, afetando outros jogadores, gerando insatisfação e possíveis desistências. Outra consequência desta restrição é o custo para efetuar a correção do estado (*rollback*). Como a quantidade de jogadores afetados é menor, menos jogadores precisarão experimentar uma alteração no fluxo de execução do jogo.

Para a solução é utilizado o conceito de estado do jogador. Define-se o estado do jogador como sendo um elemento composto pelas variáveis pertencentes ao personagem que podem ser afetados no decorrer do tempo de jogo. Exemplos de variáveis que pertencem ao estado do jogador são: quantidade de ouro, número de mortes, valor de ataque, entre outros.

Outro conceito utilizado é mensagem. A solução prevê a utilização de dois tipos de

mensagens: mensagem de jogo e mensagem de segurança. Para fins de simplificação consideramos que ambas possuem o mesmo peso (ou seja, possuem o mesmo tamanho e custo de rede para envio). Apesar disso, como pode ser notado na descrição da solução, a mensagem de segurança pode ser simplificada para possuir um custo menor do que a mensagem de jogo, pois pode ser reduzida à transmissão de certos elementos do estado relevantes ao mecanismo de detecção. Assume-se que a autenticidade e garantia de entrega e de ordem de entrega são confiáveis, pois existem soluções específicas para este tipo de problema como (GAUTHIERDICKEY et al., 2004).

Em relação à rede de conexão entre jogadores e servidores, e entre servidores e servidores, abstrai-se o conceito de rede e considera-se que o sistema possui uma rede confiável. Jogadores não perdem a conexão, nem experimentam atrasos no envio ou recebimento de mensagens, além de não haver limitações de largura de banda. Argumenta-se que, devido ao escopo deste trabalho, essas simplificações são essenciais para modelagem da solução e não impactam de forma significativa nos resultados obtidos.

4.1.2 Arquitetura

Essa seção apresenta as características da arquitetura considerada no presente trabalho. Uma definição de alto nível da arquitetura a define como um sistema de computação voluntária, com recursos de capacidade acima do considerado normal, como aquelas pertencentes a empresas ou a jogadores com maior poder aquisitivo.

Considerando a classificação de arquiteturas apresentadas na Seção 2.2, a arquitetura pode ser classificada como *peer-to-peer*. Entretanto o número de recursos disponíveis é menor do que o número de jogadores (diferindo de grande parte das soluções P2P), uma vez que espera-se que poucos jogadores possuam recursos com capacidade computacional suficiente para utilização. Entretanto, assume-se que existe a possibilidade de acesso de jogadores às máquinas contendo o estado de jogo, seja por pertencer ao próprio jogador, ou por não possuir requisitos mínimos que a protejam contra uma invasão (e.g. *firewall*). Portanto, o sistema como um todo pode ser considerado vulnerável a cheating.

Quanto à divisão do mundo virtual, é utilizada uma divisão por regiões e células conforme apresentado na Seção 2.3. O desenvolvimento da solução não leva em consideração o escalonamento das regiões. Este escalonamento pode ser utilizado para melhoria de desempenho e obtenção de uma melhor qualidade de jogo para o jogador, por exemplo, atribuindo regiões mais populosas para servidores com maior capacidade. Esse modelo de distribuição de regiões foi abstraído para fins de simplificação, onde todos os servidores possuem a mesma capacidade e as regiões são distribuídas arbitrariamente.

Também considera-se que a migração de jogador entre servidores é considerada confiável. Em relação a servidores de conexão e servidores de jogo, ou seja em qual servidor o jogador está conectado e qual servidor o estado do jogo é simulado, é definido que ambos estarão presentes na mesma máquina física, uma vez que o tratamento deste tipo de problema foge do escopo deste trabalho, além de não afetar os resultados da solução proposta.

4.1.3 Mecanismo de detecção

Conforme o objetivo descrito na seção anterior (evitar que os efeitos da ocorrência de cheating espalhem-se globalmente e afetem muitos usuários) e considerando a divisão do mundo virtual conforme descrito na Seção 2.3, utiliza-se um mecanismo de detecção para restringir os efeitos da ocorrência de cheating a uma única célula do mundo virtual.

O mecanismo de detecção proposto compara o estado do personagem do jogador

quando ele saí de uma dada célula com um *estado atual estimado*. Este estado atual estimado é calculado com base no *estado inicial* do personagem do jogador, no tempo gasto pelo jogador na célula em questão e em informações sobre a célula, sendo estas informações fornecidas pelo desenvolvedor ou obtidas através de análise. O estado inicial do personagem é obtido quando o jogador entra na célula, ou quando ele conecta-se ao jogo, o último a ter ocorrido.

A Figura 4.1 ilustra o funcionamento do mecanismo onde: (a) representa a entrada de um jogador em uma célula e o envio de seu estado para o servidor; (b) o jogador dedica um período interagindo com o ambiente virtual (ou outros jogadores); (c) o jogador sai de sua célula atual para uma outra célula; (d) antes de executar qualquer ação em sua nova célula, o jogador é declarado como cheater ou honesto, e a punição necessária é efetuada.

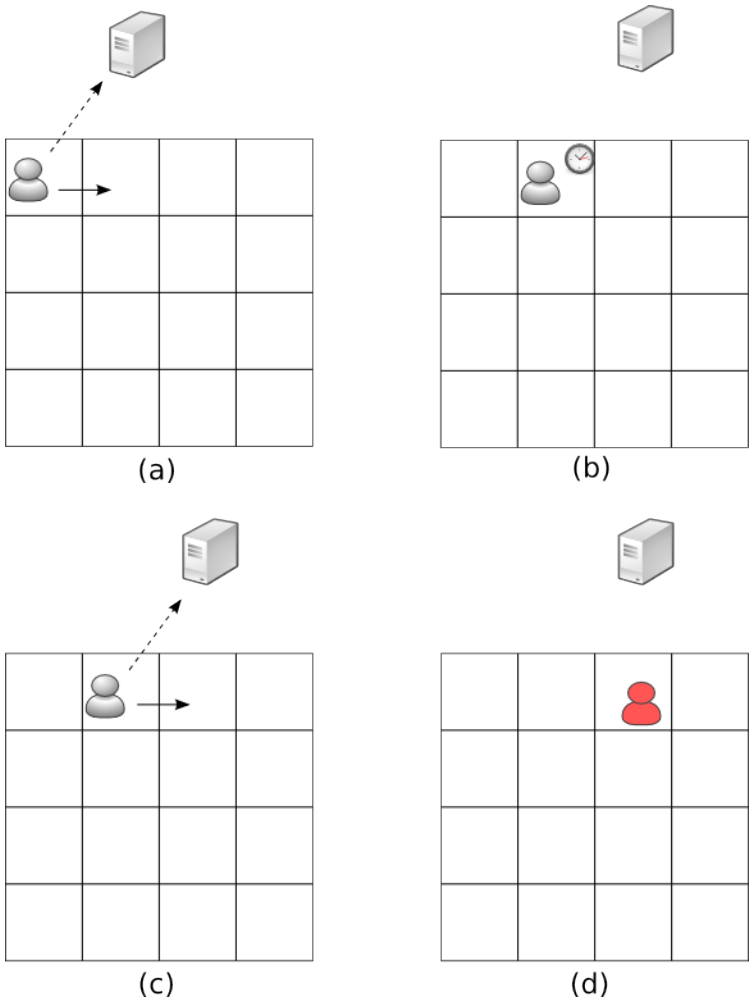

Figura 4.1: *Mecanismo de detecção.*

A detecção ocorre caso a diferença entre o estado atual do personagem em relação ao estado quando este personagem entrou na célula seja maior que um dado limite. Neste caso o sistema de detecção lança uma suspeita de cheating, que deverá ser tratado pelo sistema de acordo com a política definida, o que pode variar dependendo da empresa e jogo em questão. Valve Anti-Cheat System (VALVE, 2008), por exemplo, aguarda dias antes de efetivamente remover (banir) jogadores cheaters. De acordo com a empresa, esta técnica permite detectar mais jogadores que estejam cometendo cheating utilizando o mesmo mecanismo.

4.1.3.1 Tamanho da célula

A premissa apresentada neste trabalho considera que todas as células possuem tamanhos iguais. Entretanto, é possível realizar variações no tamanho de célula, utilizando-se células maiores e, consequentemente, menos células no espaço virtual, ou utilizando-se células menores (mais células). O tamanho da célula influencia o mecanismo de detecção em dois aspectos principais: contenção do efeito do cheating e desempenho do sistema.

Utilizando-se células maiores, a frequência com que os jogadores vão migrar para uma célula diferente será menor, portanto menos mensagens serão trocadas para fins de segurança. Entretanto o efeito do cheating se espalhará para um espaço virtual maior, podendo afetar mais jogadores. Outra consequência do uso de células maiores é o tempo para detecção de cheating, que deverá ser mais longo na maioria dos casos.

Ao diminuir-se o tamanho das células, a frequência das migrações aumenta, deteriorando o desempenho do sistema através do uso de muitas mensagens apenas para fins de segurança. Contudo, a área de efeito do cheating será menor, afetando menos jogadores, além de a detecção ser realizada mais cedo, o que também contribui para uma menor quantia de jogadores afetados.

4.1.4 Classificação de estado do jogador

Para realização do comparativo entre diferentes estados do jogador, é necessário um cálculo que permita estimar o resultado da evolução desse estado durante um dado período de tempo. Entretanto, pode-se afirmar que existem diversas formas para realização do cálculo do estado do jogador. Esse cálculo é afetado por diversas variáveis, muitas delas dependentes diretamente das regras do jogo ou da influência do ambiente em si, como outros jogadores ou eventos que alterem este estado.

Buscando limitar as características das fórmulas propõe-se uma classificação do estado do jogador. Com esta classificação é possível limitar os dados necessários para o cálculo da evolução do estado de acordo com características comuns a diferentes elementos que compõem esse estado. Na classificação apresentada nesta seção, os elementos que compõem o estado completo do jogador são divididos em dois grupos:

- estado previsível;
- estado imprevisível.

Por *estado previsível* define-se as variáveis que possuem uma relação direta entre tempo e localização no mundo virtual. Isso significa que, considerando um conjunto de tomadas de decisão de um jogador A em uma dada localidade do mundo virtual, dada a existência de um segundo jogador (B), que siga esta mesma linha de tomada de decisões, e considerando a mesma localidade, as progressões dos estados de A e B serão similares. Apesar de normalmente existirem variáveis com valor aleatório que alterem o estado do jogador (e.g. valores diferentes de ouro carregados por um monstro), esta aleatoriedade é controlada, possuindo parâmetros definidos em um dado tempo.

Como exemplo de estado previsível pode-se utilizar a coleta de recursos em uma área do mundo virtual, como a mineração de pedaços de ouro em uma caverna. Normalmente há uma quantidade limitada de recursos que um dado jogador pode adquirir em um período de tempo, sendo que este recurso torna-se indisponível por um período de tempo antes de se tornar novamente disponível para outros jogadores, ou até mesmo para o mesmo jogador. Seguindo o exemplo, pode-se considerar a existência de um veio que permite

ao jogador a mineração de 10 pedaços de ouro em um período de 1 minuto, tornando-se disponível novamente após 5 minutos de seu consumo.

A utilização de estados previsíveis permite a inferência da variação do estado num dado período de tempo. Utilizando o exemplo da mina de ouro, um jogador que permaneça menos que 1 minuto na localidade em questão não poderia possuir uma evolução do estado que indique recursos coletados com um valor superior a 10 pedaços de ouro.

A segunda categoria de estado é o *estado imprevisível*. Esta classificação compreende variáveis que são influenciadas pela ação de outros jogadores ou eventos no ambiente. Neste caso não é possível considerar uma relação direta entre tempo e localidade, devido a interferências por ações que não sejam previamente controladas (como ação de outros jogadores). Portanto, não é possível estimar um estado baseando-se em informações do espaço virtual em questão. Tipos de eventos que podem influenciar neste estado são, por exemplo, número de jogadores, alterações econômicas e níveis de personagens.

Exemplos de estado imprevisível são o número de jogadores derrotados, número de monstros enfrentados, número de tiros disparados, quantia de ouro roubado. Dois jogadores diferentes podem possuir diferentes valores para o número de vitórias ou derrotas para o mesmo tempo gasto no mesmo espaço virtual (o primeiro jogador pode ter derrotado dez jogadores, enquanto o segundo apenas um).

4.1.5 Evitando conlúio

Considerando o ambiente de computação voluntária apresentada na seção 4.1.2, assume-se que os servidores são vulneráveis e, portanto, não podem ser confiáveis. Dessa forma, confiar a decisão de declarar um jogador como cheater a um único servidor pode tornar-se um risco, pois este servidor pode ser manipulado e declarar um jogador honesto como sendo cheater em favor de outro jogador, ou grupo de jogadores, buscando retirar um possível rival de competição.

Para contornar este tipo de problema, e devido à possível presença de mais servidores voluntários do que regiões a serem controladas, esta solução considera a utilização de replicação de servidores para fins de verificação. Estes servidores de propósito específico serão chamados de servidores de segurança no contexto deste trabalho (para fins de diferencia-los de servidores de jogo, que controlam a região e recebem todas as ações dos jogadores em todas as rodadas).

Quando um dado jogador entra em uma célula, ou conecta-se ao jogo, seu estado é enviado a todos os servidores de segurança da sua região atual, além do servidor de jogo. Considerando o mecanismo de detecção descrito na Seção 4.1.3, que será ativado quando o jogador sair de sua célula atual, usa-se um mecanismo de votação utilizando as respostas das computações de todos os servidores de segurança (onde um jogador é declarado como cheater ou honesto). A decisão final é feita através de uma maioria simples, caso a maioria decida como o jogador sendo suspeito de cheating, este jogador é declarado ao sistema.

Mesmo com o mecanismo de votação existe a possibilidade de dois servidores fazerem conlúio, e acusarem falsamente um jogador honesto como cheater. Para resolver este problema considera-se a utilização de diversas réplicas para servidores de segurança, aumentando o número de servidores necessários para obter-se uma maioria na votação e, portanto, aumentando a segurança. Abstrai-se aqui o número de servidores replicados, variando conforme a quantidade de recursos disponíveis.

Diversos projetos consideram a utilização de replicação para tratar problemas similares. Um dos projetos que, conhecidamente, utiliza replicação de tarefas é o BOINC (University of California, 2010), utilizando, por padrão, três replicas para uma única ta-

refa.

4.2 Simulador

Para demonstrar a aplicabilidade da solução proposta na Seção 4.1, foi desenvolvido um simulador com características genéricas presentes em diversos MMOGs. Também foram desenvolvidas técnicas para estimativa de estado, buscando comprovar a hipótese lançada neste mesmo capítulo, de que os efeitos da ocorrência de um cheating pode ser restringida a um espaço virtual (célula). Neste sentido foram desenvolvidas duas técnicas para estimar o estado do jogador, sendo uma para estados previsíveis e uma para estados imprevisíveis.

As seções seguintes descrevem o simulador desenvolvido, descrição de comportamento dos jogadores (honestos e cheaters), bem como as técnicas para estimativa de estado.

4.2.1 Introdução

Conforme a descrição de MMOGs apresentada no Capítulo 2, o desenvolvimento de um jogo por si só pode ser tratado como um simulador. Considerando essa afirmação, esta seção descreve um jogo denominado *Miners*, onde os jogadores controlam personagens representando mineradores em uma caverna, buscando a extração de ouro.

No entanto, devido ao objetivo deste simulador ser representar uma quantidade de jogadores em larga escala simultaneamente (acima de 1000), a utilização de jogadores reais se torna proibitiva. Para resolver esse problema utiliza-se automação da ações dos jogadores, tornando o jogo (normalmente um DVE) em um simulador analítico (conforme descrito no Capítulo 2).

Nesse simulador os jogadores, que normalmente controlariam um personagem, são substituídos por um comportamento pré-definido, conforme descrito na Seção 4.2.3. Simulou-se também o comportamento de jogadores cheaters, este comportamento é descrito na Seção 4.2.4.

O desenvolvimento do jogo considera uma divisão do tempo em turnos, onde cada jogador deve enviar uma ação por turno. Outras características consideradas foram a presença de um economia virtual, interação entre jogadores e a existência de um ambiente virtual.

Cabe ressaltar-se que não faz parte dos objetivos deste trabalho a criação de uma réplica exata de um MMOG, apenas dos aspectos mais relevantes aos propósitos aqui citados. Desta forma aspectos como gerenciamento do estado do jogador, representação gráfica, oscilações de latência de rede, limitações de largura de banda, desconexões e desistências, não são consideradas ou são simplificados. Obviamente, a criação de um jogo completo, aproximando-se de exemplos comerciais de MMOGs, seria impraticável.

Para o desenvolvimento do simulador (para fins de testes) foi utilizado o Microsoft Visual Studio 2008 e utilizou-se o Windows 7 (32 bits) como ambiente de desenvolvimento e execução. A escolha deste ambiente de desenvolvimento e execução deu-se devido à similaridade com o trabalho desenvolvido por (BEZERRA, 2009), reaproveitando-se algumas idéias básicas utilizadas para o desenvolvimento do simulador utilizado neste trabalho (questões como representação gráfica dos jogadores e distribuição de células entre regiões).

4.2.2 Jogo: Miners

Essa seção descreve o jogo *Miners*, onde cada jogador controla um personagem virtual que possui a profissão de minerador de ouro. O objetivo do jogo é obter a maior quantidade de peças de ouro possível.

O jogo se ambienta em uma grande caverna, sendo possível realizar a mineração de ouro em qualquer lugar desta caverna. Também é possível realizar a interação com outros jogadores, porém limita-se à interação entre jogadores para ações de jogo, eliminando a questão de interação social por não ser pertinente aos objetivos deste trabalho.

Cada turno do jogo possui a duração de 100ms, sendo que todos os jogadores devem enviar sua ação ao servidor de jogo a cada turno. A cada jogador são possíveis três diferentes ações: movimentação, minerar ouro do solo (*gather*) e roubar ouro de outros jogadores (*steal*).

A representação gráfica do jogador é feita de forma simples e bidimensional. Cada jogador é representado no mundo virtual como um pequeno quadrado de 2 pixels de largura.

O mundo virtual como um todo possui 640 pixels de altura por 640 pixels de largura, igualmente dividido entre 16 regiões quadráticas, cada região possuindo 160x160 pixels. Cada região é subdividida em 64 células, cada célula possuindo uma área de 20x20 pixels.

O jogo considera a divisão da caverna em cavernas menores, onde os jogadores podem fazer a interação com o ambiente virtual. Para fins de simplificação, é realizada uma divisão de células semelhante à de cavernas menores, ou seja, uma caverna menor é equivalente à uma célula. Cada uma das cavernas menores (doravante células) possui um valor que é possível minerar do solo (valor de *gather*). Esse valor determina a quantidade de ouro que um jogador pode minerar em uma rodada, sendo que este valor varia entre 1 a 10 peças de ouro.

O valor de *gather* também é utilizado para determinar o máximo que um jogador pode roubar de outro jogador. Este valor (valor de *steal*) é definido como metade do valor de *gather*. Um jogador pode realizar uma ação de *steal* em qualquer jogador que esteja na mesma célula.

Em relação à ação de movimento, um jogador pode se mover entre 1 a 5 pixels por turno. Esta ação de movimento pode ser feita em qualquer direção.

A Figura 4.2 exibe uma screenshot do simulador. Na figura é possível notar a representação dos jogadores, como sendo os pontos brancos para jogadores honestos e vermelhos para cheaters. Também é possível perceber as divisões entre regiões e células, linhas coloridas e linhas verdes, respectivamente.

4.2.3 Comportamento dos jogadores

Devido à complexidade na utilização de jogadores reais para a validação e testes da solução proposta, é feita a transformação de um DVE (jogo tradicional) em uma simulação analítica, tornando possível a realização de testes envolvendo centenas ou milhares de jogadores. Entretanto, a utilização de simulação analítica levanta a questão de como será realizada a simulação dos jogadores e suas interações com o ambiente virtual.

Para solucionar este problema, utiliza-se a mesma abordagem utilizada para o desenvolvimento de *bots* em jogos. *Bots* são, essencialmente, programas criados para automatizar a ação de um jogador, usualmente objetivando ganhar vantagem sobre os demais jogadores, uma vez que o jogador não precisa estar presente para que seu personagem evolua. Estes programas são, frequentemente, desenvolvidos por jogadores experientes

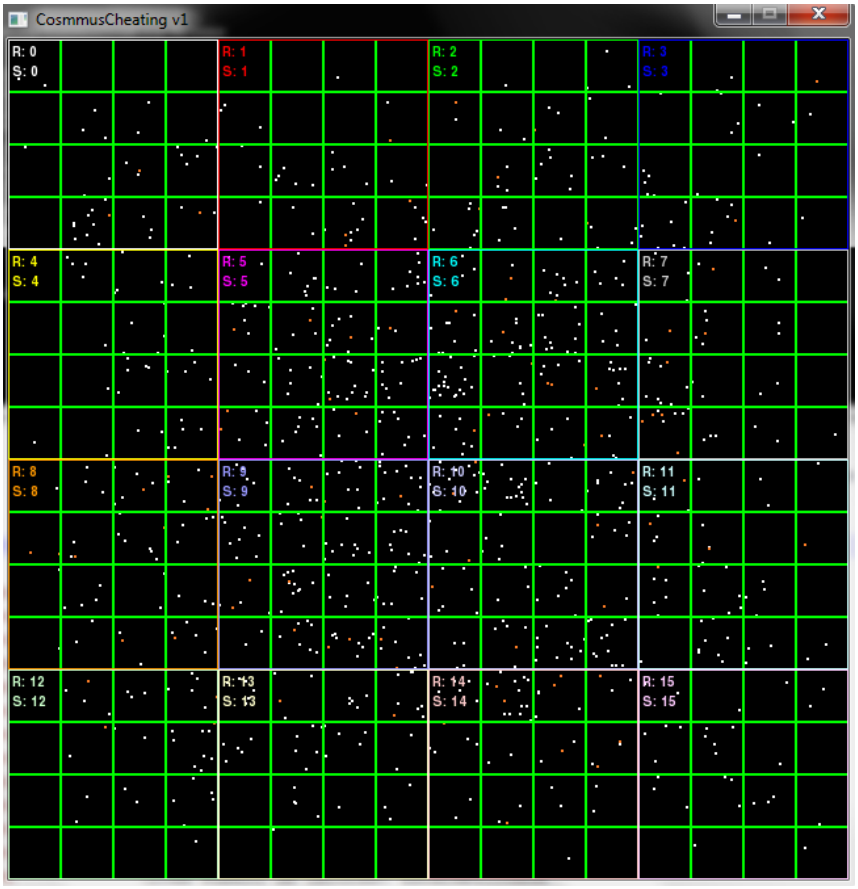

Figura 4.2: *Screenshot de Miners.*

ou desenvolvedores do jogo, que possuem um amplo conhecimento do funcionamento do jogo. São utilizadas as ações mais comuns e roteiros de ações executadas por jogadores reais, de forma que o *bot* possa se passar por um jogador, e não como ações automatizadas.

Devido ao jogo em questão não ser um jogo real, portanto não possuindo jogadores experientes, foi utilizado o conhecimento do desenvolvedor para elaborar uma sequência de ações de forma a simular as ações de um jogador. O roteiro de ações é seguido conforme o seguinte:

1. Ao iniciar, cada jogador escolhe uma área de *gather* (*gather site*), onde ele irá coletar ouro do solo;

2. Também é escolhido quanto tempo ele irá gastar nessa mesma área (máximo de 15 segundos);

3. Outro parâmetro é a quantidade de locais que o jogador irá visitar dentro desta área (máximo de 5 locais), sendo estes locais com o mínimo de 10 e máximo de 40 pixels de distância um do outro;

4. Após finalizadas todas as ações dentro deste *gather site*, o jogador escolhe outra área (outro *gather site*), a pelo menos 160 pixels de distância da sua posição atual, reiniciando o ciclo.

Para simulação das ações de roubo (*steal*) foi considerado que cada jogador possui, associada a ele, uma probabilidade de que ele irá tentar roubar um jogador caso haja

um jogador disponível em sua célula. Esta probabilidade varia entre 5 e 75 porcento (distribuição uniforme) das ações do jogador. Com esta distribuição busca-se simular jogadores que interajam com outros com muita frequência e outros que normalmente jogam individualmente (embora com um mínimo de interação).

4.2.4 Comportamento dos cheaters

De forma similar ao desenvolvimento do comportamento dos jogadores descrito na Seção 4.2.3, o comportamento de jogadores desonestos (cheaters) insere uma ação ilegal (cheat) entre ações regulares normalmente executadas pelo jogador.

Para simular a ocorrência de cheating, um dado valor é adicionado ao valor obtido numa ação de *gather* ou *steal*. Este valor é previamente determinado para cada cheater, sendo uma distribuição uniforme variando entre 1 até 500 peças de ouro inseridas por ação de cheat. Para simular a frequência com que um cheater comete uma ação ilegal também foi utilizado um valor previamente determinado, variando entre 1% à 25% (distribuição uniforme) das ações de *gather* ou *steal*.

Com estes valores busca-se representar jogadores que cometam infrações com baixa frequência ou aqueles que cometem com muita frequência. Também é possível representar jogadores ambiciosos, que criam uma quantidade considerável de recursos, e aqueles que buscam ludibriar o sistema ou outros jogadores, tentando passar desapercebidos com uma pequena quantidade de recursos.

Os efeitos de variações nestes valores para o mecanismo de detecção é discutido no Capítulo 5.

4.2.5 Estimativa de estado

O modelo proposto neste trabalho faz uso de um mecanismo de estimativa de estado com o objetivo de criar um estado passível de comparação com o estado atual do jogador. Desta forma, a eficiência do mecanismo será tão bom quanto mais preciso for o mecanismo de estimativa de estado.

Entretanto, o mecanismo de estimativa de estado depende diretamente das regras do jogo, sendo que um dado mecanismo pode não ser aplicável a dois jogos diferentes. Neste trabalho consideramos o uso de dois mecanismos genéricos, possivelmente aplicáveis a uma ampla gama de jogos, para estimativa do estado do jogador.

O objetivo é ilustrar de que forma a classificação do estado do jogador, descrita na Seção 4.1.4, pode influenciar o mecanismo de estimativa de estado. Para tanto é utilizada uma técnica para cada uma das classificações do estado do jogador. O primeiro mecanismo é dedicado para uso apenas em estados previsíveis, enquanto o outro, apesar de usado apenas em estados imprevisíveis neste trabalho, pode ser adaptado para utilização em ambos os estados.

As seções seguintes demonstram como o estado do jogador pode ser classificado no jogo descrito. Também são descritos cada um dos mecanismos de estimativa de estado do jogador, incluindo uma breve avaliação.

4.2.5.1 Estado do jogador

Conforme a premissa de que estado do jogador são todas aquelas variáveis que possam ser alteradas com o decorrer do tempo de execução da simulação, considera-se que o estado do jogador relevante ao mecanismo de detecção é composto por:

- Quantidade de ouro coletado do ambiente (*gather*);

- Quantidade de ouro roubado de outros jogadores (*steal*);

Portanto, utilizando a classificação descrita na Seção 4.1.4, pode-se dizer que a primeira classifica-se como pertencente ao estado previsível do jogador, pois o valor a ser adquirido ao coletar-se ouro do ambiente pode ser obtido diretamente do jogo (como, por exemplo, em algum arquivo de configuração ou banco de dados que descreva a região em questão).

A segunda, claramente, pertence ao estado imprevisível do jogador, por tratar-se de interações com outros jogadores, onde a quantidade de variações de cenários torna inviável o seu cálculo. Exemplos de variáveis que afetam o cálculo deste estado são o número de jogadores presentes na mesma célula (subcaverna), quantidade de ouro que o jogador carrega (caso seja inferior ao limite estipulado) e a vontade do jogador em interagir com outros.

4.2.5.2 Estado previsível

Para realizar a estimativa do estado previsível do jogador pode-se utilizar informações obtidas no ambiente, base de dados, ou pelas próprias regras do jogo, conforme descrito na Seção 4.1.4. Neste caso específico deste jogo, é possível utilizar as regras do jogo (e os valores registrados na base de dados) para realizar uma estimativa de qual o possível estado do jogador após um dado período em uma dada célula.

Para realizar tal cálculo, são necessárias as informações acerca do posicionamento do jogador, dado conhecido em todos os momentos da execução, e do estado inicial do jogador, dado informado quando jogador ingressa na célula em questão.

O estado inicial do jogador deve incluir um *timestamp* (t_0), constando o tempo da simulação em que este enviou sua mensagem ao servidor (ou seja, ingressou na célula em que se encontra atualmente). Também é necessário o *timestamp* da saída do jogador da célula (t_f), para cálculo do tempo gasto dentro da célula. O tempo gasto dentro da célula é definido como $t = t_f - t_0$. Ressalta-se que é definido $t = 0$ ao inicio do jogo e $t = 1$ ao iniciar-se o turno subsequente ao inicial.

Em relação ao estado inicial do jogador, para a análise do estado previsível considera-se apenas a variável referente à quantidade de ouro coletado (g, *gather value*). Portanto, também possui-se a quantidade coletada inicial (g_0), obtida quando o jogador entra na célula, e a quantidade coletada final (g_f), obtida quando o jogador deixa a célula. A quantidade de ouro obtida dentro da célula pode então ser calculada como sendo $g = g_f - g_0$.

Por fim, utiliza-se a quantidade de ouro que é possível ser obtido em uma ação nesta célula (c), que deve ser obtida da base de dados do jogo. E define-se que o estado previsível estimado do jogador (e_p) é calculado pela seguinte fórmula:

$$e_p = g_0 + (t * c); \tag{4.1}$$

Utilizando o estado estimado do jogador, é realizada uma comparação com o estado atual do jogador. Caso o estado atual do jogador exceda o valor estimado, ou seja $g_f > e_p$, declara-se uma suspeita de cheating.

4.2.5.3 Estado imprevisível

A estimativa do estado imprevisível do jogador não conta com elementos estáticos, tampouco pode ser calculado diretamente através da utilização de regras do jogo ou base

de dados, conforme descrito na Seção 4.1.4. Portanto, é necessário um método de estimativa que considere a volatilidade do ambiente e a evolução das variáveis relevantes, bem como leve em consideração o comportamento dos jogadores.

Para realizar o cálculo de estimativa faz-se uma análise estatística com o comportamento padrão dos jogadores (inicialmente considerados honestos). Caso haja alguma variação no comportamento padrão de um jogador em relação aos demais jogadores, este jogador é declarado como cheater.

Conforme descrito anteriormente, o estado imprevisível do jogador é basicamente composto pela quantidade de ouro roubada de outros jogadores (s, *steal value*). Para estimativa do estado final faz-se uso da média de ouro roubado por jogador (a) em uma dada célula, e utiliza-se duas vezes o desvio padrão desta média (σ) para considerar o desvio de comportamento aceitável.

Desta forma, considerando o estado inicial do jogador como s_0, o estado final do jogador como s_f, o valor total obtido através da ação de roubo (*steal*) como sendo $s = s_f - s_0$, a média de ouro roubado da célula como a e o desvio padrão como σ, o cálculo do estado imprevisível estimado do jogador (e_i) é dado como:

$$e_i = s_0 + (a + (2 * \sigma)); \tag{4.2}$$

Para detecção é realizado o comparativo entre o estado imprevisível estimado (e_i) e o estado imprevisível do jogador ao sair da célula (s_f). Caso o estado do jogador seja maior do que o estado estimado, ou seja $s_f > e_i$ é declarada uma suspeita de cheating.

A obtenção da média de estado pode ser feita com duas diferentes soluções. A primeira seria uma análise estática, considerando todos os jogadores como sendo honestos ou utilizando um grupo de controle (jogadores confiáveis, como um teste beta para participantes selecionados). A segunda solução é a utilização dos valores dos jogadores honestos para atualização do valor de média e, por consequência, o desvio padrão, com o decorrer do tempo de jogo.

Uma solução ideal poderia fazer a utilização das duas soluções para realizar uma estimativa de estado. Neste trabalho considera-se apenas o uso de uma análise estática, sendo feita através de 30 execuções de 15 minutos de jogo com 100% de jogadores honestos.

4.2.5.4 Discussão sobre estimativa de estado

Ambos os cálculos para estimativa de estado são modelos que consideram o jogo exemplo descrito na Seção 4.2.2. Conforme descrito anteriormente, os cálculos para estimativa de estado devem ser específicos para cada caso (jogo), sendo as técnicas aqui descritas exemplos da aplicabilidade do sistema de detecção.

Em relação às técnicas para estimativa em si, ambas possuem propósitos definidos de ilustrar a aplicabilidade e as características de cada um dos tipos de estado do jogador. A primeira técnica pode ser considerada mais rígida e utiliza uma estimativa de limite superior, enquanto a segunda técnica se torna mais maleável, podendo ser alterada de acordo com o comportamento dos jogadores, e utiliza um limite de comportamento aceitável de acordo com o comportamento dos próprios jogadores.

Apesar de a segunda técnica possuir maior maleabilidade e moldar-se melhor ao comportamento dos jogadores, ela permite a presença de falsos positivos, ou seja, jogadores honestos que estão fora do comportamento padrão. Embora não seja um comportamento frequente, a existência de jogadores honestos que superam outros jogadores (dominam o jogo com maior destreza do que os demais) que são declarados como cheaters pode ocasionar frustração para alguns jogadores.

Portanto é necessário que a medida tomada ao declarar-se uma suspeita de cheating preveja a possibilidade de existência de um falso positivo quando utilizando este tipo de técnica. A técnica, por sua vez, deve buscar reduzir o número de falsos positivos ao mínimo possível.

O Capítulo 5 exibe os resultados em relação às técnicas descritas, comprovando a eficiência do uso da segunda técnica em relação à primeira e exibindo a presença de falsos positivos para os mais diversos casos.

5 TESTES E RESULTADOS

O presente capítulo apresenta os testes executados utilizando o mecanismo e modelos propostos no Capítulo 4. Este capítulo divide-se conforme o seguinte: a Seção 5.1 descreve os métodos e métricas utilizados para avaliação; as Seções 5.2 e 5.3 apresentam os resultados referentes aos dois principais aspectos avaliados (eficácia e desempenho); a Seção 5.4 exibe os resultados relativos à variação no comportamento de jogadores cheaters, buscando avaliar seus efeitos no mecanismo de detecção; por fim, a Seção 5.5 apresenta algumas considerações em relação aos resultados obtidos.

5.1 Metodologia de avaliação

Esta seção descreve a metodologia utilizada nos testes para avaliação da solução proposta, sendo dividida em duas seções que descrevem os aspectos a serem avaliados e os parâmetros e configurações utilizados na simulação, respectivamente.

5.1.1 Aspectos avaliados

O objetivo dos testes aqui apresentados é validar e avaliar o modelo proposto, através da utilização do simulador e mecanismos de estimativa de estado, apresentados no Capítulo 4. Busca-se avaliar dois principais aspectos da solução: eficácia e desempenho.

Eficácia relaciona-se diretamente à capacidade da solução em conter o efeito da ocorrência de um cheating em uma área do mundo virtual. Esta contenção é realizada utilizando um mecanismo de detecção de cheating, e é este mecanismo que é avaliado. Para avaliar a eficácia, jogadores cheaters são inseridos no jogo (cometendo cheating em algumas de suas ações), e é verificado em que momentos esses cheaters foram detectados.

A avaliação de desempenho busca analisar o custo do mecanismo de detecção para o sistema. Como métrica de avaliação utiliza-se o número de mensagens trocadas para fins de segurança em todo o sistema, incluindo mensagens de servidores para jogadores, jogadores para servidores e servidores para servidores. Normalmente é feita uma comparação com o número de mensagens trocadas normalmente durante o decorrer do jogo (comandos enviados pelos jogadores ou entre servidores). A esta relação entre o número de mensagens de segurança e o número de mensagens de jogo dá-se o nome de *overhead*.

Overhead é o valor obtido ao comparar-se a quantidade de mensagens de jogo trocadas e a quantidade de mensagens de segurança trocada, sendo o *overhead* a porcentagem de mensagens adicionais trocadas devido ao uso do mecanismo de segurança. Em outras palavras, o que busca-se mostrar com a análise de *overhead* é a quantiade de investimento adicional, em recursos computacionais, necessário para utilização do mecanismo de segurança em relação ao investimento empregado para manter-se o jogo sem o mecanismo.

O número de mensagens foi escolhido como métrica devido à grande variação de tamanho de mensagens entre diferentes jogos, onde diferentes jogos podem assumir uma ampla variação de tamanho de mensagens (valores como 8 Kb e 64 Kb).

5.1.2 Parâmetros da simulação

Quatro diferentes tipos de testes foram executados, considerando os seguintes parâmetros:

- número de jogadores;
- porcentagem de jogadores cheaters;
- número de células;
- número de servidores de segurança.

Para todos os testes é utilizado o mesmo cenário padrão, exceto em testes que buscam variar o parâmetro em questão. O cenário padrão para execução é constituído de:

- 5000 jogadores;
- 96 servidores, sendo 1 servidor de jogo e 5 servidores de segurança para cada região;
- 1024 células (20x20 pixels cada uma);
- 15 minutos de jogo.

É importante ressaltar que, nos testes realizados, os servidores de jogo não podem ser servidores de segurança. Isso, obviamente, implica perda de desempenho nos testes realizados, uma vez que servidores de jogo já possuem o estado do jogador, não sendo necessário uma mensagem adicional para outro servidor de segurança.

Para cada teste são realizadas 30 repetições, sendo os resultados demonstados equivalentes à média dos resultados obtidos. Limitações foram encontradas em alguns testes realizados, em especial variando-se o número de jogadores e número de células. Essas limitações devem-se à restrição do Windows 7 (32 bits) em relação à quantidade de memória que uma única aplicação pode fazer uso (2 GB) (MICROSOFT, 2011), sendo necessário uma quantidade maior de memória para a execução de alguns dos testes.

Entretanto, argumenta-se que estas restrições não afetam as conclusões obtidas, uma vez que o comportamento geral do sistema é perceptível com os testes realizados.

5.2 Detecção de cheating

A presente seção demonstra os resultados obtidos na avaliação do mecanismo de detecção de cheating proposto na Seção 4.2.5. Os testes consideram a variação do número total de jogadores e da variação da porcentagem de jogadores cheaters para os dois mecanismos propostos, tanto para estado previsível quanto para estado imprevisível.

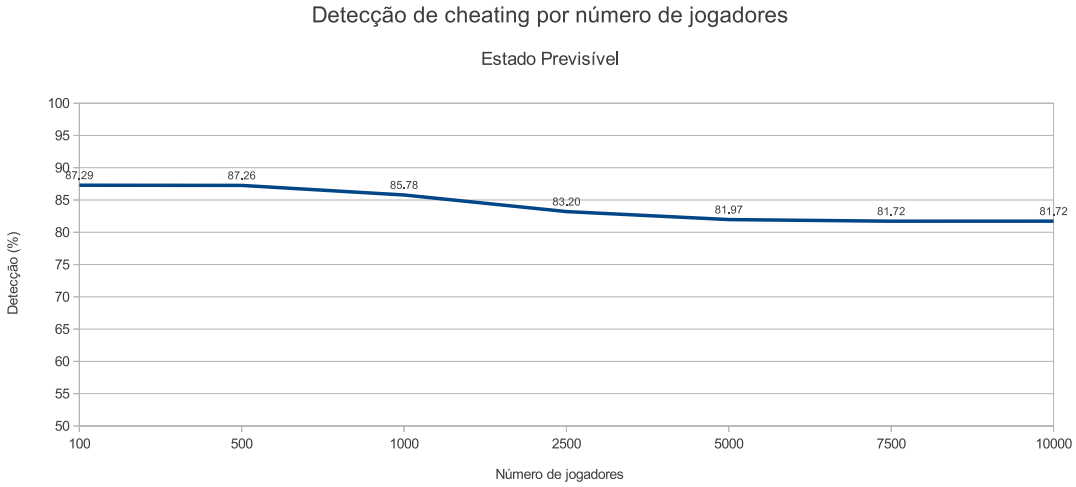

Figura 5.1: *Detecção de cheating por número de jogadores para estado previsível.*

5.2.1 Variando número de jogadores

As Figuras 5.1 e 5.2 demonstram a porcentagem de detecção do mecanismo para estado previsível e imprevisível, respectivamente, variando-se o número de jogadores. Todos os jogadores presentes são considerados cheaters.

Como pode ser percebido na Figura 5.1, há um índice de detecção de aproximadamente 87% para 100 jogadores (desvio padrão de 3.7%), com um declínio na eficácia do mecanismo ao aumentar-se o número de jogadores para 10000, ainda atingindo valores superiores a 81% de detecção (desvio padrão de 0.61%).

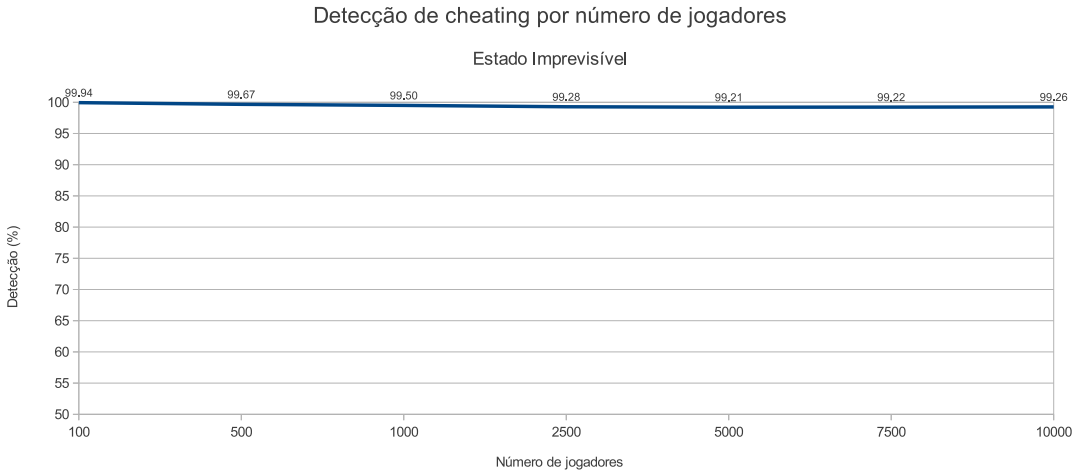

Figura 5.2: *Detecção de cheating por número de jogadores para estado imprevisível.*

A Figura 5.2 demonstra que o mecanismo para detecção para estado imprevisível possui uma eficácia superior a 99% independentemente do número de jogadores (desvio padrão de 3.1% para 100 jogadores, decrescendo até 0.39% para 10000 jogadores). Entretanto, conforme a descrição do mecanismo, esta solução pode apresentar falsos positivos. A análise de falsos positivos obtidos pelo mecanismo de detecção é apresentado na Figura 5.3.

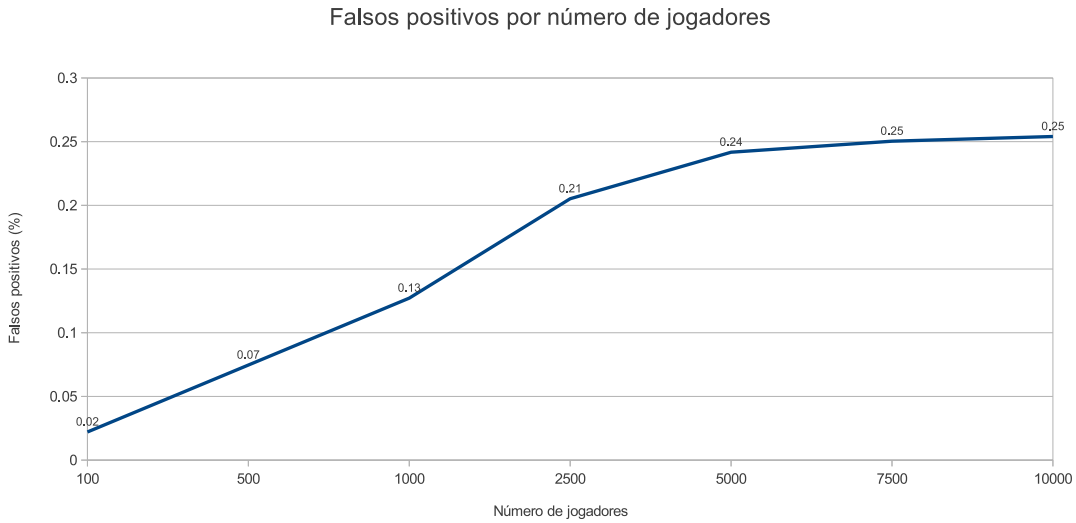

Figura 5.3: *Número de falsos positivos por número de jogadores.*

Como pode ser percebido, a quantidade de falsos positivos alcança um máximo de 0.25% dos cheatings detectados, atingindo valores considerados como aceitáveis.

5.2.2 Variando porcentagem de jogadores cheaters

Para avaliar o impacto que a quantidade de jogadores cheaters presentes no mundo virtual na solução apresentada foram realizados testes variando a porcentagem de jogadores que possuem, em algum momento de sua existência, um comportamento cheater.

As Figuras 5.4 e 5.5 demonstram o efeito que a variação na porcentagem de jogadores cheaters para os mecanismos de detecção para estados previsíveis e imprevisíveis, respectivamente.

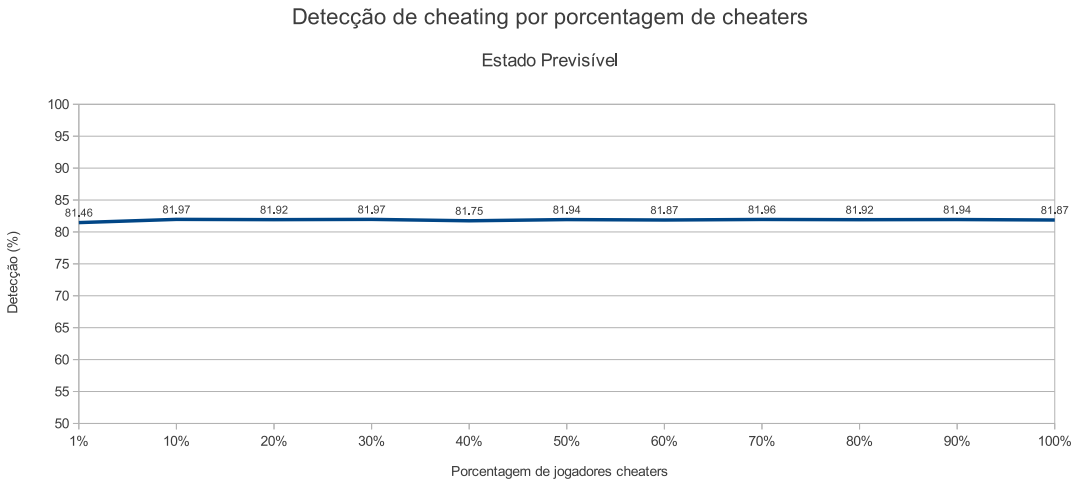

Figura 5.4: *Detecção de cheating por porcentagem de jogadores cheaters para estado previsível.*

Como os resultados expostos na Figura 5.4 demonstram, a variação na porcentagem de jogadores cheaters não impacta na eficácia do mecanismo de detecção para estados previsíveis. É atingida uma detecção superior a 81% para todos os casos (desvio padrão

de 7.21% para 1% de cheaters, decrescendo até 0.82% para 100% de cheaters), valor consistente com os resultados apresentados para 5000 jogadores (valor padrão) na Figura 5.1.

Resultado semelhante foi encontrado nos testes para estado imprevisível, conforme ilustrado na Figura 5.5.

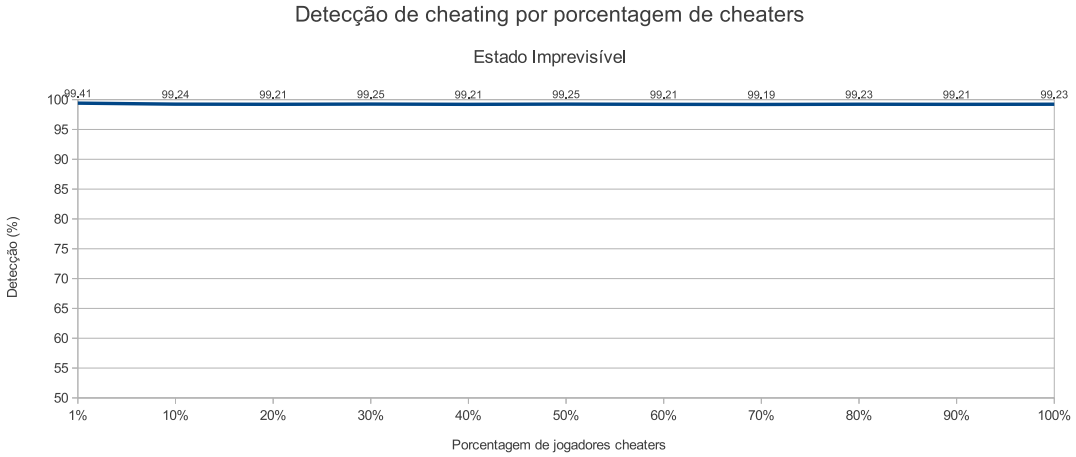

Figura 5.5: *Detecção de cheating por porcentagem de jogadores cheaters para estado imprevisível.*

Os testes indicam mais de 99% de detecção para todos os casos analisados (com desvio padrão descrescendo de 2.24% para 1% de cheaters até 0.61% para 100% de cheaters), mostrando pouca variação em relação a testes anteriores. Entretanto, há a ocorrência de uma anomalia no número de falsos positivos quando é feita a variação da porcentagem de jogadores cheaters, conforme mostra a Figura 5.6.

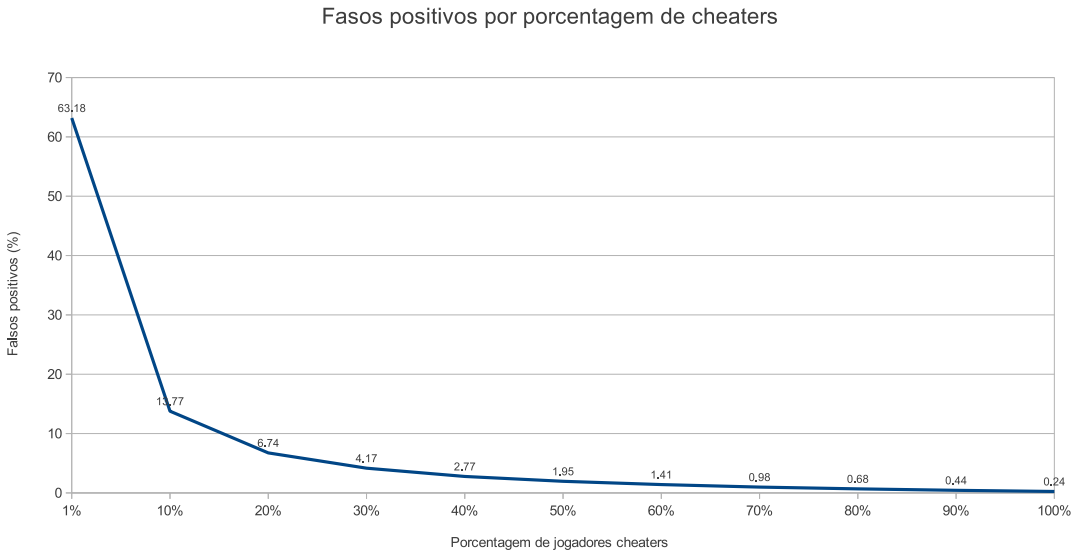

Figura 5.6: *Detecção de cheating por porcentagem de jogadores cheaters.*

Os testes indicam uma ocorrência de falsos positivos acima do esperado quando há uma pequena presença de cheaters (menos de 10%), atingindo 63% de falsos positivos

com 1% de cheaters, com um rápido declínio para 13% de falsos positivos com 10% de jogadores cheaters. Isso ocorre devido ao baixo número de cheaters no mundo virtual. Desta forma, a probabilidade da ocorrência de um jogador cujo desempenho seja acima do normal e não seja cheater (falso positivo) é mais alta quando existem poucos cheaters no jogo, caso contrário, ou seja, aumenta-se o número de cheaters no jogo, aumenta-se a probabilidade de que o jogador com desempenho acima do normal seja cheater. Porém, o modelo apresenta uma melhora em seu desempenho a medida que aumenta-se o número de jogadores cheaters.

5.3 Desempenho do mecanismo

Essa seção realiza uma análise relativa ao desempenho do mecanismo proposto. A métrica utilizada é o número de mensagens trocadas pelo sistema, onde esta mensagem pode assumir tamanhos e valores variáveis, bem como ser enviada com uma frequência diferente, variando para diferentes jogos.

Os testes apresentam variações no número de células (tamanho das células), variação do número de jogadores, variação do número de servidores de segurança utilizados e, por fim, análise do impacto que o mecanismo proposto por este trabalho terá com maiores tempos de execução do mundo virtual.

5.3.1 Variando o espaço virtual de detecção

Ao ser descrito o modelo de detecção proposto por este trabalho também discute-se o efeito que o tamanho do espaço virtual, no qual o cheating deve ser restrito, possui no mecanismo proposto (Seção 4.1.3.1). Os testes seguintes buscam mostrar o impacto que esta variação possui no desempenho do sistema.

Em um primeiro conjunto de testes foi constatado que, conforme esperado, a variação do número de células não possui impacto no número de mensagens de jogo trocadas. Esse fato auxilia a análise dos resultados seguintes.

As Figuras 5.7 e 5.8 mostram os resultados variando-se a quantidade de células presentes no mundo virtual. Um maior número de células implica em células de menor tamanho e, portanto, uma área de restrição menor e menos jogadores afetados pela ocorrência de um cheating. Um número menor de células implica em células de tamanho maior e, portanto, mais jogadores afetados.

A Figura 5.7 ilustra o impacto em número de mensagens trocadas, considerando também o número de jogadores.

As médias apresentadas possuem desvio padrão máximo de 4.57% para 100 jogadores e 16 células, decrescente conforme aumenta-se o número de células e o número de jogadores, atingindo 0.14% para 16384 células com 2500 jogadores). Como pode ser percebido, a quantidade de células aumenta exponencialmente, diminuindo-se pela metade o espaço de cada célula a cada aumento de grandeza. Entretanto, a quantidade de mensagens trocadas aumenta num ritmo muito inferior a esta escala. Portanto, pode-se afirmar que o desempenho do mecanismo é eficiente ao diminuir-se a área de restrição de cheating, mesmo considerando valores como 10000 jogadores.

Em seguida realiza-se uma análise de qual o impacto do aumento da restrição em relação ao número de mensagens normalmente trocadas (*overhead*). A Figura 5.8 ilustra os resultados obtidos.

São encontrados desvios padrão entre 0.40 e 0.57%. Os resultados ilustrados mostram que o *overhead* segue a escala exponencial apenas para pequenos números de jogadores,

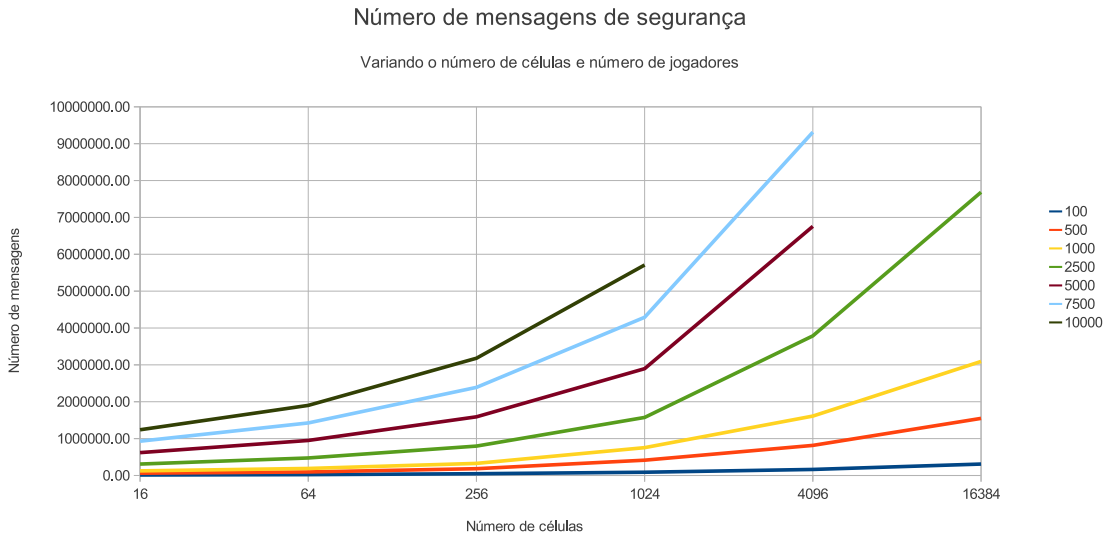

Figura 5.7: *Número de mensagens de segurança por número de células e número de jogadores.*

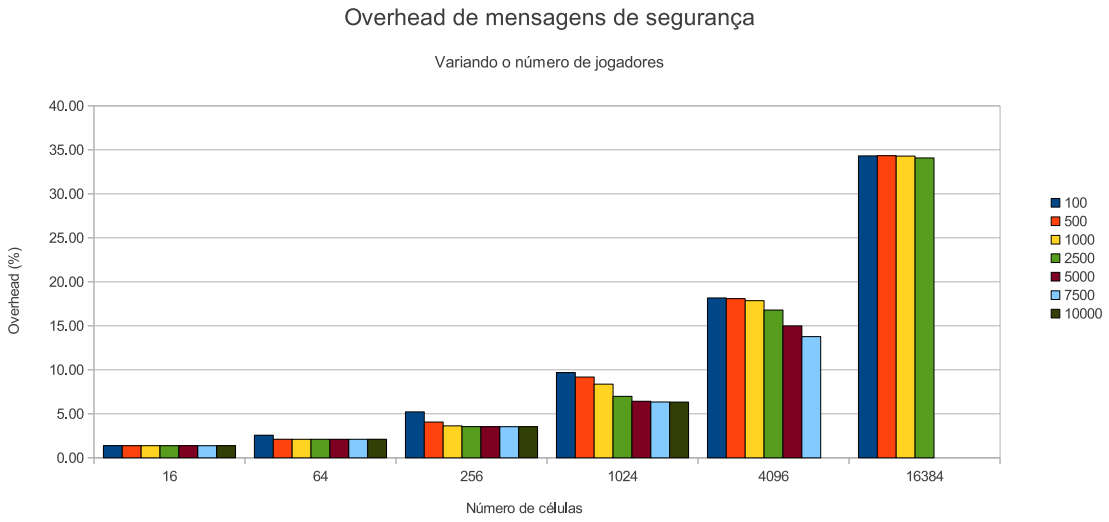

Figura 5.8: *Overhead de segurança por número de células e número de jogadores.*

enquanto seu desempenho demonstra-se mais eficáz ao aumentar-se o número de jogadores. Também é possível notar-se que atinge-se um valor inferior a 15% de *overhead* (0.4% de overhead) considerando 5000 jogadores e 4096 células (células de 10x10 pixels, lembrando que o tamanho da representação de um jogador é de 2x2 pixels).

Outro comportamento notável é que, à medida que o número de jogadores aumenta, o *overhead* do mecanismo de segurança diminui. Isso ocorre devido ao tempo gasto por cada jogador dentro de uma célula. Normalmente um jogador passa mais tempo dentro de uma célula, interagindo com jogadores com o ambiente, do que fazendo movimentos de migração de célula e, portanto, o aumento do número de mensagens de segurança tende a ser menor do que o aumento do número de mensagens de jogo.

5.3.2 Variando o número de servidores de segurança

Os testes seguintes buscam analisar o impacto que o uso de mais servidores possui no desempenho do sistema. Ou seja, o aumento do número de servidores de segurança e, portanto, a diminuição da probabilidade de conlúio entre os participantes, conforme descrito na Seção 4.1.5. As Figuras 5.9 e 5.10 exibem os resultados obtidos.

A Figura 5.10 ilustra o número de mensagens de segurança trocadas com o aumento do número de servidores, considerando o valor mínimo de 32 servidores, sendo 16 servidores de jogo e 16 servidores de segurança (1 servidor de jogo e 1 servidor de segurança para cada região), e o valor máximo de 112 servidores, sendo 16 servidores de jogo e 96 servidores de segurança (1 servidor de jogo e 6 servidores de segurança para cada região).

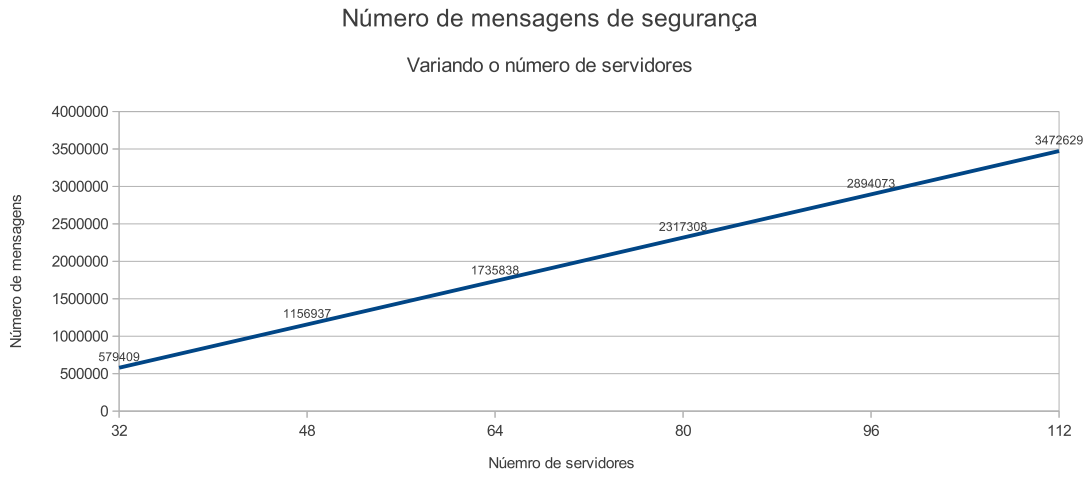

Figura 5.9: *Número de mensagens de segurança por número de servidores.*

Os resultados apresentam desvio padrão variando entre 0.44 e 0.64%.

A Figura 5.10 segue a mesma evolução de número de servidores utilizado pela Figura 5.9, entretando faz uma análise do *overhead* causado pelo aumento do número de servidores utilizados.

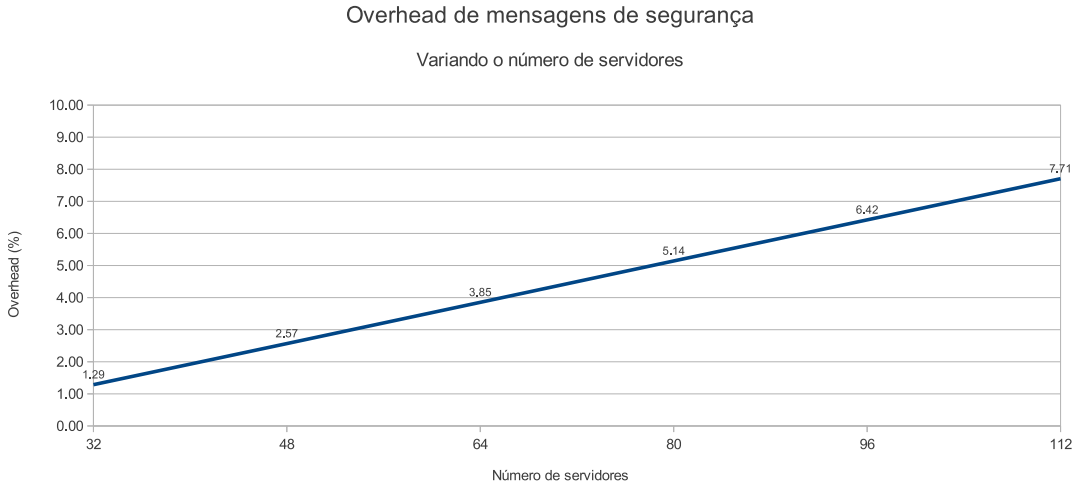

Figura 5.10: *Overhead de segurança por número de servidores.*

Como pode ser notado em ambos testes, ao aumentar-se o número de servidores obtém-se um comportamento linear, onde tanto o número de mensagens trocadas quanto o *overhead* obtidos aumentam proporcionalmente ao número de servidores utilizado. Cabe ressaltar, entretanto, que o *overhead* causado pelo uso do mecanismo, considerando o uso de 96 servidores (1 servidor de jogo e 5 servidores de segurança). Considera-se que a utilização de 5 servidores de segurança por região seja superior ao necessário para evitar-se conlúio entre servidores, seguindo os conhecidos protocolos para tratamento de falhas bizantinas, que prevém o uso de $2f + 1$ réplicas para f falhas.

5.3.3 Variando o tempo de jogo simulado

Para analisar se o tempo de utilização do mecanismo possui algum impacto no desempenho do sistema foram feitos testes variando-se o tempo de jogo simulado. Os resultados destes testes são exibidos na forma de mensagens trocadas e *overhead*, e são ilustrados nas Figuras 5.11 e 5.12.

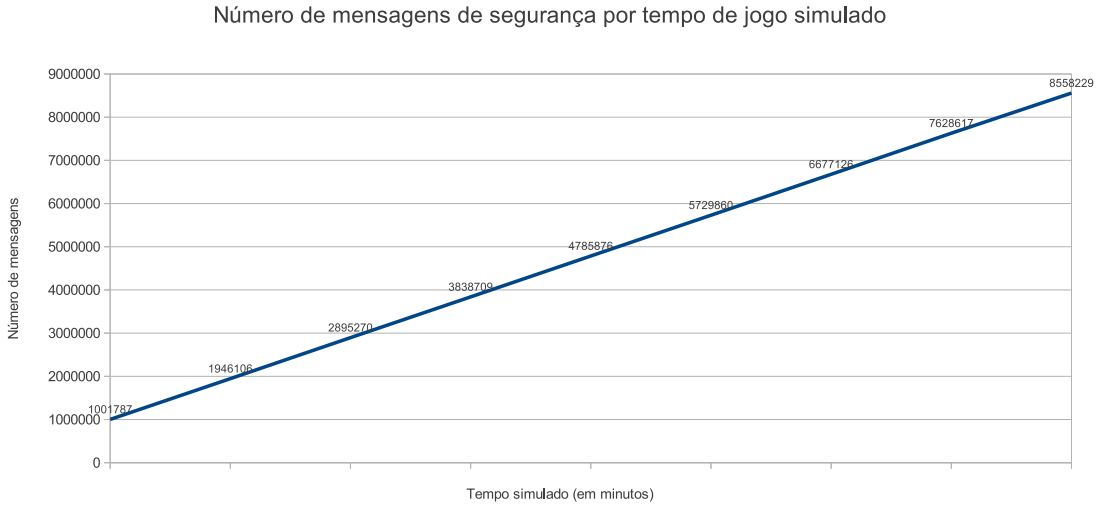

Figura 5.11: *Número de mensagens de segurança por tempo de jogo simulado.*

Como pode ser percebido na Figura 5.11 o número de mensagens aumenta linear e proporcionalmente ao tempo de jogo simulado. Os resultados apresentam desvio padrão variando entre 0.41 e 0.53%.

A Figura 5.12 ilustra o *overhead* causado pelo mecanismo de segurança. Os resultados mostram que o mecanismo não é afetado pelo tempo de jogo, possuindo um custo constante independentemente do tempo decorrido.

5.4 Comportamento dos cheaters

Para finalizar os testes e análise dos resultados obtidos, a presente seção busca analisar o impacto que variações no comportamento dos jogadores cheaters possam causar no mecanismo de detecção e, portanto, na solução como um todo. Para tanto dois principais aspectos do comportamento de um jogador cheater foram analisados, conforme descrito na Seção 4.2.4: o valor adicionado ao cometer-se um cheating e a vontade de um dado jogador cometer um cheating. A análise do impacto destes fatores é feita através da detecção de cheating apresentada pelo sistema.

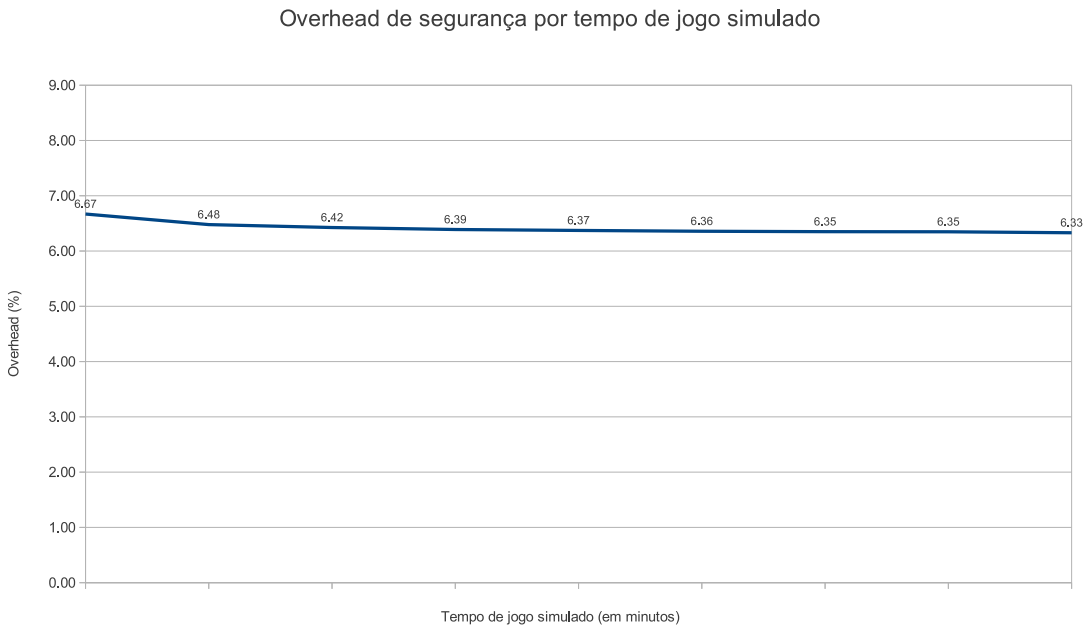

Figura 5.12: *Número de mensagens de segurança por tempo de jogo simulado.*

A Figura 5.13 exibe a detecção de cheating variando-se o valor adicionado e a variação na porcentagem das ações que o jogador cheater tentará cometer um cheating para o estado previsível.

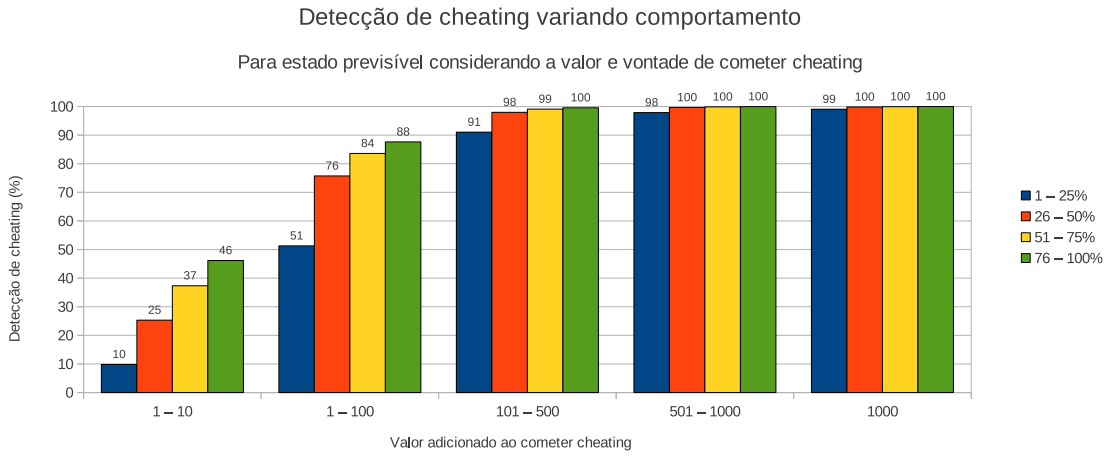

Figura 5.13: *Detecção de cheating para estado previsível considerando variação no comportamento dos jogadores.*

Como pode ser notado, a detecção de cheating para altos valores pode ser considerada ótima, atingindo valores superiores a 98% na maioria dos casos. Entretanto, ao considerar-se valores menores o mecanismo possui baixos índices de detecção, diminuindo conforme diminui-se a frequência das ações em que o cheating é cometido.

A Figura 5.14 ilustra a detecção de cheating considerando a variação no valor adicionado por cheating e a porcentagem das ações que um jogador cheater tentará cometer cheating. Os resultados são referentes à técnica de detecção para estado imprevisível.

De forma similar aos resultados da técnica para estado previsível, os resultados para

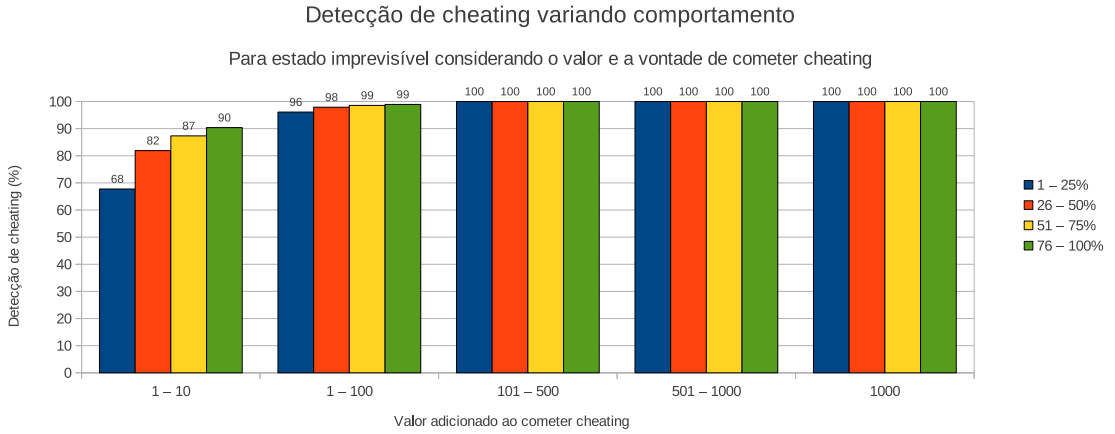

Figura 5.14: *Detecção de cheating para estado imprevisível considerando variação no comportamento dos jogadores.*

detecção de cheating variando-se o comportamento dos jogadores cheaters mostra altos níveis de detecção à medida que aumenta-se o valor adicionado em um cheat, bem como aumenta-se a detecção ao aumentar-se a frequência com que os cheaters cometem cheating.

Em relação aos falsos positivos obtidos para a técnica de detecção para estado imprevisível, foi obtido um índice de 0.34% de falsos positivos para o pior caso (1-25% de vontade, adicionando 1-10 unidades de ouro por cheating), e valores muito próximos a 0% (obtendo-se valores siginificativos apenas ao aumentar-se o número de casas decimais) para todos os casos considerando 26% ou mais de vontade de cometer cheating.

A seção seguinte discute algumas conclusões obtidas a partir dos resultados apresentados nesta seção e nas seções anteriores, descrevendo que, apesar do nível de detecção ser considerado baixo para alguns casos, a proposta de proteção da economia virtual ainda é mantida.

5.5 Considerações sobre os resultados

Os testes demonstram que as diferentes técnicas para detecção de cheating podem ser utilizadas com eficácia em um jogo com características gerais de MMOGs. Foi possível obter-se um índice de detecção superior a 81% para a técnica de detecção para estado previsível e indíce superior a 99% para a técnica de detecção para estado imprevisível, com um índice de falsos positivos inferior a 0.25% para qualquer número de jogadores, e inferior a 2% para mais de 50% de jogadores cheaters presentes no mundo virtual.

Variando-se o tamanho do espaço virtual em que o cheating será contido é possível perceber que o custo adicional causado pelo mecanismo de segurança é inferior a 10% para um espaço de 20x20 pixels. Levando em consideração o tamanho de um jogador (2x2) pixels, e as regras de jogo, considera-se que este espaço seja restrito o suficiente para um MMOG e que a solução apresenta baixo custo.

O número de servidores possui um impacto considerável no mecanismo, entretanto, utilizando 5 servidores, valor considerado suficiente para proteção contra conlúio, obtém-se um custo adicional inferior a 6.5%. Também é possível comprovar que o tempo de jogo não impacta no desempenho do mecanismo, possuindo *overhead* constante.

Por fim, nota-se que o comportamento dos jogadores cheaters afeta de forma sigini-

ficativa a acuracidade do sistema, porém as detecções superam a marca de 90% caso os jogadores cheaters adicionem valores superiores a 100 unidades de ouro por cheating.

Como consideração geral ressalta-se o objetivo do presente trabalho como sendo proteção da economia global contra jogadores maliciosos. Desta forma é possível perceber que o trabalho atinge esta meta em todos os casos analisados, mesmo em casos onde o índice de detecção de cheating é considerado abaixo do esperado. A justificativa dá-se no valor do cheating não detectado. Uma vez que o sistema não detecta cheating apenas em casos onde o cheater busca burlar o sistema atráves da adição de pequenos valores ou com uma frequencia muito baixa, argumenta-se que esta ocorrência é considerada incapaz de afetar a economia e, muitas vezes, pode ser mais custoso (em questão de produção de recursos por tempo de jogo) do que o custo para jogar honestamente.

5.6 Comparativo com trabalhos relacionados

Do conhecimento dos autores, não há um trabalho que possua como temática a proteção da economia virtual de MMOGs utilizando detecção de cheating. Desta forma, considerando este o principal objetivo desse trabalho, torna-se complexa a realização de um comparativo direto com outros trabalhos. Entretanto, ao ser considerar-se a metodologia empregada para proteção da economia, i.e. detecção de cheating, encontra-se uma métrica passível de comparativo com outros trabalhos, embora não em sua totalidade, devido à complexidade do trabalho aqui apresentado.

Considerando a detecção de cheating como aspecto a ser avaliado, pode-se perceber que os resultados aqui apresentados equiparam-se àqueles apresentados no Capítulo 3. Soluções como DaCaP (90% de detecção) e DACA (76% de detecção) estabelecem uma nova arquitetura que permite a detecção de diversas formas de cheating. Outros trabalhos, como o CCP possuem 93% de detecção, entretanto buscam tratar tipos específicos de cheating (*time cheating*).

Entre as soluções abrangentes e, portanto, mais interessantes para este trabalho, estão a solução de detecção automatizada, apresentada por Laurens et al. (LAURENS et al., 2007), e a solução através da comparação de eventos, apresentada por Izaiku et al. (IZAIKU et al., 2006). O primeiro apresenta uma solução baseada em análise do comportamento para um jogo específico utilizando uma arquitetura cliente-servidor, obtendo aproximadamente 70% de detecção de cheating.

A segunda solução (IZAIKU et al., 2006) aproxima-se mais da apresentada no corrente trabalho. Porém, há diferenças no objetivo e metodologia empregadas. A utilização de uma função de *hash* para detecção pode não possuir as variações necessárias para a detecção de estados imprevisíveis, conforme descrito no presente trabalho.

Outro fator determinante é a possibilidade de propagação do efeito do cheating antes deste ser detectado. Como a solução de Izaiku et al. utiliza o fator tempo para iniciar uma nova tentativa de detecção, é possível que o efeito do cheating já tenha sido propagado, devendo ser realizada uma detecção e, possivelmente *rollback*, em diversos jogadores afetados. Desta forma a solução aqui apresentada, utilizando espaço virtual para detecção de cheating, pode se tornar mais eficiente ao ser considerado o objetivo de proteção de outros jogadores contra o efeito do cheating.

Em relação ao desempenho apresentado pelo trabalho em questão, a limitação de 100 jogadores ao máximo por servidor permite que sejam atingidos limites máximos de 256 Kbps de *download link* (utilizando turnos de 200 ms). Considerando que o custo de mensagens de jogo é de 256 Kbps, pode-se considerar que o *overhead* adicional causado

pelo sistema é de, pelo menos, 100%, considerando o uso de apenas um servidor de segurança, enquanto a solução do presente trabalho possui um custo adicional inferior a 10% na maioria dos casos.

Por fim, pode-se concluir que o presente trabalho cumpre sua função de proteção da economia virtual através de um mecanismo de detecção que possui uma eficácia compatível com outros trabalhos similares. Também conclui-se que o desempenho apresentado pelo mecanismo atinge custos adicionais satisfatórios quando em comparação com outros trabalhos.

6 CONCLUSÃO

Com a evolução do Jogos Online Massivamente Multijogadores (MMOGs) há uma crescente preocupação em relação à infraestrutura de suporte e à quantidade de jogadores online. Seguindo a tendência histórica, deve-se experimentar nos próximos anos um aumento em relação ao número de jogadores e em relação ao número de jogadores online simultaneamente.

Outros modelos de sistemas distribuídos originaram-se de forma centralizada, arquitetura atualmente empregada na maior parte dos MMOGs comerciais. De forma similar a diversos outros sistemas, há uma busca por soluções distribuídas para suporte a MMOGs, de forma a resolver atuais limitações empregadas por essa arquitetura como: gargalo na comunicação, ponto único de falha e limitações de escalabilidade.

As arquiteturas P2P buscam resolver algumas dessas limitações, trazendo escalabilidade para o sistema. Entretanto, também apresentam limitações de segurança devido à distribuição do estado do jogo. A exploração dessas limitações de segurança, ou cheating, pode afetar todos os jogadores do mundo virtual, gerando insatisfação, desistência e dificuldades técnicas para correção da exploração.

O presente trabalho tem como foco principal a proteção da economia virtual, ou seja, evitar que a ocorrência de um cheating atinja a maior parcela dos jogadores. Para alcançar este objetivo usa-se detecção da ocorrência de cheating aliada a uma divisão celular do mundo virtual. Através da divisão do mundo virtual em células, e realizando a detecção da ocorrência de um cheating em uma dada célula, é possível evitar que o efeito deste cheating se espalhe para outras células e, portanto, limitando o número de jogadores afetados àqueles presentes nas células em que o cheating ocorreu.

O mecanismo de detecção de cheating consiste em realizar o cálculo da estimativa do estado do jogador ao deixar a célula e comparar com o estado ao ingressar na célula. Caso a evolução do estado do personagem na célula esteja acima de um dado valor, considera-se o jogador como possível cheater.

Para a criação de um mecanismo de estimativa de estado utiliza-se uma classificação de estado possuindo duas categorias: estado previsível e imprevisível. O primeiro estado consiste dos elementos que variam conforme regras pré-estabelecidas e que podem ser obtidas no banco de dados ou nas próprias regras do jogo. A segunda categoria é composta por elementos que são, normalmente, afetados por comportamento humano como o número de mortes de um dado personagem, por exemplo.

A solução abstrata é implementada em um jogo modelo, que possui características de mobilidade e interação entre jogadores, além de interação com o próprio ambiente virtual. Estas características sumarizam as principais características encontradas em MMOGs comerciais.

Para validação da classificação de estado, utilizou-se duas fórmulas de detecção, uma

para cada categoria da classificação proposta. A primeira (estado previsível) realiza um cálculo para estimativa de evolução baseado nas características da célula em questão, obtidas a partir do próprio jogo. A segunda fórmula realiza uma análise estatística do comportamento dos jogadores e considera níveis de aceitação baseando-se no desvio padrão deste comportamento. Ambas as fórmulas estabelecem limites máximos que, caso superados, indicam a provável presença de cheating. No caso da segunda fórmula, especificamente, é possível a presença de falsos positivos, ou seja, a declaração de suspeita de cheating para jogadores honestos.

Os testes de validação e análise utilizaram até 10000 jogadores e 112 servidores. Os resultados comprovam a eficiência da solução proposta, obtendo níveis de detecção de cheating aceitáveis com custos relativamente baixos, e cumprindo seu objetivo de proteção à economia virtual. A proteção à economia é comprovado pelo fato de a detecção a cheating com altos valores (ou frequência muito alta) ser realizada em 100% das vezes. A solução possui como ponto falho a detecção de cheating quando o valor obtido é muito pequeno. Entretando, um cheating com valor baixo não deve causar um impacto significativo na economia virtual, além de possuir, em muitos casos, um custo mais alto para execução do cheating do que para jogar honestamente.

Apesar da realização dos experimentos utilizar um jogo genérico, a aplicação da solução proposta em um jogo comercial não deve possuir barreiras significativas. Há a necessidade da classificação do estado, elaboração de técnicas para detecção específica para cada um destes tipos de estado (ou adaptação das técnicas propostas neste trabalho) e implementação de um mecanismo de punição, que deve variar conforme a política da empresa de desenvolvimento.

Modelos iniciais e resultados preliminares desse trabalho foram publicados em eventos regionais para análise e obtenção de contribuições. Exemplos de publicações iniciais são ERAD 2010 e WSPPD 2011. Entretanto, a principal publicação dos resultados obtidos foi realizada no periódico *International Transactions on System Science and Applications* (ITSSA) (SEVERINO; BEZERRA; GEYER, 2011).

Apesar de atender aos objetivos estabelecidos, alguns aprimoramentos e melhorias devem ser considerados para melhor utilização da proposta apresentada. Alterações podem ser realizadas nas fórmulas para estimativa de estado, procurando obter melhores índices de detecção.

Também é possível estabelecer limites mais restritos para o método de detecção para estados imprevisíveis, objetivando diminuir o número de falsos positivos, mesmo significando um decréscimo no índice de detecção. Ou seja, diminui-se a porcentagem de detecção de cheating, porém também reduz-se o número de falsos positivos, gerando menos insatisfação.

O modelo pode ser aprimorado para evitar que um jogador não seja detectado, fato que pode ocorrer caso um jogador não deixe uma dada célula após cometer cheating. Isso pode ser evitado através da utilização de um limite de tempo onde, caso o jogador não tenha deixado a célula após este limite, ativa-se o mecanismo de detecção.

REFERÊNCIAS

ALEGRETTI, F. J. P. **ChicuxBot - Genetic Algorithm Configured Behavior Network Multi-Agent for Quake II**. 2006. Dissertação (Mestrado em Ciência da Computação) — Universidade Federal do Rio Grande do Sul, Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, BR.

AOL. **Neverwinter Nights**. Disponível em: <http://en.wikipedia.org/wiki/Neverwinter_Nights_(AOL_game)/>. Acesso em: 05 jul. 2010.

ARENANET. **Guild Wars**. Disponível em: <http://www.guildwars.com/>. Acesso em: 02 abr. 2010.

ATARI. **Star Trek Online**. Disponível em: <http://www.startrekonline.com/>. Acesso em: 10 abr. 2011.

BAUGHMAN, N. E.; LIBERATORE, M.; LEVINE, B. N. Cheat-proof playout for centralized and peer-to-peer gaming. **IEEE/ACM Trans. Netw.**, Piscataway, NJ, USA, v.15, n.1, p.1–13, 2007.

BEZERRA, C. E. B. **Lidando com Recursos Escassos e Heterogêneos em um Sistema Distribuído Atuando como Servidor de MMOG**. 2009. Dissertação (Mestrado em Ciência da Computação) — Universidade Federal do Rio Grande do Sul, Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, BR.

BLIZZARD. **World of Warcraft**. Disponível em: <http://www.worldofwarcraft.com/>. Acesso em: 02 abr. 2010.

CCP. **EVE Online**. Disponível em: <http://www.eve-online.com/>. Acesso em: 02 abr. 2010.

CECIN, F. **Peer-to-peer and cheat-resistant support for massively multiplayer online games**. 2009. PhD — Universidade Federal do Rio Grande do Sul, Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, BR.

CECIN, F. R. et al. FreeMMG: a hybrid peer-to-peer and client-server model for massively multiplayer games. In: ACM SIGCOMM WORKSHOP ON NETWORK AND SYSTEM SUPPORT FOR GAMES, 3., New York, NY, USA. **Proceedings...** ACM, 2004. p.172–172. (NetGames '04).

CHEN, B. D.; MAHESWARAN, M. A cheat controlled protocol for centralized online multiplayer games. In: NETGAMES '04: PROCEEDINGS OF 3RD ACM SIGCOMM

WORKSHOP ON NETWORK AND SYSTEM SUPPORT FOR GAMES, New York, NY, USA. **Anais...** ACM, 2004. p.139–143.

DARLAGIANNIS, V.; HECKMANN, O.; STEINMETZ, R. Peer-to-peer applications beyond file sharing: overlay network requirements and solutions. **e & i Elektrotechnik und Informationstechnik**, [S.l.], v.123, p.242–250, 2006. 10.1007/s00502-006-0345-z.

FAIRFIELD, J. Virtual Property. **Boston University Law Review**, [S.l.], v.85, p.1047–1102, 2005.

FERRETTI, S.; ROCCETTI, M. AC/DC: an algorithm for cheating detection by cheating. In: NOSSDAV '06: PROCEEDINGS OF THE 2006 INTERNATIONAL WORKSHOP ON NETWORK AND OPERATING SYSTEMS SUPPORT FOR DIGITAL AUDIO AND VIDEO, New York, NY, USA. **Anais...** ACM, 2006. p.1–6.

FUJIMOTO, R. M. **Parallel and Distribution Simulation Systems**. 1st.ed. New York, NY, USA: John Wiley & Sons, Inc., 1999.

GASPARETO, O. B. **Redes Neurais Artificiais Aplicadas ao Reconhecimento de Speed Cheating em Jogos Online de Computador**. 2008. Dissertação (Mestrado em Ciência da Computação) — Universidade Federal do Rio Grande do Sul, Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, BR.

GAUTHIERDICKEY, C. et al. Low latency and cheat-proof event ordering for peer-to-peer games. In: NOSSDAV '04: PROCEEDINGS OF THE 14TH INTERNATIONAL WORKSHOP ON NETWORK AND OPERATING SYSTEMS SUPPORT FOR DIGITAL AUDIO AND VIDEO, New York, NY, USA. **Anais...** ACM, 2004. p.134–139.

IZAIKU, T. et al. Cheat detection for MMORPG on P2P environments. In: ACM SIGCOMM WORKSHOP ON NETWORK AND SYSTEM SUPPORT FOR GAMES, 5., New York, NY, USA. **Proceedings...** ACM, 2006. (NetGames '06).

JHA, S.; KATZENBEISSER, S.; VEITH, H. Enforcing Semantic Integrity on Untrusted Clients in Networked Virtual Environments. In: SP '07: PROCEEDINGS OF THE 2007 IEEE SYMPOSIUM ON SECURITY AND PRIVACY, Washington, DC, USA. **Anais...** IEEE Computer Society, 2007. p.179–186.

LAURENS, P. et al. A Novel Approach to the Detection of Cheating in Multiplayer Online Games. In: ICECCS '07: PROCEEDINGS OF THE 12TH IEEE INTERNATIONAL CONFERENCE ON ENGINEERING COMPLEX COMPUTER SYSTEMS, Washington, DC, USA. **Anais...** IEEE Computer Society, 2007. p.97–106.

LIU, H.-I.; LO, Y.-T. DaCAP - A Distributed Anti-Cheating Peer to Peer Architecture for Massive Multiplayer On-line Role Playing Game. In: CCGRID '08: PROCEEDINGS OF THE 2008 EIGHTH IEEE INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID, Washington, DC, USA. **Anais...** IEEE Computer Society, 2008. p.584–589.

LIU, H.-I.; TANG, B.-R. DACA: dynamic anti-cheating architecture for mmogs. In: AINA '09: PROCEEDINGS OF THE 2009 INTERNATIONAL CONFERENCE ON ADVANCED INFORMATION NETWORKING AND APPLICATIONS, Washington, DC, USA. **Anais...** IEEE Computer Society, 2009. p.892–897.

MICROSOFT. **Virtual Address Space**. Disponível em: <http://msdn.microsoft.com/en-us/library/aa366912(v=vs.85).aspx/>. Acesso em: 10 set. 2011.

Microsoft Games. **Age of Empires**. Disponível em: <http://www.microsoft.com/games/empires/>. Acesso em: 03 abr. 2010.

NCSOFT. **Lineage II**. Disponível em: <http://www.lineage2.com/>. Acesso em: 02 abr. 2010.

NCSOFT. **City of Heroes**. Disponível em: <http://cityofheroes.com/>. Acesso em: 05 jul. 2010.

O2 Online Entertainment. **Mankind**. Disponível em: <http://www.mankind.net/>. Acesso em: 10 abr. 2011.

SCHIELE, G. et al. Requirements of Peer-to-Peer-based Massively Multiplayer Online Gaming. In: CLUSTER COMPUTING AND THE GRID, 2007. CCGRID 2007. SEVENTH IEEE INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2007. p.773 –782.

SEVERINO, F. L.; BEZERRA, C. E.; GEYER, C. F. Protecting the Virtual Economy in MMOGs Using a Cheating Detection Mechanism. **International Transactions on Systems Science and Applications**, [S.l.], v.7, n.3/4, p.179–191, 2011.

Sony Entertainment. **PlanetSide**. Disponível em: <http://www.planetside.com/>. Acesso em: 02 abr. 2010.

University of California. **BOINC**. Disponível em: <http://boinc.berkeley.edu/>. Acesso em: 5 set. 2010.

VALVE. **Valve Anti-Cheat System**. Disponível em: <https://support.steampowered.com/kb_article.php?ref=7849-Radz-6869/>. Acesso em: 12 ago. 2009.

VILANOVA, F. J. et al. P2PSE - A peer-to-peer support for multiplayer games. In: VII BRAZILIAN SYMPOSIUM ON COMPUTER GAMES AND DIGITAL ENTERTAINMENT - COMPUTING TRACK. **Anais...** [S.l.: s.n.], 2008.

WEBB, S. D.; SOH, S. Cheating in networked computer games: a review. In: DIMEA '07: PROCEEDINGS OF THE 2ND INTERNATIONAL CONFERENCE ON DIGITAL INTERACTIVE MEDIA IN ENTERTAINMENT AND ARTS, New York, NY, USA. **Anais...** ACM, 2007. p.105–112.

WIKIPEDIA. **Massively Multiplayer Online Game**. Disponível em: <http://en.wikipedia.org/wiki/Massively_multiplayer_online_game/>. Acesso em: 05 jul. 2010.

WIKIPEDIA. **Air Warrior**. Disponível em: <http://en.wikipedia.org/wiki/Air_Warrior/>. Acesso em: 05 jul. 2010.

WIKIPEDIA. **Virtual Economy**. Disponível em: <http://en.wikipedia.org/wiki/Virtual_economy/>. Acesso em: 05 jul. 2010.

WOODCOCK, B. S. **MMOGChart**. Disponível em: <http://www.mmogchart.com/>. Acesso em: 05 abr. 2009.

YAN, J.; RANDELL, B. A systematic classification of cheating in online games. In: NETGAMES '05: PROCEEDINGS OF 4TH ACM SIGCOMM WORKSHOP ON NETWORK AND SYSTEM SUPPORT FOR GAMES, New York, NY, USA. **Anais...** ACM, 2005. p.1–9.