A System for Formation of Even Matches in Competitive Games

Rodrigo Nietiedt de Almeida* Daniel Camozzato† Rossana Baptista Queiroz‡

Universidade do Vale do Rio dos Sinos, Escola Politécnica, Brasil

RESUMO

Este artigo descreve um sistema projetado para criar partidas balanceadas em jogos digitais competitivos. Ele também fornece conhecimento fundamental para entender problema e a tecnologia envolvida na solução. O sistema implementa o algoritmo de Elo para avaliar as habilidades dos jogadores, e baseando-se nessa avaliação, procura encontrar partidas com jogadores de habilidades similares. O sistema é validado através de uma simulação e de um estudo de caso. Os resultados obtidos através da simulação mostram uma melhora no balanceamento das partidas e uma maior precisão na avaliação dos jogadores, validando as funcionalidades básicas propostas. A implementação do sistema em um caso real identificou as dificuldades relacionados com o processo de integração com o sistema de formação de partidas.

Palavras-chave: Jogos, Matchmaking, Multijogador, Balanceamento.

1 INTRODUÇÃO

Um dos principais objetivos de um design de jogo é criar uma experiência significativa, e de acordo com Katie Salen, a competitividade em um jogo pode enriquecer a experiência do jogador [6]. Diversos jogos digitais usam a competitividade para atrair jogadores, mas sofrem com o balanceamento de dificuldade do jogo, pois em jogos multijogadores com foco competitivo, ou *e-sports* [1], a dificuldade do jogo depende da habilidade dos jogadores que se enfrentam. Se o jogo é fácil ou difícil demais, as ações do jogador perdem significado e portanto não estimulam o jogador [2]. Para balancear o nível de desafio nas partidas, jogos desse tipo usam mecanismos de busca automatizada de partidas equilibradas baseada na habilidade dos jogadores.[3] [4]

Os fatores usados na formação de uma partida equilibrada variam muito de jogo para jogo, mas jogos competitivos possuem algumas necessidades em comum. O armazenamento de dados estatísticos, o gerenciamento de fila e a alocação de jogadores em salas são funcionalidades desenvolvidas em diversos casos de *matchmaking*. A necessidade de desenvolvimento dessas ferramentas torna a implementação do processo e inviável para projetos que possuem baixo investimento.

*e-mail: rodrigonietiedt@gmail.com

†e-mail: daniel.camozzato@gmail.com

‡e-mail: fellowsheep@gmail.com

[1] Eletronic-Sports

[2] Keith Burgun. Understanding Balance in Video Games. Dsiponível em http://www.gamasutra.com/view/feature/134768/understanding_balance_in_video_.php

[3] O processo de formação de partidas em jogos é chamado de matchmaking

[4] Uma partida é considerada equilibrada quando todos os jogadores envolvidos têm chance de vitória.

O projeto descrito nesse artigo tem como objetivo propor e implementar um sistema de matchmaking para o uso em motores genéricos e compreender as tecnologias envolvidas no processo.

2 REFERENCIAL TEÓRICO

Nesta seção são apresentados os conceitos principais para o entendimento da metodologia proposta.

2.1 Sistemas de Avaliação

A avaliação da habilidade de um jogador é um dos principais desafios no balanceamento de uma partida. Existem diversas técnicas para avaliar jogadores. As principais estão descritas nesta seção.

O Elo é um dos sistemas de ranqueamento mais usados, e foi popularizado por ter sido adotado pela Federação Internacional de Xadrez [3]. Nesse sistema, o jogador possui uma pontuação que representa a estimativa de sua habilidade. Essa pontuação é atualizada após cada ciclo[5] comparando a estimativa do resultado com o resultado real e ajustando a pontuação de acordo com essa diferença. Na implementação da Federação Mundial de Xadrez, a estimativa de resultado é feita de acordo com a diferença de habilidade entre dois competidores (A e B), e é calculada através da equação

$$E = \frac{1}{1+10^{-(Ra-Rb)/400}},$$

Sendo que Ra e Rb representam a pontuação dos jogadores A e B, e E representa a probabilidade de vitória do competidor A. A pontuação do jogador é atualizada após cada torneio com

$$Rpost = Rpre + K(S - Sexp),$$

aonde RPost é a pontuação do jogador após o ciclo, Rpre é a pontuação do jogador antes do ciclo, S é a pontuação real [6] e Sexp é a pontuação estimada para o jogador. O fator de atenuação K tem como objetivo definir o impacto do ajuste feito na pontuação do jogador. K costuma representar a importância da partida na avaliação do jogador, e na *USCF* é ajustado de acordo com o número de jogos do jogador: quanto maior o número de jogos, menor o K.

O sistema Elo também é usado em jogos como o GO [7] e o League of Legends [8], que usa o Elo em conjunto com um sistema de experiência para separar jogadores novos de jogadores mais experientes. Os sistemas de ranqueamento do Dota 2 [9] e da plataforma *XBox Live* [10] usam algoritmos semelhantes ao Elo, mas adicionam ao matchmaking um fator de certeza para aprimorar a precisão da avaliação do jogador com o passar do tempo.

O sistema Glicko-2 [4] [11] é usado como base nos sistemas de *matchmaking* do *Guild Wars 2* [12] e do *CounterStrike: Global Of-*

[5] O número de partidas em um ciclo varia de acordo com a implementação de cada sistema. Na implementação da *USCF*(*The United States Chess Federation*), a atualização é feita após cada torneio.

[6] O jogador pontua 0 para derrota, $\frac{1}{2}$ para empates e 1 para vitórias.

[7] http://gobase.org/studying/articles/elo/

[8] http://forums.na.leagueoflegends.com/board/showthread.php?t=12029

[9] http://dota2.gamepedia.com/Matchmaking_ratings

[10] http://research.microsoft.com/en-us/projects/trueskill/

[11] descrição atualizada e menos técnica disponível em http://www.glicko.net/glicko/glicko2.pdf

[12] https://www.guildwars2.com/en/news/finding-the-perfect-match/

fense [13]. Para o jogo *Ghost Recon*, o sistema descrito em [2] usa diversas estatísticas dos perfis de jogadores em uma rede neural para buscar partidas consideradas divertidas pelos jogadores envolvidos.

2.2 Desenvolvimento de Jogos Multijogador

Esta sessão tem como objetivo apresentar conceitos de desenvolvimento de jogos multijogadores que são necessários para o entendimento do problema apresentado e da solução proposta. Serão abordados os Motores de jogos e arquiteturas de rede em jogos multijogador.

2.2.1 Motor de jogos

Para oferecer maior produtividade, projetos de jogos mantêm separados o conteúdo artístico e o motor de jogos, que é a parte responsável pelo processo computacional do jogo. Essa divisão permite que o motor seja usado para mais de um jogo, assim como permite a reutilização do conteúdo em outros motores ou plataformas [5].

Uma dos motores mais populares atualmente é o Unity3D *Unity3D*[14]. Ele oferece um ambiente de desenvolvimento completo para diversas plataformas. Para jogos multijogadores, ele suporta implementação de comunicação em rede através de da UNet [15] e da Photon Network [16]. A *UNet* é o módulo de rede próprio da da *Unity3D*. A Photon Network é um motor de rede e uma plataforma de jogos multijogadores que oferece diversos serviços para jogos multijogadores. A *Photon PUN* é uma ferramenta oferecida pela *Photon Network* que permite o uso dos serviços da *Photon Network* na *Unity3D*. A *Photon PUN* oferece sistemas de lobby e busca de partidas aleatórias ou parametrizadas.

2.2.2 Arquitetura de Rede em Jogos Multijogador

A escolha da arquitetura de comunicação usada em um jogo digital multijogador influencia diretamente no resultado final do produto e deve ser feita de acordo com as necessidades do jogo [7].

A arquitetura mais apropriada para comunicação entre computadores jogos digitais é a cliente/servidor. Nela, todos os clientes se comunicam com um único ponto da rede, o servidor, e não precisam lidar com os outros clientes [7]. Essa arquitetura é a que permite maior liberdade no design de jogos em tempo real, pois apenas o servidor possui autoridade sobre os eventos do jogo, eliminando a necessidade de uma simulação determinística como a descrita em [1].

E-Sports e jogos jogados em partidas usam um servidor de *lobby* que redireciona os jogadores para salas em servidores externos, que processam uma única partida. Alguns jogos [17] usam como servidor um dos clientes da partida, já outros usam servidores dedicados [18]. Nos *lobbies*, existem duas principais formas usadas para formar partidas: uma delas é a criação de salas, na qual jogadores podem criar e configurar uma partida ou entrar em salas já criadas; a outra é o serviço de *matchmaking*, na qual a partida é formada automaticamente.

Apesar de ser uma arquitetura complexa, ferramentas como a *Photon PUN*, facilitam a implementação de comunicação cliente/servidor e de serviços de lobby. A *Photon Network* porém não oferece suporte à balanceamento do matchmaking, tornando necessário o desenvolvimento de ferramentas externas.

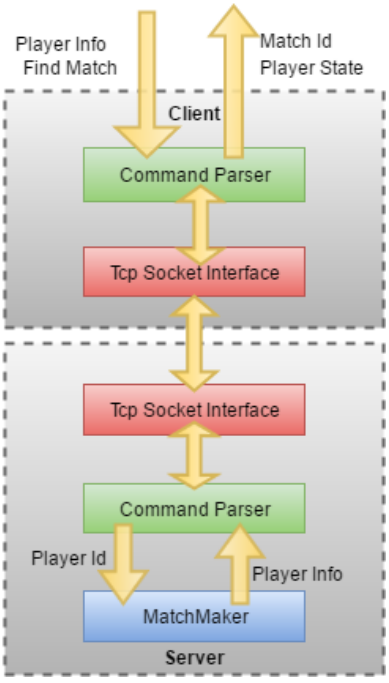

Figura 1: *Diagrama de componentes*

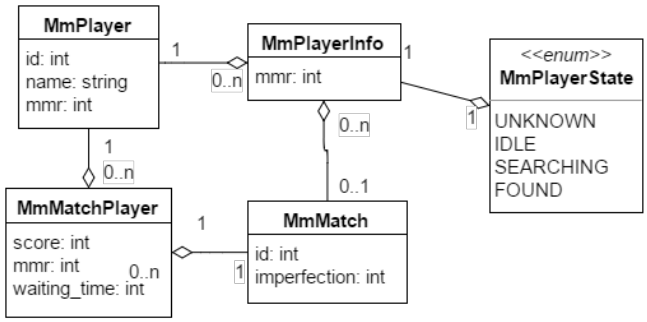

Figura 2: *Estrutura de dados do sistema*

3 Estrutura Proposta

O sistema proposto é dividido em duas partes: Servidor e Cliente. Servidor fica o motor do *matchmaking* e os dados de u suários. O Cliente representa o acesso aos serviços do Servidor, que pode ser a aplicação do jogo digital.

A estrutura do sistema usa como base o design de *middleware* proposto em [8]. Um banco de dados com dados dos usuários e das partidas ficará hospedado em um s ervidor. O serviço de matchmaking também fica rodando em um servidor dedicado, gerando partidas e gerenciando os dados desse banco de dados. Os clientes dependem dos serviços desse servidor para efetuar a busca por partidas ou para acessar informações do banco de dados. Os componentes da estrutura estão descritos na Figura 1.

3.1 Servidor de Matchmaking

O serviço de matchmaking roda em um servidor dedicado. Nele ficam a fila de jogadores esperando para jogar, e as partidas formadas. O serviço também oferece meios para inserir e acessar informações

[13] Comentário de um representante da Valve: http://www.reddit.com/r/GlobalOffensive/comments/2g3r4c/the_ultimate_guide_to_csgo_ranking/ckfhfir

[14] http://unity3d.com/pt/unity

[15] http://docs.unity3d.com/Manual/UNetOverview.html

[16] http://doc.exitgames.com/en/pun/current/getting-started/pun-intro

[17] Como o Halo 2: http://halo.bungie.net/stats/content.aspx?link=h2matchmaking

[18] Como o Planetary Anihilation http://forrestthewoods.com/the-tech-of-planetary-annihilation-chronocam/

dos jogadores e das partidas. O sistema possui 5 componentes principais: O motor, o gerenciador de ranking, o gerenciador de dados, a interface e o tradutor.

O motor, que será chamado de *matchmaker*, é a parte principal do processo. Ele gerencia a fila de jogadores e cria as partidas. Ele oferece opções para adicionar um jogador, remover um jogador, iniciar a busca de partidas, e pegar as informações de um jogador. Veja a Figura **??**. Ao adicionar um jogador, o motor guarda suas informações na estrutura *MmPlayerInfo* (Figura 2), com as informações atuais de avaliação de habilidade, estado da busca, um identificador de partida, que será nulo até que uma partida seja encontrada e o tempo de inicio de busca.

O processo de busca tem como objetivo gerar a melhor partida possível para o jogador que está esperando a mais tempo. A busca ocorre da seguinte forma: Se existe um jogador na fila, o motor busca todas as possíveis partidas para o primeiro jogador. Uma partida é considerada possível se a imperfeição da partida é menor ou igual à tolerância de todos jogadores. Se uma ou mais partidas foram encontradas, a que possui menor imperfeição é iniciada. Os jogadores envolvidos são retirados da fila e o processo recomeça. A tolerância à imperfeição do jogador é calculada da seguinte forma:

$$t = \frac{\Delta T}{Tt} * M$$

Sendo que t é a tolerância do jogador, ΔT representa a diferença de tempo entre o inicio da busca e o tempo atual, Tt é o máximo de espera desejado para o jogador em segundos e M é a maior imperfeição esperada para o jogador. Os parâmetros Tt e M são parâmetros configuráveis no sistema.

O motor de busca pode receber pedidos de inserção ou remoção de jogadores da fila a qualquer momento.

O gerenciador de dados, chamado de *DaoManager*, é o responsável pela persistência dos dados da estruturas *MmPlayer, MmMatch* e *MmMatchPlayer* descritas na Figura 2. Ele contempla a implementação da conexão com banco de dados, e dos *Data Access Objects*, que implementam escrita e leitura dos dados dos jogadores e partidas.A implementação de acesso aos dados é feita usando a biblioteca SOCI[19] em um banco *PostgreSQL*[20].

O gerenciador de ranking recebe e processa resultados das partidas. Ele é o responsável por analisar os resultados de cada partida, implementar e executar as técnicas de avaliação e atualizar a avaliação do jogador através do gerenciador de dados. Ele implementa o algoritmo de Elo usando a média de estimativa de habilidade dos adversários como uma única estimativa. Dessa forma, ele segue o algoritmo normalmente como se fosse uma partida de dois jogadores para cada um dos envolvidos na partida. A nova avaliação gerada a partir do algoritmo é salva no cadastro de usuário e a partida é marcada como fechada.

O tradutor, é o componente responsável por traduzir os comandos da interface para ações do sistema. Esses comandos serão descritos em seguida. **echo** retorna o parâmetro recebido. é usado para testar a comunicação; **addPlayer** adiciona um jogador no banco de dados. Recebe como parâmetros o nome do novo jogador e a avaliação inicial e retorna o númer o identificador do jogador adicionado; **findMatch** adiciona um jogador na fila de busca; **playerInfo** busca informações de um jogador. textbfcloseMatch: Fecha uma partida, e processa os resultados. Recebe como parâmetros o identificador da partida e uma lista de identificadores de jogadores seguidos do resultado de cada um deles.

A interface é o componente responsável pela comunicação com os clientes. Ele cria *sockets* de comunicação *TCP* e abre uma *thread* para cada cliente conectado. De forma semelhante à um serviço de *RPC*, a interface recebe comandos pela rede e envia a resposta processada pelo tradutor. O design proposto tem como objetivo permitir em trabalhos futuros a implementação de interfaces alternativas para acesso via *HTTP* ou *RPC*.

[19]Mais informações em http://soci.sourceforge.net/

[20]Mais informações em http://www.postgresql.org

3.2 Matchmaking Client

O cliente realiza o acesso aos serviços oferecidos pelo servidor. Em um jogo digital, ele representa a implementação das duas ações principais do jogador: a busca por partida e envio de resultados. As ações envolvem os passos:

1. **Estabelecer uma conexão *TCP* com o servidor**: Ao estabelecer a conexão, o servidor fica aguardando comandos. Ao enviar um comando, o servidor processa e responde informando se houve erro ou sucesso, e em casos de pedidos de dados, os dados solicitados

2. (a) **Carregar informações do jogador**: Se o usuário já está cadastrado no sistema, o cliente pode buscar no banco de dados as informações do usuário através do comando **playerInfo**.

   (b) **Ou, Adicionar o jogador ao sistema de matchmaking**: Através do comando **addPlayer**, o cliente pode registrar um jogador no sistema de matchmaking. O servidor responde com o número identificador do jogador.

3. **Iniciar a busca de partida para o jogador**: enviando o comando **findMatch**

4. **Verificar o estado de busca do jogador**: O cliente pode pedir ao servidor de matchmaking o estado atual do jogador. O servidor responde com o estado do busca do jogador, contendo, quando uma partida foi encontrada, o identificador da partida. Esse identificador da partida é o mesmo para todos os usuários envolvidos. Através dele o cliente pode iniciar a sala e conectar todos os jogadores no servidor aonde ocorrerá a partida.

5. **Fechar a partida**: Enviando a lista de jogadores com a pontuacão de cada um.

4 Experimentos

O sistema proposto tem como objetivo facilitar a implementação de um sistema de busca de partidas balanceadas em jogos. Para verificar a funcionalidade do sistema, foi feita uma simulação. Para analisar o processo de implementação do sistema em um cliente foi estudada a implementação do sistema no lobby de um jogo na *Unity3D*. Os experimentos realizados serão descritos e analisados abaixo.

4.1 Simulação

Para validar o funcionamento de matchmaking, foi executada uma simulação de ambiente de matchmaking de partidas de xadrez.

O objetivo da simulação foi validar o funcionamento do sistema em um universo ideal. A simulação é feita por um programa que age como cliente no servidor de *matchmaking*. Esse programa carrega uma lista de jogadores com diversos valores de avaliação, e os coloca buscando partida constantemente, de forma que as seguintes regras são respeitadas: jogadores possuem tolerância à imperfeição de 500 pontos para cada 1 minuto de espera; Foram carregados 100 jogadores no sistema; Apenas o cliente conhece o valor de habilidade real dos jogadores. todos os jogadores possuem uma avaliação inicialmente igual no sistema de matchmaking, mas possuem também uma avaliação real conhecida apenas pelo cliente. No inicio da simulação, todos os jogadores são inseridos na fila de busca por partidas; Cada partida é formada por 2 jogadores; Quando

uma partida é encontrada, ocorre uma simulação de partida perfeita: O jogador com maior valor de habilidade vence, e essa vitória é informada ao sistema; Ao terminar uma partida, os jogadores envolvidos voltam ao sistema de busca.

O principal comportamento esperado no sistema é o aumento do equilíbrio de habilidade entre os jogadores das partidas. Foi possível verificar esse comportamento com a diminuição e estabilização da imperfeição das partidas com o tempo. Foi possível ver também alguns casos isolados de partidas com alta imperfeição. Esses casos aconteceram com jogadores que poissuiam avaliação muito distante dos outros jogadores mas estava a muito tempo esperando na fila. Esses jogadores ficam com uma alta tolerância à imperfeição. Esse é um comportamento previsto, e que garante que nenhum jogador deixará de jogar.

Outra funcionalidade do sistema é a avaliação da habilidade do jogador. Na simulação, foi possível observar uma diminuição na diferença entre habilidade real e estimada de cada jogador com o tempo.

4.2 Estudo de Caso

Para testar a viabilidade do uso do modelo proposto, foi desenvolvido um estudo de caso. O caso estudado é um sistema de matchmaking balanceado para um na Unity3D usando o serviço de *lobby* da *Photon Network*.

Para abstrair a camada de comunicação com o serviço de matchmaking, foram recriadas as classes da estrutura do servidor descritas na Figura 2. Para efetuar a comunicação foram criadas duas classes: *MmClient*, e *MmCommandParser*. A primeira classe tem como objetivo gerenciar a conexão e a comunicação com o servidor via *sockets*, e também esperar as respostas. A segunda traduz funções do sistema para comandos para enviar pela rede, e converte as respostas em *JSON* para objetos do sistema.

Para inserir novos jogadores ao sistema, o cliente deve estabelecer conexão com o servidor, enviar o comando de inserção e esperar a resposta contendo o identificador do jogador. Depois disso o sistema cliente é responsável por guardar o identificador e relaciona-lo com o usuário. No estudo feito, foi criado um cadastro de usuários local, que permite inserir um jogador no sistema e usar o mesmo usuário em sessões futuras.

Após cadastrar um usuário ou selecionar um previamente cadastrado, o jogador pode iniciar uma busca no servidor. A busca é implementada da seguinte forma: O cliente envia ao servidor um pedido de busca. Em seguida, em um laço, espera do servidor o identificador da partida encontrada. O cliente então é conectado à sala de jogo usando o serviço de salas da *Photon Network*. Quando ambos estão conectados o jogo começa. Quando o jogo termina um dos cliente envia ao servidor o resultado da partida.

5 Considerações Finais e Trabalhos futuros

O sistema proposto e implementado oferece um serviço de balanceamento simples para busca de partidas. Este não é um sistema completo. Existem diversos assuntos relacionados à busca por partidas que não foram abordados aqui mas podem ser estudados a partir do sistema proposto. A segurança da comunicação, por exemplo, pode ser implementada com camadas de criptografia entre as camadas de comunicação e os componentes tradutores (Figura 1 no cliente e no servidor. Alterando o módulo gerenciador de rankings seria possível fazer o uso de algoritmos de avaliação mais avançados como o *Glicko-2*, ou ainda experimentar novos algoritmos em um sistema de lobby existente. Também é possível aumentar a compatibilidade do sistema adicionando suporte a protocolos de comunicação de alto nível como o *HTTP/REST*, ou um sistema de *RPC*.

Os resultados da simulação do sistema proposto e implementado mostra que o sistema consegue gerar um equilíbrio nas partidas e, de acordo com os fundamentos introduzidos, maior imersão para jogos competitivos.

O trabalho apresentado incentiva o estudo e o desenvolvimento de jogos competitivos com baixo custo. A implementação do sistema se mostrou modular e pouco invasiva. O sistema de *matchmaking* proposto permite a integração do sistema de balanceamento com sistemas de lobby já existentes, como o sistema de lobby da *Photon Network*, de forma harmoniosa.

Referências

[1] P. Bettner and M. Terrano. 1500 archers on a 28.8: Network programming in age of empires and beyond. *Presented at GDC2001*, 2:30p, 2001.

[2] O. Delalleau. Beyond skill rating: Advanced matchmaking in ghost recon online. *IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES*, 4:167–177, 2012. Autores: Olivier Delalleau, Emile Contal, Eric Thibodeau-Laufer, Raul Chandias Ferrari, Yoshua Bengio, and Frank Zhang.

[3] M. E. Glickman. A comprehensive guide to chess ratings. *American Chess Journal*, 1995.

[4] M. E. Glickman. Dynamic paired comparison models with stochastic variances. *Journal of Applied Statistics, 28, 673-689*, 28(6):673–689, 2001.

[5] J. Gregory. *Game Engine Architecture, Second Edition*. CRC Press, 2014.

[6] K. Salen and E. Zimmerman. *Rules of play: Game design fundamentals*. MIT press, 2004.

[7] J. Smed, T. Kaukoranta, and H. Hakonen. Aspects of networking in multiplayer computer games. *The Electronic Library*, 20(2):87–97, 2002.

[8] J. H. S. Tobias Fritsch, Benjamin Voigt. The next generation of competitive online game organization. 2015.