Arquitetura do servidor de um jogo de cartas multijogador online

Universidade Federal de Minas Gerais
Belo Horizonte, Minas Gerais, Brasil

Abstract

Piratas é um jogo de cartas de turnos de 2 a 6 jogadores que consiste na disputa da conquista de tesouros, onde quem conseguir 5 tesouros primeiro ganha a partida. O presente trabalho pretende apresentar uma arquitetura de um servidor online multijogador e avaliar do ponto de vista de qualidade de software as escolhas de design, estrutura de código, práticas de versionamento e padronização de nomeação. Foi criado um sistema de ação para permitir reuso e facilitar a criação de novas expansões para o jogo. O objetivo do trabalho é avaliar a flexibilidade, testabilidade, legibilidade e escalabilidade do servidor.

CCS Concepts: • **Métricas e qualidade de software → Sistemas distribuídos**.

Keywords: qualidade, sistemas distribuídos, testes, refatoração

ACM Reference Format:
. 2022. Arquitetura do servidor de um jogo de cartas multijogador online. In *Proceedings of UFMG Workshop on Software Engineering (Conference WSE).* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUÇÃO

Piratas é um jogo de cartas de turnos de 2 a 6 jogadores que consiste na disputa da conquista de tesouros, onde quem conseguir 5 tesouros primeiro ganha a partida. O presente trabalho pretende apresentar uma arquitetura de um servidor online multijogador e avaliar do ponto de vista de qualidade de software as escolhas de design, estrutura de código, práticas de versionamento e padronização de nomeação. O servidor consiste numa arquitetura em camadas com o objetivo de isolamento das regras de negócio do jogo de forma a facilitar testes de unidade de cada entidade do domínio. Foi desenvolvido um sistema de ações que permite definir um fluxo de operações e escolhas dos jogadores de forma a reutilizar e isolar código em suas devidas responsabilidades. Também foi desenvolvido um protocolo para que isola completamente o cliente das regras do jogo, convertendo regras complexas de fluxos em escolhas simples para o cliente, isso é importante pois aumenta a segurança do jogo, diminuindo a possibilidade de trapaceiros burlarem alguma regra do jogo, manter a regra apenas no servidor também facilita o processamento de testes de unidade do início ao fim do jogo, aumentando a segurança de implementação de novas funcionalidades. O objetivo do trabalho é introduzir a arquitetura desenvolvida, apresentando suas vantagens de forma a facilitar a implementaçao de novas mecânicas de jogo.

Conference WSE, Dezembro, 2022, Belo Horizonte, MG

ACM ISBN 978-1-4503-XXXX-X/18/06.
https://doi.org/XXXXXXX.XXXXXXX

2 ARQUITETURA

O servidor segue uma arquitetura em camadas com foco no isolamento do domínio do restante das camadas. Esse modelo se apresenta adequado uma vez que facilita escalabilidade e troca de troca de camadas alterando apenas a camada anterior. Apesar do projeto multijogador online, é possível substituir camadas para caso o jogo se torne offline contra uma inteligência artificial sem alterar o domínio já existente.

É importante pontuar que o protocolo se apresenta dentro da estrutura do servidor como um namespace para facilitar o teste do servidor com o cliente em linha de comando. O resultado final desse namespace é se tornar um pacote privado completamente isolado de qualquer projeto, dessa forma, tanto cliente quanto deverão adicionar esse pacote como dependência externa para obter as classes necessárias para a comunicação.

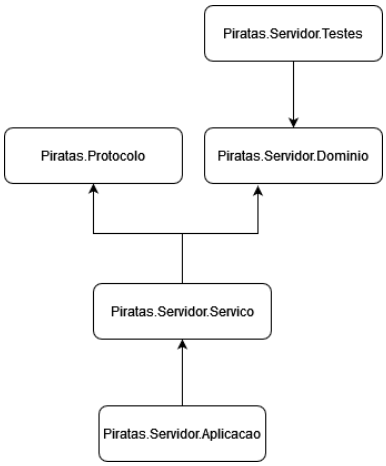

Figure 1. Diagrama de namespaces

A arquitetura foi pensada para possuir toda a lógica de processamento de regras no servidor, isolando o cliente de qualquer conhecimento de domínio do jogo. O objetivo principal dessa decisão é servir como uma barreira que tenta evitar trapaças no jogo. Os autores [4] e [5] classificam essa abordagem como focada no servidor e apresentam mais vantagens para esse modelo tais como diminuir processamento do lado do cliente e facilidade de modificação de regras de domínio sem precisar necessariamente gerar uma nova build e fazer uma nova publicação do cliente.

A estrutura de nomenclatura, formatação de código e semântica foi seguida a partir dos coneitos apresentados por [3] com o intuito de facilitar a leitura do código para novos desenvolvedores e facilitar a manuteção da estrutura do código a longo prazo. Para garantir a padronização da estrutura entre múltiplos desenvolvedores, é utilizado o 'Editor Config', tecnologia de formatação de código que permite a criação de arquivos de configuração que descrevem qual é a estrutura correta a se seguir. Atualmente a maioria dos ambientes de desenvolvimentos integrados e editores de texto dão suporte a ela.

2.1 Domínio

O sistema isola de forma lógica toda a regra de negócio através de um 'namespace' dedicado, esse espaço é dedicado a processamento de regras do jogo. Esse isolamento permite maior organização do domínio e facilita a criação de testes de unidade. Outra vantagem é a possibilidade de mudar a natureza do sistema para um jogo offline sem que seja necessário alterar qualquer código desse espaço. A ideia de pensamento do domínio de forma isolada, prioritária e simples surge com princípios criados por [2].

2.1.1 Principais classes. As principais classes do domínio são 'Jogador', 'Mesa', 'BaseCarta' e 'BaseAcao' de forma a representar a estrutura básica do jogo onde o jogador realiza uma ação envolvendo uma carta na mesa.

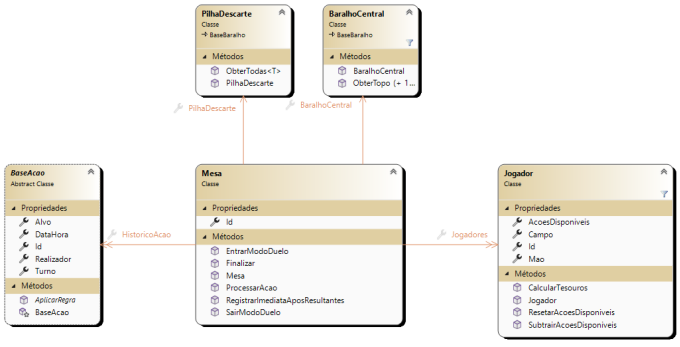

Figure 2. Diagrama de classe com as principais classes do sistema

A classe 'Mesa' é onde todas as ações realizadas pelos jogadores são executadas, ela é responsável por ditar todo o fluxo da gameplay, possuindo todos os elementos necessários para o jogo rodar similar ao que foi implementado por [4] classificando o padrão 'Gameplay Engine'.

2.1.2 Sistema de ações. A interação do jogador consiste num sistema de ação que agrupa escolhas e operações a serem realizadas na mesa e cada tipo de ação tem seu propósito bem definido no sistema. As ações podem ser encadeadas com o objetivo da realização de algum fluxo. A figura 3 mostra um diagrama UML com as principais entidades.

- Primária: Ações principais que o jogador pode realizar em seu turno.
- Resultante: Ações resultado da execução de outra ação e que necessita de alguma escolha do jogador.
- Passiva: Ação a ser executada no fim do turno do jogador gerada por alguma carta no campo.
- Imediata: Ação a ser executada imediatamente ao ser retornada por outra ação.

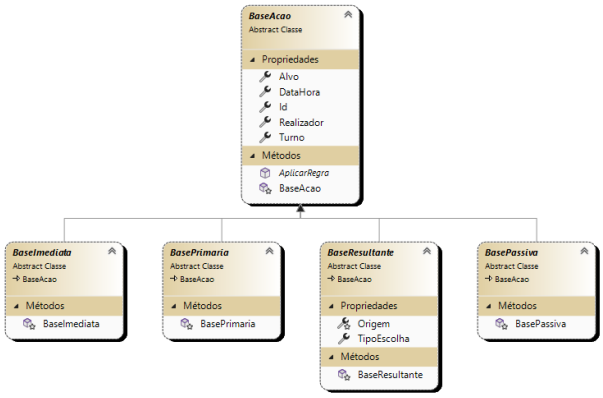

Figure 3. Diagrama de classe das principais entidades do sistema de ação

O fluxo de duelo é o mais complexo do jogo, dessa forma, é o melhor fluxo para exemplificar como o sistema de ação resolve os problemas de comunicação e decisão apresentados pelo mesmo. O fluxo apresentado na figura 4 do jogo consiste em um jogador escolhendo uma carta de duelo iniciadora e decidindo qual com qual jogador vai começar o duelo. Após o início do duelo, o jogador alvo deve descer uma carta de duelo virada para baixo. Ambos os jogadores podem descer cartas do tipo duelo surpresa para aumentar a possibilidade de vitória no duelo. Os jogadores então devem virar suas cartas duelo e quem tiver mais pontos de duelo é declarado vencedor e pode roubar uma carta da mão do perdedor.

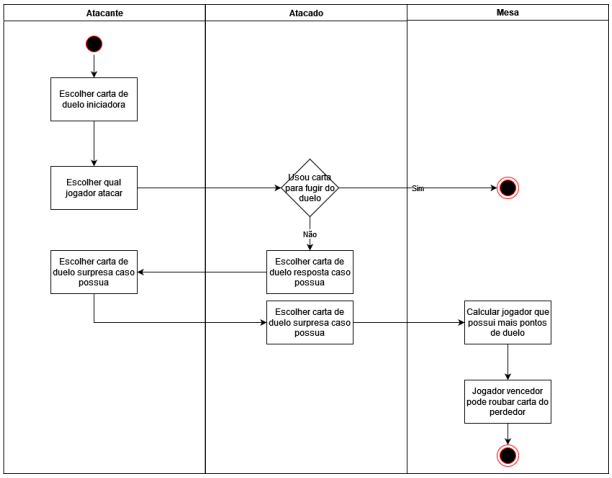

Figure 4. Diagrama de atividade com o fluxo de duelo do jogo

O servidor implementa o fluxo anterior usando o sistema de ações ilustrado na figura /refdiagrama-atividade-sistema-acao-duelo da seguinte maneira: Ao executar a ação primária "Duelar", o servidor devolve a ação resultante "EscolherJogador", após escolher com quem vai duelar, o servidor devolve a ação "EscolherCartaIniciadora" para que o jogador escolha qual carta em sua mão vai iniciar o duelo, após realizar essa escolha o servidor envia a ação "DescerCartasDuelo" para o jogador alvo descer a sua carta duelo resposta. O servidor verifica se algum dos jogadores possui uma carta duelo surpresa e envia a ação "DescerCartaDueloSurpresa" para os que tiverem. O servidor então executa a ação imediata "CalcularResultadoDuelo" para calcular o vencedor e perdedor do duelo e envia a ação resultante "RoubarCarta" para o vencedor.

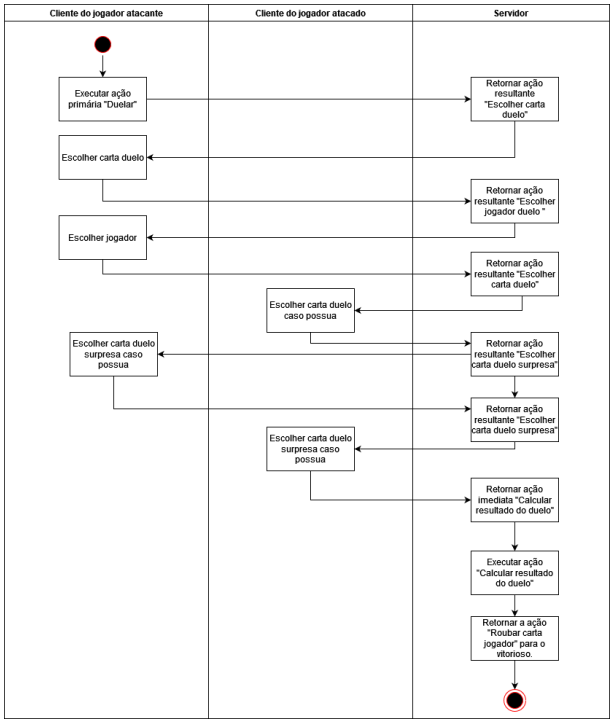

Figure 5. Diagrama de atividade com o funcionamento do sistema de ação no fluxo de duelo

2.1.3 Testes de unidade. Os testes de unidade são implementados em cima de todas as classes do domínio do projeto. A estrutura de pastas é análoga ao do domínio para facilitar navegação.

Para facilitar a escrita dos testes, foi utilizado o pacote de mock de testes NSubstitute. Ele cria em tempo de execução uma classe que herda a passada por parâmetro genérico e permite definir o que é retornado em cada método e propriedade sem a necessidade de invocação formal do construtor e outros métodos da classe.

Listing 1. Teste de unidade da carta Rum

```
namespace Piratas.Servidor.Testes.Cartas.ResolucaoImediata
{
    using System;
    using System.Collections.Generic;
    using Dominio;
    using Dominio.Acoes;
    using Dominio.Cartas;
    using Dominio.Cartas.ResolucaoImediata;
    using NSubstitute;
    using NUnit.Framework;

    public class RumTestes
    {
        [Test]
        public void AplicarEfeitoDeveComprarCartasParaJogador()
        {
            var jogadoresNaMesa = new List<Jogador>();
            var cartasNaMao = new List<Carta>();

            var mesa = new Mesa(jogadoresNaMesa);

            var jogadorRealizador = new Jogador(
                Guid.NewGuid(),
```

```
                null,
                null,
                null,
                null);

            var cartasNoBaralhoCentral = new List<Carta>
            {
                Substitute.For<Carta>(),
                Substitute.For<Carta>(),
            };

            mesa.BaralhoCentral.InserirTopo(cartasNoBaralhoCentral);
            jogadorRealizador.Mao.Adicionar(cartasNaMao);
            var acao = Substitute.For<BaseAcao>(jogadorRealizador, null);
            var rum = new Rum();

            rum.AplicarEfeito(acao, mesa);

            foreach (Carta carta in cartasNoBaralhoCentral)
                Assert.IsTrue(jogadorRealizador.Mao.Possui(carta));

            Assert.IsNull(mesa.BaralhoCentral.ObterTopo());
        }
    }
}
```

O pacote de testes de unidade utilizado foi o NUnit por sua ampla adoção pela comunidade, facilitando na obtenção de respostas para dúvidas.

2.2 Comunicação cliente servidor

2.2.1 Web Socket. Web socket é um protocolo de comunicação construído sobre uma conexão TCP contínua que permite envio ativo de de pacotes tanto pelo cliente quanto pelo servidor. Facilitando o transporte de dados em tempo real para os vários clientes conectados. [1] Se vale desse protocolo para implementar um jogo multiplayer baseado em browser pela bidirecionalidade, de forma a permitir que o servidor envie mensagens ao cliente sem que o mesmo faça uma requisição pedindo aquela mensgem.

O servidor utiliza esse protocolo pois facilita o envio de pacotes para todos os jogadores, permitindo que o servidor avise aos jogadores quando uma carta é removida da mão de um dos jogadores, por exemplo. É mencionado por [5] a ideia do uso do TCP como protocolo de comunicação principalmente pelo fato dele ficar resopnsável pela ordenação dos pacotes, ponto que para um jogo de cartas é essencial. Considerando que o fluxo de comunicação do jogo é consideravelmente menor do que servidores de jogos em tempo real, o custo desse protocolo não se apresenta relevante para a solução atual.

2.2.2 Protocolo. Foi desenvolvido um protocolo para uniformizar a comunicação e isolar o conhecimento do domínio pelos clientes. Existem pacotes de cliente e servidor para os dois tipos de contexto implementados: Sala e partida.

O fluxo de comunicação consiste no envio de uma mensagem de servidor e resposta do cliente a mensagem recebida.

Pacote de sala do servidor:

- Id: Id do pacote.
- IdErro: Id do erro caso tenha ocorrido.
- DescricaoErro: Descrição do erro caso tenha ocorrido.
- PossuiErro: Possui erro no pacote. Feito para facilitar verificação de erro.
- IdJogador: Id jogador que recebe o pacote.
- IdSala: Id da sala em que a operação ocorreu.
- IdPartida: Id da partida caso tenha sido inicializada.
- IdJogadorRealizouOperacao: Id do jogador que realizou a operação.
- TipoOperacaoSala: Tipo da operação realizada (entrar, sair, iniciar partida).

Listing 2. JSON do pacote da sala do servidor

```
{
    "Id": "91b7388b-6a53-4df5-bad2-562d1e98f610",
    "IdErro": "",
    "DescricaoErro": "",
    "PossuiErro": false,
    "IdJogador": "2536612e-84be-46c6-879c-d9e8755fbfa3",
    "IdSala": "ba1b1b64-08a6-4a45-b955-eeb9e9f39bd0",
    "IdPartida": "",
    "IdJogadorRealizouOperacao": "1adeb4ef-9c90-46df-b61b-9c57b90ebd87",
    "TipoOperacaoSala": 1
}
```

Pacote de sala do cliente:

- Id: Id do pacote.
- IdJogador: Id jogador que recebe o pacote>
- IdSala: Id da sala em que a operação ocorreu.
- IdJogadorRealizouOperacao: Id do jogador que realizou a operação.
- TipoOperacaoSala: Tipo da operação realizada (entrar, sair, iniciar partida).

Listing 3. JSON do pacote da sala do cliente

```
{
    "Id": "ba1b1b64-08a6-4a45-b955-eeb9e9f39bd0",
    "IdJogadorRealizouOperacao": "ffa93170-17d5-4118-9ad5-0afdc12c8fb4",
    "IdSala": "2536612e-84be-46c6-879c-d9e8755fbfa3j",
    "TipoOperacaoSala": 1
}
```

Pacote de partida do servidor:

- Id: Id do pacote.
- IdErro: Id do erro caso tenha ocorrido.
- DescricaoErro: Descrição do erro caso tenha ocorrido.
- PossuiErro: Possui erro no pacote. Feito para facilitar verificação de erro.
- Eventos: Eventos de que ocorreram na mesa. (Carta adicionada, removida na mão de jogador etc).
- Tesouros: Quantidade de tesouros que o jogador possui.
- AcoesRestantes: Quantidade de ações restantes do jogador.
- IdMesa: Id da mesa.
- DataHora: Data hora da criação do pacote.
- Escolha: Escolha para o jogador fazer (escolha entre cartas, ações etc).
- IdJogadorRealizador: Id do jogador que realizou a ação

Listing 4. JSON do pacote da partida do servidor

```
{
    "Id": "dadbfc3c-bff6-4970-aa23-c2c78eeac721",
```

```
    "IdErro": "",
    "AcoesRestantes": 2,
    "Tesouros": 1,
    "Eventos":
    {
        "888b4316-f6da-4e46-93f9-afc8df092763":
        [
            {
                "Local": 3,
                "IdCarta": "rum",
                "Adicionado": false
            }
        ]
    },
    "DescricaoErro": "",
    "PossuiErro": false,
    "IdMesa": "a61dab79-3046-4072-a229-9821f5d620b5",
    "IdJogadorRealizador": "767507ef-3784-440a-b86b-e0a8b707611f",
    "DataHora": 1667923144,
    "Escolha":
    {
        "Opcoes":
        [
            "rum",
            "papagaio",
            "kraken",
            "batata"
        ],
        "LimiteQuantidadeEscolha": 1
    }
}
```


Pacote de partida do cliente:

- IdMesa: Id da mesa.
- IdJogadorRealizador: Id do jogador que realizou a ação.
- DataHora: Data hora da criação do pacote.
- IdAcaoExecutada: Id da ação que o jogador está executando.
- Escolha: Resposta da escolha.

Listing 5. JSON do pacote da partida do cliente


```
{
    "IdMesa": "9b315933-487e-4ef0-8cc1-85c9f2ee9cee",
    "IdJogadorRealizador": "ffa93170-17d5-4118-9ad5-0afdc12c8fb4",
    "DataHora": 1667923144,
    "IdAcaoExecutada": "escolher-carta",
    "Escolha": "papagaio"
}
```


2.3 Serviço

Essa camada é responsável por servir como intermediária entre o domínio e a camada de aplicação, isolando o conhecimento de uma pela outra. Essa camada possui inúmeros serviços com própósitos diferentes mas que servem como a camada que lida com "tecnologia" do projeto, ou seja, leitura/escrita de arquivos, acesso a banco de dados, requisições web entre outras.

2.3.1 Configuração. Responsável por obter o arquivo de configuração no formato JSON e convertê-lo numa instância do domínio para que os serviços e domínios sejam capazes de se configurar de acordo. O projeto segue o padrão de criar um arquivo de configuração por ambiente seguindo o seguinte padrão de nomenclatura: configuracao.<AMBIENTE>.json

2.3.2 Log. O serviço de log é responsável por configurar o pacote de log Serilog para escrita simultânea num arquivo de texto para futuras análises e no console. O serviço também garante que exceções não tratadas são capturadas e devidamente logadas a partir do "AppDomain.CurrentDomain".

2.3.3 Sala. Responsável por gerenciar todas as salas abertas e processar as operações realizadas pelos clientes. Notifica o gerenciador de partidas quando uma nova deve ser iniciada.

2.3.4 Partida. Consiste num serviço responsável por receber mensagens do cliente, repassar para a instância da mesa e fazer as devidas conversões do resultado do domínio para a estrutura do protocolo.

2.3.5 Gerenciador de partida. Responsável por armazenar partidas em andamento. Esse serviço serve como intermediário entre os controladores da partida e o serviço de partida em si. Ele também é o responsável por criar partidas quando uma sala está completa e removê-la quando o jogo termina.

3 TRABALHOS FUTUROS

3.1 Testes com partidas reais

É possível persistir toda a comunicação do cliente/servidor de partidas válidas e reprocessar esses dados, comparando os dados de entrada e saída da partida persistida com o que o servidor acabou de processar. Considerando que o servidor deve funcionar de forma determinística, essa solução permite rodar infinitas partidas reais no servidor, testando se o servidor continua consistente no que diz respeito ao processamento de partidas. Essa solução pode aumentar a segurança do código de forma a aumentar a garantia de que o fluxo que já existe não foi quebrado por implementações novas. Vale citar que existe o cenário em que funcionalidades introduzidas alteram os fluxos já existentes, dessa forma, os testes não deveriam rodar as partidas não compatíveis.

3.2 Otimização de uso de memória das instâncias de cartas

Atualmente o servidor cria uma instância de carta para cada carta disponível no jogo. Porém, essa instância não possui nada de diferente de uma instância de mesma carta de um jogador de outra mesa. Dessa maneira, o servidor desperdiça memória. Uma possível solução seria criar uma instância de cada carta e manusear apenas ID de carta no baralho e mão dos jogadores e, quando precisar executar o efeito de uma carta, pedir para que essa entidade a execute baseado em seu id. Isso permite que apenas uma instância de cada carta exista para todas as mesas ativas no servidor.

4 CONCLUSÃO

Piratas é um jogo de cartas de turnos de 2 a 6 jogadores que consiste na disputa da conquista de tesouros, onde quem conseguir 5 tesouros primeiro ganha a partida. O presente

trabalho pretende apresentar uma arquitetura de um servidor online multijogador e avaliar do ponto de vista de qualidade de software as escolhas de design, estrutura de código, práticas de versionamento e padronização de nomeação. O servidor consiste numa arquitetura em camadas com o objetivo de isolamento das regras de negócio do jogo de forma a facilitar testes de unidade de cada entidade do domínio. O principal objetivo do trabalho é avaliar a arquitetura e modalegem desenvolvidos para o jogo, de modo a verificar a flexibilidade de implementação de fluxos existentes e novos possíveis fluxos a serem introduzidos por expansões com novas cartas e mecânicas. O trabalho apresenta o sistema de ação como principal mecanismo de flexibilização de estrutura de fluxo para controlar os eventos do jogo e é avaliado como satistatório para suportar novas mecânicas. Outro ponto positivo é o protocolo pois permitiu o isolamento completo dos clientes da regra de negócio do jogo a partir da abstração de escolhas. Apesar de possuir múltiplos pontos de melhorias de design e otimização de memória e processamento, o servidor se apresenta adequado para entrar num ambiente de produção.

References

[1] Bijin Chen and Zhiqi Xu. 2011. A framework for browser-based Multiplayer Online Games using WebGL and WebSocket. In *2011 International Conference on Multimedia Technology*. 471–474. https://doi.org/10.1109/ICMT.2011.6001673

[2] Eric Evans. 2004. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley.

[3] Robert C. Martin and James O. Coplien. 2009. *Clean code: a handbook of agile software craftsmanship*. Prentice Hall, Upper Saddle River, NJ [etc.].

[4] D. Varun Ranganathan, R. Vishal, Vallidevi Krishnamurthy, Prashant Mahesh, and Roopeshwar Devarajan. 2017. Design patterns for multiplayer card games. In *2017 International Conference on Computer, Communication and Signal Processing (ICCCSP)*. 1–3. https://doi.org/10.1109/ICCCSP.2017.7944107

[5] Cai Lan Zhou, Yu Kui Wang, and Sha Sha Li. 2011. Design of server for network casual games. In *2011 International Conference on Computer Science and Service System (CSSS)*. 2193–2196. https://doi.org/10.1109/CSSS.2011.5972196