Ataíde – CSU04 – Bloquear Jogador

Descrição da funcionalidade:

- Evitar interações com jogadores indesejados.

Problemas e inconsistências encontrados:

- A descrição do DA04 mostra que o jogador pode informar um motivo para bloqueio (como “abuso verbal”, “incompatibilidade”), mas isso não está representado nos atributos da classe Jogador nem no relacionamento com ServicoDeModeracao.
- O caso de uso trata o bloqueio como ação do jogador, mas a existência da classe ServicoDeModeracao e dos métodos bloquearJogadorSistema() e desbloquearJogadorSistema() sugere também uma ação administrativa. Essa ambiguidade precisa ser resolvida.
- Embora o DA04 sugira uma opção de desbloqueio, não há uma descrição clara no CSU04 sobre como desbloquear, nem se existe confirmação para o desbloqueio. Também não há controle de tempo ou histórico.
- Não fica claro se o jogador bloqueado é notificado sobre o bloqueio, mas isso poderia ser mencionado como uma política do sistema.
- A modelagem trata o bloqueio como um simples método dentro da classe Jogador, mas seria mais robusto representá-lo com uma classe associativa como Relacionamento, contendo tipo (amizade, bloqueio), data, motivo, etc.

Propostas de melhoria:

- Adicionar uma entidade Relacionamento entre jogadores com atributos como tipo, motivo, dataInicio, dataFim, ativo, etc.
- Especificar melhor a separação entre bloqueios feitos por jogadores e por administradores.
- Incluir no fluxo do CSU04 o processo de desbloqueio.
- Adicionar uma regra de negócio que proíba interações mesmo indiretas após o bloqueio.
- Garantir que o diagrama de atividade represente o bloqueio como decisão confirmada e reversível.

Ataíde – DA05 – Enviar Mensagem

Descrição da funcionalidade:

Detalha o funcionamento do sistema de chat.

Problemas e inconsistências encontrados:

- O DEM04 indica que é necessário verificar permissão para envio a não amigos, mas isso não está detalhado.
- O DA05 permite escolha entre chat global e pessoal, mas não há modelagem específica para o chat global nas classes.
- O envio de notificações é citado, mas não fica claro se a notificação é gerada por nova mensagem, por visualização, por leitura, etc.
- Os diagramas e as regras não explicam a integração entre sistema de mensagens e sistema de bloqueios.

Propostas de melhoria:

- Criar uma classe CanalDeChat, separando ChatGlobal de ChatPrivado, com regras e atributos distintos.
- Detalhar no diagrama de estados quais permissões são avaliadas no envio a não amigos.
- Adicionar regras para moderação do chat global (palavrões, flood, etc.).
- Ampliar a função de ServicoDeNotificacao para identificar tipo de notificação (mensagem recebida, lida, ignorada).
- Criar integração clara entre Jogador.bloquearJogador() e a impossibilidade de envio de mensagens.

Nalanda – CSU01 – Buscar Jogadores

•Explicação da funcionalidade:

- Essa funcionalidade permite que os jogadores encontrem outros usuários que sejam compatíveis com eles, usando filtros como jogo, estilo, plataforma e região. O sistema usa um matchmaking inteligente para facilitar essa busca.

•Análise:

- Ao analisar os artefatos, como o caso de uso CSU01, o diagrama de atividade DA03, o diagrama de estado DEM02 e as classes relacionadas, percebi alguns pontos que poderiam ser melhorados. Por exemplo, a **pré-condição** do caso de uso só fala que o jogador precisa estar logado e com o perfil preenchido, mas na prática o sistema depende de informações bem detalhadas, como estilos de jogo, horários e

plataformas. Acho que isso poderia estar mais claro e o sistema deveria validar esses dados antes de permitir a busca.

- Outro ponto é que o **fluxo alternativo** só cobre o caso de nenhum jogador compatível ser encontrado, mas não tem nada sobre o que acontece se der erro no sistema, ou se os filtros forem preenchidos de forma errada.

•Sugestão:

- Deixar a pré-condição mais explícita e funcional. Além disso, os diagramas poderiam contemplar melhor os diferentes caminhos que o usuário pode seguir, como cancelar, corrigir filtros ou lidar com erros.

Nalanda – DA03 – Buscar Jogadores

•Explicação da funcionalidade:

- O diagrama representa como o jogador realiza a busca por outros jogadores, podendo aplicar filtros ou simplesmente receber sugestões automáticas do sistema. A ideia principal é facilitar o encontro de perfis compatíveis para partidas, com base em jogo, estilo, plataforma, região e outros dados.

•Análise:

- Durante a análise do DA03, junto com o caso de uso CSU01, o diagrama de estado DEM02 e as classes envolvidas (como ServicoDeMatchmaking e Sugestao), percebi que o processo **está muito direto e simples, talvez até demais**. O jogador basicamente aplica os filtros e o sistema já parte para a sugestão de perfis, sem prever situações mais realistas, como a pessoa querer **refinar os filtros, cancelar a busca, ou até mesmo tentar de novo se não gostar dos resultados.**

- Outra coisa que me chamou atenção é que o próprio **jogador** parece estar envolvido demais na **execução da sugestão**, quando isso deveria ser algo mais automático e feito pelo serviço. O papel do jogador deveria ser apenas o de iniciar a busca e receber o resultado.

•Sugestão:

- O sistema poderia permitir que o jogador refinasse os filtros, cancelasse ou repetisse a busca com facilidade. Ademais, a forma como o método receberSugestao() está implementado na classe Jogador faz parecer que o próprio jogador está chamando e controlando o processo de sugestão, o que dá a ele uma responsabilidade que não

deveria ter. O mais adequado seria que essa lógica ficasse totalmente no ServicoDeMatchmaking, que receberia os filtros, processaria os dados e, ao final, retornaria uma lista de sugestões já pronta para o jogador apenas visualizar.

Matheus Tavares – CSU02 – Enviar Convite

Descrição da funcionalidade:

Permite que um jogador envie um convite de amizade para outro jogador, desde que já tenha ocorrido uma interação prévia via chat.

Artefatos analisados:

- Caso de Uso CSU02 – Enviar Convite
- Diagrama de Atividade DA04 – Visualizar Perfil
- Diagrama de Estado DEM03 – Visualizar Perfil
- Classe Jogador
- Classe Convite
- Classe Notificacao
- Classe ServicoDeNotificacao

Problemas e inconsistências encontrados:

1. **Pré-condição mal aplicada no diagrama:**

O caso de uso especifica que é necessário ter havido interação via chat para o envio do convite. No entanto, nem o DA04 nem o DEM03 tratam ou verificam essa pré-condição antes de liberar a ação “Enviar Convite”.

2. **Extensão ambígua:**

As extensões “CSU02.1 – Aceitar convite” e “CSU02.2 – Notificações em tempo real” são citadas no CSU02 como extensões, mas na prática fazem parte de outro contexto (resposta ao convite e gerenciamento de notificação). Isso pode causar confusão conceitual sobre o escopo da funcionalidade original.

3. **Papel do chat mal definido:**

A relação entre chat e convite é apenas descrita na pré-condição textual, mas não representada nos diagramas. Isso compromete a rastreabilidade entre requisitos e modelagem.

4. **Falta de tratamento de exceções:**

- E se o outro jogador já estiver na lista de amigos?
- E se o convite já tiver sido enviado anteriormente?
- O sistema permite múltiplos convites simultâneos?

5. **Problema de segurança:**

A restrição do envio de convites a apenas jogadores que já conversaram é uma decisão interessante para evitar spam, mas ela precisa ser validada programaticamente e essa lógica está ausente dos diagramas e da modelagem de classe.

Justificativa técnica:

Um bom caso de uso precisa estar alinhado com os diagramas de atividade e estado, especialmente em funcionalidades sensíveis como convites, que envolvem notificações e segurança de interação. A ausência de validação explícita pode gerar comportamentos inconsistentes ou expor falhas na lógica de controle de amizade.

Propostas de melhoria:

1. Representar no DA04 e DEM03 uma verificação de pré-condição antes de liberar o botão de convite.
2. Criar um diagrama de estado específico para o convite, mostrando os possíveis estados (pendente, aceito, expirado, recusado).
3. Separar o caso de uso de envio de convite do gerenciamento da notificação.
4. Adicionar um fluxo alternativo que trate a tentativa de envio de convite repetido ou inválido.
5. Refletir no modelo de classe do convite os estados de controle e vínculo com a conversa prévia.

Conclusão:

Embora o caso de uso esteja bem-intencionado e realista, a falta de validação e o excesso de responsabilidades implícitas causam risco de ambiguidade e falhas de segurança. A funcionalidade se tornaria mais clara e segura com a representação explícita das restrições e o controle do ciclo de vida dos convites.

Matheus Tavares – DEM03 – Visualizar Perfil

Descrição da funcionalidade:

Representa os diferentes estados possíveis ao visualizar o perfil de outro jogador, incluindo as ações que podem ser realizadas: enviar convite, mensagem, avaliação ou bloqueio.

Artefatos analisados:

- Diagrama de Estado DEM03 – Visualizar Perfil
- Diagrama de Atividade DA04 – Visualizar Perfil
- Casos de Uso relacionados: CSU02 – Enviar Convite, CSU03 – Avaliar Jogador, CSU04 – Bloquear Jogador
- Classe Jogador
- Classe Notificacao
- Classe Convite
- Classe Avaliacao

Problemas e inconsistências encontrados:

1. Estado Composto mal aproveitado:

O diagrama apresenta um “Estado Composto Funcionalidades” com ações paralelas (avaliar, bloquear, etc.), mas trata como se apenas uma dessas ações pudesse ser executada por vez. Isso não representa corretamente a natureza da interface, que permite múltiplas interações dentro da tela de perfil.

2. Ausência de transições internas:

Não há representação do que acontece após cada ação dentro do perfil. Por exemplo, se o jogador envia uma avaliação e depois deseja bloquear o mesmo perfil, isso exige nova entrada no estado ou reuso do estado atual? O diagrama não esclarece.

3. Saída única genérica:

Após qualquer ação, o sistema transita para “Saindo do Perfil”. Isso reduz a fidelidade do modelo, pois o usuário pode simplesmente voltar sem fazer nenhuma ação, ou pode fazer várias. Essa rigidez no diagrama não reflete o comportamento real da interface.

4. Ausência de exceções ou cancelamento:

- E se a ação for cancelada?
- E se a tentativa de envio de convite ou avaliação falhar?
- Nenhuma dessas possibilidades está representada.

5. Falta de controle de permissões:

Algumas ações deveriam depender de pré-condições. Por exemplo, só é possível avaliar após uma partida, e o convite depende de chat prévio. O diagrama não demonstra essas restrições.

Justificativa técnica:

Diagrama de estado é usado para modelar o comportamento reativo do sistema, considerando eventos externos, transições internas, e condições de guarda. Neste caso, o uso de estados compostos está simplificado demais e oculta a complexidade real da tela de perfil, o que prejudica a análise de navegabilidade e resposta do sistema a diferentes entradas do usuário.

Propostas de melhoria:

1. Redefinir o “Estado Composto Funcionalidades” como um estado que permite múltiplas transições internas, mantendo o usuário no estado até que ele opte por sair manualmente.

2. Adicionar eventos de falha para ações críticas como envio de convite ou bloqueio.

3. Incluir condições de guarda para validar se ações são permitidas (ex: só avaliar após partida).

4. Dividir as ações em subestados com suas próprias transições, refletindo melhor a interação real.

5. Inserir estado intermediário “Aguardando ação do jogador” para dar maior controle sobre o fluxo de eventos.

Conclusão:

O diagrama é útil para visualizar os recursos disponíveis no perfil, mas peca por falta de flexibilidade e fidelidade à experiência real do usuário. Uma reformulação baseada em

múltiplas interações e tratamento de exceções tornaria o sistema mais bem representado e fácil de manter.

Matheus Yuri – CSU03 – Avaliar Jogador

Descrição da funcionalidade:

Permite que um jogador avalie outro jogador após o término de uma partida. A nota e/ou comentário influenciam o sistema de sugestões futuras.

Artefatos analisados:

- Caso de Uso CSU03 – Avaliar Jogador
- Diagrama de Atividade DA04 – Visualizar Perfil
- Diagrama de Estado DEM03 – Visualizar Perfil
- Classe Avaliacao
- Classe Jogador
- Classe ServicoDeMatchmaking

Problemas e inconsistências encontrados:

1. Falta de validação da pré-condição:

A descrição textual diz que só é possível avaliar após uma partida concluída entre os dois jogadores, mas isso não está representado nos diagramas de atividade nem no modelo de classes. Não há controle que garanta que a avaliação não seja feita arbitrariamente.

2. Fluxo superficial:

O fluxo principal tem apenas quatro passos simples, sem variações ou validações. Não há menção a avaliações duplicadas, anonimato, reversão ou edição da avaliação.

3. Ausência de fluxo alternativo ou de exceção:

- O que acontece se o jogador se recusar a avaliar?
- E se houver erro ao enviar?
- E se ele quiser editar uma avaliação feita por engano?

4. Modelagem de classe subutilizada:

A classe Avaliacao só armazena nota, comentário e data, sem vinculação explícita à partida ou jogador avaliador/avaliado. Isso dificulta a rastreabilidade e integridade da informação.

5. Avaliação sem controle de frequência:

O sistema não define se é possível avaliar o mesmo jogador mais de uma vez. Isso abre margem para abusos, como avaliações negativas em massa.

Justificativa técnica:

O sistema de reputação (baseado em avaliação) impacta diretamente o matchmaking. Se for mal modelado, pode ser manipulado por jogadores mal-intencionados. Além disso, é essencial garantir que somente jogadores que jogaram juntos possam avaliar, o que exige validação cruzada com histórico de partidas.

Propostas de melhoria:

1. Adicionar verificação explícita (pré-condição) de que houve partida válida antes de liberar a avaliação.

2. Criar um relacionamento entre a Avaliacao e a entidade “Partida” (mesmo que esta não tenha sido modelada, pode ser sugerida).

3. Inserir fluxos alternativos para falha no envio, avaliação recusada ou edição posterior.

4. Limitar a quantidade de avaliações por jogador por período ou por partida.

5. Tornar o fluxo mais detalhado, com validações visuais no diagrama de atividade (ex: nota inválida, botão “enviar”, confirmação).

Conclusão:

A funcionalidade é simples, mas por lidar com impacto social entre usuários, precisa de controle mais robusto e validação. A ausência desses controles enfraquece o sistema de reputação e pode comprometer a justiça do matchmaking.

Matheus Yuri – DA01 – Acesso ao Sistema

Descrição da funcionalidade:

Descreve o processo de login e cadastro de um jogador no sistema, desde a escolha da ação (entrar ou registrar-se) até o acesso à conta.

Artefatos analisados:

- Diagrama de Atividade DA01 – Acesso ao Sistema
- Diagrama de Estado DEM01 – Acesso ao Sistema
- Classe Usuario
- Classe ServicoDeUsuario
- Requisitos RF01 e RF06

Problemas e inconsistências encontrados:

1. Validações não específicas:

O diagrama cita validação de nickname, e-mail e senha, mas não especifica os critérios (por exemplo: formato do e-mail, complexidade da senha, unicidade do nickname). Isso enfraquece a modelagem, pois não representa as regras de negócio de segurança.

2. Falta de tratamento de falhas:

O fluxo de login mostra uma tentativa com senha errada que retorna para o início, mas não há limite de tentativas nem sistema de bloqueio. Isso pode comprometer a segurança da aplicação.

3. Cadastro sem confirmação de e-mail:

Não há passo para verificar ou confirmar o e-mail via link ou código. Essa omissão é crítica em sistemas com interação entre usuários, pois pode gerar contas falsas.

4. Inexistência de caminho de recuperação de senha:

Embora a funcionalidade esteja descrita na classe Usuario (recuperarSenha), não há nenhum fluxo ou diagrama que represente esse processo.

5. Confusão entre diagramas de atividade e estado:

O DEM01 também trata o login, mas de forma redundante ao DA01. Falta coordenação entre os dois – não fica claro o que complementa o quê.

Justificativa técnica:

Login e cadastro são pontos sensíveis de segurança. A ausência de validações explícitas, confirmação de identidade e fluxos de erro bem modelados pode permitir o acesso de usuários indevidos ou dificultar a recuperação de contas legítimas.

Propostas de melhoria:

1. Especificar nos diagramas os critérios de validação (ex: “senha com no mínimo 8 caracteres, contendo letra maiúscula e número”).

2. Adicionar fluxos alternativos de falha (ex: senha errada 3x → conta bloqueada).

3. Criar um novo diagrama de atividade exclusivo para recuperação de senha.

4. Incluir confirmação de e-mail no processo de cadastro.

5. Revisar o papel do DEM01 para que ele complemente e não repita o DA01.

Conclusão:

Apesar de ser um processo básico, o login/cadastro mal especificado pode abrir brechas críticas no sistema. Com ajustes simples nos fluxos e nas validações, a robustez e a segurança do acesso seriam muito maiores.

Mirelle CSU05 Cadastro de Horários (Casos de uso)

Descrição da funcionalidade:
Permite que o jogador informe seus horários disponíveis para jogar, seja manualmente ou com base em login anteriores.
Artefatos analisados:

- Caso de Uso CSU05 – Cadastro de Horários
- Diagrama de Atividade DA02 – Editar Perfil
- Diagrama de Estado DEM01 – Acesso ao Sistema
- Classe Jogador
- Regra de Negócio RN04

Problemas e inconsistências encontrados:

1. Ação de definir horários está “escondida” dentro de outro caso de uso - Apesar de ser uma funcionalidade central, o fluxo está modelado somente dentro do DA02 – Editar Perfil, o que enfraquece a rastreabilidade e independência do CSU05 como caso de uso separado.
2. Falta de detalhamento sobre sugestão automática de horários - A funcionalidade permite aceitar sugestões automáticas com base nos horários de login, mas nenhum artefato explica como isso é feito, como funciona a lógica por trás disso?
3. Ausência de feedback sobre sobreposição de horários ou conflitos - Não há qualquer verificação prevista sobre horários conflitantes, sobreposição com outros compromissos ou alertas visuais na interface.

4. Interface para o cadastro de horários não é representada em nenhum artefato visual - A RNF06 cita uma “interface intuitiva”, mas não há evidência visual (mockup, descrição ou comportamento específico) dessa interface no fluxo.
5. Falta integração com a funcionalidade de matchmaking no nível de modelagem - O texto diz que os horários serão usados no matchmaking, mas não há nenhuma associação explícita ou direta com o ServicoDeMatchmaking ou suas classes.

Justificativa técnica:
O gerenciamento de horários influencia diretamente a compatibilidade entre jogadores. Se mal representado, pode causar problemas de combinação incorreta, frustração do usuário e dificuldade de adesão à plataforma.

Propostas de melhoria:
1. Criar um subcaso de uso próprio para “Definir Horários”, com pré-condições e fluxos bem descritos.
2. Modelar explicitamente a lógica de sugestão automática, vinculando-a à frequência de login ou sessões ativas.
3. Representar o cadastro com feedback visual e prevenção de conflitos de horários.
4. Adicionar um relacionamento visível entre os horários e o ServicoDeMatchmaking.
5. Descrever os fluxos alternativos para situações como: não salvar os horários, tentar definir horários duplicados, etc.

Conclusão:
A funcionalidade é importante, mas está “diluída” dentro de outras ações e carece de destaque. Separar, refinar e integrar melhor essa funcionalidade pode aprimorar significativamente a precisão das sugestões no sistema.

Mirelle - Buscar Jogadores (Diagrama de Estados)

Descrição da funcionalidade:
Modela os diferentes estados durante o processo de busca por jogadores compatíveis, desde a filtragem até o convite e notificação.

Artefatos analisados:
- Diagrama de Estados DEM02 – Buscar Jogadores
- CSU01 – Buscar Jogadores
- CSU02 – Enviar Convite
- Classe Jogador, ServicoDeMatchmaking, Notificacao
- Regras de Negócio RN02 e RN03

Problemas e inconsistências encontrados:
1. Falta de verificação de login/estado inicial no diagrama - O diagrama começa diretamente no estado “Buscando Jogadores”, sem verificação se o usuário está logado e com perfil preenchido, o que é uma pré-condição clara no CSU01.

2. Falta de estado para resultados vazios - O CSU01 prevê um fluxo alternativo FA01 (“Nenhum jogador encontrado com esses critérios”), mas esse estado está ausente no diagrama de estados, o que impede o rastreio visual dessa exceção comum.
3. Transições pouco detalhadas entre ações principais - Após o estado “Sugerindo jogadores”, o diagrama transita diretamente para “Visualizando o Perfil” apenas se o jogador for selecionado, mas não prevê outros caminhos como uma opção clara para redefinir filtros, voltar à tela anterior, encerrar sem ação ou modificar critérios dentro do fluxo.
4. Estado de envio de convite está muito linear - As ações de “enviar convite” e “notificar” estão representadas como transições diretas e sequenciais. Porém, não há validação de elegibilidade do convite (chat prévio) nem consideração de bloqueios (RN03), que deveriam constar.

Justificativa técnica:
Um bom diagrama de estados deve contemplar exceções, validações, e opções de navegação lateral. Isso garante clareza de comportamento para o sistema e confiabilidade para o usuário.

Propostas de melhoria:
1. Adicionar um estado de “Verificando Pré-condições” antes de iniciar a busca.
2. Incluir estado “Nenhum jogador encontrado” com possibilidade de voltar e ajustar os filtros.
3. Adicionar transições alternativas para cancelar, modificar filtros, ou encerrar busca sem interação.
4. Incluir verificação de bloqueios antes do envio de convite no fluxo de estados.
5. Considerar estados compostos dentro da “Sugerindo jogadores”, com visualização de múltiplos perfis ou ações paralelas.

Conclusão:
O diagrama de estados apresenta a lógica básica da busca por jogadores, mas omite elementos importantes para robustez e usabilidade. Com pequenos ajustes, o comportamento do sistema pode ficar muito mais realista e seguro.