1.2 Mudanças em Relação à Etapa Anterior

Na etapa anterior do trabalho, o grupo elaborou um esquema UML para representar as classes do jogo de tabuleiro Trilha. Contudo, esse esquema sofreu diversas modificações ao longo do desenvolvimento. Inicialmente, superestimamos a complexidade das classes e métodos relacionados às mecânicas do jogo (como colocação, movimentação e remoção de peças no tabuleiro) e subestimamos as necessidades para implementar a interface gráfica.

Por exemplo, no diagrama anterior, utilizamos apenas uma classe para toda a implementação da interface gráfica, que se revelou extremamente simplificada. Nesta etapa, no entanto, dedicamos muito mais esforço à interface, o que resultou na criação de mais classes e métodos para atendê-la adequadamente.

Adicionalmente, as classes responsáveis por elementos principais do jogo, como casas, peças e jogadores, originalmente incluíam atributos ou métodos relacionados à posição no grid da interface, como o atributo `int pos_x`. Porém, percebemos que tais implementações poderiam ser transferidas para classes específicas da interface, tornando o design mais limpo.

Também identificamos que algumas classes, como *Moinhos* e *Jogadores*, não precisaram ser criadas. Essas entidades puderam ser representadas dentro de outras classes sem comprometer a manutenibilidade ou modularidade do projeto, considerando sua simplicidade. Essa abordagem tornou o design mais enxuto e eficiente, alinhando-se melhor às necessidades reais do jogo.

1.3 Implementação

O que foi implementado:

Classe Casa: responsável por representar as casas do tabuleiro no jogo Trilha. Cada casa possui um identificador único (`id`), um caractere (`ocupante`) que indica se a casa está ocupada por uma peça (e qual jogador a ocupa) ou se está vazia (representado por `'-'`), e uma lista de casas adjacentes (`adjacentes`).

Atributos:

- `id`: identifica de forma única cada casa no tabuleiro.
- `ocupante`: caractere que representa quem está ocupando a casa. O valor inicial é `'-'`, indicando que a casa está vazia.

- `adjacentes`: lista de casas conectadas diretamente à casa atual, representando as possíveis movimentações no tabuleiro.

Métodos:

- **`getOcupante()`**: retorna o caractere que representa o ocupante atual da casa.
- **`setOcupante(char ocupante)`**: define o ocupante da casa. Pode ser um caractere representando o jogador ou `'-'` para indicar que está vazia.
- **`getAdjacentes()`**: retorna a lista de casas adjacentes à casa atual.
- **`adicionarAdjacente(Casa casa)`**: adiciona uma casa à lista de adjacentes, permitindo criar conexões entre as casas do tabuleiro.

Classe Tabuleiro: responsável por representar e gerenciar a estrutura do tabuleiro do jogo Trilha. Essa classe encapsula a lógica principal do tabuleiro, incluindo as casas, conexões entre elas, grupos de moinho, e operações como colocar, mover e remover peças. A classe também lida com a verificação de moinhos, bloqueios de movimento e fornece funcionalidades para reiniciar o estado do tabuleiro.

Estrutura:

Atributos:

- `casas`: Lista que contém todas as casas do tabuleiro, representadas pela classe `Casa`.
- `gruposDeMoinho`: Matriz de combinações de casas que formam um moinho, utilizada para verificar a formação de moinhos no jogo.
- `conexoes`: (interno ao método) Matriz que define as conexões entre casas adjacentes no tabuleiro.

Constantes:

- `NUM_CASAS`: Define o número total de casas no tabuleiro (24 no jogo Trilha).

Funcionalidades:

1. **Conexão entre casas:**

    - Método `conectarCasas`: Cria as relações de adjacência entre as casas, de acordo com as regras do jogo.
2. **Colocação de peças:**
    - Método `colocarPeca`: Permite ao jogador colocar uma peça em uma casa vazia no tabuleiro.
3. **Movimentação de peças:**
    - Método `moverPeca`: Realiza o movimento de uma peça de uma casa para outra adjacente, verificando a ocupação e a conexão entre elas.
4. **Remoção de peças:**
    - Método `removerPeca`: Remove uma peça de uma casa, caso o jogador tenha permissão para realizar essa ação.
5. **Verificação de moinho:**
    - Método `verificarMoinho`: Verifica se uma peça colocada em uma posição específica forma um moinho com outras peças do mesmo jogador.
6. **Verificação de bloqueio:**
    - Método `verificarBloqueio`: Determina se todas as peças de um jogador estão bloqueadas, ou seja, não têm movimentos válidos.
7. **Gerenciamento do estado do tabuleiro:**
    - Método `resetarTabuleiro`: Restaura o tabuleiro ao estado inicial, removendo todas as peças.
    - Método `imprimirTabuleiro`: Exibe o tabuleiro em um formato visual estruturado, com os símbolos das peças ou espaços vazios.
8. **Acessos auxiliares:**
    - Métodos `getPeca` e `getCasa`: Permitem acessar diretamente o ocupante ou o objeto `Casa` de uma posição específica no tabuleiro.

Classe ControleJogoGUI: esta classe implementa a interface gráfica e a lógica do jogo *Trilha*, permitindo interações entre os jogadores e o tabuleiro. Ela gerencia o estado do jogo, alterna entre as fases do jogo (colocação de peças, movimentação e captura), e atualiza os componentes da interface com base nas ações dos jogadores.

Atributos:

- **`NUM_CASAS`**: constante que define o número total de casas do tabuleiro.
- **`tabuleiro`**: objeto que representa o tabuleiro do jogo, responsável pela lógica das casas e suas conexões.

- **`nomeJogadores` e `jogadores`**: arrays que identificam os nomes e os símbolos ('b' para branco, 'p' para preto) de cada jogador.
- **`pecasDisponiveis` e `pecasNoTabuleiro`**: arrays que rastreiam o número de peças disponíveis para colocação e as já posicionadas no tabuleiro para cada jogador.
- **`jogadorAtual`**: índice do jogador que está na vez.
- **`botoesTabuleiro`**: array de botões que representam graficamente as casas do tabuleiro.
- **`GRID_SIZE`**: constante que define o tamanho da grade da interface.
- **`faseInicial`, `modoCaptura`, `origemSelecionada`**: controlam o estado do jogo (colocação, captura e movimentação).

Métodos:

Inicialização:

- **`ControleJogoGUI()`**: construtor que inicializa o tabuleiro e configura a interface gráfica.
- **`inicializarInterface()`**: cria os componentes gráficos da interface e os configura com os estados iniciais.

Atualização:

- **`atualizarInterface()`**: atualiza o estado dos botões e as informações exibidas para os jogadores na interface com base no progresso do jogo.

Interação com os jogadores:

- **`tratarClique(int posicao)`**: processa os cliques nos botões, diferenciando entre as fases do jogo:
    - Na fase inicial, permite posicionar peças.
    - No modo de captura, permite remover peças do adversário.
    - Na fase de movimentação, permite selecionar uma peça e movê-la.
- **`alternarJogador()`**: alterna a vez entre os jogadores.
- **`verificarVitoria()`**: verifica as condições de vitória (adversário com menos de 3 peças ou bloqueado).

Reinicialização:

- **`reiniciarJogo()`**: reinicia o jogo, restaurando o estado inicial para começar uma nova partida.

Comentários sobre implementação:

1. **Estrutura da interface gráfica:**
    - A interface usa um *GridLayout* para organizar os botões do tabuleiro com base na configuração de 7x7.
    - O mapeamento do tabuleiro (matriz `mapaTabuleiro`) garante que apenas as posições válidas tenham botões.
2. **Fases do jogo:**
    - A lógica do jogo está claramente separada nas fases de **colocação**, **movimentação** e **captura**.
    - O atributo `modoCaptura` é habilitado após a formação de um moinho para permitir capturar peças do adversário.
3. **Gerenciamento do jogo:**
    - A classe atualiza dinamicamente a interface após cada ação para refletir o estado atual, incluindo o jogador ativo, peças disponíveis e ocupação das casas.
4. **Reforço na usabilidade:**
    - Mensagens de diálogo (*JOptionPane*) são usadas para comunicar eventos importantes, como a formação de um moinho ou erros em jogadas.

A classe `BotaoCircular` é uma extensão da classe `JButton` que desenha botões com formato circular. A principal funcionalidade dessa classe é substituir a aparência padrão dos botões do Swing, tornando-os esteticamente mais agradáveis e adequados para jogos como o Trilha. Aqui está o resumo das principais características:

Características Principais

1. **Formato Circular**:
    - A aparência do botão é alterada para um formato circular, desenhado com `fillOval` no método sobrescrito `paintComponent`.
2. **Interação Visual**:
    - O botão muda de cor ao ser clicado, alternando para a cor amarela (`Color.yellow`) no estado "armed".
3. **Bordas Invisíveis**:
    - A renderização de bordas padrão é desativada com `setBorderPainted(false)` para evitar interferências no design circular.

4. **Detecção de Clique Precisa**:
    - O método `contains(int x, int y)` é sobrescrito para calcular se um clique ocorreu dentro da área circular do botão, garantindo maior precisão.
5. **Tamanho Personalizado**:
    - O método `getPreferredSize` assegura que o botão seja perfeitamente quadrado, ajustando largura e altura para o mesmo valor.

A classe chamada `Main` serve como o ponto de entrada principal para a aplicação gráfica do jogo. Ele utiliza o framework Swing para iniciar a interface gráfica do jogo de Trilha.

Características Principais

1. **Extensão de `ControleJogoGUI`**:
    - A classe `Main` estende a classe `ControleJogoGUI`, que contém a lógica e os componentes da interface gráfica para o jogo.
2. **Execução da Interface Gráfica**:
    - O método `main` chama o método estático `SwingUtilities.invokeLater` para garantir que a interface gráfica seja inicializada na *Event Dispatch Thread* (EDT), como recomendado para aplicações Swing.
3. **Simplificação da Inicialização**:
    - A chamada `ControleJogoGUI::new` cria uma nova instância da interface gráfica, iniciando automaticamente o jogo.

1.4 Testes

Para a classe Tabuleiro, temos:

Testes Realizados:

`colocarPeca()`:

- **Objetivo**: Verificar se uma peça é corretamente colocada no tabuleiro em uma casa específica.

- **Descrição**: O teste coloca uma peça 'X' na casa com ID 1 e verifica se a peça foi colocada corretamente na casa.
- **Resultado Esperado**: A peça na casa com ID 1 deve ser 'X'.

`moverPeca()`:

- **Objetivo**: Verificar se uma peça pode ser movida de uma casa de origem para uma casa de destino.
- **Descrição**: O teste coloca uma peça 'X' na casa de origem e move para a casa de destino, verificando se a peça foi movida corretamente.
- **Resultado Esperado**: A peça deve ser removida da casa de origem e colocada na casa de destino.

`removerPeca()`:

- **Objetivo**: Verificar se uma peça pode ser removida corretamente de uma casa.
- **Descrição**: O teste coloca uma peça 'X' em uma casa e a remove, verificando se a casa se torna vazia após a remoção.
- **Resultado Esperado**: A peça deve ser removida da casa, e o ocupante da casa deve ser `'-'`.

`verificarMoinho()`:

- **Objetivo**: Verificar se o sistema detecta corretamente um "moinho" (3 peças alinhadas).
- **Descrição**: O teste coloca 3 peças 'X' em casas adjacentes e verifica se o método `verificarMoinho` detecta o moinho.
- **Resultado Esperado**: O método deve retornar `true` indicando que há um moinho.

`verificarBloqueio()`:

- **Objetivo**: Verificar se o sistema detecta corretamente o bloqueio de um jogador.
- **Descrição**: O teste coloca peças de dois jogadores em casas específicas e verifica se a função `verificarBloqueio` reconhece o bloqueio de um dos jogadores.
- **Resultado Esperado**: O método deve retornar `true` indicando que o jogador 'Y' está bloqueado.

`getPeca()`:

- **Objetivo**: Verificar se o método retorna corretamente a peça em uma casa.
- **Descrição**: O teste coloca uma peça em uma casa e verifica se o método `getPeca` retorna corretamente a peça.
- **Resultado Esperado**: O método deve retornar a peça presente na casa.

`getCasa()`:

- **Objetivo**: Verificar se o método retorna corretamente a casa com um ID específico.
- **Descrição**: O teste coloca uma peça em uma casa e verifica se o método `getCasa` retorna corretamente a casa com o ID especificado.
- **Resultado Esperado**: O método deve retornar a casa correta.

`resetarTabuleiro()`:

- **Objetivo**: Verificar se o método de reset do tabuleiro limpa corretamente todas as casas.
- **Descrição**: O teste coloca uma peça em uma casa, reseta o tabuleiro e verifica se a casa foi limpa.
- **Resultado Esperado**: Após resetar o tabuleiro, a casa deve estar vazia (`'-'`).

Para a classe Casa, temos:

Testes realizados:

Casa Vazia

- **Objetivo**: Verificar se a casa está vazia no momento da criação.
- **Método**: `testCasaVazia()`
- **Descrição**: Inicialmente, a casa deve estar vazia, ou seja, seu ocupante deve ser representado pelo caractere `'-'`.
- **Resultado Esperado**: A casa deve retornar `'-'` como ocupante.

Casa com Peça Preta

- **Objetivo**: Verificar se a casa corretamente armazena uma peça preta.
- **Método**: `testPecaPreta()`
- **Descrição**: Ao definir o ocupante da casa como uma peça preta (`'p'`), a função `getOcupante()` deve retornar `'p'`.

- **Resultado Esperado**: O ocupante da casa deve ser `'p'`.

Casa com Peça Branca

- **Objetivo**: Verificar se a casa corretamente armazena uma peça branca.
- **Método**: `testCasaBranca()`
- **Descrição**: Após definir o ocupante da casa como uma peça branca (`'b'`), a função `getOcupante()` deve retornar `'b'`.
- **Resultado Esperado**: O ocupante da casa deve ser `'b'`.

Casa e Seu Vizinho

- **Objetivo**: Verificar a interação de uma casa com uma casa vizinha, especificamente a verificação do ocupante da casa vizinha.
- **Método**: `testVizinhoBranca()`
- **Descrição**: Uma casa deve ser capaz de adicionar uma casa vizinha e, ao verificar o vizinho, a peça da casa vizinha deve ser retornada.
- **Resultado Esperado**: A casa vizinha deve ter como ocupante `'b'`, que é a peça definida para ela.

Lista de Vizinhos

- **Objetivo**: Validar o comportamento da casa quando se tem uma lista de casas vizinhas.
- **Método**: `testListaVizinhos()`
- **Descrição**: A função deve ser capaz de retornar corretamente a lista de vizinhos de uma casa e garantir que o ocupante da casa da lista seja o esperado.
- **Resultado Esperado**: A primeira casa da lista de vizinhos deve ter o ocupante correto.

A utilização de testes unitários durante a implementação foi crucial para garantir que os métodos estivessem funcionando conforme o esperado, especialmente nas classes menores que compõem a estrutura geral do sistema. Esses testes ajudaram a manter o controle sobre o funcionamento individual das classes, como `Tabuleiro` e `Casa`, prevenindo problemas antes que se manifestassem em classes mais complexas, como a `ControleJogoGUI`. Sem os testes, seria provável que muitos erros aparecessem durante a implementação da `ControleJogoGUI`, já que ela depende diretamente das classes menores. Nesse cenário, a falta de testes unitários resultaria em um processo de manutenção mais demorado e com maior potencial de falhas. Portanto, os testes

unitários não apenas garantiram a qualidade das classes individuais, mas também otimizaram o desenvolvimento da classe maior, economizando tempo e esforço ao detectar problemas precocemente.

1.5 Executável (Aplicação, Interface)

A aplicação implementada apresenta as funcionalidades principais de um jogo de tabuleiro Trilha, possível de ser encontrado em diversas lojas. Nele há o gerenciamento de peças, movimentação e verificação de condições do jogo (como verificação de bloqueios e moinhos - 3 peças de um jogador formando uma linha reta horizontal ou vertical). A interface gráfica foi projetada para permitir uma interação simples e direta.

Nessa interface, as casas do tabuleiro permanecem em cor acinzentada quando indicam que não há peça para colocar nelas. Como existem dois jogadores com peças na cor Branco e Preto, quando um desses jogadores coloca um peça em uma casa, ela recebe essa cor e uma pequena letra indicando o jogador (P - preto, B - branco). Mais do que isso, existe uma imagem de fundo para deixar o tabuleiro com um aspecto de tabuleiro de verdade, melhorando um pouco a visualização.

Agora explicaremos as funcionalidades da interface:

1. **Tabuleiro interativo**: O tabuleiro é mostrado na interface com as casas dispostas em três quadrados, um deles externo, outro médio e o último interno. Cada casa pode conter uma peça (representada por uma letra ou, caso vazia, pelo símbolo ‘-’). A cor dessas casas reflete a peça em que se encontra sobre ela.
2. **Movimento de peças**: O jogador pode selecionar uma peça já posicionada no tabuleiro e movê-la para uma casa adjacente. A movimentação é feita com simples cliques nas casas de origem e destino.
3. **Colocação de peças**: Na fase de colocação de peças, elas podem ser colocadas em casas vazias no tabuleiro por meio de um simples clique. Se a casa não estiver sozinha e o jogador tentar colocar ali, um pop-up vai informá-lo para tentar colocar em outro lugar.
4. **Verificação de Condições de Jogo**: Quando um jogador forma um moinho, o jogo vai mostrar um pop-up dizendo para o jogador atual retirar a peça do outro jogador. Depois de selecionar, a peça inimiga some. Caso o jogador tente retirar uma peça própria ou clicar em uma casa vazia, uma janela pop-up vai informá-lo para realizar uma remoção de peça correta. O jogo verifica também bloqueios, nesse caso, uma janela vai informar a vitória do jogador que causa o bloqueio.

5. **Reiniciar**: Quando um jogador remove várias peças do adversário, fazendo com que ele tenha apenas duas, uma janela vai surgir e informar que o jogador ganhou. Quando ele clicar em “ok”, o jogo vai reiniciar do zero. Caso um jogador causar um bloqueio, o mesmo acontece.

Manual de uso:

1. O jogo começa na fase de posicionamento de peças. Os jogadores, com 9 peças cada, devem colocar todas elas no tabuleiro. Eles só podem colocar peças em casas vazias e cada um deles possui um turno. Logo, quando o Branco começa o jogo (ele sempre começa, por convenção), o Preto vai logo em seguida.
2. Depois das 18 peças totais colocadas, começa a fase de movimentação. Nessa fase os jogadores, em turno, devem clicar em uma casa com uma de suas peças e clicar na peça a qual desejam que essa peça se mobilize. Movimentos só podem ocorrer para casas adjacentes.
3. Os jogadores devem tentar formar moinhos, que são a formação de 3 peças em linha reta horizontal ou vertical. Quando isso ocorre, eles possuem o direito de retirar qualquer peça do jogador oponente. Depois de retirarem uma peça, o jogador oponente recebe o turno.
4. Quando sobrarem apenas duas peças para um dos jogadores, eles perdem e o oponente ganha. Caso nenhuma peça do jogador possam ser movimentadas, ocorre um bloqueio, e esse jogador perde.

Na fase inicial do desenvolvimento, a interface foi projetada e prototipada com base nas funcionalidades essenciais do jogo. A interface prototipada incluía um tabuleiro simples, com funcionalidades básicas de colocação e movimentação de peças, além de mensagens informativas sobre o estado do jogo. Infelizmente, queríamos ter deixado a interface atual mais bonita, com mais recursos visuais e com uma estética mais trabalhada, sem falar em proporcionar animações ao colocar, movimentar e retirar as peças (esse é o ponto mais interessante da interface que acabou por estar ausente). No requisito de interatividade, entretanto, acreditamos que fizemos o trabalho que nos propusermos a fazer, pois, seguindo as regras do jogo Trilha, o jogador iria se familiarizar rapidamente e jogar o jogo como o desejado e esperado. As funcionalidades seguem as regras do jogo Trilha normalmente, apenas trocando os movimentos manuais que um jogador em pessoa faria com as peças e tabuleiro por cliques na interface gráfica.

Em resumo, a aplicação foi construída de forma que as funcionalidades planejadas na prototipagem fossem implementadas de maneira eficiente e fácil de usar. A interface

gráfica, embora simples, é funcional e cumpre seu papel de proporcionar uma experiência de jogo interativa e acessível.