Trabalho Final – Etapa 1 – 2025/1

INF01120 – Técnicas de Construção de Programas

Rayan Raddatz, Enzo Lisboa, Natan Tristão, Gabriel Henrique e *Iuri Kali* – Grupo 2

Mudanças em relação à etapa anterior

Houve poucas mudanças em relação ao projeto elaborado na etapa anterior. A interface foi levemente alterada e o visual das peças foi modificado. As classes principais do domínio permanecem, agora nomeadas em Inglês – *Match*, *Player*, *Clock*, *Board*, *Move*, a classe abstrata *Piece* e suas filhas. Há uma nova classe, *Blank*, que herda de piece *Piece*, e serve para representar espaços em branco no tabuleiro envolvidos em movimentos.

Foram criados novos métodos para representar a lógica do jogo, e algumas responsabilidades foram transferidas de uma classe para outra. O exemplo principal é o Tabuleiro (*Board*), que ganhou vários métodos para verificações e atualização da sua matriz de peças, tratando de movimentos especiais como roque, en passant, promoção do peão, xeque e xeque-mate.

Algumas relações entre classes, como *Match* — *Play* e *Player* — *Piece* foram removidas, pois durante o desenvolvimento notou-se que não havia necessidade de relacioná-las diretamente, bastando as relações indiretas por meio das classes intermediárias, como *Board*. A figura 1 mostra uma versão atualizada do diagrama de classes com base nas mudanças realizadas.

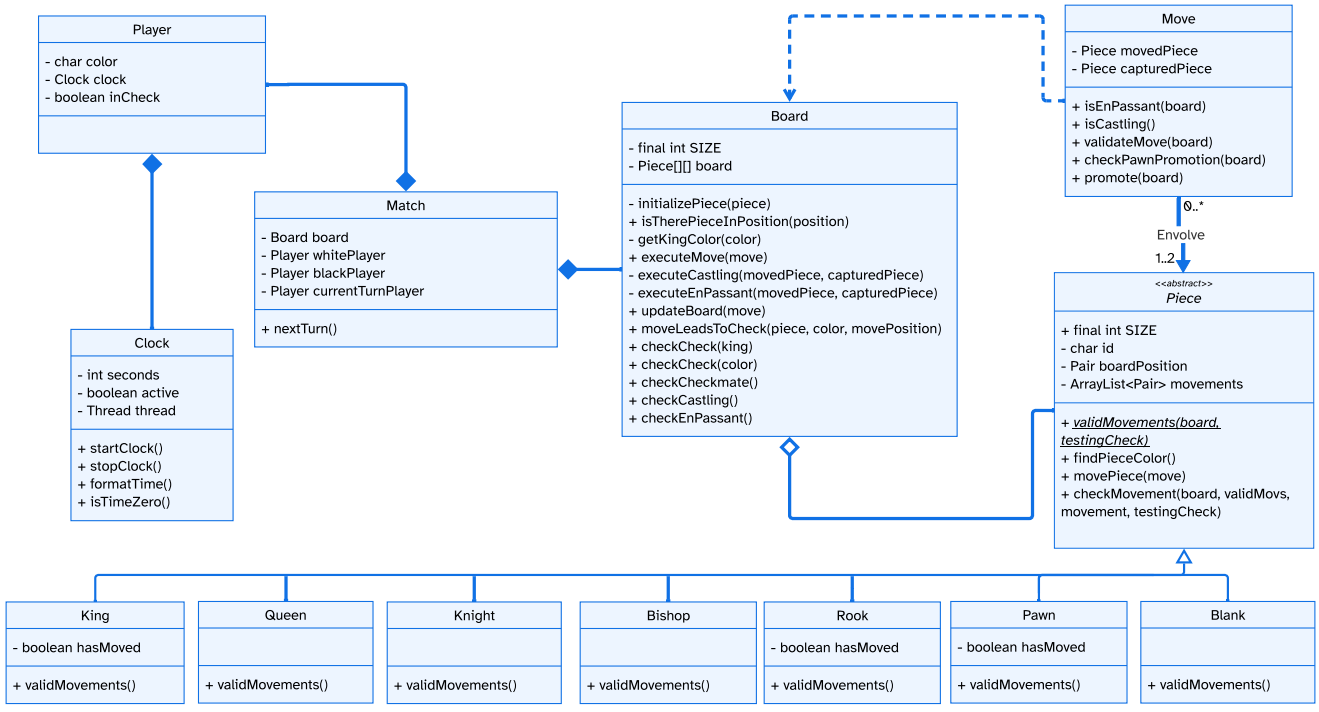


Figura 1: Diagrama de classes atualizado.

Implementação

O jogo foi criado inteiramente em Java, com auxílio de uma versão da biblioteca gráfica Raylib para Java intitulada Jaylib. O processo de implementação foi então dividido entre a parte gráfica e a parte lógica. A parte gráfica, utilizando

a *Jaylib*, abrangeu a interação com o usuário, criação dos menus e visualização do jogo. Enquanto a parte lógica abrangeu a implementação e validação dos movimentos das peças, validação das jogadas, atualizações no tabuleiro e no jogo e verificação de finalização do jogo.

Toda a implementação da lógica seguiu o diagrama planejado na etapa anterior, todas as classes foram de fato implementadas, além de serem criadas classes auxiliares para auxiliar na parte gráfica ou representar estruturas de dados. Por outro lado, muitas das relações entre as classes foram alteradas ou removidas. O número total de relações foi reduzido de 10 para 7. Como exemplo, e já citado anteriormente, a classe *Board* ficou responsável por relacionar indiretamente algumas classes, removendo a relação direta entre elas planejada anteriormente.

Focando em cumprir com os requisitos funcionais e não funcionais estabelecidos na etapa anterior, diversos novos métodos foram criados que não haviam sido introduzidos no diagrama de classes anterior. Muitos dos métodos introduzidos são responsáveis pela validação das jogadas (buscando cumprir os requisitos **RF-2**, **RF-3** e **RF-6**). Para validar uma jogada, foi necessário primeiro identificar o movimento válido de cada peça. Para isso, cada uma das peças tem um método `ValidMoviments()`, o qual, baseado no tabuleiro, retorna quais movimentos a peça pode ou não pode fazer. Tal método também deve verificar se o movimento leva ou não seu rei a cheque antes de afirmar que tal movimento é válido. Assim, quando o usuário clicar em uma de suas peças na tela, verificamos os movimentos válidos da peça e mostramos ao usuário (**RF-3**), que pode selecionar um deles para, então, criar sua jogada (**RF-2**). Quando o jogador faz sua jogada, criamos uma nova instância da classe *Move* (anteriormente *Jogada*), e verificamos se tal jogada é válida utilizando o método `validateMove()`. Em caso positivo, a jogada é executada utilizando o método `executeMove()`, o qual utiliza métodos auxiliares como `updateBoard()` para atualizar o tabuleiro, além de métodos para realização de movimentos especiais, como `executeMoveCastling()` para realizar o roque por exemplo. Todos os requisitos (funcionais e não funcionais), estabelecidos na etapa anterior foram atendidos na visão do grupo.

Testes

Foram desenvolvidos testes unitários para todas as classes do pacote game, o domínio com a lógica do projeto. Sua implementação foi feita utilizando a ferramenta *JUnit*, em 46 funções de teste localizadas no diretório `/test`, espelhando a hierarquia de classes presentes em `/src`. Em resumo, os testes automatizados cobrem os seguintes contextos:

Testes para Piece e suas filhas

- Encontrar a cor de uma peça (preta, branca ou nenhuma);
- Movimentar uma peça e checar se sua posição interna foi atualizada corretamente;
- Verificar o conjunto válido de movimentos para espaço em branco, Bispo, Rei, Cavalo, Peão, Dama e Torre. Os testes foram feitos se baseando na organização abaixo, onde no lugar do rei preto coloca-se a peça que se deseja testar.

Testes para Board

- Conferir que o tabuleiro foi atualizado corretamente após alguns movimentos de exemplo;
- Detecção de movimentos que levam os reis a xeque;
- Detecção de xeque-mate;
- Tratamento de roque e en passant;

Testes para Clock

- Formatação do tempo do relógio;
- Conferir que o relógio inicia e para quando solicitado;

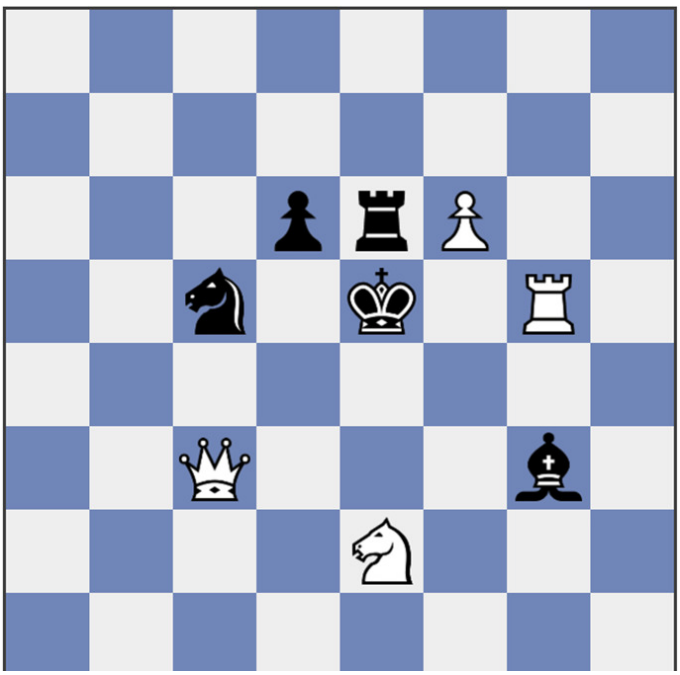


Figura 2: Tabuleiro utilizado nos testes de movimentos válidos.

Testes para Match

- Inicialização correta de uma nova partida e controle da vez de cada jogador;

Testes para Move

- Detecção de En Passant e Roque;
- Promoção do Peão para Bispo, Cavalo, Rainha e Torre;
- Validação de movimento conforme a lista de movimentos válidos da peça;

Testes para Player

- Inicialização de um novo jogador

Houve uma dificuldade inicial em implementar os testes automatizados, visto que grande parte da lógica do jogo é não trivial e os testes envolvem inicializar peças em posições específicas no tabuleiro bidimensional, de forma a fazer a validação e execução de movimentos especiais. Além disso, a ferramenta *JUnit* apresentou problemas no seu uso conjunto com a biblioteca gráfica, não permitindo rodar testes em classes e métodos que fizessem chamadas à *JayLib*, então foram necessárias refatorações para permitir a separação entre lógica e UI antes que os testes pudessem rodar sem lançar exceções.

Executável (aplicação, interface)

Interface

A interface gráfica, abrangida pela parte gráfica discutida anteriormente, foi implementada utilizando a biblioteca externa *Jaylib*, que consiste em uma versão em Java da biblioteca open source *Raylib*, originalmente escrita em `C`.

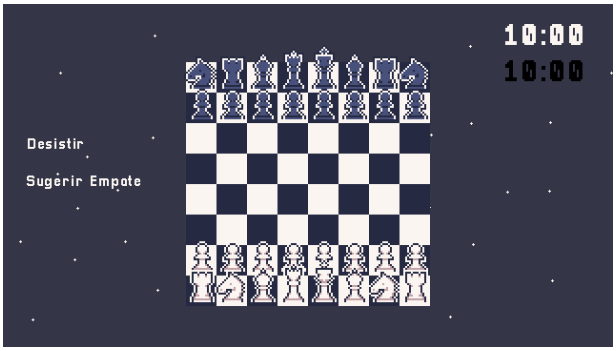
O visual segue o que foi definido na etapa anterior, e poucas mudanças foram feitas. Abaixo seguem algumas fotos comparando a versão ”protótipo”com a versão final de algumas das telas da aplicação. [1]

Menu Final

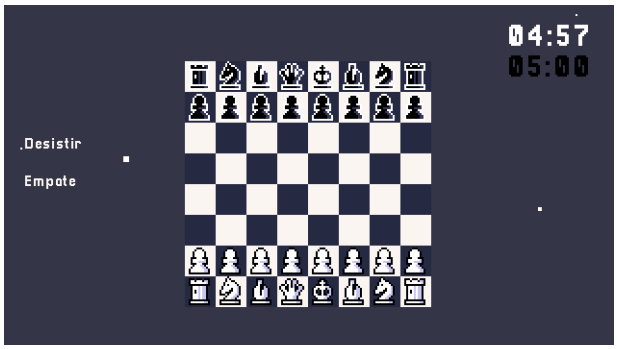
Protótipo

Versão final

Tela do Jogo



Protótipo



Versão final

Menu Final

Protótipo

Versão final

[1] As partículas na versão final do jogo são dinâmicas.

Compilando a partir do código fonte

Uma das formas de executar o jogo, e a mais direta, envolve compilar o código fonte disponível no Github.

Clonando o repositório

O código a seguir copia o código do Github para a pasta `"tcp"` na máquina do usuário.

```
git clone https://github.com/SW-Engineering-Courses-Karina-Kohl/tcp-final-20251-grupo02.git tcp
```

Após isso, o usuário pode abrir a pasta `"tcp"` em seu computador e compilar e executar o jogo com base em seu sistema operacional:

Compilando e executando no Windowns através do PowerShell

Os comandos a seguir compilam e executam, respectivamente, o código copiado do repositório do github utilizando o PowerShell do Windows.

```
javac -cp lib/jaylib-5.5.0-2.jar -d bin (Get-ChildItem -Recurse -Filter *.java -Path src).FullName
java -cp lib/jaylib-5.5.0-2.jar‘;bin app.Main
```

Compilando e executando no Linux com Make [2]

Junto ao repositório é possível encontrar um arquivo `"Makefile"`[3] configurado para compilar e também executar a aplicação. Com isso, para executar o programa, basta digitar o seguinte comando na mesma pasta em que se encontra o arquivo `"Makefile"`:

```
make run
```

Compilando e executando no Linux com .jar

Junto ao repositório também é possível encontrar um arquivo configurado `".jar"` que pode ser utilizado para compilar e executar a aplicação. Com isso, para executar o programa, basta digitar o seguinte comando na mesma pasta em que se encontra o arquivo `".jar"`:

```
java -jar tcp.jar
```

Através do link do itch.io no Windows

É possível baixar diretamente o arquivo executável do jogo pela página do jogo no itch.io. Após baixado, extraia os arquivos do `".zip"` baixado e execute o arquivo `"TCP.exe"`.

Jogando

A seguir temos algumas informações sobre o funcionamento do jogo.

- O jogo foi criado com o intuito de promover coop-local, ou seja, não existe a opção de jogar contra o computador.
- O primeiro turno é sempre do jogador branco (metade inferior da tela).
- Para mover uma peça basta clicar com o botão esquerdo em cima da peça desejada e depois clicar em uma posição válida.
- Aperte o botão esquerdo do mouse para cancelar um movimento.
- É possível alterar a duração da partida através do menu de opções, possuindo um limite de até 60 minutos.

[2] https://www.gnu.org/software/make/
[3] Requer no mínimo OpenJDK 17.0.15

- Clicar no botão de desistir acabará com a partida e o jogador que estava com o turno ativo no momento é declarado perdedor.
- Clicar no botão de empate acabará a partida imediatamente em empate. **NÃO HÁ UMA TELA DE ”TEM CERTEZA?”!**
- Quando o tempo de um jogador acabar, ele é considerado perdedor e o jogo acaba.