Relatório do Projeto 2- SCC 0504

Integrantes do Grupo:
Andrey Lucas Esteves Garcia **NroUsp:** 10734290
Gabriel Muniz Morão **NroUsp:** 7236785

O que foi feito:

Foi utilizado como base para o projeto o código do primeiro projeto da disciplina, um jogo de Xadrez na linguagem JAVA. No primeiro projeto fizemos as peças herdeiras como peão, cavalo, etc. Fizemos o movimento de cada uma (específicos do tipo), se preocupando se eram válidos ou não, além de turnos entre dois jogadores,e ataque peças (“comer”).. O segundo projeto, assim como o primeiro, foi desenvolvido em ide NetBeans 10.0 e NetBeans 8.2.

A princípio, um pequeno menu em texto foi apresentado na main. Nele são exibidas mensagens de “Bem vindo” e sobre qual opção de jogo é desejado, sendo possível começar um novo jogo (“0”) ou carregar o último jogo salvo (“1”). Caso seja selecionado a opção 0 (“Novo Jogo”), o jogo carrega normalmente. Já caso a opção 1 for selecionada (“Carregar Jogo”), caso haja um arquivo de “save”, ele será carregado e o jogo continuará de onde parou. As imagens abaixo mostram as mensagens no terminal.

```
run:
Bem Vindo ao Xadrez !
Para iniciar um novo jogo, digite 0.
Para carregar o último jogo, digite 1.
0
Começando nova Partida
```

Figura 01. Exemplo de menu quando se quer novo jogo

```
run:
Bem Vindo ao Xadrez !
Para iniciar um novo jogo, digite 0.
Para carregar o último jogo, digite 1.
1
Abrindo o jogo...
```

Figura 02. Exemplo de menu quando se quer carregar último jogo

Para a opção 1, caso não haja jogos salvos, é apresentado no terminal a mensagem de erro “Não há jogos salvos” além da mensagem de erro do programa identificando que “O sistema não pode encontrar o arquivo especificado”. Aqui já vemos o conceito de **exceptions** sendo utilizado, com a aplicação de “try, catch” ao código de abertura e carregamento do jogo.Ele é observado com mais afinco ao testarmos os movimentos das peças. No clique, que controla o movimento, temos o try-catch referido. o movimento leva ao método de movimento, que cada peça sobrescreve e caso o vimento não se enquadre nos possíveis para tal peça, um “MovimentoException” é jogado invocando a mensagem de erro no movimento, com a sobrescrita do método toString.

A segunda funcionalidade criada neste projeto, foi o autosave. Para tal, foi criada uma classe “AutoSave” que estende **thread**, que cuida do salvamento. Inicialmente iríamos aplicar o salvamento a cada determinado período de tempo, porém caso o programa seja fechado (falta de energia, erro no computador,etc) antes do período de um novo salvamento, pode-se perder algum movimento realizado. Por tanto, o salvamento automático foi implementado a medida em que se move ou ataca com as peças. Assim, garante-se que após uma importante ação no jogo, o mesmo sempre salvará, minimizando os riscos de algo se perder. Assim, ao se movimentar as peças ou atacar outras, ao fim do movimento é chamado a funcionalidade “salva_jogo” que tenta salvar o jogo utilizando da biblioteca “java.io.FileOutputStream” e “java.io.ObjectOutputStream”. Para tal salvamento acontecer, foi necessária utilizar a biblioteca “java.io.Serializable”, acrescentando “Serializable” ao “implements” das classes necessárias. Começamos assim a observar a utilização dos conceitos de **thread** e **serialização** no projeto. O conceito de **exceptions** é novamente observado aqui ao tentar salvar o jogo, com “try, catch”. Para melhores desempenho e implementação, foram serializados o model, e partes do controller, em 3 arquivos separados. Um para o model, um para a matriz de verificação e um para o jogador o qual possui a vez. Ambos são salvos no mesmo método, que é chamado apenas uma vez passando os três parâmetros.

A terceira funcionalidade também utiliza do conceito de **thread**. Nela contamos o tempo de jogo e de cada jogador, através da nova classe “Relógio”. A classe apresenta atributos de segundos e minutos além de uma flag para saber se está ligado ou não. Também possui funções para pegar e mostrar tais variáveis. Por fim, seu método run com @override, que funciona acrescendo as variáveis em 1 e então dormindo por 1000 milésimos de segundo, utilizando da funcionalidade “.sleep”. A flag funciona dividindo o tempo de cada jogador, onde apenas na vez do jogador 1, o relógio 1 conta o tempo enquanto o 2 fica parado. Ao passar a vez para o jogador 2, o relógio 1 para e o 2 começa a contar. Há ainda o relógio principal que conta o tempo total de partida. As labels para cada relógio foram acrescentadas e enviadas para o topo do tabuleiro, deixando mais agradável sua visualização. Dentro da classe “Tabuleiro.java” temos a funcionalidade que atualiza os relógios com a passagem do tempo, cuidado para que sua visualização no painel siga perfeitamente o modelo de “00:00:00” (HH:MM:SS). Por fim, na mesma classe, a cada atualização, o método repaint é chamado para atualizar o painel com as novas informações. Os relógios não são salvos no salvamento automático, começando novamente em uma partida carregada. As imagens abaixo mostram o relógio principal e o relógio do jogador 1 (peças brancas) e do jogador 2 (peças pretas) em funcionamento.

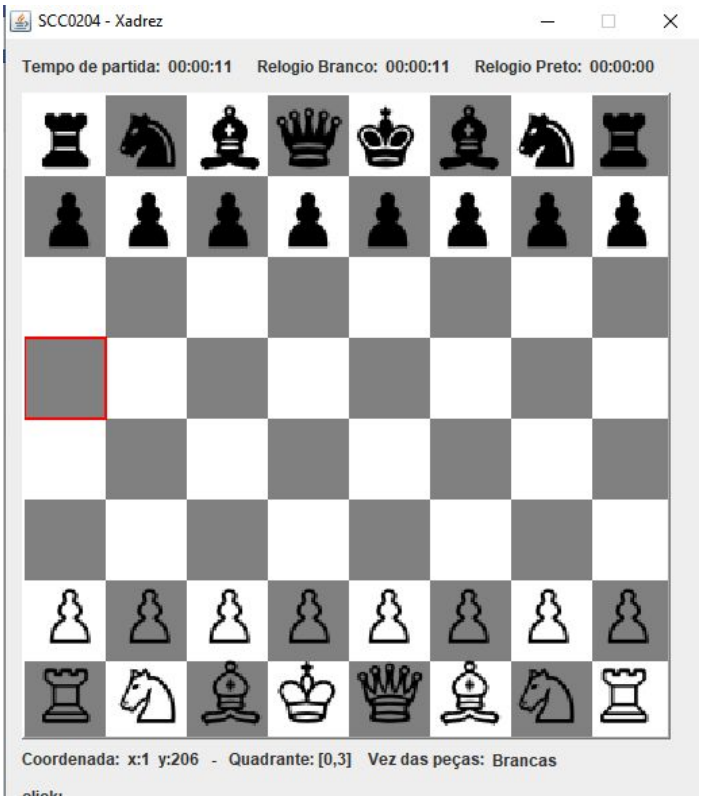

Figura 03. Exemplo de relógio de partida e relógio das peças brancas em funcionamento

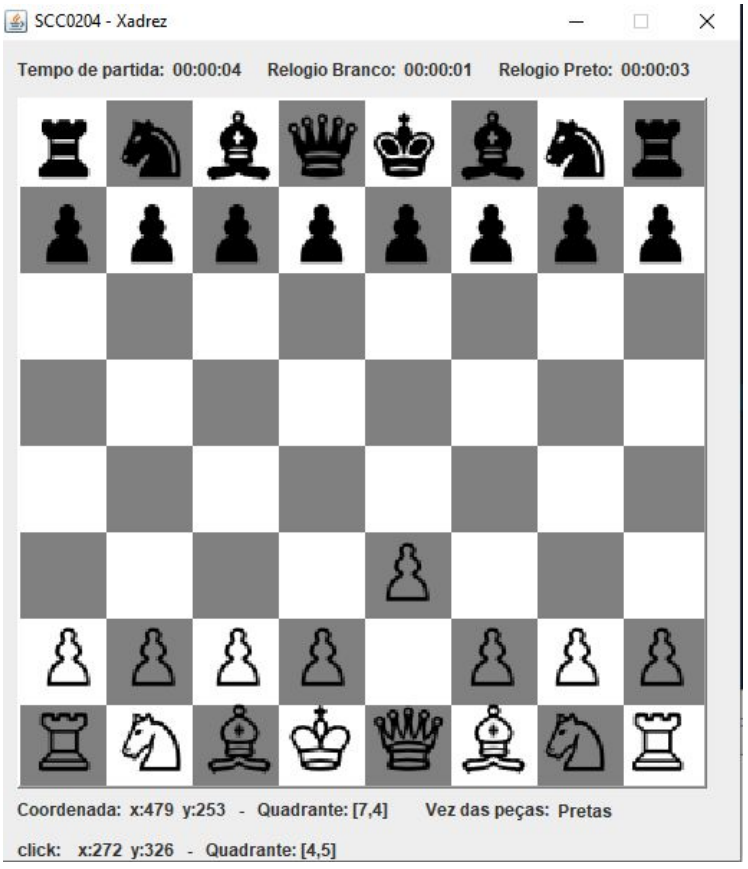

Figura 04. Exemplo de relógio de partida e relógio das peças brancas em funcionamento

Em um futuro projeto utilizando o jogo, provavelmente em outras disciplinas, pode-se pensar no salvamento dos relógios também para que se possa ter mais exatidão de tempo de partida, novas exceções e partilhamento em rede.