Universidade Federal de Minas Gerais (UFMG)

Aynoa Souza Jorgino
Emmanuel Van Petten de Oliveira
Leticia Ribeiro Miranda
Luiza Barbosa de Andrade
Victor de Paula Pinto Ferreira

Trabalho prático da matéria Programação e Desenvolvimento de Software II

Belo Horizonte, 2024

I. PROBLEMA E VISÃO GERAL

Neste trabalho prático, tínhamos o desafio de desenvolver um programa contendo dois jogos, ‘Lig4’ e ‘Reversi’, utilizando as técnicas e boas práticas que aprendemos durante a disciplina de Programação e Desenvolvimento de Software II.

Para este problema, o nosso grupo resolveu implementar os jogos utilizando uma interface gráfica chamada SFML. Nela, temos diversas ferramentas interessantes que deixam o programa visualmente mais atrativo.

A. COMPILAÇÃO DO JOGO

O jogo foi desenvolvido utilizando o sistema operacional Windows. O Makefile está todo configurado para atender as especificidades desse sistema. Logo, ao tentar rodá-lo no Linux, provavelmente não irá funcionar.

Para acessar o jogo utilizando o Windows, será necessário baixar a biblioteca gráfica SFML, através do link https://www.sfml-dev.org/download/sfml/2.6.1/. Caso use o VSCode ou qualquer outro tipo de IDE sem ser o Visual C++, baixe o SFML no campo *“GCC 13.1.0 MinGW (SEH) - 64-bit”*. Coloque a pasta da biblioteca SFML diretamente em ***C:*** (não mova para nenhuma subpasta). Além disso, você deverá baixar a versão do GCC que o SFML disponibiliza neste mesmo site, no campo que está destacado em vermelho, escrito *“WinLibs MSVCRT 13.1.0”*.

Por fim, não se esqueça de configurar a variável de ambiente PATH no Windows, adicionando o caminho da pasta onde você colocou a biblioteca SFML.

B. FUNCIONAMENTO DO JOGO

De maneira generalizada, nosso programa começa em um ‘Menu’, onde temos os seguintes botões e seus respectivos objetivos:

Login Jogador 1 e Login Jogador 2: Para acessar os jogos, os usuários deverão inserir dois apelidos já cadastrados. Após inseri-los no campo correspondente, cada jogador deverá pressionar a tecla **Enter**. Ambos os jogadores precisam pressionar Enter em seus campos, para que os jogos Lig4 e Reversi sejam liberados.

- No terminal, ao inserir o apelido e pressionar **Enter**, será impresso um dos dois tipos de mensagem:
    - 1º - “Jogador foi logado com sucesso!”, caso o jogador já tenha realizado o cadastro.
    - 2º - “Erro: Apelido não encontrado no histórico - Jogador não foi logado com sucesso!”, caso o jogador não tenha realizado o cadastro.

Cadastro: No cadastro, o jogador deverá inserir seu nome e um apelido que não esteja na base de dados. Após isso, basta clicar no botão ‘confirma’ e o cadastro é realizado.

- No terminal, ao tentar cadastrar um novo jogador, será impresso um dos dois tipos de mensagem:
    - 1º - “Cadastro criado com sucesso!”, caso o apelido não esteja cadastrado ainda.
    - 2º - “Erro: Apelido já existente no histórico.”, caso o apelido já exista no banco de dados.

Lista de Jogadores: Esse botão lista todos os jogos e suas respectivas estatísticas.

- Além da lista de jogadores aparecer na tela, será mostrado no terminal também.

Excluir Conta: Já neste botão, precisamos inserir um apelido que esteja cadastrado no campo e apertar o botão de ‘Excluir Conta’.

- No terminal, ao tentar excluir um jogador já existente, será impresso um dos dois tipos de mensagem:
    - 1º - “Conta excluída com sucesso!”, caso o apelido esteja cadastrado.
    - 2º - “Erro: Apelido não encontrado no histórico - Conta não existente!”, caso o apelido não exista no banco de dados.

Estatísticas: Por último, neste campo o usuário precisará inserir seu apelido (cadastrado), clicar em Pesquisa e verificar suas estatísticas nas partidas de Reversi ou Lig4.

- No terminal, ao tentar acessar as estatísticas de um jogador específico, caso o mesmo não tenha cadastro na base de dados, será impresso a seguinte mensagem:
    - “Erro: Apelido não encontrado no histórico - Erro! Conta não existente!”

Feito o login, os botões para entrar no jogo Reversi ou Lig4 ficará disponível, o jogador terá a opção de jogá-los, mas, durante a sessão, caso tenha o interesse, ele poderá desistir da partida clicando no botão ‘desistir’.

No Lig4, durante a partida, será impresso no terminal de quem é a vez do jogador, sua respectiva cor e a coluna que ele inseriu a peça. Já no Reversi, será mostrado no terminal também, a vez do jogador, sua respectiva cor, a quantidade de peças que cada um tem e, caso cliquem é um campo indisponível, será mostrado a mensagem de “Jogada inválida". Tente novamente.”

Ao final, aparecerá a tela de encerramento do jogo, contendo o nome do vencedor e os botões de ‘Restart’ ou ‘Menu’. Caso a pessoa não deseje jogar mais, ela simplesmente pode fechar a janela.

II. MODELAGEM E PRÁTICA

Inicialmente, o grupo se reuniu para desenvolver o User Stories e os CRCs (em anexo), bem como seus colaboradores e suas responsabilidades. Após isso, decidimos que cada tarefa seria realizada por dois membros do grupo, para todos participarem praticamente em conjunto da produção do jogo.

Assim, as tarefas desenvolvidas ficaram da seguinte forma:

- As classes referente às interações dos botões, ao desenho das telas, o desenho do tabuleiro e às dinâmicas de troca de páginas, foram de responsabilidade dos membros Emmanuel e Leticia.
- As classes responsáveis por cadastrar o jogador, manter o histórico armazenado, conectar com a planilha.csv, exibir as estatísticas e conectar os jogadores aos jogos, foram de responsabilidade dos membros Luiza e Victor.
- Já a classe abstrata e a herança das classes Lig4 e Reversi, foi desenvolvida por Leticia. Além disso, a lógica do funcionamento dos jogos foi feita pelos membros Emmanuel e Luiza.
- A interface gráfica do jogo foi desenvolvida pelo Emmanuel, enquanto a tela final do jogo foi elaborada pela Letícia.
- Por fim, os tratamentos de exceções foram feitos pela Letícia, enquanto os testes foram desenvolvidos pela Aynoa.

Os detalhes da implementação se encontram na pasta documento\html, no arquivo chamado index.html.

III. TRATAMENTO DE EXCEÇÕES E TESTES

As classes CampoTexto, Interacao, Jogador, Telas e Wallpaper são responsáveis por operações de entrada e leitura de arquivos. Por isso, cada uma delas inclui mecanismos para o tratamento de exceções.

Os testes foram aplicados em Jogador, visto que eles garantem que todas as funcionalidades da classe Jogador, como métodos para registrar, atualizar e exibir informações dos jogadores, estejam funcionando conforme o esperado. Isso ajuda a identificar e corrigir erros e garantir que a classe cumpra seu propósito.

Além disso, a classe Jogador geralmente implementa regras e lógicas específicas, como validações de movimentos, regras de pontuação e outras funcionalidades relacionadas ao comportamento do jogador. Testar essas regras assegura que o jogador segue as regras corretamente e que o jogo se comporta como esperado.

IV. DIFICULDADES ENCONTRADAS

Desenvolver um código padrão e aplicar boas práticas de programação foi um dos nossos desafios, especialmente porque nosso projeto envolveu diversos arquivos e um ambiente com diferentes pessoas. Além disso, ordenar e gerenciar a execução de eventos foi um ponto complicado, mas importante, visto que era a primeira vez que desenvolvemos um programa com duas funcionalidades distintas.

Além disso, enfrentamos grandes desafios no uso do Git, mas, após nos acostumarmos, reconhecemos a importância dessa ferramenta para o trabalho colaborativo com múltiplos arquivos. Outro desafio significativo, foi a gestão de estados e a implementação da lógica do jogo para o Lig4 e Reversi. Garantir que as regras do jogo fossem corretamente aplicadas e que a interface refletisse o estado atual do jogo era um ponto muito importante e que exigiu muito dos membros.

Em relação a alocação dinâmica de memória, em muitas momentos esquecemos de desalocar a memória, o que eventualmente se tornava um problema, mas conseguimos melhorar nesse aspecto. Acredito que, para um upgrade do código, seria interessante usar os smart pointers. Outro ponto foram os testes, enfrentamos dificuldade justamente por não estarmos habituados com ele e termos pouco tempo para estudá-los de fato. Essa etapa será melhorada em um upgrade também.

Por último, em relação à compilação do jogo, foi uma dificuldade garantir que o programa funcionasse no Windows, principalmente porque era a primeira vez que todos os membros utilizavam a biblioteca SFML e estamos buscando assegurar que o jogo funcione perfeitamente em todos os sistemas operacionais.

V. EXTRAS

Um dos maiores diferenciais do nosso trabalho foi a implementação da interface gráfica, que desempenhou um papel importante na experiência do usuário. A criação de uma interface gráfica não se limitou apenas à exibição básica dos jogos, mas envolveu um cuidadoso desenvolvimento de cada detalhe interativo. A interface foi projetada para exibir estatísticas de forma clara e acessível, permitindo que os jogadores visualizassem informações relevantes, como desempenho e histórico de jogos.

Além disso, o sistema de cadastro foi integrado de maneira intuitiva, facilitando o processo de criação e gerenciamento de perfis de jogadores. A interação com a interface foi aprimorada através de respostas visuais e feedback em tempo real, como alterações dinâmicas ao clicar em botões ou ao inserir dados corretos.

VI. ANEXO I (USER STORIES)

1) Acesso ao Menu Inicial

Descrição: Como jogador, acessar o menu inicial, quero poder logar ou cadastrar e consultar jogadores cadastrados.

Critérios de aceitação:

- Apresentar uma interface (tela inicial) que permite ao jogador inicializar a partida.
- Capacitar a utilização do mouse para interação do jogador com a tela inicial.
- Criar uma tela de login.
- Criar um botão “cadastro”, para caso o jogador não tenha cadastro ainda.
- Não permitir os usuários cadastrarem apelidos que já existem.
- Criar um botão para acessar os jogadores cadastrados.
- Criar um botão para excluir a conta.
- Criar botão de confirmar excluir a conta.
- Criar um botão para acessar minhas estatísticas.

2) Inicializar uma partida

Descrição: Inicializar um dos jogos para jogar uma partida.

Critérios de aceitação:

- Criar botão para a opção de jogo lig4.
- Criar botão para a opção de jogo reversi.
- Permitir selecionar o jogo quando dois apelidos forem válidos.
- Criar um botão para voltar para o menu inicial.

3) Realizar uma jogada

Descrição: Como jogador, quero poder fazer uma jogada.

Critérios de aceitação:

- Tornar as peças móveis através do mouse.
- Não permitir que a peça ultrapasse o tabuleiro.
- Não permitir jogadas que infrinjam as regras do jogo.
- Ver as atualizações da minha jogada.

4) Verificar informações da Partida

Descrição: Como jogador, quero, durante a partida, observar o tempo, a quantidade de peças, verificar a possibilidade de vitória ou empate e desistir da partida.

Critérios de aceitação:

- Criar uma barra superior na tela, contendo o tempo da partida, a quantidade de movimentos de cada jogador e qual jogador está jogando.
- Criar um botão para desistir da partida.

5) Finalizar a partida

Descrição: Como jogador, quero poder derrotar o adversário para finalizar a partida e ser o vencedor.

Critérios de aceitação:

- Finalizar a partida quando o critério de derrota for alcançado.

6) Tela de fim de jogo

Descrição: Como jogador, quero ver quem venceu a partida, ter acesso às minhas estatísticas no final do jogo e do adversário. Quero poder visualizar a pontuação do jogo e compará-la com o record.

Critérios de aceitação:

- Anunciar na tela final o jogador que venceu.
- Exibir as estatísticas dos dois jogadores (número de vitórias, empate e derrotas).
- Exibir a pontuação do jogo e exibir o record.

7) Opções pós-partida

Descrição: Como jogador, quero voltar para o menu, começar nova partida ou sair do jogo.

Critérios de aceitação:

- Criar um botão para voltar ao menu inicial.
- Criar um botão para jogar novamente.
- Criar um botão para sair do jogo.

Ideias para mais extras:

8) Acessar instruções

Descrição: Como jogador, quero acessar as instruções de cada jogo.

Critérios de aceitação:

- Criar um botão para acessar a seção de ajuda com informações sobre como jogar.

9) Ajustar configurações do jogo

Descrição: Como jogador, quero ajustar as configurações do jogo, como volume, dificuldade e tema.

Critérios de aceitação:

- Criar um botão para acessar as configurações do jogo.
- Permitir ajuste de volume.
- Permitir seleção de dificuldade do jogo.
- Permitir mudança de tema/estilo visual do jogo.

VII. ANEXO II (CARTÕES CRCs)

Classe: Botão

Atributos:

- largura
- altura
- posição (x, y)
- cor
- corHover
- retangulo
- circulo
- isCirculo
- texto
- fonte

- text
- corFonte
- tamanhoFonte
- windowLargura
- windowAltura

Método:

- criarBotoes ( )
- criarCampo ( )
- getDimensão ( )
- getPosição ( )
- getCor ( )
- mudarCor ( )
- getRetangulo()
- getCirculo()
- setCor()
- setCorHover()
- setTamanhoFonte()
- desenhar()
- passouMouse()
- foiClicado()

Classe: Historico

Atributos:

- nome_arquivo
- cabecalho

Método:

- Historico()
- excluirLinha()
- Editar()
- criarLinha()
- acessarDados()
- acessarDados()

- acessarDados()
- addEstatistica()
- getNomeArquivo()

Colaborações:

- Pode colaborar com `Jogador` para armazenar e manipular os dados históricos dos jogadores.

Classe: Jogador

Atributos:

- nome
- apelido
- vitorias_reversi
- derrotas_reversi
- empates_reversi
- vitorias_lig4
- derrotas_lig4
- empates_lig4
- nome_jogo
- histórico

Métodos:

- Jogador()
- criarCadastro ( )
- setNome()
- setApelido( )
- getNome ( )
- getApelido ( )
- setResultado( )
- getResultado ( )
- vencedor()
- empate()
- excluirConta ()
- existeConta()
- getVitoriasReversi()

- getDerrotasReversi()
- getEmpatesReversi()
- getDerrotasLig4()
- getEmpatesLig4()

Colaborações:

- Colabora com Historico para manter registros dos jogos.
- Interage com jogo_principal para realizar as ações específicas do jogador.

Classe abstrata: Jogo

Métodos:

- condicao_vitoria( )
- jogada_valida( )
- get_jogador_atual( )
- get_status

Colaborações:

- Colabora com Jogador para gerenciar as ações e status dos jogadores durante o jogo.

Classe: JogoLig4

Herança:

- Herda de: Jogo

Atributos:

- window
- fonte
- evento
- origemX
- origemY
- qtd_celulaX
- qtd_celulaY
- tamanho_celula
- borda
- iocupado
- jocupado

- animando
- jogador1
- jogador2
- botaoApelido
- botaoVoltar
- fimDeJogo
- circulo
- tabuleiroLIG4
- jogadorAtual
- nomeVencedor

Métodos:

- desenharJogo()
- LimpaTabuleiro()
- setJogadores()
- FazJogada
- getNomeVencedor()
- verificaCondicaoVitoria()
- anima()

Colaborações:

- Colabora com Tabuleiro para gerenciar o estado do jogo Lig4.
- Colabora com Interacao para desenhar o jogo Lig4
- Colabora com Jogador para pegar os dados dos jogadores

Classe: JogoReversi

Herança:

- Herda de: Jogo

Atributos:

- jogador1
- jogador2
- jogadorAtual
- window
- fonte
- evento
- borda

- origemX
- origemY
- qtd_celulaX
- qtd_celulaY
- tamanho_celula
- ganhador

Métodos:

- contaPeca()
- desenharJogo()
- acao()
- TemJogadasValida()
- LimpaTabuleiro()
- setJogadores()
- FazJogada()
- getNomeVencedor()
- verificaCondicaoVitoria()
- jogada_valida()
- VerificaJogadaDireca()

Colaborações:

- Colabora com Tabuleiro para gerenciar o estado do jogo Lig4.
- Colabora com Jogador para pegar os dados dos jogadores

Classe: Celula

Atributos:

- estado
- posicaoX
- posicaoY
- _tamanho_celula
- isCirculo
- cor

Métodos:

- Celula()
- setEstado()
- getEstado()

Classe: Tabuleiro

Atributos:

- qtd_celulaX
- qtd_celulaY
- tamanho_celula
- origemX
- origemY
- borda
- indice_i
- indice_j
- jogadorAtual
- jocupado
- get_qtd_celulaX
- get_qtd_celulaY

Métodos:

- matriz()
- slots()
- deuClique()
- get_celula_status()
- set_celula_status()
- poePeca()
- acao()
- anima()
- desenhar()

Colaborações:

- Colabora com botões, pois as peças são botões também

Classe: Wallpaper

Atributos:

- textura
- sprite

Métodos:

- Wallpaper()
- desenhar()
- redimensionar()

Classe: Telas

Atributos:

- window
- fonte
- evento
- botaoJogador1
- botaoJogador2

Métodos:

- desenhaTela()

Colaborações: colabora com Campotexto, Wallpaper, Interacao, Historico, Jogador para desenhar as telas dos jogos