Especificações do projeto

O objetivo deste projeto é implementar um jogo de tabuleiro baseado no famoso Jogo de Damas utilizando a biblioteca gráfica Gloss.

O jogo se baseia num confronto entre dois jogadores, peças pretas contra peças brancas. Ao iniciar a aplicação, a interface gráfica já carregará o tabuleiro e as peças, sendo o jogador das peças brancas responsável pelo primeiro movimento, e na sequência o jogador das peças pretas. O jogo segue alternando entre os jogadores, até que um seja o vencedor (eliminar todos as peças do adversário) ou ambos possuam apenas uma peça, nesse caso é considerado empate.

- Antes de compilar: baixar o ghci, stack e a biblioteca gloss. Clonar o repositório e abrir o terminal no diretório do projeto.
- Compilar: executar o comando “cd minigames” e depois stack build.
- Rodar: executar o comando “stack run” ou “stack exec minigames”
- Rodar testes automatizados: executar o comando “stack test”

Na primeira etapa do desenvolvimento, configuramos os ambiente, conforme explicado na parte “antes de compilar”. Depois disso, iniciamos desenvolvendo a estrutura do tabuleiro, do jogo, os controladores de renderização da tela e de jogo (regras de movimentação, início do jogo e fim de jogo), nos baseando na implementação do jogo “Tic-Tac-Toe” disponível em https://www.youtube.com/watch?list=PLguYJK7ydFE7KBfUplAk_qLp--xG6gw3R&time_continue=383&v=VxLvaHpAK-U.

A maior dificuldade da dupla foi no momento de criar as regras de movimentação, que foi dividida em etapas. Primeiro, criamos a função responsável por criar, na casa clicada, uma peça de acordo com o jogador. Depois, ao clicar numa peça já existente, alteramos a função de criação para no mesmo momento de criar a peça na outra posição, excluir a peça que foi selecionada previamente. Para que isso fosse possível, foi preciso criar dois estados para tratar o evento de clique, onde um corresponde ao primeiro evento de click (estado Running) e outro corresponde ao segundo evento de click (estado Move). O Running é responsável por selecionar a peça, e o Move por movê-la e caso necessário remover peças que foram comidas. Com esses dois momentos, podemos ter mais controle do jogo, e fazer as devidas validações de movimentos. Se o estado do jogo for Running, fazemos a validação da peça escolhida e verificamos se há movimentações possíveis para ela, caso existam, então mudamos o estado do jogo para Move e trocamos a cor da peça selecionada, caso contrário o estado permanece o mesmo até que seja selecionada uma peça com movimentos válidos. No estado de jogo Move, verificamos se o segundo clique é uma casa válida e se faz sentido a peça se movimentar naquela posição, pois cada peça só pode andar uma cada por vez, na diagonal, sem andar pra trás - a não ser que seja para comer uma peça adversária - e deve se movimentar para uma casa vazia, caso ela se mova apenas uma casa, então o estado do jogo voltará para Running, caso contrário avaliamos se é possível comer outra peça adversária, caso seja possível, mantemos o estado Move, caso contrário voltamos ao Running.

Em todo evento de primeiro click avaliamos se o jogo acabou (GameOver (Just Player)) ou empatou (GameOver Nothing), que é o nosso terceiro estado de jogo. Para saber se o jogo acabou avaliamos se a quantidade de peças presentes no tabuleiro de algum dos jogadores (Black ou White) é igual a zero, o jogador que tiver suas peças zeradas, perde o jogo, adotamos que caso ambos tenham uma única peça então temos um empate. Quando o jogo chegar no estado de GameOver o fundo da tela terá cor laranja e o próximo clique reiniciará o jogo, para que seja possível jogar novamente.

Nossos testes automatizados contém as seguintes funções:

1. playerTest: Realiza a verificação da existência de algumas peças nas posições corretas no tabuleiro;
2. peçasIniciaisTest: Realiza a verificação da lista que possui posição e player para popular o tabuleiro;
3. initialGameBoardTest: Realiza a verificação do tabuleiro inicial;
4. initialGameTest: Realiza a verificação do estado inicial do game;
5. isCoordCorrectTest: Realiza a verificação para saber se a função que valida se as coordenadas recebidas estão dentro do tabuleiro;
6. inversePlayerTest: Realiza a verificação para saber se a função inverte o player recebido;
7. nextPlayerTest: Realiza a verificação da função que retorna o próximo jogador;
8. possibleMovesWhiteTest: Realiza a verificação da função que retorna se aquela peça branca possui movimentos possiveis;
9. possibleMovesBlackTest: Realiza a verificação da função que retorna se aquela peça preta possui movimentos possíveis;
10. playerTurnTest: Realiza a verificação para saber se o click e o jogador atual podem prosseguir com a pedra selecionada;
11. moveRockBlackTest: Realiza a verificação da função que trata o segundo click do jogador preto;
12. moveRockWhiteTest: Realiza a verificação da função que trata o segundo click do jogador branco;
13. gameStateAfterMoveTest: Realiza a verificação da função que retorna o novo estado do game após uma movimentação de uma peça.