Relatório de decisão de projeto do trabalho final da cadeira de Projeto Detalhado de Software de 2019.2.

Equipe:
Francisco Davi Rodrigues Xavier - 416738
Javel Queiroz Freitas - 414943
João Elias Lima Viana - 427532

Repositório no GitHub

Data de Criação: 04/12/2019
Data de Última Modificação: 04/12/2019

Decisões de Projeto quanto à padrões:

1. Strategy Pattern: Nós decidimos usar este padrão nas casas para ter mais flexibilidade no uso de funções que elas teriam em comum. Em algumas casas (CasasEspeciais) quando o jogador é alocado para ela é ativado um efeito especial, o Strategy Pattern nos ajudou a usar métodos parecidos de diferentes classes para calcular o destino de um jogador caso o mesmo pare em uma dessas casas.
2. Builder Pattern: A decisão de como o builder seria aplicado no código foi uma das mais demoradas, atualmente o builder está sendo utilizado para construir o Jogo por conta de ele ser uma classe complexa que é construída em partes.
3. Observer Pattern: Deparamo-nos com um impasse a respeito de como manter a interface gráfica atualizada com as informações do Jogo, então, decidimos usar o padrão Observer, onde o Jogo seria um objeto observável e a interface gráfica do jogo um observador. Desse modo o Jogo não necessitaria conhecer a interface para notificá-la de eventos ocorridos dentro do mesmo, pois ele iria conhecer só uma abstração de um observador qualquer.
4. Iterator Pattern: Foi usado para encapsular a coleção de jogadores e a lista de casas do mapa.

Outras Decisões de Projeto:

Movimentação ou Iterator de movimento?

A abstração do movimento do jogador foi uma questão discutida desde o começo do projeto, no final a melhor solução para o nosso problema foi implementar as duas classes.

Pouca ou Muita divisão por pacotes?

O código possui mais de 30 classes, a partir de um momento a idéia de ter mais pacotes apareceu, o argumento contra foi: os imports ficariam mais complexos e a divisão diminuiria a visibilidade, o argumento a favor foi: uma melhor divisão tornaria o projeto mais organizado, assim facilitando sua escalabilidade. O argumento vencedor foi o segundo, logo dividimos os pacotes.

Interfaces e Classes abstratas?

Em certas partes do sistema vimos a necessidade de haver uma interface para ser implementada por alguma classe, por exemplo, a interface Sorteável torna a introdução de outros tipos de dado mais fácil, essa mesma estratégia foi utilizada em diversas partes do sistema para torná-lo o mais expansível possível. Classes abstratas, por exemplo nas casas, foram utilizadas para evitar código duplicado.