Resumo do Processo de Refatoração do Gilded Rose

O objetivo principal do processo da refatoração foi conseguir tornar o sistema Gilded Rose mais modular, limpo e fácil de manter, sem alterar o comportamento esperado dos itens. Na versão original, toda a lógica da atualização estava concentrada em um apenas um único método, o que dificultava a compreensão e a inclusão de novas categorias de produtos.

A principal mudança foi a adoção de um padrão de estratégia (Strategy Pattern). Cada tipo de item agora tem uma função para atualizar que foi criada em novos arquivos chamados ItemTypes.c e ItemTypes.h. Essas funções são atribuídas a cada item usando ponteiros de função. Isso faz com que o código escolha a regra de atualização correta automaticamente. Assim, o arquivo GildedRose.c só fica responsável por criar, mostrar os itens e passar a atualização de qualidade para a função certa. Como resultado, o código ficou mais organizado, mais fácil de manter e de fazer novas implementações. Ele também evita repetir lógica. Por exemplo, para incluir o novo tipo Conjured, basta criar a função update_conjured e adicionar uma linha na seleção de estratégias.

Dificuldades Encontradas:

A primeira foi conflito de tipos (conflicting types): Isso aconteceu porque havia declarações iguais e diferentes da estrutura Item e das funções em arquivos de headers diferentes. Para resolver, garantimos que a definição de Item aparecesse uma única vez. Também verificamos que todos os arquivos incluíssem os headers na ordem correta.

A segunda dificuldade foi a organização dos arquivos no CodeBlocks. Havia confusão sobre qual arquivo era o ponto de entrada do programa. Isso aconteceu porque o projeto tinha um arquivo main.c padrão do CodeBlocks. Para resolver, definimos GildedRoseTextTests.c como o arquivo principal de testes. Os outros arquivos passaram a ser módulos auxiliares.

A terceira foi a preservação das regras originais, foi necessário garantir que regras específicas, como o aumento da qualidade do “Aged Brie” e o comportamento de queda rápida do “Backstage Passes”, continuassem funcionando. Foram feitos

vários testes comparando a saída com a versão original. Isso ajudou a confirmar que o comportamento não mudou.
A quarta dificuldade foi a compatibilidade de funções: Alguns avisos de compilação apareceram porque o protótipo da função print_item no teste não era igual ao do módulo. A solução foi padronizar todas as assinaturas para evitar diferenças.

Lições aprendidas

1. Modularizar o código o torna mais simples de entender e de mudar. Separar regras de negócio em funções ajuda a diminuir a dependência entre partes do código e torna mais claro.
2. Cuidar dos headers em C é essencial: um erro na ordem ou uma duplicata pode causar problemas difíceis de resolver.
3. Ponteiros de função são uma ferramenta forte para fazer diferentes comportamentos, mesmo sem usar classes ou herança.
4. Manter testes automatizados ou saídas comparativas é importante ao fazer mudanças no código. Assim, você sabe que a lógica do programa não muda.
5. Entender como o ambiente de desenvolvimento (IDE) compila e organiza os arquivos é fundamental. Isso vale especialmente para projetos com muitos módulos ou arquivos.

Resumindo, a refatoração do Gilded Rose foi uma oportunidade para usar conceitos de design simples, dividir o código em partes menores e seguir boas práticas em C. O código ficou mais organizado. Assim, fica mais fácil fazer mudanças no sistema no futuro.