Relatório

Programação Orientada a Objetos - 2024/2025

Trabalho Prático

Simulador de Viagens no Deserto

Alunos

Luccas Galvan – 2022100410

Introdução

Este relatório apresenta a análise detalhada do código desenvolvido para o simulador de viagens no deserto, destacando as principais escolhas de design, estrutura do código, heranças, uso de smart pointers e funcionalidades principais de cada módulo implementado.

1. Estrutura Geral do Projeto

O projeto é composto por vários módulos organizados em arquivos .cpp e .h, cada um responsável por funcionalidades específicas. A seguir, detalharemos cada componente do sistema:

2. Arquivo Buffer

Função:

O **Buffer** é responsável pela representação do mapa do jogo, lidando com a exibição e manipulação gráfica dos elementos no mapa.

Principais Métodos:

- **carregarMapa**: Carrega um mapa a partir de um arquivo de texto, inicializando as dimensões e as configurações iniciais do jogo.
- **imprimirChar** e **imprimirInt**: Atualizam visualmente o conteúdo no buffer com caracteres ou números.
- **getAllCaravans, getAllCities e getAllItems**: Recuperam as coordenadas de elementos específicos no mapa.
- **ajustarCoordenada**: Trata coordenadas fora dos limites, permitindo movimentação circular no mapa.

Destaques:

O uso de um array dinâmico para representar o mapa foi eficiente, e os métodos oferecem flexibilidade para manipulação dos elementos.

3. Arquivo Caravana

Função:

Gerencia os diferentes tipos de caravanas (comercial, militar, secreta (fantasma) e bárbara) e suas operações, como movimento, batalha e interações com o mapa.

Herança e Polimorfismo:

A classe base **Caravana** utiliza herança para definir diferentes comportamentos para os tipos de caravanas:

- **CComercial**: Focada no transporte de mercadorias.
- **CMilitar**: Especializada em combate.
- **CSecreta**: Caravana fantasma que assombra bárbaros.
- **CBárbara**: Controlada automaticamente, representando ameaças.

Cada classe derivada implementa métodos virtuais como:

- **movimentoAutonomo()**: Movimenta a caravana autonomamente.
- **consumirAgua()**: Gerencia o consumo de recursos.
- **tempestadeHandler()**: Lida com as tempestades de areia.

Uso de Smart Pointers:

As caravanas são manipuladas por **`shared pointers`** para garantir gerenciamento seguro de memória, permitindo múltiplas referências e controle automático de desalocação, assim evitando a inconveniência de dangling pointers. Os shared pointers foram escolhidos pois permitem que tanto a cidade quanto o mapa compartilhem o mesmo estado sem cópias desnecessárias.

4. Arquivo Cidade

Função:

Gerencia as operações relacionadas às cidades, como compra e venda de mercadorias, contratação de tripulantes e aquisição de novas caravanas.

Principais Métodos:

- **aceitaCaravana()**: Aceita uma caravana na cidade e atualiza seus recursos.
- **comprarCaravana()**: Permite a compra de novas caravanas.
- **sairCaravana()**: Remove uma caravana da cidade e a posiciona no mapa.

- **listarCaravanas()**: Lista todas as caravanas na cidade.

Uso de Smart Pointers:

A cidade armazena as caravanas disponíveis como **`std::shared_ptr`**, garantindo que o estado da caravana seja compartilhado entre a cidade e o mapa.

5. Arquivo Item

Função:

Gerencia os itens espalhados pelo mapa, aplicando efeitos específicos quando interagem com caravanas.

Polimorfismo:

Diferentes tipos de itens implementam o método virtual **aplicarEfeito()**:

- **CaixaDePandora**: Reduz a tripulação.
- **ArcaDoTesouro**: Aumenta as moedas do jogador.
- **Mina**: Destrói a caravana.
- **Surpresa**: Caixa de suprimentos que aumenta os recursos da caravana.

6. Arquivo mapManager

Função:

Gerencia todas as entidades no mapa, incluindo caravanas, cidades e itens, além de realizar atualizações durante o jogo.

Principais Métodos:

- **spawnDefaultEntities()**: Inicializa cidades e caravanas no mapa.
- **moverCaravana()**: Move uma caravana e lida com colisões ou interações.
- **removerCaravanasDestruidas()**: Remove caravanas destruídas do jogo.
- **comprarMercadoria() e venderMercadoria()**: Facilitam transações entre caravanas e cidades.
- **criarTempestade()**: Simula tempestadesdes areia que afetam caravanas.

Uso de Smart Pointers:

Gerencia caravanas, itens e cidades como **`std::shared_ptr`**, garantindo segurança de memória e consistência entre referências.

7. Arquivo Player

Função:

Armazena informações sobre o jogador, como moedas, estatísticas e progresso.

Principais Métodos:

- **adicionarMoedas()**: Atualiza a quantidade de moedas.
- **mostrarEstatisticas()**: Exibe as estatísticas do jogador.

Integração:

O jogador é integrado globalmente, acessível por diferentes partes do sistema para manter dados consistentes.

8. Arquivo GameData

Função:

Gerencia as configurações globais do jogo, como preços, durações e limites.

Principais Métodos:

- **getters e setters**: Configuram e recuperam os valores globais do jogo.

Uso Global:

Implementado como uma variável global (`gameData`) para acesso rápido por diferentes componentes.

9. Arquivo Main

Função:

Gerencia o loop principal do jogo e interpreta comandos inseridos pelo usuário.

Principais Comandos:

- **/config** : Carrega o mapa do jogo.
- **/move:** Move uma caravana.

- **/prox** : Avança o tempo do jogo.
- **/comprac** : Compra uma caravana.
- **/areia** : Simula uma tempestade de areia.

10. Lista de requisitos implementados

- **Buffer de memória**: Implementação de um buffer para controle da exibição do mapa e atualizações dinâmicas.
- **Movimento de Caravanas**: As caravanas podem ser movidas manualmente e automaticamente, respeitando as regras descritas.
- **Gerenciamento de Itens**: Itens são gerados aleatoriamente e possuem durações configuráveis, sendo removidos após expirarem.
- **Compras e Vendas em Cidades**: Implementação de transações econômicas envolvendo mercadorias e compra de caravanas seguem as regras descritas no enunciado.
- **Combates com Bárbaros**: Combates são realizados de forma automática quando caravanas estão próximas.
- **Gestão de Tempo e Turnos**: Simulação avança em instantes configuráveis pelo usuário.
- **Configuração via Arquivo**: Parâmetros são carregados de arquivos de configuração conforme especificação, além de permitir o uso de um arquivo de texto auxiliar que executa os comandos nele contidos.
- **Comportamento das Caravanas em Situações Específicas**: Caravanas seguem as lógicas de comportamento autônomo, sendo capazes de analisar seus arredores em busca de aliados ou inimigos.
- **Tempestades de Areia**: Tempestades podem ser criadas, e seus impactos sobre as caravanas seguem as regras impostas no enunciado.
- **Interações com Itens**: Implementação dos efeitos dos itens segue todas as regras.
- **Caravana Secreta**: A caravana secreta foi definida como uma caravana fantasma, que tem 75 turnos para encontrar bárbaros para assombrar antes de ir embora. Em caso de batalha, tem 50% de chance de assombrar os bárbaros para fora do deserto, resetando o número de turnos até desaparecer.
- **Visualização de Estatísticas Finais**: No final do jogo, as estatísticas são mostradas.
- **Sistema de Salvamento e Carregamento do Buffer**: Comandos como 'saves', 'loads', 'lists' e 'dels' foram implementados e funcionam apropriadamente.
- **Movimentos Aleatórios para Caravanas Sem Tripulação**: A lógica para movimentação automática de caravanas sem tripulação não foi

apropriadamente implementada para caravanas militares, que deveriam se deslocar na direção do seu último movimento, mas ficam paradas.

- **Representação visual de caravanas intuitiva**: Ao chegar no final do projeto, reparei que cometi um erro por usar a representação visual das caravanas como seu ID, e isso leva a caravanas que são compradas serem inicializadas muito antes de entrarem no buffer, o que as faz ter um ID de mais de um digito, dificultando a possibilidade de reconhecimento ou discernimento visual entre outras caravanas.

11. Conclusão

O simulador apresenta uma estrutura modular bem organizada, com uso extensivo de **herança**, **polimorfismo** e **smart pointers** para garantir flexibilidade e segurança na manipulação de objetos. O design permite expansões futuras, como novos tipos de caravanas, eventos ou melhorias na interface.

Testes adicionais e otimizações podem ser implementados para aprimorar o desempenho e prevenir problemas.