isec
Politécnico de Coimbra

2024/2025

Relatório
Projeto POO

Autores:
Bruno Martins 2022147149
Nuno Cerejeira 2022113359

Índice

1. Introdução

1.1. Objetivo do Projeto

Este projeto tem como objetivo o desenvolvimento de um simulador de viagens no deserto, utilizando a linguagem de programação C++. O simulador foi projetado para permitir a gestão de caravanas que percorrem um mapa dinâmico, interagindo com diferentes elementos, como itens mágicos e combates contra adversários bárbaros. O projeto aplica conceitos de programação orientada a objetos (POO) para estruturar e organizar o código, garantindo modularidade e reutilização.

1.2. Descrição Geral

O simulador apresenta um ambiente dinâmico representado por um mapa bidimensional composto por desertos, montanhas e cidades. O utilizador controla caravanas que podem ser de diferentes tipos, cada uma com características específicas para transporte de mercadorias ou combate. As caravanas enfrentam desafios, como tempestades de areia e escassez de recursos, sendo necessário gerir estrategicamente água, mercadorias e tripulação.

A interação com o simulador é realizada através de comandos escritos na consola, permitindo ao utilizador comprar e vender mercadorias, movimentar caravanas, criar tempestades de areia e até iniciar combates. O tempo avança por turnos, nos quais ocorrem eventos dinâmicos, como o aparecimento de novos itens e caravanas bárbaras.

1.3. Estrutura do Relatório

Este relatório está organizado em capítulos para facilitar a análise e compreensão do projeto:

- Capítulo 1 - Introdução - Apresenta o objetivo do projeto, a descrição geral do simulador e a estrutura do relatório.
- Capítulo 2 - Requisitos e Implementação - Descreve os requisitos funcionais, a arquitetura do sistema, a implementação das funcionalidades e os métodos utilizados.
- Capítulo 3 - Conclusão e Trabalho Futuro - Resume os resultados alcançados e sugere melhorias para o projeto em versões futuras.

2. Requisitos e Implementação

2.1 Requisitos Funcionais

Categoria	Requisito	Métodos
Buffer em Memória	Criar uma ou mais classes para armazenar caracteres em memória, simulando um ecrã em memória.	Buffer::Buffer, Buffer::clear, Buffer::moveCursor, operator<<
	A dimensão do buffer (linhas e colunas) deve ser definida em tempo de execução.	Buffer::Buffer
	Não usar mais espaço de armazenamento do que o necessário.	Buffer usa unique_ptr para otimizar memória
	Não usar classes de biblioteca como `string`, `ostringstream`, ou coleções (`vector`, `set`, etc.).	Sem uso de vector, string, etc.
	Suportar múltiplos objetos da classe buffer.	Múltiplas instâncias permitidas pela classe Buffer
	Funcionalidades mínimas: esvaziar o buffer, transcrever o conteúdo para a consola, suportar cursor, mover cursor, imprimir caracteres, strings e inteiros.	clear, moveCursor, operador <<
	Suportar o operador `<<` para imprimir dados no buffer.	operator<< para caracteres, strings e inteiros
	Suportar a transcrição de objetos de outras classes para o buffer usando o operador `<<`.	operator<< para transcrever objetos personalizados
Mapa do Deserto	O mapa deve ser uma grelha retangular com zonas de deserto, montanhas e cidades.	Mapa::Mapa, Mapa::loadFromFile, Mapa::renderToBuffer
	Montanhas são intransponíveis e ocorrem em grupos contíguos.	Mapa::setCell, Mapa::getCell
	Cidades são isoladas e acessíveis, ocupando uma única posição.	Mapa::encontrarCidade
	O deserto é esférico (movimentos nas bordas do mapa levam ao lado oposto).	Mapa::wrapPosition
	Representação visual do mapa: deserto (`.`), montanhas (`+`), cidades (letras minúsculas), caravanas (números), caravanas bárbaras (`I`).	Mapa::renderToBuffer, renderItemsToBuffer

Categoria	Requisito	Métodos
Caravanas	Caravanas podem ser de comércio, militar ou secreta.	Caravana::move, Caravana::consumirRecursos, Caravana::renderToBuffer
	Caravanas de comércio: lentas, alta capacidade de carga, 20 tripulantes iniciais, 40 toneladas de carga, 200 litros de água.	CaravanaComercial::move, consumirRecursos
	Caravanas militares: rápidas, baixa capacidade de carga, 40 tripulantes iniciais, 5 toneladas de carga, 400 litros de água.	CaravanaMilitar::move, consumirRecursos
	Caravanas secretas: características definidas pelo grupo.	Definido no código
	Caravanas bárbaras: 40 tripulantes, movem-se aleatoriamente, não usam água.	CaravanaBarbara::move
	Caravanas podem transportar mercadorias e tripulantes, precisam de água para sobreviver.	Caravana::transportarMercadorias, consumirRecursos
	Caravanas podem entrar em cidades para reabastecer, comprar e vender mercadorias.	Caravana::mudaCelula, setAgua
Itens	Itens aparecem aleatoriamente no deserto: Caixa de Pandora, Arca do Tesouro, Jaula, Mina, Surpresa.	Item::aplicarEfeito, Item::reduzirDuracao
	Itens permanecem no mapa por 20 instantes e desaparecem.	Item::getDuracao, reduzirDuracao
	Itens são apanhados automaticamente por caravanas adjacentes.	Item::aplicarEfeito
	Efeitos dos itens: Caixa de Pandora (mata 20% da tripulação), Arca do Tesouro (aumenta moedas em 10%), Jaula (aumenta tripulação), Mina (destrói caravana), Surpresa (efeito definido pelo grupo).	Efeitos definidos nas classes específicas
Tempestades de Areia	Tempestades de areia são criadas pelo utilizador, afetando uma região quadrada do mapa.	Simulador::criarTempestadeAreia
Tempestades de Areia	Duração de 1 instante.	Duração definida no simulador
	Efeitos: destruição de caravanas, perda de tripulantes ou carga.	Efeitos aplicados por Simulador::aplicarEfeitoTempestade

Categoria	Requisito	Métodos
Combates	Combates ocorrem quando caravanas do utilizador e bárbaras estão adjacentes.	Simulador::executarCombate, calcularForcaCombate
	O resultado do combate é decidido por sorteio com base no número de tripulantes.	Simulador::resolverVitoria
	A caravana vencedora perde 20% da tripulação, a perdedora perde o dobro.	Simulador::verificarDestruicao
	Caravanas podem fugir do combate.	Simulador::condicoesCombate
Comandos do Utilizador	Comandos para configurar o mapa, mover caravanas, comprar/vender mercadorias, criar tempestades de areia, etc.	GestorComandos::executar, Comando::validar
	Comandos específicos: `config`, `exec`, `prox`, `comprac`, `precos`, `cidade`, `caravana`, `compra`, `vende`, `move`, `auto`, `stop`, `barbaro`, `areia`, `moedas`, `tripul`, `saves`, `loads`, `lists`, `dels`, `terminar`.	Todos os comandos implementados (config, exec, prox, etc.)
	O mapa e parâmetros iniciais são lidos de um ficheiro de texto.	Simulador::carregarMapa
Configuração Inicial	O ficheiro de configuração define o tamanho do mapa, posições de montanhas, cidades, caravanas, e valores configuráveis (moedas iniciais, preços, etc.).	Definido no Simulador
Restrições de Implementação	Não usar `vector<vector<X>>` em nenhum local do programa.	Sem uso de vector<vector>
Regras Gerais	Configurações iniciais de mapa, moedas, itens, etc., são lidas de ficheiros configuráveis.	Configuráveis no Simulador
Configurações Específicas	Intervalo configurável para itens e bárbaros, preços de mercadorias e duração de turnos.	Definidos em ficheiros de configuração Preços e durações ajustáveis

dade	Inicializacao
CaravanaComercial	**iD** ∈ [0,9] **MaxTripulacao** = 20; ≥ **MaxMercadoria** = 40; **MaxAgua** = 200;
CaravanaMilitar	**iD** ∈ [0,9] **MaxTripulacao** = 40; **MaxMercadoria** = 5; **MaxAgua** = 400;
CaravanaSecreta	**iD** ∈ [20,30[**MaxTripulacao** = 10; **MaxMercadoria** = 10; **MaxAgua** = 100;
CaravanaBarbara	**iD** **MaxTripulacao** = 40; **MaxMercadoria** = 25; **MaxAgua** = 400; **autoGestao** = True
Simulador	**Moedas** = 500 (configurável) **Turnos** = 1 (começa no primeiro turno) **proximoIdNormal** = 0 **proximoIdBarbaro** = 100 **proximoIdSecreta =** 20 **instantesEntreNovosItens** = 10 (configurável) **duracaoItem** = 10 (configurável) **maxItens** = 5 (configurável), **precoVendaMercadoria** = 2 (configurável), **precoCompraMercadoria** = 1 (configurável), **precoCaravana** = 100 (configurável), **instantesEntreNovosBarbaros** = 40 (configurável), **duracaoBarbaros** = 60 (configurável)

Implementação de tópicos extra

1. As **caravanas bárbaras** têm como objetivo saquear a mercadoria das outras (também ficam com água, apesar de não usarem). Num combate quem ganhar recebe a mercadoria e a água da caravana derrotada;
2. As **caravanas bárbaras** não gastam água, só têm água para efeitos de transferência entre combates.
3. Comando auxiliar 'listacarvanas' mostra todas as caravanas ativas no simulador (ID, tipo, posição, tripulantes, água, carga, cidade);
4. Descrição da **caravana secreta**:

   4.1. Nunca está fora das cidades a não ser para transportar mercadorias entre cidades durante apenas um turno, ficando inativa durante os 5 turnos seguintes;

   (novo comando: secreta <id> mercadoria <c>, c é a cidade de destino)

   4.2. Pode também ser usada para destruir outra caravana (id2) durante um mesmo turno; depois de destruir a caravana regressa à cidade de origem e só depois fica inativa durante os 5 turnos seguintes;

   (novo comando: secreta <id1> destroi <id2>)

   4.3. Não interage com nenhuma outra caravana ou item durante o seu percurso e também não é afetada por tempestades de areia, nem consome recursos;
5. Quatro dos cinco itens aparecem no mapa com os seus caracteres específicos; só o **item Surpresa** aparece com um ponto de interrogação e é aleatoriamente um desses 4; achámos que não fazia sentido os itens Caixa de Pandora, Jaula, Tesouro e Mina estarem escondidos, porque a sua ocultação causa efeito surpresa efeito esse que nós achamos apropriado para o **item Surpresa**;

2.2 Arquitetura do Sistema

Classe Simulador

Descrição: Responsável por coordenar todo o simulador, gerir turnos, eventos, combates e interações entre as caravanas e o mapa.

Principais Métodos:
- **void carregarMapa(const std::string &ficheiro)** - Carrega o mapa inicial do ficheiro.
- **void avancarTurno(int n)** - Avança o tempo do simulador por um número especificado de turnos.
- **void adicionarCaravana(char tipo, char cidade)** - Adiciona uma nova caravana ao simulador.
- **void executarCombate(Caravana &c1, Caravana &c2)** - Executa combates entre caravanas.
- **void gerarEventos()** - Gera eventos aleatórios como tempestades e novos itens.

Classe Mapa

Descrição: Representa o mapa do deserto, contendo células que podem ser desertos, cidades ou montanhas.

Principais Métodos:
- **bool loadFromFile(const std::string &filename)** - Carrega o mapa a partir de um ficheiro de texto.
- **void criarTempestadeAreia(int l, int c, int _linha, std::vector<std::shared_ptr<Caravana>> &caravanas)** - Cria uma tempestade de areia.
- **void renderToBuffer(Buffer &buffer, const std::vector<std::shared_ptr<Caravana>> &caravanas)** - Renderiza o estado do mapa no buffer.

Classe Buffer

Descrição: Simula um ecrã em memória para representar visualmente o mapa e as caravanas.
Principais Métodos:
- **void moveCursor(int linha, int coluna)** - Move o cursor virtual para uma posição.
- **void clear()** - Esvazia o buffer.
- **std::ostream &operator<<(std::ostream &os, const Buffer &buffer)** - Permite imprimir o buffer na consola.

Classe Pixel

Descrição: Classe base abstrata que representa um ponto no mapa com coordenadas e um símbolo.
Classes derivadas:
- **Elemento**;
- **Item**;

Principais Métodos:
- **virtual char getSimbolo() const = 0**; - Método virtual puro para obter o símbolo do pixel.
- **int getX() const**; - Retorna a posição X.
- **int getY() const**; - Retorna a posição Y.

Classe Elemento

Descrição: Subclasse de Pixel que representa diferentes elementos do mapa, como desertos, montanhas e cidades.

Classes derivadas:
- **Deserto** - Representa áreas vazias no mapa.
- **Montanha** - Representa áreas intransitáveis.
- **Cidade** - Representa locais onde podem ser realizadas trocas e compras.

Classe Item

Descrição: Subclasse de `Pixel` que representa itens mágicos disponíveis no mapa.

Classes derivadas:
- **CaixaPandora** - Reduz a tripulação em 20%.
- **ArcaTesouro** - Acrescenta 10% das moedas do jogador.
- **Mina** - Destrói a caravana.
- **Surpresa** - Executa efeitos aleatórios, como ganhar moedas ou destruir a caravana.

Classe Caravana

Descrição: Representa as caravanas e suas interações no mapa.

Classes derivadas:
- **CaravanaComercial** - Focada no transporte de mercadorias.
- **CaravanaMilitar** - Otimizada para combate.
- **CaravanaSecreta** - Com comportamentos personalizados.
- **CaravanaBarbara** - Movimentação e combates automáticos com inimigos.

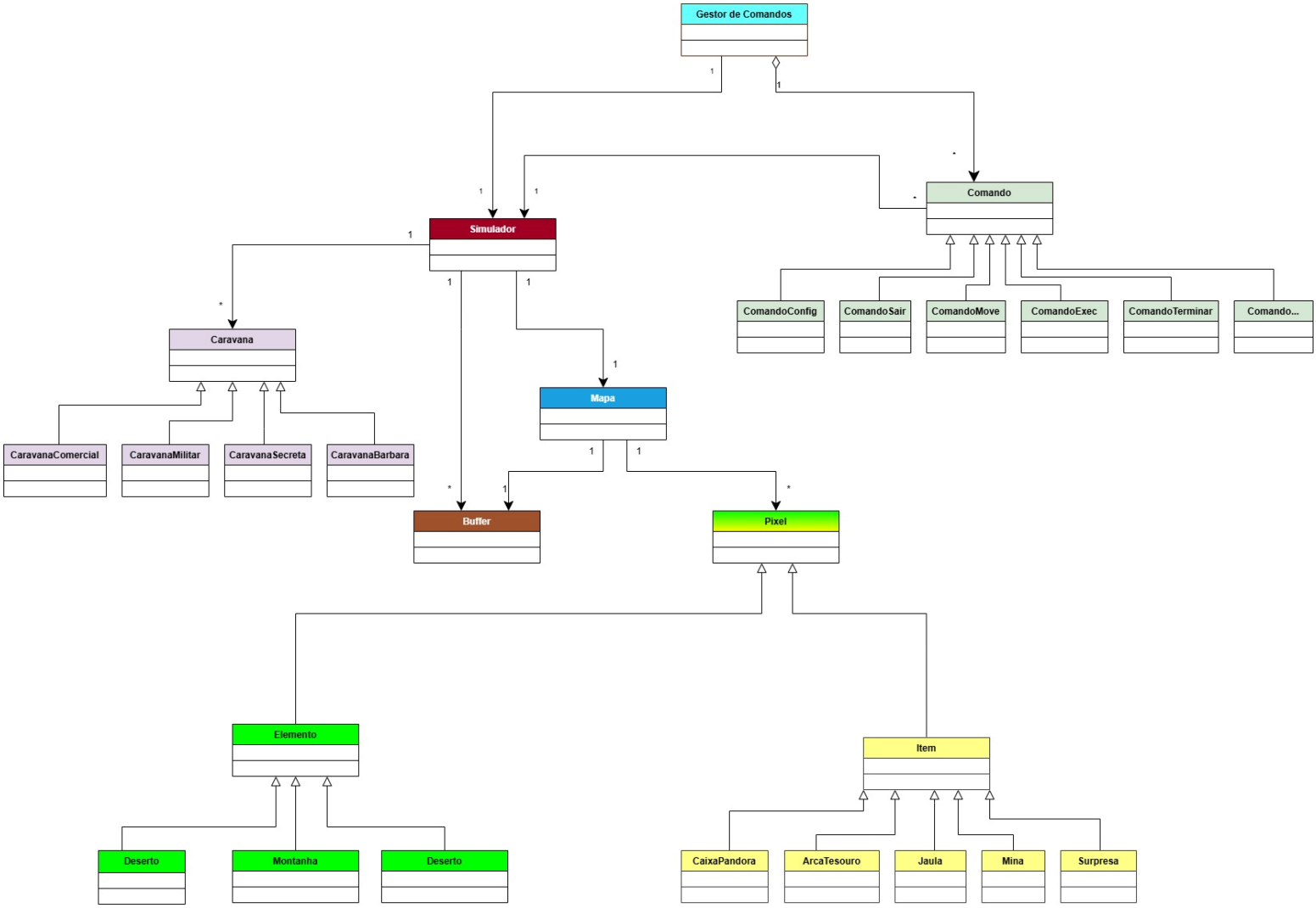
Classe GestorComandos

Descrição: Processa os comandos introduzidos pelo utilizador para controlar as caravanas e o simulador.

Principais Métodos:

- **void executar()** - Lê e executa os comandos do utilizador.
- **bool validarFase(const std::string &input)** - Valida comandos permitidos em diferentes fases do simulador.

Diagrama de Classes



2.3 Análise dos Conceitos de Programação Orientada a Objetos em C++

Modularidade

- O projeto está dividido em múltiplas classes (Simulador, Mapa, Buffer, Caravana, Item, etc.), cada uma responsável por funcionalidades específicas, promovendo a reutilização e organização do código.
- A classe Simulador é dedicada apenas à execução do jogo;
- A classe GestorComandos tem como função gerir os comandos que irão fazer correr o simulador, isto é, os comandos são de classes externas à classe Simulador e são também geridos de forma autónoma pela Classe GestorComandos;
- Os ficheiros separados (.h e .cpp) garantem a separação entre declarações e implementações, facilitando a manutenção e extensão do código.

Abstração e Encapsulamento

- **Abstração:** As classes fornecem interfaces claras para interagir com os objetos (por exemplo, Simulador oferece métodos como carregarMapa e avancarTurno).
- **Encapsulamento:** Os atributos privados (como moedas e turnos no Simulador) estão protegidos e só podem ser manipulados através de métodos públicos, garantindo segurança e integridade dos dados.

Herança e Polimorfismo

- **Herança:**
  - Elemento e Item herdam de Pixel, permitindo o reaproveitamento de código para coordenadas e símbolos.
  - As classes Montanha, Deserto e Cidade derivam da classe Elemento (2º nível de herança);
  - As classes CaixaPandora, Tesouro, Jaula e Surpresa derivam da classe Item (2º nível de herança);
  - Caravana possui subclasses (CaravanaComercial, CaravanaMilitar, etc.), que herdam e especializam funcionalidades específicas.

- **Polimorfismo:**
  - Métodos virtuais como getSimbolo() permitem comportamentos específicos para cada tipo de elemento ou item.
  - A função aplicarEfeito() nas subclasses de Item define efeitos diferenciados para cada item.
  - os vector<Caravana> contêm entidades das classes derivadas (CaravanaMilitar, CaravanaComercial, CaravanaBarbara e CaravanaSecreta);
  - os vector<Comandos> contêm entidades das classes derivadas (ComandoConfig, …, ComandoTerminar);

Operadores

- Sobrecarga do operador << na classe Buffer para facilitar a impressão de dados diretamente na consola.
- Uso de operadores de atribuição e comparação para simplificar a manipulação de objetos.

Utilização de Ponteiros Inteligentes (SmartPointers)

- Utilização de std::shared_ptr e std::unique_ptr para garantir a gestão automática de memória.
- Exemplo: std::shared_ptr<Caravana> é utilizado para gerir as caravanas no simulador, evitando fugas de memória e simplificando a partilha de objetos entre diferentes componentes.

3. Conclusão

Este projeto atingiu os seus objetivos, fornecendo um simulador funcional de viagens no deserto que incorpora elementos de gestão de recursos, movimentação estratégica e eventos dinâmicos. A implementação em C++ utilizando conceitos de Programação Orientada a Objetos garantiu modularidade e escalabilidade. Todas as funcionalidades especificadas foram desenvolvidas e testadas com sucesso.

Trabalho Futuro:
Melhorias para versões futuras, como:

- Interface gráfica.
- Novos tipos de itens, eventos e mercadorias.
- Expansão das funcionalidades automáticas das caravanas.
- Melhoramento da funcionalidade de jogo:

  - Cada cidade só vende um tipo de mercadoria e compra todos os outros; assim as caravanas serviriam para abastecer as diversas cidades das mercadorias que elas precisam para sobreviver;

  - Os preços devem variar o longo dos turnos de acordo com a lei da oferta e da procura (tempestades de areia, guerras e raides bárbaros fazem encarecer os produtos enquanto durarem);

  - Podem ser introduzidos tratados de paz, parcerias militares e comerciais, etc;

4. Bibliografia

[1] Brokken, F. B. (2024). C++ Annotations Version 13.0.0, University of Groningen publishing, http://www.icce.rug.nl/documents/cplusplus/ - Online
Stroustrup, B. (2000). A linguagem de programação C++, Bookman – 1ª-1-359

[2] Eckel, B. (2000). Thinking in C++ Vol.1 (2nd ed.), Prentice Hall Inc., http://www.bruceeckel.com/ - 1A-1-380 / Online

[3] https://chatgpt.com/