POO

TP 2024 – 2025

Duarte Santos a2022149622
Gustavo Costa a2023145800

Licenciatura de Engenharia Informática

Índice

Índice

Introdução:ge

Este trabalho tem como objetivo o desenvolvimento de um simulador de viagens no deserto utilizando C++. O simulador apresenta um ambiente interativo onde o utilizador gere caravanas em missões que envolvem transporte de mercadorias, exploração de tesouros e combates com bárbaros. Estas caravanas movem-se por um mapa que retrata o deserto, um cenário hostil e dinâmico.

O mapa da simulação é composto por uma grelha retangular que representa diferentes tipos de terreno: zonas de deserto, que são transitáveis, montanhas, que são intransponíveis e cidades, onde as caravanas podem descansar, reabastecer recursos, comprar ou vender mercadorias e interagir com o ambiente. A representação visual do mapa é uma parte fundamental do simulador, destacando as diferentes zonas, a localização das caravanas, e outros elementos relevantes, utilizando símbolos específicos para facilitar a compreensão.

A simulação ocorre de forma sequencial, baseada em turnos. Em cada turno, o utilizador pode realizar ações específicas consoante a introdução de comandos. Este sistema de interação exige que os comandos sejam escritos completamente, com palavras-chave e argumentos necessários numa única linha, tornando-a assim uma interface simples, mas muito prática para se controlar as ações no jogo.

Além disso, o simulador tem elementos autónomos que se comportam de forma independente ao longo jogo. A passagem do tempo, neste caso "instantes", permite que o ambiente evolua dinamicamente, introduzindo novos desafios a cada turno. O utilizador pode, também, avançar o tempo por meio de comandos, simulando a progressão de 1 ou vários instantes.

Pormenores da Implementação:

O projeto contém as seguintes classes:

Mapa e Buffer:

- **Buffer**: Representa o armazenamento de informações associadas ao mapa, contendo elementos como a matriz de representação do jogo.
- **Mapa**: Gere o ambiente do jogo, controlando a configuração do mapa, as zonas de deserto, montanhas, cidades, e a interação com as caravanas.

Caravana:

- **Caravana**: Classe base que representa as caravanas do jogo, utiliza herança para especificar os diferentes tipos de caravanas.
- **CaravanaMilitar**: Especialização de Caravana voltada para combates no deserto.
- **CaravanaComercio**: Especialização para transporte e comercialização de mercadorias.
- **CaravanaSecreta**: Uma classe que nos coube a nós inventar, a explicação estará mais adiante.
- **CaravanaBarbara**: Representa caravanas bárbaras que podem atacar outras caravanas.

Itens:

- **Item**: Classe abstrata que serve como base para os diferentes tipos de itens que podem ser encontrados no jogo.
- **CaixaPandora**: Um item irá tirar 20% da tripulação atual.
- **ArcaTesouro**: Um item que irá acrescentar 10% das moedas do utilizador.
- **Jaula**: Irá adicionar tripulantes à caravana que o apanha, sem exceder o máximo.
- **Mina**: Destrói a caravana.
- **Surpresa**: Um item que nós implementamos, tal como a classe CaravaSecreta, irá ser explicado mais adiante.

Eventos :

- **TempestadeAreia**: As tempestades de areia que podem ocorrer no deserto.

Cidade:

- **Cidade**: Representa os locais onde as caravanas podem realizar transações de compra e venda, além de reabastecimento.

Interface:

- **Interface**: Controla a interação entre o jogador e o simulador, gere comandos e exibindo informações do estado atual.

Dados:

- **Dados**: Atua como núcleo central do jogo, gere o estado global, incluindo caravanas, cidades, itens, bárbaros e tempestades.

Especificação das classes:

Buffer:

A classe Buffer é uma representação de um buffer de memória bidimensional, com as seguintes funcionalidades principais:

- **Atributos:**
  - **Uma matriz de caracteres (char **matriz).**
  - **Posições de um cursor que navega pela matriz (cursorLinha e cursorColuna).**
  - **O número de linhas e colunas do buffer (linhas e colunas).**
- **Construtores e Destruidor:**
  - **Construtores para inicializar o buffer com dimensões específicas ou com valores padrão.**
  - **Destruidor para liberar a memória alocada.**
- **Funções de Acesso (getters e setters):**
  - **Funções para acessar e modificar o número de linhas e colunas.**
  - **Uma função para acessar um caractere específico no buffer via coordenadas.**
- **Funções de Manipulação:**
  - **Funções para limpar o buffer (esvaziar()), imprimir seu conteúdo (imprimir()), e mover o cursor (moverCursor()).**
  - **Funções para inserir caracteres, strings e inteiros no buffer na posição atual do cursor (inserirChar(), inserirString(), inserirInt()).**
- **Sobrecarga de Operadores:**
  - **Sobrecarga do operador << para inserir strings, caracteres ou inteiros no buffer de forma prática.**
- **Construtor de Cópia e Operador de Atribuição:**
  - **Funções para copiar o conteúdo de outro Buffer.**

Mapa:

Em resumo, a classe Mapa é responsável por gere um mapa com configurações e um buffer, permitindo carregar dados de um arquivo, manipular elementos no mapa e alterar configurações.

- **Atributos**:
  - Um map (std::map<std::string, int> configuracoes) que armazena as configurações do mapa, associando uma chave (nome) a um valor.
  - Um ponteiro para um objeto Buffer (Buffer *buf), que representa o conteúdo do mapa em forma de buffer.
- **Construtores**:

    - Um construtor que recebe o caminho de um arquivo de texto e carrega o mapa e suas configurações a partir desse arquivo.
    - Um construtor padrão.
- **Funções de Manipulação e Acesso**:
    - **carregarDeFicheiro(const std::string& caminhoFicheiro)**: Carrega o mapa e suas configurações a partir de um ficheiro de texto.
    - **mostrarConfiguracoes() const**: Exibe as configurações do mapa.
    - **getBuffer()**: Retorna uma referência ao buffer associado ao mapa.
    - **getConfiguracao(const std::string& chave) const**: Retorna o valor da configuração associada .
    - **isAdjacente(int linha1, int coluna1, int linha2, int coluna2) const**: Verifica se duas posições no mapa são adjacentes.
- **Funções de Modificação**:
    - **inserirElemento(int linha, int coluna, char elemento)**: Insere um elemento em uma posição específica no mapa.
    - **alterarConfiguracao(const std::string& chave, int novoValor)**: Modifica o valor de uma configuração existente.

Caravana:

A classe Caravana representa uma caravana com diversos atributos e comportamentos relacionados aos recursos e ações da caravana.

Atributos:

- **Identificação e Estado**: Possui um ID único, a posição no mapa, e indica se está destruída ou em modo autónomo.
- **Recursos**: Controla a capacidade e a quantidade atual de carga e água, além do número de tripulantes.
- **Movimento e Interações**: Registra o número de movimentos realizados e permite a atualização de sua posição no mapa.

Funções:

- **Manipulação de Recursos**: Permite adicionar ou remover carga e água, bem como alterar o número de tripulantes.
- **Movimento e Posição**: Oferece funções para mover a caravana no mapa, definir sua posição, e atualizar seu estado após o movimento.
- **Comportamento e autonomia**: Possui funções para simular comportamentos como reabastecimento de água, modo autónomo e interação com tempestades, e permite o comportamento sem tripulantes.

Funções Virtuais:

- A classe contém várias funções virtuais puras que serão implementadas em subclasses, como o movimento autónomo, comportamento em tempestades e clonagem da caravana.

Em resumo, a classe Caravana é responsável por gerir os aspetos básicos de uma caravana, incluindo movimento, recursos e interações com o ambiente, oferecendo uma estrutura que pode ser especializada através de herança.

CaravanaMilitar:

A classe CaravanaMilitar é uma especialização da classe Caravana e adiciona funcionalidades específicas para o contexto de uma caravana militar.

Atributos:

- **int turnosSemTripulantes**: Armazena o número de turnos que a caravana ficou sem tripulantes.
- **std::string ultimaDirecao**: Armazena a última direção em que a caravana se moveu.

Funções:

- **getMaxMovimentos()**: Retorna o número máximo de movimentos por turno permitidos para a caravana militar.
- **moverAutonomo(Mapa& mapa, const std::vector<Caravana*>& caravanasJogador, const std::vector<Cidade>& cidadesJogador)**: Implementa o movimento autónomo da caravana militar.
- **isTempestade()**: Define o comportamento da caravana em caso de tempestade.
- **semTripulantes(Mapa &mapa)**: Define o comportamento da caravana quando não há tripulantes disponíveis.
- **consumirAgua()**: Define como a caravana militar consome água, consoante o seu número de tripulantes.
- **moverCaravanaUmaPosicao(Mapa& mapa, std::string direcao)**: Move a caravana militar uma posição no mapa, dependendo da direção fornecida.
- **verificarBarbarosProximos(Mapa &mapa)**: Verifica se há bárbaros próximos à caravana militar.
- **moverParaBarbaros(Mapa &mapa)**: Move a caravana militar em direção aos bárbaros mais próximos.
- **getMaxTripulantes()**: Retorna o número máximo de tripulantes que a caravana militar pode ter.
- **clone()**: Cria uma cópia da caravana militar.

CaravanaComercio:

A classe CaravanaComercio é uma especialização da classe Caravana, adaptada para o contexto de uma caravana de comércio.

Atributos:

- **int turnosSemTripulantes**: Conta o número de turnos em que a caravana de comércio fica sem tripulantes, permitindo que o comportamento da caravana seja alterado quando isso ocorre.

Funções:

- **getMaxMovimentos()**: Retorna o número máximo de movimentos por turno permitidos para a caravana de comércio.
- **moverAutonomo(Mapa& mapa, const std::vector<Caravana*>& caravanasJogador, const std::vector<Cidade>& cidadesJogador)**: Implementa o movimento autónomo da caravana de comércio.
- **isTempestade()**: Define o comportamento da caravana de comércio durante uma tempestade.
- **semTripulantes(Mapa &mapa)**: Define o comportamento da caravana quando não há tripulantes disponíveis.
- **consumirAgua()**: Define como a caravana de comércio consome água, consoante o número da sua tripulação.
- **getMaxTripulantes()**: Retorna o número máximo de tripulantes que a caravana de comércio pode ter.
- **clone()**: Cria uma cópia da caravana de comércio.

CaravanaSecreta:

A classe CaravanaSecreta é uma especialização da classe Caravana, que introduz a capacidade de invisibilidade e funcionalidades relacionadas à furtividade e ocultação no jogo ou simulação.

Atributos:

- **int turnosSemTripulantes**: Contabiliza os turnos em que a caravana está sem tripulantes, o que pode alterar seu comportamento.
- **int turnosVisivel**: Controla o número de turnos em que a caravana está visível no mapa.
- **bool invisivel**: Indica se a caravana está em estado de invisibilidade, o que permite que ela se oculte de outras caravanas e das tempestades.

Funções:

- **atualizarInvisibilidade(Mapa &mapa)**: Atualiza o estado de invisibilidade da caravana, possivelmente dependendo das condições do mapa ou do jogo.
- **tornarVisivel(Mapa &mapa)**: Torna a caravana visível novamente, interrompendo sua invisibilidade.
- **moverAutonomo(Mapa& mapa, const std::vector<Caravana*>& caravanasJogador, const std::vector<Cidade>& cidadesJogador)**: Implementa o movimento autónomo da caravana.
- **isTempestade()**: Define o comportamento da caravana secreta durante uma tempestade.

- **semTripulantes(Mapa &mapa)**: Define o comportamento da caravana quando não há tripulantes.
- **consumirAgua()**: Especifica como a caravana secreta consome água.
- **getMaxMovimentos()**: Retorna o número máximo de movimentos por turno permitidos para a caravana secreta.
- **getMaxTripulantes()**: Retorna o número máximo de tripulantes que a caravana secreta pode ter.
- **clone()**: Cria uma cópia da caravana secreta, permitindo replicá-la no jogo.

A **Caravana Secreta** é uma unidade discreta e ágil, projetada para missões de furtividade, destacando-se pela capacidade de se tornar invisível no deserto. Embora tenha uma capacidade de carga e número de tripulantes limitados, sua principal vantagem é a habilidade de operar sem ser detetada.

- **Capacidade e Recursos**: Inicialmente, possui 10 tripulantes, capacidade para 20 toneladas de carga e 150 litros de água. Consome 1 litro de água por turno, ou 0,5 litros se tiver menos de metade dos tripulantes, e nada se estiver sem tripulantes.
- **Movimento e Invisibilidade**: Pode mover-se até duas posições por turno e alterna entre visível e invisível. Fica visível por 3 turnos e invisível por 5, com o estado de invisibilidade atualizado automaticamente. Em estado invisível, não é visível no mapa e não interage com outras entidades, como bárbaros ou tempestades.
- **Comportamento em Tempestades**: Durante tempestades, a caravana tem 50% de chance de perder 10% dos seus tripulantes e carga, mas permanece intacta. Se estiver invisível, a invisibilidade é desativada temporariamente.
- **Quando Fica Sem Tripulantes**: Se ficar sem tripulantes, a caravana se move aleatoriamente por até 7 turnos antes de desaparecer silenciosamente do mapa.
- **Propósito e Utilidade**: Ideal para missões de reconhecimento e transporte de itens valiosos em áreas perigosas, a caravana secreta é útil para evitar interações indesejadas e se destacar pela furtividade.

CaravanaBarbara:

A **Caravana Bárbara** é uma classe que representa caravanas com comportamentos agressivos, com foco em destruição. Diferente das caravanas tradicionais, ela pode desaparecer após um certo número de turnos e apresenta características únicas de movimento e interação com o ambiente.

- **Atributos**:
  - **int turnosAtivos**: Contador que monitora o número de turnos em que a caravana está ativa.
  - **bool desaparecida**: Flag que indica se a caravana desapareceu do mapa.
  - **int turnos_max**: Número máximo de turnos antes do desaparecimento da caravana bárbara.

- **Funções**:
    - **moverAutonomo(Mapa& mapa, const std::vector<Caravana*>& caravanasJogador, const std::vector<Cidade>& cidadesJogador)**: Implementa o movimento autónomo da caravana bárbara.
    - **isTempestade()**: Define o comportamento da caravana bárbara durante tempestades.
    - **getMaxMovimentos()**: Retorna o número máximo de movimentos que a caravana bárbara pode fazer.
    - **getMaxTripulantes()**: Retorna o número máximo de tripulantes para a caravana bárbara.
    - **semTripulantes(Mapa &mapa)**: Define o comportamento da caravana quando não há tripulantes disponíveis.
    - **getDesaparecida()**: Retorna se a caravana bárbara desapareceu.
    - **setIncrementaTurno()**: Incrementa o contador de turnos ativos.
- **Funções irrelevantes**:
    - **consumirAgua()** e **clone()** são funções não aplicáveis ou não implementadas de forma relevante para uma caravana bárbara, visto que esta não consome água e nem poderão haver duas caravanas bárbaras ao mesmo tempo.

Item:

A **classe Item** representa a classe baze de um item no jogo que pode ter efeitos quando interage com uma caravana. Esses itens possuem uma duração limitada e podem impactar o comportamento das caravanas de diversas maneiras.

Atributos:

- **int linha, coluna**: Coordenadas do item no mapa.
- **int duracao**: Tempo restante até o item expirar ou perder seu efeito.
- **std::string tipo**: Tipo do item, que pode determinar seu efeito ou funcionalidade (por exemplo, água, alimento, etc.).

Funções:

- **Item(Mapa& mapa, const std::string& tipo, int linha, int coluna)**: Construtor que inicializa o item com as informações de tipo, localização e duração.
- **virtual ~Item() = default**: Destruidor virtual padrão.
- **int getDuracao() const**: Retorna a duração restante do item.
- **std::string getTipo() const**: Retorna o tipo do item.
- **int getLinha() const, int getColuna() const**: Retornam as coordenadas do item no mapa.
- **void decrementarDuracao()**: Decrementa a duração do item a cada turno ou ação.

- **virtual void efeito(Caravana& caravana, Mapa& mapa) = 0**: Método virtual puro que define o efeito do item ao ser apanhado por uma caravana. Este efeito pode variar dependendo do tipo do item.
- **virtual Item* clone() const = 0**: Método virtual que permite criar uma cópia do item.
- **Item& operator=(const Item& other)**: Sobrecarga do operador de atribuição para copiar os atributos de um item para outro.

CaixaPandora:

A **classe CaixaPandora** é uma especialização da classe **Item**, representando um item específico no jogo com um efeito único quando interage com uma caravana. O seu principal efeito é o de remover 20% da tripulação existente na caravana.

Construtor:

- **CaixaPandora(Mapa& mapa, int linha, int coluna)**: Construtor que inicializa a caixa pandora no mapa, passando as coordenadas (linha, coluna) e o mapa onde o item está localizado. Chama o construtor da classe base **Item**.

Funções:

- **void efeito(Caravana& caravana, Mapa& mapa) override**: Implementação do método **efeito** da classe base **Item**. Esta função define o efeito que a **CaixaPandora** tem quando é apanhada por uma caravana.
- **Item* clone() const override**: Sobrecarga do método **clone()** da classe base **Item**, que cria uma cópia da **CaixaPandora**.

Jaula:

A **classe Jaula** é uma especialização da classe **Item**. Neste caso, a **Jaula** é um item que irá aumentar a tripulação da caravana que a apanha.

Construtor:

- **Jaula(Mapa& mapa, int linha, int coluna)**: Construtor que inicializa a **Jaula** no mapa, posicionando-a nas coordenadas especificadas (linha, coluna). Ele chama o construtor da classe base **Item** para configurar o tipo de item e sua localização.

Funções:

- **void efeito(Caravana& caravana, Mapa& mapa) override**: Implementação do método **efeito** da classe base **Item**. Esta função define o efeito específico que a **Jaula** tem quando é apanhada por uma caravana.
- **Item* clone() const override**: Sobrecarga do método **clone()** da classe base **Item**, que cria uma cópia da **Jaula.**

Mina:

A **classe Mina** é uma especialização da classe **Item**, representando a explosão e destruição da caravana que a apanha.

Construtor:

- **Mina(Mapa& mapa, int linha, int coluna)**: Construtor que inicializa a **Mina** no mapa, atribuindo as coordenadas (linha, coluna) e passando o mapa onde a mina está localizada. Chama o construtor da classe base **Item** para definir o tipo de item e a sua posição.

Funções:

- **void efeito(Caravana& caravana, Mapa& mapa) override**: Implementação do método **efeito** da classe base **Item**. Esta função define o efeito da mina quando é ativada por uma caravana.
- **Item* clone() const override**: Sobrecarga do método **clone()** da classe base **Item**, que cria uma cópia da **Mina**.

Supresa:

A **classe Surpresa** é uma especialização da classe **Item**, representando no nosso trabalho uma perda de 20% da tripulação porém com a adição de "o gato da fortuna" que enquanto a caravana não estiver destruída, irá adicionar 1 moeda ao utilizador por cada turno que passe.

Construtor:

- **Surpresa(Mapa& mapa, int linha, int coluna)**: Construtor que inicializa a **Surpresa** no mapa, atribuindo as coordenadas (linha, coluna) e passando o mapa onde o item está localizado. Chama o construtor da classe base **Item** para configurar o tipo de item e a sua posição no mapa.

Funções:

- **void efeito(Caravana& caravana, Mapa& mapa) override**: Implementação do método **efeito** da classe base **Item**. Esta função define o efeito da **Surpresa** ao ser apanhada por uma caravana.
- **Item* clone() const override**: Sobrecarga do método **clone()** da classe base **Item**, que cria uma cópia da **Surpresa**.

TempestadeDeAreia:

A **classe TempestadeAreia** representa uma tempestade de areia no jogo, que irá afetar as caravanas que estejam dentro dela.

Atributos:

- **int linhaC, colunaC**: Coordenadas do centro da tempestade de areia no mapa.
- **int raio**: O raio de alcance da tempestade de areia.
- **bool ativa**: Uma flag se a tempestade está ativa ou não.

Construtores:

- **TempestadeAreia(int linhaCentro, int colunaCentro, int raio)**: Construtor que inicializa a tempestade de areia com o centro e o raio especificados, além de definir se está ativa ou não.
- **TempestadeAreia()**: Construtor padrão.

Funções:

- **bool isAtiva() const**: Retorna **true** se a tempestade de areia estiver ativa, e **false** caso contrário.
- **bool isDentro(int linha_caravana, int coluna_caravana) const**: Verifica se as coordenadas da caravana (linha, coluna) estão dentro do raio de alcance da tempestade de areia. Retorna **true** se a caravana estiver dentro da área afetada pela tempestade, caso contrário, retorna **false**.
- **void setInativa()**: Define a tempestade de areia como inativa.

Cidade:

A **classe Cidade** representa uma cidade no jogo, possuindo atributos como localização no mapa, mercadorias disponíveis para venda e tipos de caravanas associadas a ela. Cada cidade pode ter caravanas de diferentes tipos (comercial, militar, secreta) disponíveis .

Atributos:

- **char nome**: O nome da cidade, representado por um único caractere.
- **int linha, coluna**: Coordenadas da cidade no mapa.
- **int qntdMercadorias**: Quantidade de mercadorias disponíveis para venda na cidade.
- **bool cComercio**: Indica se a cidade possui uma caravana de tipo comercial disponível.
- **bool cMilitar**: Indica se a cidade possui uma caravana de tipo militar disponível.
- **bool cSecreta**: Indica se a cidade possui uma caravana de tipo secreta disponível.

Construtores:

- **Cidade()**: Construtor padrão.
- **Cidade(char nome, int linha, int coluna)**: Construtor que inicializa uma cidade com o nome, linha e coluna fornecidos. Pode definir a posição da cidade no mapa e seu nome.

Funções:

- **int getLinha() const**: Retorna a linha da cidade no mapa.
- **int getColuna() const**: Retorna a coluna da cidade no mapa.
- **char getNome() const**: Retorna o nome da cidade.
- **void setCComercio(bool estado)**: Define se a cidade tem uma caravana de comércio disponível.

- **void setCMilitar(bool estado)**: Define se a cidade tem uma caravana militar disponível.
- **void setCSecreta(bool estado)**: Define se a cidade tem uma caravana secreta disponível.
- **bool getCComercio() const**: Retorna se a cidade possui uma caravana de comércio.
- **bool getCMilitar() const**: Retorna se a cidade possui uma caravana militar.
- **bool getCSecreta() const**: Retorna se a cidade possui uma caravana secreta.
- **Cidade* operator[](char nome)**: Sobrecarga do operador [] para buscar a cidade pelo nome. Retorna um ponteiro para a cidade com o nome fornecido.

Interface:

A **classe Interface** representa a interface do usuário, responsável por interagir com o jogador, processar comandos e gerir a execução do jogo.

Atributos:

- **Dados* dados**: Ponteiro para um objeto da classe Dados.
- **bool fase1**: Um booleano que indica se o jogo está na primeira fase ou não.

Construtor:

- **Interface(Dados* d)**: Inicializa a interface, recebendo um ponteiro para um objeto da classe Dados. Esse ponteiro permite à interface acessar e modificar os dados do jogo.

Funções:

- **void executa()**: Função principal que executa a interface.

Funções privadas:

  - **void configComando(const std::string& nomeFicheiro)**: Configura os comandos do jogo a partir de um arquivo.
  - **void execComandoFicheiro(const std::string& nomeFicheiro)**: Executa comandos a partir de um arquivo de texto, lendo e processando os comandos listados.
  - **void proxComando(int n)**: Avança para o próximo comando (avançando de turno assim).
  - **void processaComando(const std::string& comando)**: Processa um comando dado em formato de string, executando as ações correspondentes no jogo.
  - **void sair()**: Finaliza a execução da interface e pode encerrar o jogo ou o programa.

Dados:

A **função Dados** é responsável por gerir e controlar todo o jogo, incluindo as entidades principais (caravanas, cidades, itens, etc.), o mapa e o progresso do jogo. Ela oferece funcionalidades para carregar e salvar o estado do jogo, interagir com o ambiente e realizar diversas ações, como mover caravanas, comprar mercadorias, criar itens ou eventos (como bárbaros e tempestades de areia) e gerir turnos. Basicamente, a classe **Dados** centraliza todas as operações e dados importantes do jogo.

Conclusão

Este projeto permitiu criar uma aplicação funcional que responde aos objetivos propostos, mostrando-se eficiente na gestão de comandos e no controlo do sistema. Além disso, a solução desenvolvida é flexível e pode ser facilmente ajustada para incluir novas funcionalidades no futuro. No geral, o trabalho alcançou os resultados esperados, abrindo espaço para possíveis melhorias.