Programação Orientada a Objetos 2021

16 JANEIRO

Instituto Superior De Engenharia de Coimbra
Docente: João Durães
Realizado por: Francisco Almeida - 2020138795
Realizado por: Diogo Pinto – 2020133653

isec Engenharia

Índice

Estrutura

Classes

O programa encontra-se dividido em diversas classes de forma a manter uma boa organização e fácil manutenção do código. Estas classes estão sujeitas a alteração posterior, e são as seguintes:

Ilha

Esta classe foi criada com o intuito de armazenar o mapa onde o utilizador trabalha

Tipo	Nome	Objetivo
int	linhas	Armazenar o número de linhas no mapa
int	colunas	Armazenar o número de colunas no mapa
Zona* **	tab	Ponteiro para o array de bidimensional de ponteiros para classes Zona

Com estas 3 variáveis vamos conseguir controlar o tamanho do mapa bem como ter acesso a cada uma das células(zonas) do mesmo.

A class ilha possui funções para criar o tabuleiro, para o limpar e para obter uma certa zona a partir de uma linha e uma coluna.
A classe ilha encontra-se numa associação de composição com a classe Zona que vamos referir a seguir.

Zona

A classe zona foi criada com o propósito de armazenar uma série de variáveis, tais como: edifícios, trabalhadores, tipos de zona etc. E também como forma de termos diferentes tipos de zonas por toda a ilha tendo assim um "tabuleiro de zonas"
Esta classe possui diversas funções importantes para o funcionamento de qualquer zona, como a função **getMultiplier(), getAdjacente()**, **efeito()**, etc.

Tipo	Nome	Objetivo
Ilha*	ilha	Armazenar um ponteiro para a ilha
array<Recursos*,7>	recursos	Armazenar um array de ponteiros para os recursos do jogo
int	linha	Numero da linha onde a zona se encontra
int	coluna	Numero da coluna onde a zona se encontra
Edificios*	edificio	Ponteiro para um edificio presente na zona
vector<Trabalhadores*>	tipoTrabalhadores	Armazenar um contador que irá ser utilizado no construtor de cada Zona
int	nTrabalhadores	Numero de trabalhadores na zona

Classes derivadas de Recursos:

- **Floresta**
  Classe que representa a zona de floresta onde os Lenhadores trabalham. Esta zona possui um número de árvores inicial crescendo uma árvore nova a cada dois dias e produz 1kg de madeira por cada Lenhador que trabalhar aqui.

- **Pastagem**
  Classe que representa a zona de pastagem onde os trabalhadores começam. Os trabalhadores nesta zona não se despedem.

- **Deserto**
  Classe que representa a zona do deserto. As minas nesta zona produzem 50% menos ferro.

- **Montanha**
  Classe que representa a zona de montanha. As minas nesta zona produzem 100% mais ferro e ela tem a característica especial de produzir 0.1kg de ferro por dia e de aumentar a chance de despedimento de todos os trabalhadores por 5%.

- **Pântano**
  Classe que representa a zona de pântano. Os edificios nesta zona são destruidos passados dez dias e os trabalhadores são movidos para o purgatório(ZonaX).

- **Purgatório**
  Classe que representa a zona de purgatório. Todos os trabalhadores que se despedem são movidos para esta zona onde irão ficar a trabalhar para o resto da vida, fornecendo 5€ ao jogador por trabalhador na zona.

Comandos

Esta classe foi criada de forma a facilitar a execução e verificação de comandos escritos pelo utilizador, por exemplo: cons, cont, list etc.

Tipo	Nome	Objetivo
Ilha&	refIlha	Armazenar uma referência para a classe Ilha onde irão ser executados os comandos
Save	savess	Guarda os saves para futura utilização no comando “save” e “load”
Interface	interface	Guarda a class interface para avisar de quaisquers erros que possam ter acontecido

Esta classe irá dar origem a um conjunto de outras classes derivadas que irão dar override da função “**execComando()**” de modo a cada uma ter uma função diferente

Classes derivadas de Recursos:

- **Constroi**
  Classe que representa o comando “**cons <tipo> <linha> <coluna>**” que verifica se os parâmetros estão corretos e trata de remover os recursos e construir o edificio especificado caso seja necessário

- **Contrata**
  Classe que representa o comando “**cont <tipo>**” que verifica se os parâmetros estão corretos e trata de remover os recursos e contratar o trabalhador especificado colocando-o numa zona do tipo “Pastagem”

- **List**
  Classe que representa o comando “**list <linha> <coluna>**” e “**list**” que verifica se os parâmetros estão corretos e de demonstrar a zona especificada nos parâmetros, caso estejam errados simplesmente mostra o tabuleiro todo através da sua função privada **printTab().**

- **Vende**
  Classe que representa o comando “**vende <linha><coluna>**” que verifica se os parâmetros estão corretos e trata de vender o edificio especificado e adicionar os recursos respetivos. Caso falhe tenta chamar o comando “vende <tipo> <quantidade>” que irá vender uma certa quantidade do recursos especificado.

- **Liga**
  Classe que representa o comando ”**liga <linha> <coluna>**” que verifica se os parâmetros estão corretos e liga o edificio(caso exista) nessa zona.Se o edificio ja estiver ligado este permanece ligado.

- **Desliga**
  Classe que representa o comando ”**desliga <linha> <coluna>**” que verifica se os parâmetros estão corretos e desliga o edificio(caso exista) nessa zona.Se o edificio ja estiver desligado este permanece desligado.

- **Debash**
  Classe que representa o comando ”**debcash <valor>**” , como é um comando de debugging não possui verificações e apenas serve para acrescentar / remover dinheiro.

- **Debed**
  Classe que representa o comando ”**debed <tipo> <linha> <coluna>**” , como é um comando de debugging não possui verificações de custos e apenas verifica se já existe um edificio nessa zona. Caso não exista constroi la um de graça.

- **Debkill**
  Classe que representa o comando ”**debkill <id>** ” , remove o trabalhador que corresponde ao id passado no comando.

- **Move**
  Classe que representa o comando ”**move <id> <linha> <coluna>**” que verifica se os parâmetros estão corretos e move o trabalhador com esse id para a zona especificada.

- **Upgrade**

Classe que representa o comando ”**upgrade <linha> <coluna>**” que verifica se os parâmetros estão corretos e se o jogador possui recursos. Upgrada o edifício caso ele tenha upgrades disponíveis.

- **Config**
  Classe que representa o comando ”**config <nome>**” que verifica se os parâmetros estão corretos e se o ficheiro existe. Caso existe lê o ficheiro linha a linha fazendo update dos custos dos edificios e trabalhadores parando quando encontra uma linha mal formatada ou chega ao fim do ficheiro.

- **Save**
  Classe que representa o comando ”**save <nome>**” que salva o estado do jogo em memória.

- **Loadgame**
  Classe que representa o comando ”**load <nome>**” que carrega o save com o nome especificado caso ele exista.

- **Deletesave**
  Classe que representa o comando ”**apaga <nome>**” que apaga o save com o nome especificado caso ele exista.

Dias

Esta classe abstrata foi criada com o propósito de executar o “loop” do programa. Ela dá origem a 3 outras classes que herdam dela e fazem um override das funções herdadas, nomeadmente a funçao **cloneDias(),** que irá servir para criar dias especificos sem necessitar de referir o nome do dia e bastando utilizar o ponteiro para a classe Dias, e a função **executa(Jogo& jogo)** que cada dia irá mudar consoante a sua necessidade.

Tipo	Nome	Objetivo
Ilha&	ilha	Referência para a ilha para executar ações nel
Comandos&	comandos	Referência para a classe comandos onde estão todos os comandos do jogo e as suas proteções
static int	nDia	Inteiro que é incrementado sempre que um dia passa(comando next)
Interface	handler	Classe interface para permitir uma comunicação de mensagens de erro ao UI

Classes derivadas de Dias:

- **Manha**
  Classe que irá dar override da função **executa()** para executar todos os eventos presentes na ilha(desabamentos de edificios,despedir trabalhadores,etc).
- **MeioDia**
  Classe que irá dar override da função **executa()** de forma a intrepertar e verificar se um comando existe e caso exista executa-lo através de um mapa de comandos.
- **Noite**
  Classe que irá dar override da função **executa()** para executar a extração de todos os recursos e a sua adição aos recursos do jogador.

Recursos

Esta classe abstrata foi criada de forma a proporcionar um facil armazenamento de recursos que o jogo utiliza. Possui funções para saber o valor de cada recurso**(getValor())**, a sua abreviatura**(getAbreviatura())** e permite alterar e obter a sua quantidade**(setRecurso(const float quant)/getRecurso()).**
Possui também uma função **cloneRecursos()** que permite criar os recursos utilizando o polimorfismo

Tipo	Nome	Objetivo
float	quantidade	Armazena a quantidade de um certo tipo de recurso
float	valor	Armazena o valor de um certo tipo de recurso em €/kg
string	abreviatura	Guarda a abreviatura de cada recurso(ferro,carvao,din,etc)

Classes derivadas de Recursos:

- **Ferro**
  Classe que representa o ferro
- **BarradeAco**
  Classe que representa as barras de aço
- **Carvao**
  Classe que representa o carvao
- **Madeira**
  Classe que representa a madeira
- **VigasDeMadeira**
  Classe que representa as vigas de madeira
- **Eletricidade**
  Classe que representa a eletricidade
- **Dinheiro**
  Classe que representa o dinheiro

Armazem

A classe armazem foi criada para permitir aos edificios herdarem e usarem as suas funções de modo a todos possuirem um “armazém” onde guardam os recursos. Possui a função **removeRecurso(const float quant)** que permite remover uma certa quantidade de recursos do armazém **aumetaArmazenado(const float quant)** que permite aumentar uma certa quantidade de recursos no armazém e por último possui a função **getArmazenado()** que permite obter a quantidade de recursos que o armazém tem dentro de si.

Tipo	Nome	Objetivo
float	armazenamento	Guarda o número de recursos armazenado neste armazém
float	capacidade	Guarda a capacidade máxima que este armazém possui

A classe armazem também acaba por ser utilizada pelas zonas que necessitam de armazenar recursos, por isso foi feita como uma classe à parte ao invés de a incorporar diretamente na zonas e nos edificios, pois a lógica por detrás dos dois é a mesma.

Assim se aparecer alguma outra classe que necessite de um funcionalidade de armazenamento basta herdar a class armazém o que evita a repetição de código desnecessária

Edificios

Esta é a classe abstrata que contêm toda a lógica e implementação do funcionamento dos edificios presentes no jogo. Contêm funções para os ligar **(setLigado(const bool estado)),** obter o seu custo, executar o seu "evento"(por exemplo as minas desabarem), etc. A função mais importante, de entre todo o conjunto de funções que esta classe possui, acaba por ser a função **extrai().** Esta função é alterada consoante o tipo de edíficio mas a sua função "básica" é utilizar o edificio para extrair/criar recursos.

Tipo	Nome	Objetivo
bool	podeMelhorar	Permite saber se o edificio possui mais upgrades ou não
int	nivel	Guarda o nivel do edificio
bool	ligado	Guarda o estado do edificio, ligado/desligado
array<Recursos*,7>	recursos	Array com ponteiros para os recursos do jogo
vector<tuple<Recursos*,float>>	upgradeCusto	Vetor de tuples que contêm o recurso que é preciso para upgradar o edificio e a sua quantidade em float
Zona*	zona	Ponteiro para a zona onde o edificio está localizado

map<tuple<Recursos*,Recursos*>,tuple<float,float>	custo	Mapa com dois tuples, um que guarda o recurso principal e o recurso de substituição(NULL caso nao haja) e outro com a quantidade de cada recurso

Classes derivadas de Edificios:

- **MinaFerro**
  Classe que representa o edificio responsável pela extração do recurso **"Ferro"**, possui um evento chamado pela função **evento()** em que a mina tem uma chance de 15% de desabar e desaparecer, a sua função **extrai()** é bastante simples vendo apenas se existem mineiros na zona e se existem cria 2kgs de ferro + 1kg por cada nivel que possui(começando ao nivel 0) e multiplicando pelo multiplicador da zona onde ele se encontra(obtido pela funçao **getMultiplier(Recurso*)**). Tem uma capacidade de armazenamento de 100kg

- **MinaFerro**
  Classe que representa o edificio responsável pela extração do recurso **"Carvão"**, é um edificio praticamente igual ao anterior porém a sua chance de desabar é de apenas 10% e o seu custo de upgrade é menor (10€ e uma viga vs 15€ e uma viga).

- **CentralEletrica**
  Classe que representa o edificio responsável pela criação do recurso **"Eletricidade"** e **"Carvão"**, é um edificio que se destaca dos outros pois a sua função de extração é a mais complexa, requerendo a utilização de 3 mapas. O mapa "funcs" que contém ponteiros para funçoes Armazem(**removeRecurso()** e **aumenta Armazenado()**), este irá ser utilizado para saber o que fazer com cada zona e edificio adjacente(se irá aumentar o remover recursos nessa zona), o mapa "func_parameters", que contêm a quantidade a ser adicionada ou removida pelo mapa "funcs" e o mapa "relacoes" que ira servir como guia para a função **extrai()** saber se sucedeu em tudo ou se falhou.

- **Fundição**
  Classe que representa o edificio responsável pela criação do recurso **“Barras de aço”.** Funciona de igual forma ao edificio “**CentralEletrica**” sendo a unica diferença o tipo de recurso que cria e a quantidade e tipo de recursos que usa para o criar.

- **Bateria**
  Classe que representa o edificio responsável pelo armazenamento gerado pelo edificio “**CentralEletrica**”

- **Serraria(EdificioX)**
  Classe que representa o edificio responsável pela criação do recurso **“Vigas de Madeira”.** Funciona de igual forma ao edificio “**CentralEletrica**” sendo a unica diferença o tipo de recurso que cria e a quantidade e tipo de recursos que usa para o criar.

Trabalhadores

A classe Trabalhadores contém todas as funçoes necessárias ao funcionamento dos trabalhadores no jogo. Contêm funçoes para verificar se eles podem mudar de zona, para executar um evento(demissão),obter o seu custo e para verificar se estão a descansar.

Tipo	Nome	Objetivo
bool	descanso	Permite saber se o trabalho se encontra em descanso
char	simbolo	Simbolo que representa o trabalhador
int	diasDespedir	Quantos dias até se poderem despedir
int	diaContrato	Dia em que foi contratado
int	identificador	Id que identifica o trabalhador
Zona*	zona	Ponteiro para a zona onde o trabalhador está localizado
static int	id	Numero incrementado a cada novo trabalhador
bool	mover	Permite saber se o trabalhador pode ja foi movido
int	pagamento	Custo de contratar o trabalhador
float	probIrEmbora	Chance de o trabalhador se despedir

Classes derivadas de Trabalhadores:

- **Mineiros**
  Classe que representa o trabalhador "**Mineiro**" que trabalha nas minas, tem uma percentagem de 10% de se despedir a partir do segundo dia de contrato e custa 10€ para contratar.

- **Lenhadores**
  Classe que representa o trabalhador "**Lenhador**" que trabalha nas florestas, tem uma percentagem de 0% de se despedir(a não ser que uma zona lhe afete esta chance e aumente) a partir do segundo dia de contrato e custa 20€ para contratar. Possui a caracteristica unica de não trabalhar no 5º dia.

- **Operario**
  Classe que representa o trabalhador "**Mineiro**" que trabalha nos edificios industriais(fundição, serraria, central eletrica), tem uma percentagem de 5% de se despedir a partir do 10º dia de contrato e custa 15€ para contratar.

Save

Esta classe é responsável pelo armazenamento, criação e carregamento de saves para o jogo. Tem 3 funções **savegame()**, **loadgame()** e **deletesave()** para este efeito.

Tipo	Nome	Objetivo
static vector<tuple<string,Jogo*>>	saves	Armazena os saves que o jogador faz ao longo do jogo, guardando um nome e associando lhe um ponteiro para a class Jogo guardada

Jogo

Esta é a classe principal que armazena todos os aspetos importantes para o jogo funcionar. É constituida por funções que lhe permitem associar uma interface(UI) através do "**observer pattern**" e notifica essas interfaces de quaisquer alterações que tenham ocorrido. Também é responsavel por criar e armazenar os recursos do jogo e os trabalhadores e edificios que irão ser usados com padrão.

Tipo	Nome	Objetivo
Ilha	ilha	Guarda a classe Ilha onde o jogo irá decorrer
Comandos	comando	Guarda uma referência a classe Comandos onde irá executar os comandos
vector<Dias*>	dias	Guarda o conjunto de dias por onde o jogo irá dar "loop"
array<Recursos*,7>	listaRecursos	Array com ponteiros para os recursos
map<string,Edificios*>	tipos	Guarda um mapa com os tipos de edificios que existem no jogo onde os outros se irão basear

map<string,Trabalhadores*>	tiposTrabalhadores	Guarda um mapa com os tipos de trabalhadores que existem no jogo onde os outros se irão basear

UI

Esta uma classe que herda de uma classe "**observer**" e que apenas serve para mostrar informação ao utilizador. Não é uma classe totalmente completa pois certas informações ainda são mostradas através de "**couts**" dentro da classe dos comandos(comando "**list**" por exemplo). Ela faz uso da class Interface onde o resto do programa guarda mensagems de erro ou simples informações como "**edificio construido**".