

2022_1 - PROGRAMAÇÃO E DESENVOLVIMENTO DE SOFTWARE II - TA_TN - METATURMA

PAINEL > MINHAS TURMAS > 2022_1 - PROGRAMAÇÃO E DESENVOLVIMENTO DE SOFTWARE II - TA_TN - METATURMA > GERAL
> L02E03 - CAMPO MINADO (3,0 PTS)


 Descrição



 Enviar

 Editar

 Visualizar envios

L02E03 - Campo Minado (3,0 pts)

 Data de entrega: sexta, 1 Jul 2022, 23:59

 Arquivos requeridos: Coordenada.hpp, Coordenada.cpp, Bloco.hpp, Bloco.cpp, BlocoMina.hpp, BlocoMina.cpp, BlocoContador.hpp, BlocoContador.cpp ( [Baixar](#))

Tipo de trabalho:  Trabalho individual

Polimorfismo Básico	
VPL: 3	Nome: Campo Minado

Objetivo:

Seu objetivo neste exercício é usar os conceitos de polimorfismo para simular uma parte de um jogo de campo minado. No final, sua aplicação deverá permitir que um tabuleiro de campo minado seja implementado e o jogador seja capaz de revelar os blocos deste campo e ver as consequências relativas de cada bloco revelado. Neste jogo **os blocos irão se auto gerenciar**, ao invés do tabuleiro gerenciar os blocos. Não será necessário implementar a *main.cpp*. Você deve considerar que o tabuleiro do jogo será passado como parâmetro para que o bloco possa fazer ações sobre ele.

Você pode baixar a *main.cpp* [aqui](#) se desejar reproduzir o exercício em sua máquina.

TADs do seu programa:

Classe Coordenada	
Atributos:	private int row → Armazena o número da linha da coordenada na matriz. private int col → Armazena o número da coluna da coordenada na matriz.
Métodos: (Todos os métodos descritos abaixo devem ser públicos)	
Coordenada() → Construtor de coordenadas Coordenada(int _row, int _col) → Construtor da coordenada que inicializa suas propriedades void getCoordenadasAdjacentes(std::vector<Coordenada>& adjacentes, int rowBoundary, int colBoundary) → Preenche o vector "adjacentes" com as coordenadas adjacentes à coordenada. rowBoundary e colBoundary são a quantidade de linhas e colunas do tabuleiro, respectivamente. Eles devem auxiliar nesse processo para que não sejam retornadas coordenadas que vão além do tamanho do tabuleiro. Dica1: as coordenadas adjacentes são aquelas que estão nas linhas/colunas logo antes ou logo em seguida da linha/coluna atual. Dica2: Você consegue pegar os limites diretamente pelo tabuleiro.	

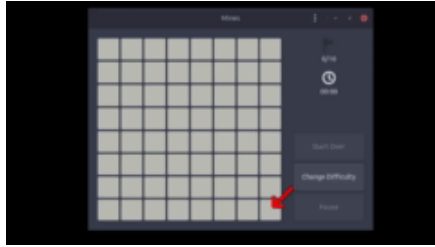
Classe Bloco	
Atributos:	protected Coordenada coord → Coordenada do bloco protected bool revelado → Indicador se o bloco foi revelado ou não protected int valor → Indicador do tipo do bloco: -1, se é uma bomba, 0 se nenhum de seus blocos adjacentes possui uma bomba (ou seja, não é Contador) > 0 caso nenhuma das alternativas anteriores se aplique (ou seja, é Contador)
Métodos: (Todos os métodos descritos abaixo devem ser públicos)	

Bloco(Coordenada _coord) → Construtor de blocos que inicializa suas propriedades

virtual ~Bloco() → Destrutor de blocos, o virtual faz com que o destrutor de classes derivadas também sejam acessados. Não precisa de nenhuma especificação adicional

virtual bool revelar(std::vector<std::vector<Bloco*>>& tabuleiro) → Revela o bloco no tabuleiro fazendo com que o atributo "revelado" seja true. Deve retornar true indicando que o jogo pode continuar.

Ao revelar um bloco da classe Bloco, ele deve revelar também todos seus blocos adjacentes. Note que se um desses blocos adjacentes for da classe Bloco, é causado um efeito cascata automático e os adjacentes dele também serão revelados. Atenção, só deve-se tentar revelar blocos ainda não revelados!



Dica: use o método getCoordenadasAdjacentes para recuperar e acessar diretamente as posições vizinhas no tabuleiro.

std::string getSimbolo() → Retorna o símbolo que representa o bloco:

"#" se o bloco ainda não foi revelado,

"*" caso seja uma bomba",

valor do bloco caso nenhuma das opções anteriores sejam atendidas.

bool ehRevelado() → Retorna um indicador se o bloco já foi revelado ou não

Classe BlocoContador : Bloco

Atributos: Nenhum novo atributo

Métodos: (Todos os métodos descritos abaixo devem ser públicos)

BlocoContador(Coordenada _coord) → Construtor de blocos que inicializa suas propriedades

virtual bool revelar(std::vector<std::vector<Bloco*>>& tabuleiro) override → Revela o bloco no tabuleiro fazendo com que o atributo "revelado" seja true, também retorna true indicando que o jogo pode continuar.

virtual ~BlocoContador() → Não precisa de nenhuma especificação adicional

void incrementarValor() → Incrementa o valor do bloco. Note que o "valor" no bloco contador representa a quantidade de bombas que tem em volta dele.

Classe BlocoMina : Bloco

Atributos: Nenhum novo atributo

Métodos: (Todos os métodos descritos abaixo devem ser públicos)

BlocoMina(std::vector<std::vector<Bloco*>>& tabuleiro, Coordenada _coord) → Ao criar um BlocoMina **m** devemos fazer a seguinte análise:..

Se um bloco **b** for adjacente ao **BlocoMina m** e **b** for da classe **Bloco**, então você deve criar e colocar um novo BlocoContador **bc** no lugar de **b**. Lembre-se de liberar a memória de **b** antes de descartá-lo. O valor de **bc** já deve ser incrementado em 1, pois ele é adjacente a uma bomba.

Observe que o atributo "tipo" pode ajudar a identificar qual o tipo do bloco (**Bloco**, **BlocoMina** ou **BlocoContador**).

O tabuleiro armazena blocos do tipo **Bloco***, e o método de incrementarValor() é específico do tipo **BlocoContador**. Se um bloco **b** é do tipo **BlocoContador*** (ou seja, o atributo "valor" é > 0), então você pode convertê-lo para um ponteiro **BlocoContador*** usando o **dynamic_cast** do C++ para fazer a conversão. Exemplo:

```
Bloco* b = new BlocoContador(Coordenada());
BlocoContador* bc = dynamic_cast<BlocoContador*>(b);
```

virtual ~BlocoMina() → Não precisa de nenhuma especificação adicional

virtual bool revelar(std::vector<std::vector<Bloco*>>& tabuleiro) override → Revela o bloco no tabuleiro e todos os outros blocos que existem e ainda não foram revelados. Deve sempre retornar false, já que ao revelar uma bomba o jogador perde o jogo e não pode continuar.

Você tem liberdade para implementar quaisquer outros métodos na TAD que julgar necessário. Lembre-se que getters e setters podem ser importantes quando atributos são privados ou protegidos e precisamos acessá-los de fora da classe.

Exemplos de entrada e saída:

Exemplo 1

Entrada:

```
2
b 0 0
b 1 0
e
r 0 1
r 1 0
```

Saída:

```
##
##
```

```
# 2
##
```

```
* 2
* 2
```

Você perdeu!

Exemplo 2	
Entrada: 4 b 0 0 b 0 2 b 3 3 e r 0 1 r 2 1 r 1 3 r 2 3 r 0 3 e	Saída: # # # # # # # # # # # # # # # # # 2 # # # # # # # # # # # # # # # 2 # # 1 2 1 # 0 0 1 # 0 0 1 # # 2 # # 1 2 1 1 0 0 1 # 0 0 1 # # 2 # # 1 2 1 1 0 0 1 1 0 0 1 # # 2 # 1 1 2 1 1 0 0 1 1 0 0 1 # Jogo encerrado!

- Links Úteis:
- [Explicação do jogo](#) na Wikipedia
 - [Goodle Doodle](#) do jogo para ter uma ideia de como o jogo funciona
 - [Dynamic_cast](#)

[VPL](#)

◀ [L02E02 - Makefile \(1,0 pt\)](#)

Seguir para...

[L02E04 - Conversor de Arquivos \(4,0 pts\)](#) ▶