Traços de Execução do Código

Eloísa Bazzanella e Maria Eduarda Buzana

Criação da Malha

Classe: Main

```
private void onClickCarregar(java.awt.event.ActionEvent evt) {
    int fileChooseResponsee = FileChooser.showSaveDialog(null);

    if (fileChooseResponsee == JFileChooser.APPROVE_OPTION) {
        File arquivoSelecionado = new File(FileChooser.getSelectedFile().getAbsolutePath());

        try {
            controllerSimulacao.carregaSimulacao(arquivoSelecionado);
            ViewSimulacao telaSimulacao = new ViewSimulacao(controllerSimulacao);
            telaSimulacao.setVisible(true);

        } catch (IOException ex) {
            Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

Classe: ControllerSimulacao

```
  public void carregaSimulacao(File arquivo) throws IOException {
      this.arquivo = arquivo;

      this.criaMalha(false);
      this.inicializarMalha();
      notifyTableModel(new TableModelMalha(this));
  }
```

```
public void criaMalha(boolean usaMonitor) throws FileNotFoundException, IOException {
    BufferedReader in = new BufferedReader(new FileReader(arquivo));
    this.linhas = Integer.parseInt(in.readLine());
    this.colunas = Integer.parseInt(in.readLine());

    this.factoryMalha(false);

    for (int linhaAtual = 0; linhaAtual < linhas; linhaAtual++) {
        String[] listaTipos = in.readLine().split("\t");
        for (int colunaAtual = 0; colunaAtual < colunas; colunaAtual++) {
            int tipoPista = Integer.parseInt(listaTipos[colunaAtual]);
            PosicaoPista posicaoPista = new PosicaoPista(linhaAtual, colunaAtual);
            malhaViaria.adicionarItemPista(posicaoPista, new ItemPista(tipoPista, posicaoPista
        }
    }
}
```

```
public void factoryMalha(boolean usaMonitor) {
    if(usaMonitor) {
        AbstractMalhaFactory factory = new MalhaViariaMonitorFactory();
        malhaViaria = factory.criarMalha(linhas, colunas);
    } else {
        AbstractMalhaFactory factory = new MalhaViariaSemaforoFactory();
        malhaViaria = factory.criarMalha(linhas, colunas);
    }
}
```

No onClickCarregar, ele busca o arquivo e chama o carregarSimulacao que cria a malha conforme as informações do .txt. Cada ponto no plano cartesiano da malha, foi definido como ItemPista, portanto a malha é composto por várias ItemPista. Inicialmente a malha é criada para utilizar Semáforo, mas se o usuário posteriormente trocar para Monitor, a malha é somente recriada para usar Monitor, para a criação foi utilizado o Factory.

Renderização da JTable

Classe: TableModelMalha

```
public class TableModelMalha extends AbstractTableModel {

    private InterfaceControllerObserved controller;

    public TableModelMalha(InterfaceControllerObserved controller) {
        this.controller = controller;
    }

    @Override
    public int getRowCount() {
        return this.controller.getMalhaRodoviaria().length;
    }

    @Override
    public int getColumnCount() {
        return this.controller.getMalhaRodoviaria()[0].length;
    }

    @Override
    public Object getValueAt(int rowIndex, int columnIndex) {
        return this.controller.getMalhaRodoviaria()[rowIndex][columnIndex];
    }

}
```

Classe: DefaultTableCellRenderMalha

```
public class DefaultTableCellRendererMalha extends DefaultTableCellRenderer {
```

A malha visualmente nada mais é do que uma JTable, em que cada célula, ou seja, ItemPista, é renderizada com um icone definido conforme o tipo que consta no .txt, assim é possível visualizar graficamente os veículos e as pistas, bem como as graminhas. No nosso caso ainda adicionamos uns detalhezinhos randomicamente na grama para ficar mais agradável visualmente.

Inicialização da Simulação

Classe: ControllerSimulacao

```
public void iniciarSimulacao(int quantidadeVeiculo, boolean usaSemaforo, boolean usaMonitor) throws
    notifyButtonChanged(true);

    if(usaMonitor) {
        this.malhaViaria = null;
        this.criaMalha(true);

        this.inicializarMalha();
        notifyTableModel(new TableModelMalha(this));
    }

    new Thread(() -> {
        while (start) {
            if (this.veiculos.size() < quantidadeVeiculo) {
                iniciarNovoVeiculo(malhaViaria.entradaAleatoria());
                notifyTableModelChanged();
            }
            sleepThread();
        }
    }).start();
}
```

Ao clicar no botão de iniciar a simulação, o sistema verifica se todos os parâmetros necessários foram informados, e caso não tenham sido, retorna uma mensagem de alerta. Caso o usuário tenha informado tudo corretamente, a simulação inicia verificando se é semáforo ou monitor. Se for semáforo, continua como está pois a malha já foi criada para utilizar semáforo, caso contrário a malha é recarregada. Depois, inicia-se a entrada de veículos aleatoriamente nos pontos de entrada, respeitando a quantidade máxima informada pelo usuário.

Definição dos pontos de Entrada de Saída

Classe: MalhaViaria

```java
private void definirEntradasSaidas() {
    for (int linhaAtual = 0; linhaAtual < linhas; linhaAtual++) {
        for (int colunaAtual = 0; colunaAtual < colunas; colunaAtual++) {
            ItemPista itemPistaAtual = malha[linhaAtual][colunaAtual];

            adicionarPistasAdjacentes(itemPistaAtual, new PosicaoPista(linhaAtual, colunaAtual));

            if (primeiraLinha(linhaAtual)) {
                definirEntradaSe(itemPistaAtual, ESTRADA_BAIXO);
                definirSaidaSe(itemPistaAtual, ESTRADA_CIMA);
            }

            if (ultimaLinha(linhaAtual)) {
                definirEntradaSe(itemPistaAtual, ESTRADA_CIMA);
                definirSaidaSe(itemPistaAtual, ESTRADA_BAIXO);
            }

            if (primeiraColuna(colunaAtual)) {
                definirEntradaSe(itemPistaAtual, ESTRADA_DIREITA);
                definirSaidaSe(itemPistaAtual, ESTRADA_ESQUERDA);
            }

            if (ultimaColuna(colunaAtual)) {
                definirEntradaSe(itemPistaAtual, ESTRADA_ESQUERDA);
                definirSaidaSe(itemPistaAtual, ESTRADA_DIREITA);
            }
        }
    }
}
```

```java
private void definirEntradaSe(ItemPista itemPistaAtual, int direcaoPistaAtual) {
    if (itemPistaAtual.getTipo() == direcaoPistaAtual) {
        itemPistaAtual.setIsEntrada(true);
        adicionarNovaEntrada(itemPistaAtual);
    }
}

private void definirSaidaSe(ItemPista itemPistaAtual, int direcaoPistaAtual) {
    if (itemPistaAtual.getTipo() == direcaoPistaAtual) {
        itemPistaAtual.setIsSaida(true);
    }
}
```

Para a definição dos pontos de entrada e saída, foram analisadas as pistas na beirada da malha e seu sentido de direção.

Criação dos Veículos

Classe: ControllerSimulacao

```java
public void iniciarNovoVeiculo(ItemPista itemPistaAtual) {
    Veiculo veiculo = new Veiculo(malhaViaria);
    this.addVeiculo(veiculo);
    veiculo.setPistaAtual(itemPistaAtual);
    new Thread(veiculo::start).start();
}
```

Classe: Veiculo

```
public synchronized void start() {
    while (ControllerSimulacao.getInstance().isRunning()) {
        try {
            if (this.pistaAtual.isSaida()) {
                this.pistaAtual.setOcupada(false);
                this.pistaAtual.setVeiculo(null);

                removeVeiculo();

                return;
            }

            if (this.percurso == null || !this.percurso.hasNext()) {
                this.percurso = this.getPercurso().iterator();
            }

            ItemPista proximaPista = this.percurso.next();

            this.pistaAnterior = this.pistaAtual;
            this.pistaAnterior.setVeiculo(null);
            this.pistaAnterior.setOcupada(false);

            this.pistaAtual = proximaPista;

            malhaViaria.adicionarVeiculo(pistaAtual.getPosicaoPista(), this);

            ControllerSimulacao.getInstance().notifyTableModelChanged();

            Thread.sleep(this.getVelocidade());
```

Ao iniciar um novo veículo, ele adicionar esse veiculo num array de veículos ativos na malha e define a sua pista atual. Então ele dá um start no veículo. Enquanto o veículo se movimenta na malha, ele sempre verifica se chegou já num ponto de saída, para então ser removido o carro, caso contrário, ele monta o percurso e executa o percurso, enquanto isso atualiza a tableModel da tela. E por fim, ele dá um thread.sleep conforme a velocidade definida aleatoriamente.

Montagem do Percurso

Classe: PercursoPistaBase

```java
@Override
public List<ItemPista> getPercurso(ItemPista pistaAtual, ItemPista pistaAnterior) {
    List<ItemPista> percurso = new ArrayList<>();
    ItemPista proximaPista = null;

    do {
        proximaPista = this.getProximaPista(pistaAtual, pistaAnterior);
    } while (proximaPista.isOcupada() || !proximaPista.isTransitavel());

    proximaPista.setOcupada(true);

    percurso.add(proximaPista);

    if(proximaPista.isCruzamento()) {
        percurso.addAll(this.getPercurso(proximaPista, pistaAtual));
    }

    return percurso;
}

public ItemPista getProximaPista(ItemPista pistaAtual, ItemPista pistaAnterior) {
    ItemPista pista = null;

    switch (pistaAtual.getTipo()) {
        case 1: {
            if(pistaAtual.getPistaDireita().getTipo() == pistaAtual.getTipo()){
                Random random = new Random();
                int opcao = random.nextInt(5);

                if (opcao == 1) {
                    pista = pistaAtual.getPistaDiagonalCimaDireita();
                } else {
                    pista = pistaAtual.getPistaCima();
                }
            }
            else if(pistaAtual.getPistaEsquerda().getTipo() == pistaAtual.getTipo()) {
                Random random = new Random();
                int opcao = random.nextInt(5);

                if (opcao == 1) {
                    pista = pistaAtual.getPistaDiagonalCimaEsquerda();
                } else {
                    pista = pistaAtual.getPistaCima();
                }
            }
            else {
                pista = pistaAtual.getPistaCima();
            }
            break;
        }
```

```
        case 12: {
            if (pistaAnterior.getTipo() == 10 || pistaAnterior.getTipo() == 12) {
                pista = pistaAtual.getPistaEsquerda();
            } else {
                Random random = new Random();
                int opcao = random.nextInt(2);

                if (opcao == 1) {
                    pista = pistaAtual.getPistaBaixo();
                } else {
                    pista = pistaAtual.getPistaEsquerda();
                }
            }
            break;
        }
    }
```

O percurso é montado sempre verificando se a pista está ocupada, ou seja, se foi reservada por algum outro veículo, ou se já possui algum veículo na pista, quando existe uma dessas opções, ele monta outro percurso. Senão, ele monta o percurso pegando sempre a próxima pista, e em casos de cruzamento, ele vai montando o percurso até sair do cruzamento. Vale ressaltar ainda, que em vias de mão dupla, o veículo ainda pode trocar de pista, essa decisão é tomada aleatoriamente.

```
public MalhaViariaMonitor(int linhas, int colunas) {
    super(linhas, colunas);
}

@Override
public synchronized void adicionarVeiculo(PosicaoPista posicaoPista, Veiculo veiculo) {
    int linha = posicaoPista.getLinha();
    int coluna = posicaoPista.getColuna();
    malha[linha][coluna].setVeiculo(veiculo);
}

@Override
public synchronized void removerVeiculo(PosicaoPista posicaoPista, Veiculo veiculo) {
    int linha = posicaoPista.getLinha();
    int coluna = posicaoPista.getColuna();
    malha[linha][coluna].setVeiculo(null);
    notificarController(veiculo);
}
```

```java
private final Semaphore mutex;

public MalhaViariaSemaforo(int linhas, int colunas) {
    super(linhas, colunas);

    mutex = new Semaphore(1);
}

@Override
public void adicionarVeiculo(PosicaoPista posicaoPista, Veiculo veiculo) {
    try {
        mutex.acquire();
        int linha = posicaoPista.getLinha();
        int coluna = posicaoPista.getColuna();
        malha[linha][coluna].setVeiculo(veiculo);
    } catch (InterruptedException e) {
        e.printStackTrace();
    } finally {
        mutex.release();
    }
}

@Override
public void removerVeiculo(PosicaoPista posicaoPista, Veiculo veiculo) {
    try {
        mutex.acquire();
        int linha = posicaoPista.getLinha();
        int coluna = posicaoPista.getColuna();
        malha[linha][coluna].setVeiculo(null);
        notificarController(veiculo);
    } catch (InterruptedException e) {
        e.printStackTrace();
    } finally {
        mutex.release();
    }
}
```

Ao utilizar Monitor, a inserção e a remoção do veículo na Pista, ou seja, quando ele faz um movimento, são feitas com método "synchronized". E quando é Semáforo, é criado o atributo mutex do tipo Semaphore, e é realizado um "aquire" antes de adicionar ou remover o veículo na pista, ou seja, quando ele está se deslocando para outra pista, e é feito um "release" depois do processo. Isso garante que o veículo esteja reservando aquela pista pra ele durante o processo.

Encerramento da Simulação

Classe: ViewSimulacao

```
private void btFinalizarDefActionPerformed(java.awt.event.ActionEvent evt) {
    this.controllerSimulacao.setStart(false);
    this.controllerSimulacao.setRunning(false);
    this.controllerSimulacao.removeTodosVeiculos();
    this.controllerSimulacao.inicializarMalha();
    this.controllerSimulacao.notifyTableModelChanged();
    this.atualizaButton(false);
```

Ao finalizar a execução, todos os veículos são removidos e a malha é reiniciada para uma próxima execução e, claro, a malha é "repintada" na tela.