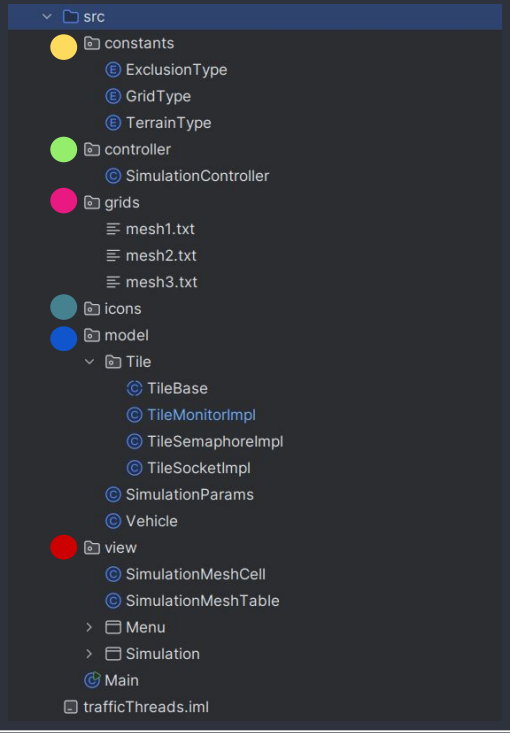
</ Simulador de Tráfego em Malha Viária />

} /> [

Eduardo, Luíza e Maria

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

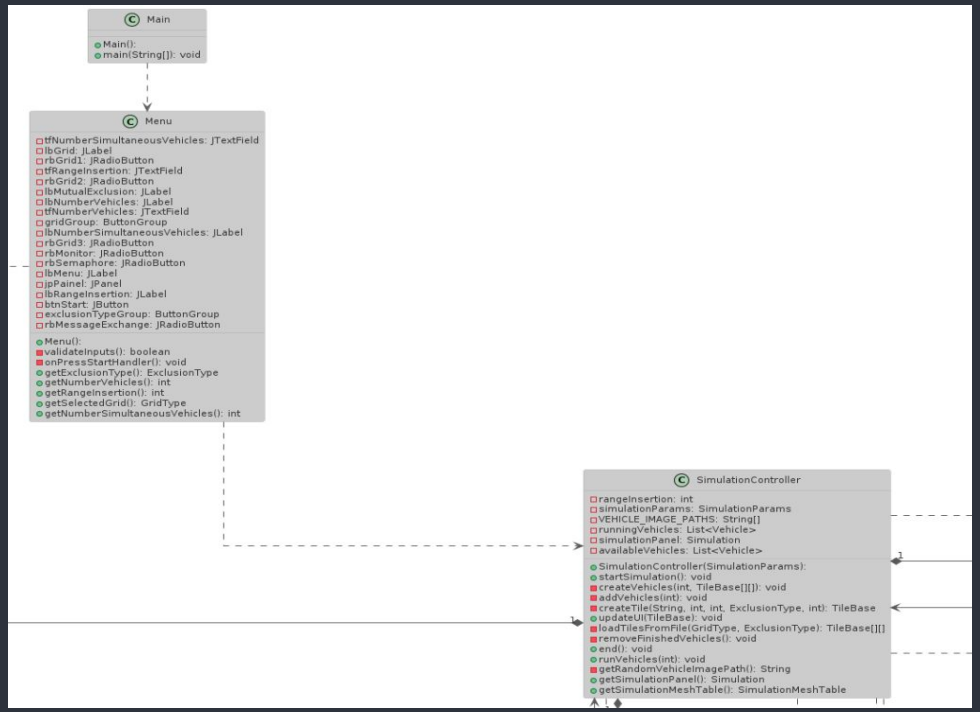
</ Estrutura Geral



- **Constants**
  Tem todas as constantes do sistema
- **Controller**
  Tem toda a gestão dinâmica dos carros dentro da simulação
- **Grids**
  Tem os arquivos TXT das malhas
- **Icons**
  Tem todos os icones de setas, grama e veículos
- **Model**
  Tem os modelos associados a exclusão mútua e veículo
- **View**
  Tem a estrutura com swing associados as telas

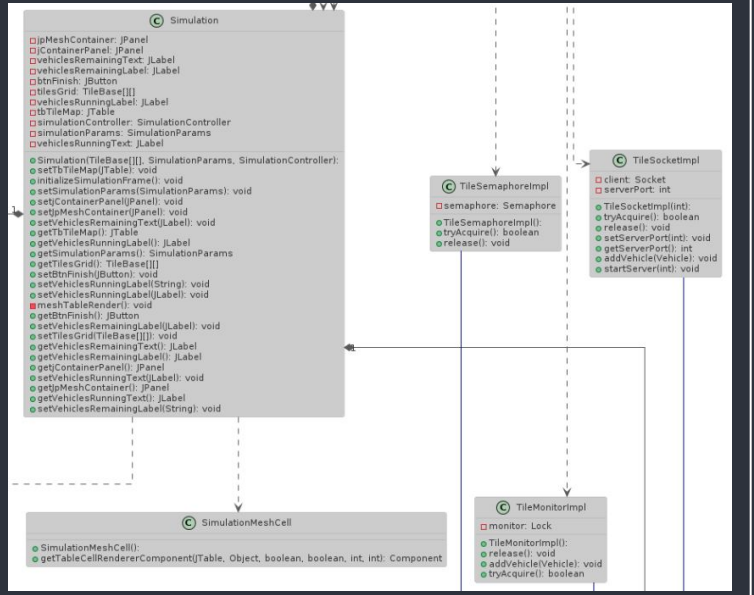
1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Diagrama de Classes



Type	Symbol	Drawing
Extension	<|--	
Composition	*--	
Aggregation	o--	

Character	Icon for field	Icon for method	Visibility
-			private
#			protected
~			package private
+			public



1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 1 0 1 1 1 0 1

</ Diagrama de Classes

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Diagrama de Classes

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Telas

```java
public class Simulation extends JFrame {  6 usages  Eduardo Schork +2
    private void meshTableRender() {  1 usage  Eduardo Schork
    }

    public void initializeSimulationFrame() {  1 usage  maariacecilliaholler +2
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize( width: 650, height: 900);
        setResizable(false);
        setLocationRelativeTo(null);

        jContainerPanel = new JPanel();

        meshTableRender();

        JPanel buttonPanel = new JPanel();
        buttonPanel.setLayout(new BoxLayout(buttonPanel, BoxLayout.Y_AXIS));
        buttonPanel.setAlignmentX(Component.CENTER_ALIGNMENT);

        btnFinish = new JButton( text: "Finalizar");
        btnFinish.setAlignmentX(Component.CENTER_ALIGNMENT);
        buttonPanel.add(btnFinish);

        JPanel vehiclesRunningPanel = new JPanel();
        vehiclesRunningPanel.setAlignmentX(Component.CENTER_ALIGNMENT);
        vehiclesRunningPanel.add(vehiclesRunningText);
        vehiclesRunningPanel.add(vehiclesRunningLabel);
        buttonPanel.add(vehiclesRunningPanel);

        JPanel vehiclesRemainingPanel = new JPanel();
        vehiclesRemainingPanel.setAlignmentX(Component.CENTER_ALIGNMENT);
        vehiclesRemainingPanel.add(vehiclesRemainingText);
        vehiclesRemainingPanel.add(vehiclesRemainingLabel);
        buttonPanel.add(vehiclesRemainingPanel);

        add(tbTileMap, BorderLayout.CENTER);
        add(buttonPanel, BorderLayout.SOUTH);
```

Gerencia a exibição do mapa de tiles onde os veículos se movem e fornece controles para interação do usuário.

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Veículo

Gerencia o movimento do veículo ao longo de um caminho predefinido, considerando os diferentes tipos de tiles no mapa, como cruzamentos e estradas.

Por meio de métodos como setupVehicle para configuração inicial, run para execução da lógica de movimento e generateTilePath para geração de um caminho aleatório

```
public class Vehicle extends Thread {  17 usages  Eduardo Schork +2 *
    protected TileBase currentTile;  6 usages
    protected ArrayList<TileBase> path = new ArrayList<>();  8 usages
    protected String imagePath;  2 usages
    protected int vehicleSpeed;  2 usages
    protected TileBase[][] tileMap;  1 usage
    protected SimulationController simulationController;  3 usages
    protected boolean ended;  2 usages

    public Vehicle(String imagePath, SimulationController simulationController) {
        this.imagePath = imagePath;
        this.vehicleSpeed = generateRandomVehicleSpeed(100, 300);
        this.simulationController = simulationController;
    }

    public void setupVehicle(TileBase[][] tileMap) {  1 usage  Eduardo Schork
        ArrayList<TileBase> tilePath = generateTilePath(tileMap);

        this.tileMap = tileMap;
        this.path = tilePath;
    }

    public boolean isVehicleStopped() {  1 usage  Eduardo Schork
        return this.path.isEmpty();
    }

@   private ArrayList<TileBase> findCrossingTiles() {  1 usage  Eduardo Schork
        ArrayList<TileBase> crossingTiles = new ArrayList<>();

        for (int i = 0; i < this.path.size(); i++) {
            TileBase tile = this.path.get(i);
            crossingTiles.add(tile);

            if (!tile.isCrossing()) {
                break;
            }
        }
```

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Monitor 

```java
import ...

public class TileMonitorImpl extends TileBase {  2 usages  maariaceciliaholler +1
    private final Lock monitor = new ReentrantLock();  2 usages

    @Override  2 usages  maariaceciliaholler +1
    public boolean tryAcquire() {
        boolean acquired = false;
        try {
            acquired = this.monitor.tryLock(generateRandomCooldown(1000, 2000), TimeUnit.MILLISECONDS);
        } catch (InterruptedException e) {
            System.out.println(e.getStackTrace());
        }
        return acquired;
    }

    @Override  4 usages  maariaceciliaholler +1
    public void release() {
        try {
            this.monitor.unlock();
        } catch (Exception e) { }
    }

    @Override  1 usage  maariaceciliaholler +1
    public void addVehicle(Vehicle vehicle) {
        synchronized (this) {
            this.vehicle = vehicle;
            this.setTileCurrentImage();
        }
    }
}
```

Utiliza um monitor para controle de acesso exclusivo por veículos. Ela implementa métodos para tentar adquirir o monitor dentro de um limite de tempo, liberar o monitor e adicionar um veículo ao tile de forma sincronizada.

Essa abordagem garante que apenas um veículo por vez possa ocupar o tile, proporcionando um comportamento seguro e coordenado dentro da simulação de tráfego.

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Semáforo 

Utiliza um semáforo para controlar o acesso exclusivo por veículos.

Ela inicializa um semáforo com uma permissão e implementa métodos para tentar adquirir o semáforo dentro de um limite de tempo e liberar o semáforo.

```java
import java.util.concurrent.Semaphore;
import java.util.concurrent.TimeUnit;

public class TileSemaphoreImpl extends TileBase {  2 usages  Eduardo Schork
    private final Semaphore semaphore = new Semaphore( permits: 1);  2 usages

    @Override  4 usages  Eduardo Schork
    public void release() {
        this.semaphore.release();
    }

    @Override  2 usages  Eduardo Schork
    public boolean tryAcquire() {
        if (super.vehicle != null) {
            return false;
        }

        boolean acquired = false;

        try {
            acquired = this.semaphore.tryAcquire(generateRandomCooldown(1000, 2000), TimeUnit.MILLISECONDS);
        } catch (InterruptedException e) {
            System.out.println("Failed to acquire Tile. Vehicle cannot move to tile: " + this);
        }

        return acquired;
    }
}
```

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Troca de Mensagens

```java
public class TileSocketImpl extends TileBase {  2 usages  luizanumberg +1 *
    private int serverPort;  2 usages
    private Socket client;  4 usages

    public TileSocketImpl(int port) {  1 usage  luizanumberg
        this.startServer(port);
    }

    public void startServer(int port) {  1 usage  luizanumberg
        this.setServerPort(8000 + port);
        int portNumber = this.getServerPort();
        try {
            ServerSocket serverSocket = new ServerSocket(portNumber);
            new Thread(() -> {
                while (true) {
                    try {
                        serverSocket.accept();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
            }).start();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @Override  2 usages  luizanumberg +1 *
    public boolean tryAcquire() {
        try {
            this.client = new Socket();
            int portNumber = this.getServerPort();
            this.client.connect(new InetSocketAddress( hostname: "localhost", portNumber));
            if (this.client.isConnected()) {
                return true;
```

Utiliza a troca de mensagens com sockets para garantir a exclusão mútua. Ao invés de depender de mecanismos internos de sincronização como semáforos ou monitores, ela se comunica com os veículos através de sockets para coordenar o acesso exclusivo ao tile.

Os veículos devem solicitar ao tile enviando através do socket, garantindo assim a segurança e coordenação no contexto da simulação de tráfego.

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ OBRIGADA! />

} /> [

Alguma pergunta?

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1