

根据您提供的 `main.nice.js` 文件内容，模块 46（即 `46 : [function(require, module, n){ ... }]`）定义了一个名为 `update` 的类，尽管其在代码中被命名为 `update`，但在上下文中被用作场景内的汽车管理类（以下称为 `Car` 类）。这个类负责管理场景中的汽车对象，包括其移动、碰撞检测以及与其他汽车的交互。以下是对该类的详细分析，包括其接口和功能，以及每个接口的主要实现方式。

---

## 类概述

`Car` 类从模块 55 (`self = require("55")`) 继承，`self` 是一个基类，可能提供了基本的场景对象管理功能。`Car` 类的主要职责是管理单个汽车对象的状态和行为，包括位置、速度、方向、碰撞检测等。它通过与场景中的其他对象（如车道、路口、其他汽车）交互，模拟汽车在无限城市中的移动。

## 构造函数

---

javascript



```
var update = function(obj, data, transform) {
```

```
self.call(this, obj);
this.name = "car";
this.maxSpeed = .25;
this.minSpeed = 0;
this.speed = this.maxSpeed;
this.stuck = false;
this.restartTimer = null;
this.add(data);
this.mesh = data;
this.radarRadius = (this._isLargeVehicle(), 20);
this.position.copy(transform.position);
var point = new THREE.Vector3(3.4, 0, 0);
data.rotation.copy(transform.rotation);
point.applyAxisAngle(normal, data.rotation.y);
if (result.random() > .5) {
    this.position.add(point);
} else {
    data.rotation.y += Math.PI;
    this.position.sub(point);
}
this.direction = data.getWorldDirection().negate();
this.direction.set(Math.round(this.direction.x), Math.round(this.direction.y));
this._initCollisionPoints();
};
```

- 参数：

- obj：场景或父对象，通常是汽车所属的区块（chunk）。

- `data`：汽车的网格对象 (`THREE.Mesh`)，表示汽车的几何形状和外观。
- `transform`：变换对象，包含位置 (`position`) 和旋转 (`rotation`) 信息。
- 功能：
  - 初始化汽车的基本属性，如名称 ("car")、最大速度 (0.25)、当前速度、是否卡住 (`stuck`) 等。
  - 将汽车的网格对象添加到自身，并根据传入的 `transform` 设置初始位置和方向。
  - 根据随机值调整汽车的初始位置（偏移 `point` 或反向偏移），模拟车道上的不同起始点。
  - 计算汽车的移动方向 (`direction`)，并初始化碰撞检测点。
- 实现细节：
  - 使用 `result.random()`（模块 59 的随机函数）决定汽车在车道上的初始偏移。
  - 调用 `_initCollisionPoints` 方法设置碰撞检测点，用于后续的汽车检测。

## 属性

- `name`："car"，标识对象类型。
- `maxSpeed`：最大速度，默认值为 0.25。
- `minSpeed`：最小速度，默认值为 0，在卡住时可能调整。
- `speed`：当前速度，初始为 `maxSpeed`。
- `stuck`：布尔值，表示汽车是否卡住。
- `restartTimer`：定时器，用于在卡住一段时间后调整最小速度。
- `mesh`：汽车的网格对象。
- `radarRadius`：雷达半径，默认值为 20，用于检测附近汽车。
- `direction`：移动方向向量 (`THREE.Vector3`)。
- `collisionPoints`：碰撞检测点数组，用于检测与其他汽车的距离。

---

## 接口与功能分析

以下是 `Car` 类的所有方法（接口），包括其功能和实现细节：

## 1. addRadarHelper()

- 功能：添加一个可视化的雷达辅助对象，用于调试或显示汽车的检测范围。

- 实现：

```
javascript
```



```
addRadarHelper: function() {  
    var pregeom = new THREE.CircleGeometry(this.radarRadius, 32, 0, Math.PI / 2);  
    var wrapper = new THREE.MeshBasicMaterial({ color: 16711935 });  
    var el = new THREE.Mesh(pregeom, wrapper);  
    el.rotation.order = "YXZ";  
    el.position.y = 1;  
    el.rotation.x = -Math.PI / 2;  
    el.rotation.y = this.mesh.rotation.y;  
    this.add(el);  
    this.helper = el;  
}
```

- 创建一个圆形几何体 (CircleGeometry)，半径为 radarRadius，并使用紫色材质 (颜色值为 16711935，即 #FF00FF)。
  - 设置圆形的旋转和位置，使其位于汽车上方并与汽车方向一致。
  - 将圆形添加到汽车对象中，作为辅助对象 (this.helper)。
- 主要用途：调试时可视化汽车的雷达范围。

## 2. detectCars(\_data)

- 功能：检测附近的汽车并调整自身速度，以避免碰撞或卡住。

- 参数：

- \_data：附近汽车对象的数组。

- 实现：

```
javascript
```



```
detectCars: function(_data) {  
    var speed = 0.075;
```

```

var _speed = .005;
var n = true;
this.detectedCar = null;
var dataIndex = 0;
for (; dataIndex < _data.length; dataIndex++) {
    var i = this.detectCar(_data[dataIndex]);
    if (i) {
        n = false;
        this.detectedCar = i;
        break;
    }
}
if (n) {
    if (this.speed < this.maxSpeed) {
        this.speed += _speed;
        this.speed = Math.min(this.speed, this.maxSpeed);
    }
    if (this.stuck) {
        clearTimeout(this.restartTimer);
        this.stuck = false;
        this.minSpeed = 0;
    }
} else {
    this.speed -= _speed;
    this.speed = Math.max(this.speed, this.minSpeed);
    if (!(this.stuck || 0 !== this.speed)) {
        this.stuck = true;
        this.restartTimer = setTimeout(function() {
            this.minSpeed = .25 * this.maxSpeed;
        }.bind(this), 2E3);
    }
}
}

```

- 遍历 `_data` 中的汽车对象，调用 `detectCar` 方法检测是否有汽车在雷达范围内。
- 如果没有检测到汽车 (`n = true`)：

- 增加速度（增量为 0.0075），但不超过 maxSpeed。
  - 如果之前卡住（stuck），清除定时器并恢复正常状态。
  - 如果检测到汽车（n = false）：
    - 减小速度（减量为 0.0075），但不低于 minSpeed。
    - 如果速度降为 0 且未卡住，标记为 stuck，并在 2 秒后设置 minSpeed 为 maxSpeed 的 25%，以尝试重新启动。
  - 主要逻辑：
    - 通过速度调整实现简单的交通流控制，避免碰撞或长时间停滞。
3. detectCar(obj)
- 功能：检测单个汽车是否在雷达范围内，并判断是否需要减速。
  - 参数：
    - obj：另一个汽车对象。
  - 实现：

javascript



```
detectCar: function() {  
    var v1 = new THREE.Vector3();
```

```

var v1 = new THREE.Vector3,
var v2 = new THREE.Vector3;
var startGround = new THREE.Vector3;
var endGround = new THREE.Vector3;
var orig = new THREE.Vector3;
return function(obj) {
    var c = obj.detectedCar === this;
    var length = false;
    if (c) return null;
    if (this.isOnIntersection() && !obj.isOnIntersection() && !this.direction)
        obj.updateMatrix();
    v1.copy(this.direction);
    v1.applyAxisAngle(normal, -Math.PI / 4);
    result.getTablePosition(this.position, this.parent.tableX, this.parent.tableY);
    var i = 0;
    for (; i < obj.collisionPoints.length; i++) {
        var pos = obj.collisionPoints[i];
        orig.copy(pos).applyMatrix4(obj.matrix);
        result.getTablePosition(orig, obj.parent.tableX, obj.parent.tableY);
        var length = startGround.distanceTo(endGround);
        if (length <= this.radarRadius) {
            v2.subVectors(endGround, startGround).normalize();
            var delta = v1.dot(v2);
            if (delta > .5) {
                length = true;
                break;
            }
        }
    }
    return length ? obj : null;
};
}()

```

· 排除条件：

- 如果目标汽车已将当前汽车标记为检测对象 (`obj.detectedCar === this`)，忽略。
- 如果碰撞点在桌子边缘或桌子之外

如果当前汽车不在路口 (`isUnintersection`)，而目标汽车不在，且方向不同，忽略。

- 检测逻辑：

- 计算当前汽车的前方检测方向 (`v1`)，偏转 45 度。
- 获取当前汽车和目标汽车在全局坐标系中的位置 (`startGround` 和 `endGround`)。
- 遍历目标汽车的碰撞点，计算距离 (`length`)。
- 如果距离小于 `radarRadius`，计算方向夹角 (`delta`)，若夹角余弦大于 0.5 (约 60 度以内)，认为目标汽车在前方。
- 返回值：检测到的汽车对象，或 `null`。
- 主要用途：支持 `detectCars` 方法，判断是否需要调整速度。

#### 4. `update()`

- 功能：更新汽车的位置和状态，每帧调用。

- 实现：

```
update: function() {
    var value = new THREE.Vector3;
    return function() {
        value.copy(this.direction).multiplyScalar(this.speed);
        this.position.add(value);
        result.roundVector(this.position, 2);
        this._updateTablePosition();
        var fakeMutation = this.table.getNeighboringCars(this);
        this.detectCars(fakeMutation);
    };
}()
```

- 根据当前速度和方向更新位置 (position)。
- 使用 result.roundVector 保留两位小数，防止浮点误差。
- 调用 \_updateTablePosition 更新汽车在场景表格中的位置。
- 获取附近汽车 (getNeighboringCars)，并调用 detectCars 调整速度。
- 主要逻辑：
  - 实现汽车的移动和动态交互，保持场景的实时更新。

## 5. isOnIntersection()

- 功能：判断汽车是否位于路口区域。
- 实现：

```
isOnIntersection: function() {
    return this.position.x < -20 && this.position.x > -40 && this.position.z
}
```

- 检查汽车的 `x` 和 `z` 坐标是否在特定范围内（`-20` 到 `-40`）。

- **主要用途：**

- 用于 `detectCar` 中判断是否忽略某些检测条件。

## 6. `_initCollisionPoints()`

- **功能：** 初始化汽车的碰撞检测点。

- **实现：**

```
_initCollisionPoints: function() {
    var self = new THREE.Box3;
    self.setFromObject(this.mesh);
    var p = new THREE.Vector3;
    p.copy(self.min);
    this.worldToLocal(p);
    p.y = 1;
    if (Math.abs(this.direction.x) > 0) p.z = 0; else p.x = 0;
    var b = new THREE.Vector3;
    b.copy(self.max);
    this.worldToLocal(b);
    b.y = 1;
    if (Math.abs(this.direction.x) > 0) b.z = 0; else b.x = 0;
    this.collisionPoints = [p, b];
    this.collisionPoints.forEach(function(canCreateDiscussions) {}, this);
}
```

- 使用 `THREE.Box3` 计算汽车网格的边界框。
- 提取最小点 (`min`) 和最大点 (`max`)，转换为局部坐标。
- 根据移动方向 (`direction`) 调整点的坐标 (沿 `x` 或 `z` 轴对齐)。
- 将两个点存储在 `collisionPoints` 数组中。

- 主要用途：

- 为 `detectCar` 提供碰撞检测的基础数据。

## 7. `_isLargeVehicle()`

- 功能：判断汽车是否为大型车辆（影响雷达半径等参数）。
- 实现：

```
_isLargeVehicle: function() {
    return this.mesh.name.indexOf("Bus") !== -1 || this.mesh.name.indexOf("Cc
}
```

- 检查网格名称是否包含 "Bus"、"Container" 或 "Truck"。

- **主要用途：**

- 在构造函数中调用，用于设置 `radarRadius`（尽管当前实现中未直接使用返回值）。

## 总结

`Car` 类是一个功能完整的汽车管理类，主要用于在无限城市场景中模拟汽车的移动和交互。其核心功能包括：

- **移动管理：**通过 `update` 方法更新位置，结合速度和方向。
- **碰撞检测：**通过 `detectCars` 和 `detectCar` 实现与其他汽车的动态交互。
- **状态调整：**根据检测结果调整速度，处理卡住状态。
- **辅助功能：**提供雷达可视化（`addRadarHelper`）和位置判断（`isOnIntersection`）。

## 实现特点

- **模块化：**依赖模块 55 的基类功能和模块 59 的工具函数（如 `random` 和 `getTablePosition`）。
- **动态性：**通过每帧更新和邻近检测实现实时交通模拟。
- **简单性：**使用基本的向量运算和距离判断实现碰撞检测，适合性能敏感的场景。

这个类的设计与无限城市漫游的目标一致，支持动态加载的区块（chunks）中汽车的自然移动和交互，为用户提供流畅的探索体验。