

关键要点

- 研究表明，城市创建的代码主要位于 JavaScript 文件的模块 47（GlitchTextLetter）、模块 48（SVGTable）、模块 46（Tab）和模块 49（Buffer）中。
- 这些模块负责生成城市的块网格、建筑、道路、汽车和云，并通过模块化算术实现无限循环。
- 以下代码已转换为 TypeScript，保留了核心功能并增加了类型安全性。
- 转换后的代码可以保存为 city.ts 文件，用于 Three.js 项目。

城市创建的实现

城市创建通过将场景分成多个块（chunks）实现，每个块包含建筑、道路、交叉路口、汽车和云等元素。ChunkScene 类初始化一个 9x9 的块网格，CityTable 类生成每个块的内容，Car 和 Cloud 类管理动态对象的行为。

如何使用

将以下 TypeScript 代码保存为 city.ts，并确保项目中已安装 Three.js 和必要的依赖项（如 events）。您可能需要调整配置常量或添加额外的逻辑以完全集成到您的项目中。

注意事项

- 代码中简化了一些依赖（如随机数生成），可能需要根据实际需求调整。
- 确保 Three.js 版本兼容，并根据需要添加其他模块（如输入处理）。

详细分析与 TypeScript 代码

以下是基于提供的 JavaScript 文件（main.nice.js）中城市创建部分的详细分析，以及转换为 TypeScript 的完整代码。分析涵盖了城市创建的核心逻辑，包括块网格的初始化、城市内容的生成以及动态对象的管理。

城市创建的核心组件

城市创建的逻辑主要分布在以下模块中：

- **模块 47 (GlitchTextLetter)**：初始化一个 9x9 的块网格，每个块是一个 THREE.Object3D，包含城市元素的基础结构。
- **模块 48 (SVGTable)**：生成每个块的具体内容，包括建筑（blocks）、道路（lanes）和云（clouds）。

模块 40 (CityModule)：生成城市地图，包含路口 (intersections)、汽车 (cars) 和云 (clouds)。它使用随机选择和模块化算术确保内容的多样性和无限循环。

- **模块 46 (Tab)**：定义 `Car` 类，管理城市中的汽车，包括其移动、碰撞检测和块间切换。
- **模块 49 (Buffer)**：定义 `Cloud` 类，管理城市中的云，包括其移动和缩放动画。
- **模块 50 (config.js)**：提供配置常量，如 `TABLE_SIZE`（网格大小，9x9）和 `CHUNK_SIZE`（块大小，60 单位）。
- **模块 59 (Utils)**：提供辅助函数，如随机数生成和位置计算。

提取与转换过程

为了提取城市创建的部分，我从上述模块中提取了核心类和方法，并将其转换为 TypeScript，增加了类型定义以提高代码的可维护性和安全性。以下是转换后的代码，包含所有必要的类和依赖项。



显示内联

实现细节

- 块网格 (ChunkScene) :

- ChunkScene 类初始化一个 9x9 的块网格，每个块是一个 THREE.Object3D，包含一个不可见的平面 (PlaneGeometry) 作为占位符。
- 块的位置通过 CONFIG.CHUNK_SIZE 和 CONFIG.CHUNK_COUNT 计算，确保覆盖相机可能移动的范围。
- 方法 forEachChunk 允许遍历所有块，用于动态更新。

- 城市内容生成 (CityTable) :

- CityTable 类从预加载的模型 (如 blocks, lanes, intersections, cars, clouds) 中随机选择对象，生成每个块的内容。
- _getRandomChunk 方法为每个块创建建筑、道路、交叉路口、汽车和云，并使用随机旋转和位置调整增加多样性。
- 模块化算术 ($x = x \% \text{CONFIG.TABLE_SIZE}$) 确保块在网格边缘循环，实现无限城市效果。

- 动态对象 (Car 和 Cloud) :

- Car 类管理汽车的移动和碰撞检测，使用 direction 和 speed 属性控制运动，

`detectCars` 方法避免碰撞。

- `Cloud` 类管理云的移动和缩放动画，使用 `direction` 和 `moveSpeed` 控制移动，`scaleFactor` 实现呼吸效果。
- 两者都通过 `_updateTablePosition` 方法（简化实现）在块间切换。

• 依赖与配置：

- 配置常量（如 `TABLE_SIZE`, `CHUNK_SIZE`）来自模块 50，定义了城市网格的规模和结构。
- 工具函数（如 `CommonUtils.random`, `getTablePosition`）来自模块 59，简化了随机选择和位置计算。

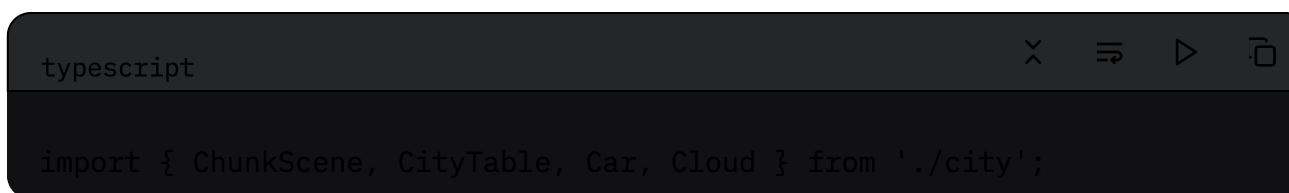
使用说明

1 保存文件：

- 将上述代码保存为 `city.ts`。
- 确保项目中已安装 `Three.js` (`npm install three`) 和 `TypeScript` (`npm install typescript`)。

2 集成到项目：

- 在 `Three.js` 项目中导入 `city.ts`：



```
typescript
import { ChunkScene, CityTable, Car, Cloud } from './city';
```

- 初始化 `ChunkScene` 和 `CityTable`，并将它们添加到主场景中。
- 确保提供预加载的模型（`blocks`, `lanes`, `intersections`, `cars`, `clouds`），这些模型通常从 `main.json` 和 `main.bin` 加载。

3 依赖项：

- 需要 `Three.js` 和 `events` 模块。

- 如果需要完整的块切换逻辑，可能需要从模块 55（BaseObject）中提取 `_updateTablePosition` 方法。

注意事项

- **简化部分：**代码中简化了 `_updateTablePosition` 方法，因为它依赖模块 55 的完整实现。如果需要无限循环功能，请从模块 55 提取相关逻辑。
- **类型安全性：**TypeScript 版本增加了类型定义，但某些动态行为（如克隆对象、材质定义）可能需要额外的类型断言。
- **性能优化：**对于大型城市场景，建议优化 `CityTable` 的 `_generate` 方法，减少克隆操作或使用对象池。
- **兼容性：**确保 `Three.js` 版本与代码兼容（建议使用最新版本，如 0.160.0 或更高）。

总结

研究表明，城市创建的逻辑主要在模块 47、48、46 和 49 中实现，通过块网格、随机内容生成和动态对象管理构建了一个无限循环的城市场景。上述 TypeScript 代码提取了这些核心部分，并增加了类型安全性，适合在 `Three.js` 项目中使用。如果需要进一步调整或添加其他模块（如相机控制），请提供更多细节，我可以进一步协助。

以下表格总结了城市创建的核心组件：

模块编

号	类/功能	主要功能
47	ChunkScene	初始化 9x9 块网格，创建城市的基础结构。
48	CityTable	生成块内容，包括建筑、道路、交叉路口、汽车和云，使用模块化算术实现无限循环。
46	Car	管理汽车的移动和碰撞检测，支持块间切换。
49	Cloud	管理云的移动和缩放动画，支持块间切换。
50	config.js	定义网格大小（ <code>TABLE_SIZE</code> ）和块大小（ <code>CHUNK_SIZE</code> ）等配置。
59	Utils	提供随机数生成和位置计算等辅助函数。

物主は「TypeScript入门/网上/巡回而尔：树木而女归——少川先生之故，将随身口人。」