a2

Creating Custom Maps

TCSS 305 Programming Practicum

Custom maps are a powerful tool for testing your vehicle implementations. Instead of waiting for specific scenarios to occur in the large default city map, you can create small, focused maps that test exactly the behavior you want to verify. This guide explains the map file format and shows you how to create your own maps.

---

1 Changing the Map in the GUI

The Road Rage application loads its map from a file specified in `io/FileLoader.java`. To use a different map:

1. Open `src/edu/uw/tcss/io/FileLoader.java`
2. Find the constant `CITY_FILE` near the top of the class:

```
/** The filename of the city map. */
private static final String CITY_FILE =
        "maps" + File.separator + "city_map1.txt";
```

1. Change the filename to your custom map:

```
private static final String CITY_FILE =
        "maps" + File.separator + "my_test_map.txt";
```

1. **Restart the application** - the map is loaded once at startup

> **Keep the Original**
>
> Rather than modifying the original map file, create new files for your custom maps. This makes it easy to switch between maps by changing just the filename in the constant.

---

2 Map File Format

Map files are plain text files (`.txt`) stored in the `maps/` directory. Each file has two sections: the **grid section** and the **vehicle section**.

2.1 File Structure Overview

```
[rows] [columns]
[grid of terrain characters]
[number of vehicles]
[vehicle entries, one per line]
```

2.2 Grid Section

The first line contains two integers: the number of rows and columns in the map grid.

```
13 24
```

This example declares a 13-row by 24-column map.

Following the dimensions is the actual grid. Each character represents one terrain cell. The grid must be **rectangular** - every row must have exactly the same number of characters as declared in the column count.

2.3 Vehicle Section

After the grid, a single integer declares how many vehicles follow:

```
9
```

Each vehicle is then specified on its own line with four values:

```
[type] [x] [y] [direction] [terrain_char]
```

Field	Description
`type`	Single character identifying the vehicle type
`x`	X-coordinate (column number, 0-indexed from left)

Field	Description
`y`	Y-coordinate (row number, 0-indexed from top)
`direction`	Initial facing direction (`N`, `S`, `E`, `W`)
`terrain_char`	The terrain character at this position (for map rendering)

Example vehicle entry:

```
H 18 5 N -
```

This places a Human at position (18, 5), facing North, on grass terrain.

---

3 Terrain Characters

The following characters represent different terrain types in the map grid:

Character	Terrain	Description
`-`	GRASS	Open field - humans can travel here
`|`	STREET	Road for vehicles
`+`	LIGHT	Traffic light intersection (cycles green/yellow/red)
`X`	WALL	Impassable barrier
`.`	TRAIL	Bicycle path
`#`	CROSSWALK	Pedestrian crossing

Important

The pipe character `|` represents streets. When viewing map files in a text editor, streets appear as vertical bars.

3.1 Terrain Behavior Summary

Different vehicles can travel on different terrain types:

Vehicle	Allowed Terrain
Truck	Street, Light, Crosswalk
Car	Street, Light, Crosswalk
Taxi	Street, Light, Crosswalk
Bicycle	Street, Light, Crosswalk, Trail
ATV	Everything except Wall
Human	Grass, Crosswalk

---

4 Vehicle Type Characters

Vehicle entries in the map file use single-character type codes:

Character	Vehicle Type
`T`	Truck
`C`	Car
`X`	Taxi
`B`	Bicycle
`A`	ATV
`H`	Human

Direction Values

Direction uses compass abbreviations: `N` (North/up), `S` (South/down), `E` (East/right), `W` (West/left).

---

5 Map Design Guidelines

Creating effective test maps requires careful planning. Follow these guidelines to avoid common pitfalls.

5.1 Ensure Valid Paths

Every vehicle needs a valid path to travel. A vehicle with no valid moves (except reverse) will constantly reverse direction, which may not be what you want to test.

```
// BAD: Car trapped with no forward path
XXXXXX
X|XXXX
X|XXXX
XXXXXX

// GOOD: Car has room to navigate
XXXXXXX
X|+||XX
X|X||XX
X|||+|X
XXXXXXX
```

5.2 Connect Streets Logically

Streets should form a connected network. Isolated street segments create dead ends that may trap vehicles.

5.3 Place Crosswalks at Boundaries

Crosswalks connect grass areas to streets, allowing humans to cross. Place them where grass meets street terrain:

```
---#|||
---#|||
```

```
---#|||
```

5.4 Keep Trails Straight

Bicycles travel along trails but have specific seeking behavior. For predictable testing, keep trails in straight horizontal or vertical lines:

```
// GOOD: Straight trail
|||...|||
|||...|||

// Confusing: Trail with turns (harder to predict behavior)
|||..||||
|||.|||||
||||.||||
```

5.5 Test Incrementally

Start with simple maps testing one behavior at a time:

1. First, test basic movement on a single terrain type
2. Add traffic lights to test light-stopping behavior
3. Add crosswalks to test crosswalk rules
4. Add multiple vehicles to test collision behavior

---

6 Example: Simple Test Map

Here is a small test map for verifying Truck and Human behavior around a crosswalk:

```
6 10
XXXXXXXXXX
X|||#|---X
X|||#|---X
X|||+|---X
X|||#|---X
XXXXXXXXXX
3
T 2 2 E |
H 6 2 W -
H 7 4 N -
```

6.1 Breakdown

Dimensions: 6 rows by 10 columns (including walls)

Grid visualization:

```
XXXXXXXXXX    Row 0: All walls (boundary)
X|||#|---X    Row 1: Street, street, street, crosswalk, street, grass, grass,
grass
X|||#|---X    Row 2: Same pattern - vehicles placed here
X|||+|---X    Row 3: Light intersection in the street area
X|||#|---X    Row 4: Another crosswalk row
XXXXXXXXXX    Row 5: All walls (boundary)
```

Vehicles: - Truck at (2, 2) facing East, on street - Human at (6, 2) facing West, on grass - Human at (7, 4) facing North, on grass

What this tests: - Truck driving on streets - Truck behavior at crosswalks (stops for red only, drives through yellow/green) - Human walking on grass - Human seeking crosswalks - Human crosswalk behavior (stops for green, crosses on yellow/red)

---

7 Example: Traffic Light Test Map

This map isolates traffic light behavior:

```
5 8
XXXXXXXX
X||+||XX
X||X||XX
X||+||XX
XXXXXXXX
2
C 2 1 S |
C 4 3 N |
```

What this tests: - Cars approaching traffic lights from opposite directions - Car stopping behavior at red lights - Cars meeting at an intersection

---

8 Example: Trail-Seeking Bicycle Map

This map tests bicycle trail-seeking behavior:

```
7 12
XXXXXXXXXXXX
X|||...|||XX
X|||...|||XX
X|||...|||XX
X|||...|||XX
X|||||||||XX
XXXXXXXXXXXX
1
B 2 5 N |
```

What this tests: - Bicycle starting on a street - Bicycle discovering and preferring nearby trails - Bicycle following a trail

---

9 Debugging Map Issues

When your map does not work as expected, check these common issues.

9.1 Non-Rectangular Grid

Symptom: Application crashes or displays incorrectly on startup.

Cause: Rows have different lengths.

Fix: Count characters in each row. Every row must have exactly the number of characters specified in the column count.

```
// WRONG: Row 2 is missing a character
13 24
XXXXXXXXXXXXXXXXXXXXXXX     <- Only 23 characters!
X|||+|||||||+||+|||||||X
...
```

9.2 Invalid Terrain Characters

Symptom: Unexpected terrain appears (usually defaults to GRASS).

Cause: Using characters not defined in the `Terrain` enum.

Fix: Use only valid characters: `-`, `|`, `+`, `X`, `.`, `#`

9.3 Vehicle Position Out of Bounds

Symptom: Vehicle does not appear or error on startup.

Cause: Vehicle coordinates exceed map dimensions.

Fix: Ensure vehicle x-coordinate is less than column count and y-coordinate is less than row count. Remember: coordinates are 0-indexed.

9.4 Vehicle on Invalid Terrain

Symptom: Vehicle appears but behaves unexpectedly (may immediately stop or reverse).

Cause: Vehicle placed on terrain it cannot travel.

Fix: Match vehicle types to valid terrain: - Place trucks, cars, taxis on streets/lights/crosswalks - Place bicycles on streets/lights/crosswalks/trails - Place humans on grass/crosswalks - ATVs can go anywhere except walls

9.5 Mismatched Vehicle Count

Symptom: Some vehicles missing or file parsing error.

Cause: Number of vehicle lines does not match declared count.

Fix: Ensure the number after the grid matches the actual number of vehicle entries.

9.6 Debug Mode

Use the GUI's **Debug Mode** checkbox to display coordinates and `toString()` output for each vehicle. This helps verify:

- Vehicles are at expected positions
- Vehicles are facing expected directions
- Vehicle state (enabled/disabled)

---

10 Creating Maps for Specific Tests

Custom maps are especially useful for testing edge cases that rarely occur in the default city map.

10.1 Testing Reverse Behavior

Create a dead-end where the vehicle must reverse:

```
5 5
XXXXX
X|XXX
X|XXX
X|XXX
XXXXX
1
T 1 1 N |
```

The truck starts facing the wall and must reverse.

10.2 Testing Collision Recovery

Place two vehicles on collision course with controlled timing:

```
5 8
XXXXXXXX
X||+||XX
X||X||XX
X||+||XX
XXXXXXXX
2
T 2 1 S |
C 2 3 N |
```

The truck (mass 100) and car (mass 50) will collide. The car should become disabled.

10.3 Testing Random Direction Selection

For vehicles that choose randomly (Truck, ATV, Human), create a map where multiple directions are valid:

```
5 7
XXXXXXX
X|+||XX
X|||XXX
X|+||XX
XXXXXXX
1
T 2 2 N |
```

Run multiple times to verify the truck sometimes goes straight, sometimes turns.

Summary

Concept	Key Point
Map Location	Change `CITY_FILE` constant in `io/FileLoader.java`
File Format	Dimensions, grid, vehicle count, vehicle entries
Terrain Characters	`-` grass, `|` street, `+` light, `X` wall, `.` trail, `#` crosswalk
Vehicle Characters	`T` Truck, `C` Car, `X` Taxi, `B` Bicycle, `A` ATV, `H` Human
Grid Requirement	Must be rectangular (all rows same length)
Testing Strategy	Create small, focused maps for specific behaviors

---

Further Reading

External Resources

- Road Rage Assignment - Full assignment requirements and vehicle specifications
- Terrain enum source - View the actual Terrain enum implementation

---

References

Course Materials:

- TCSS 305 Assignment 2: Road Rage — Vehicle specifications and terrain rules

Language Documentation:

- Oracle JDK 25 Documentation — Official Java SE reference

This guide is part of TCSS 305 Programming Practicum, School of Engineering and Technology, University of Washington Tacoma.