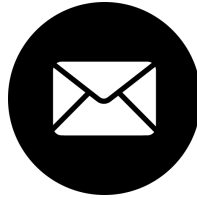


RAILWAY CONSTRUCTOR

Copyright (c) [Pampel Games](#) e.K. All Rights Reserved.



Support



Contact



More Tools

UNLOCKED DISCOUNTS



SUMMARY

[RAILWAY CONSTRUCTOR](#)

[SUMMARY](#)

[INSTALLATION](#)

[Requirements](#)

[Installation](#)

[QUICK START](#)

[Start](#)

[Editor](#)

[Player](#)

[Terrain](#)

[HOW IT WORKS](#)

[General](#)

[Construction](#)

[Railway Set](#)

[Railway Builder](#)

[RAILWAY CREATION](#)

[General](#)

[Rails and Sleepers](#)

[Lanes](#)

[Object Presets](#)

[Bridges](#)

[TRAFFIC SYSTEM](#)

[Traffic Component](#)

[Train Controller](#)

[SAVE SYSTEM](#)

[INTEGRATIONS](#)

[Road Constructor](#)

[CUSTOMIZATION](#)

[Railway Builder](#)

[API](#)

[RailwayConstructor.cs](#)

INSTALLATION

Requirements

- Minimum Unity Version 2022 LTS +

Dependencies

- unity.burst
- unity.collections
- unity.splines

Installation

If you have other tools from Pampel Games installed in your project, it is recommended that you update them to the latest version before importing the asset.

After importing, the 'PampelGames' folder can be moved to a different location within the 'Assets/' folder if desired.

QUICK START

If you want to start building railways immediately without reading further details, here's how you can do that.

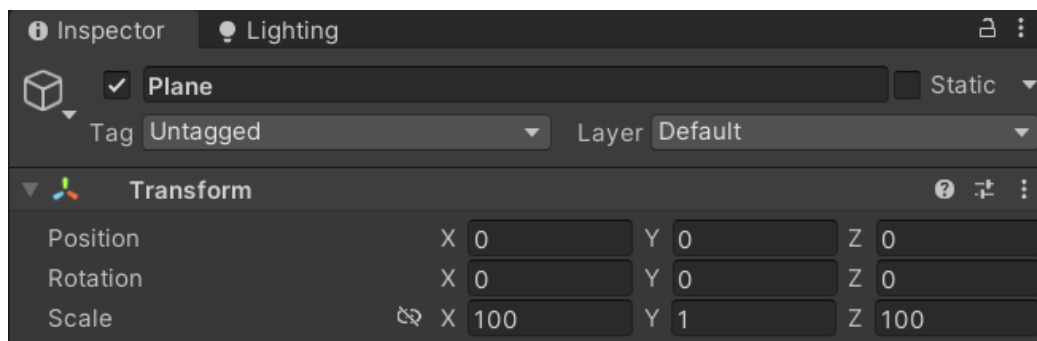
However, it is recommended to at least take a quick look at the [How It Works](#) section, as a basic understanding of how the tool functions can help you achieve your goals more quickly.

Start

First, navigate to the 'PampelGames/RailwayConstructor/Demo' folder, where you will find the prefabs you can use to start building.

Railway Constructor allows you to build railways either in the editor or during gameplay. Depending on your use case, you'll need either the 'Railway Builder Editor' or 'Railway Builder Player' prefab, which can be dragged and dropped into your scene.

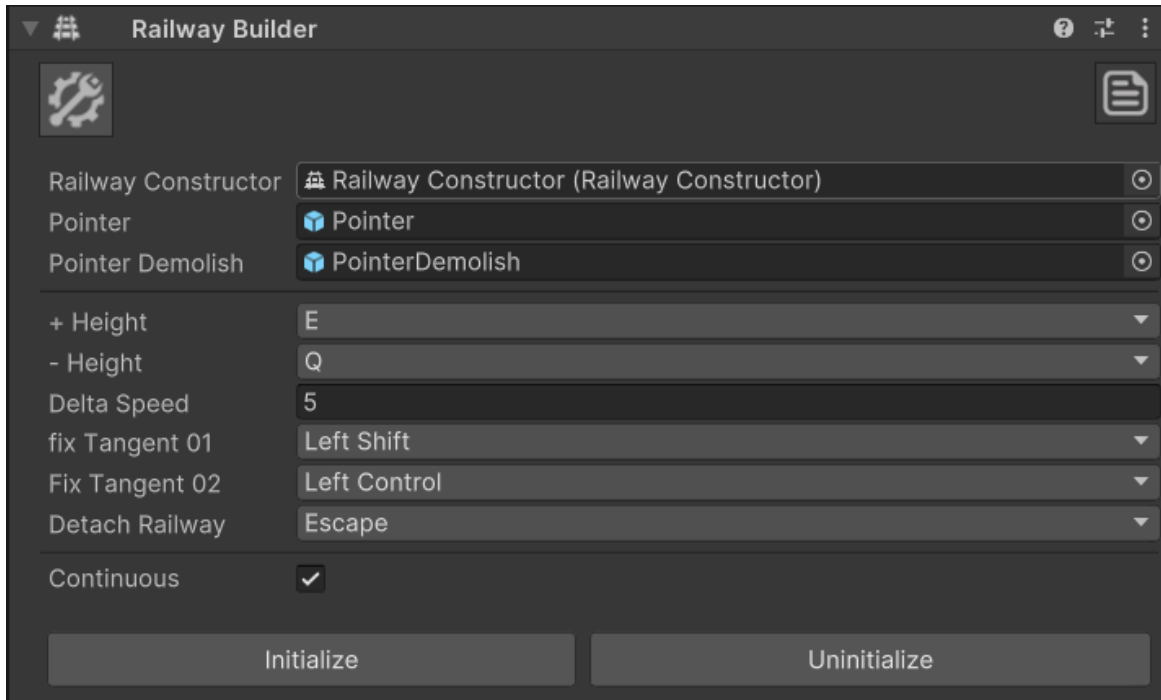
Either way, you'll need a ground to place the objects on as well. If you don't already have a mesh or terrain to use, create a plane in your hierarchy and set its scale to 100.



Editor

Important: To use the Railway Builder Editor, you must enable Gizmos in the scene view.

Drag the 'Railway Builder Editor' prefab into your scene and select the parent GameObject.



To show or hide the builder settings, click the construction icon at the top left of the component.

Next, click on the 'Initialize' button.

The component should look something like this:

Uninitialize

Register Scene Objects

► Post Construction Options

► Export Mesh

Railway

Ramp

Turnout

Station

Demolish

Move

Reverse

Crossing

Undo

Rail Count

2

Delta Height

0

Connections

Align

Tangent Length

0.5

Parallel Railway

☐

Mainline | Stone

Mainline | Stone Demo

Mainline | Stone Paving

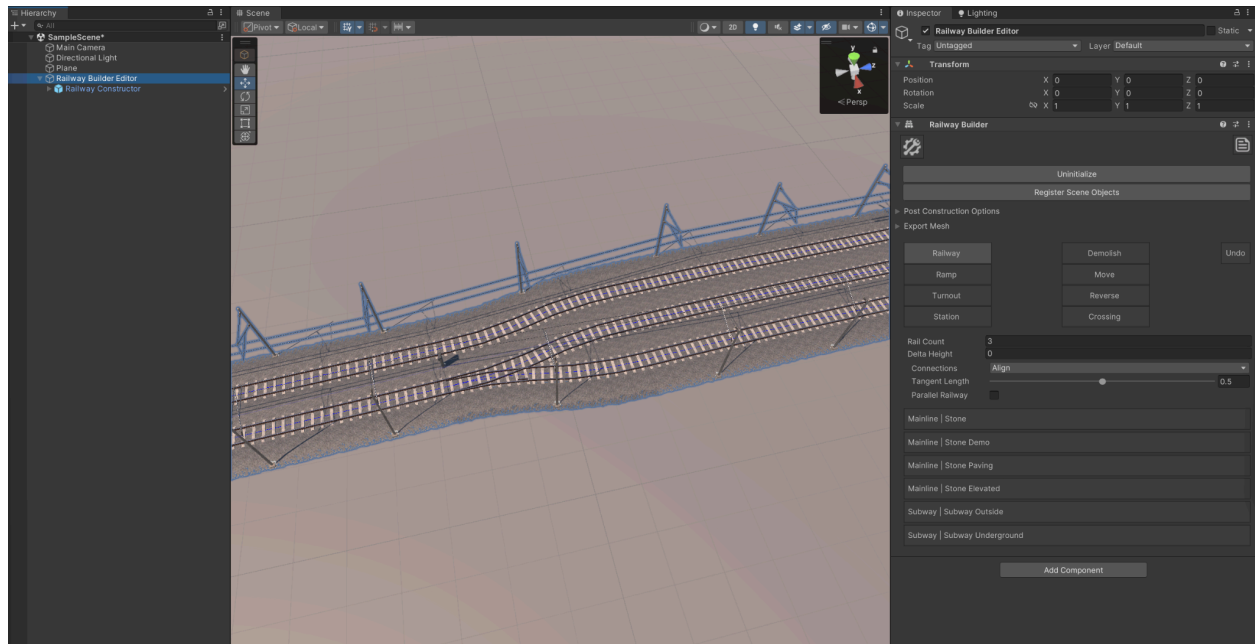
Mainline | Stone Elevated

Subway | Subway Outside

Subway | Subway Underground

Now you can select the different railways and place them onto your ground, demolish them, or undo the last operations using the dedicated buttons.



Feel free to experiment with the different railway types, builder settings as well as the component settings on the 'Railway Constructor' child GameObject - more on that in the next chapter.



Player

Drag the 'Railway Builder Player' prefab into your scene and select the parent GameObject.

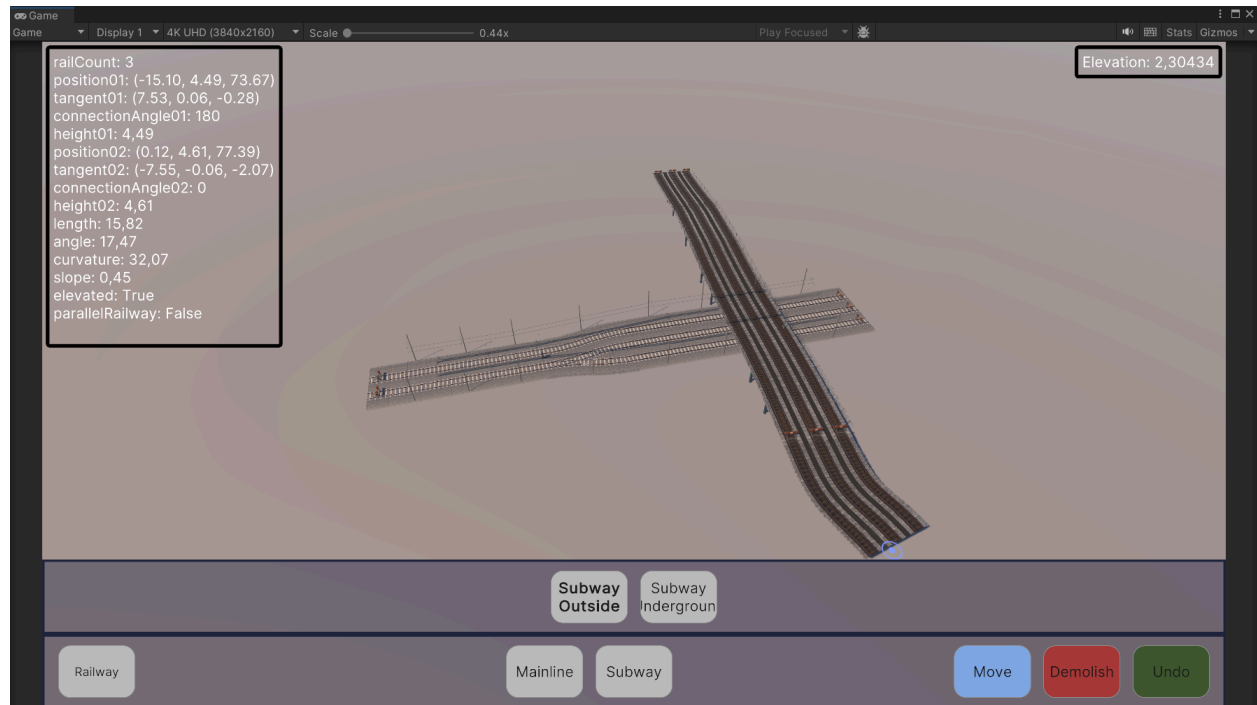
Here, you can find the current hotkeys for the player.

Increase Height	E
Decrease Height	Q
Delta Speed	5
Fix Tangent 1	Left Shift
Fix Tangent 2	Left Control
Detach Railway	Escape
Register Scene Objects	<input checked="" type="checkbox"/>
Continuous	<input checked="" type="checkbox"/>
Rail Count	2
Delta Height	0
Position 01 Set	<input type="checkbox"/>
UI Document	 Railway Builder Player (UI Document) 

Next, make sure the Main Camera has its view focused on the ground from a distance of about 100 units.

Hit play, and you can immediately start building railways using the menu panel.

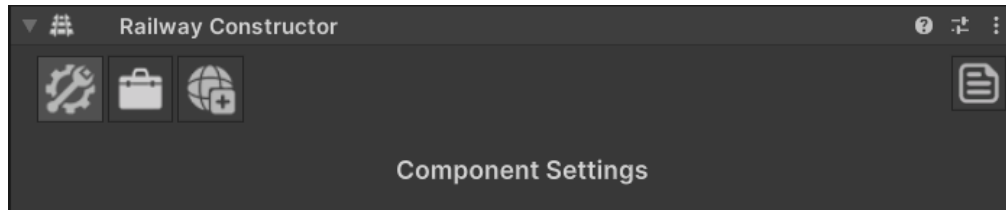
Rail counts can be switched using the keyboard alpha keys, for example '2' to build 2 adjacent railways.



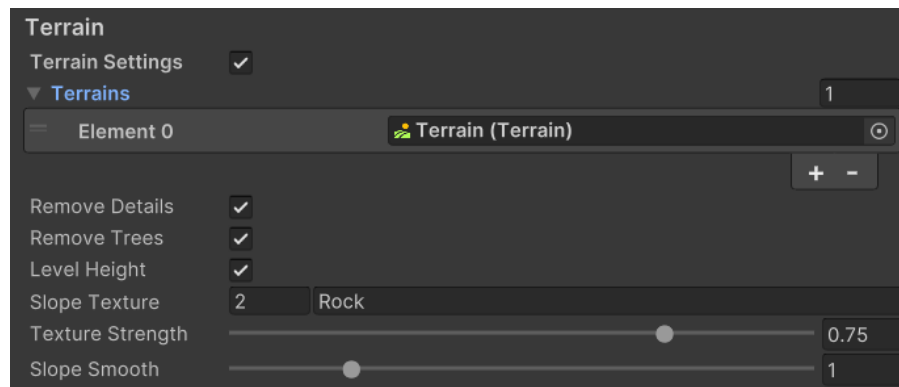
Terrain

Whether you are working in the player or editor, if you have ground terrain on which you want to build, you can apply helpful terrain settings.

Select the 'Railway Constructor' GameObject in the hierarchy and make sure you have the settings tab enabled.

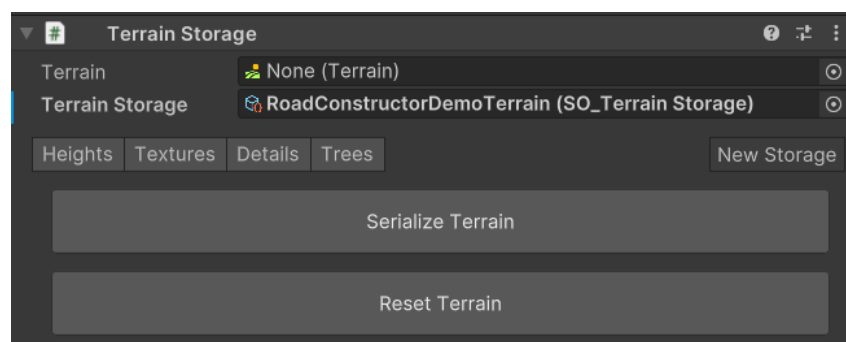


Locate and enable the terrain settings, reference your terrains and then toggle the terrain sub-settings as desired.



Remember that a Unity terrain is an asset, meaning that all modifications made to it are permanent.

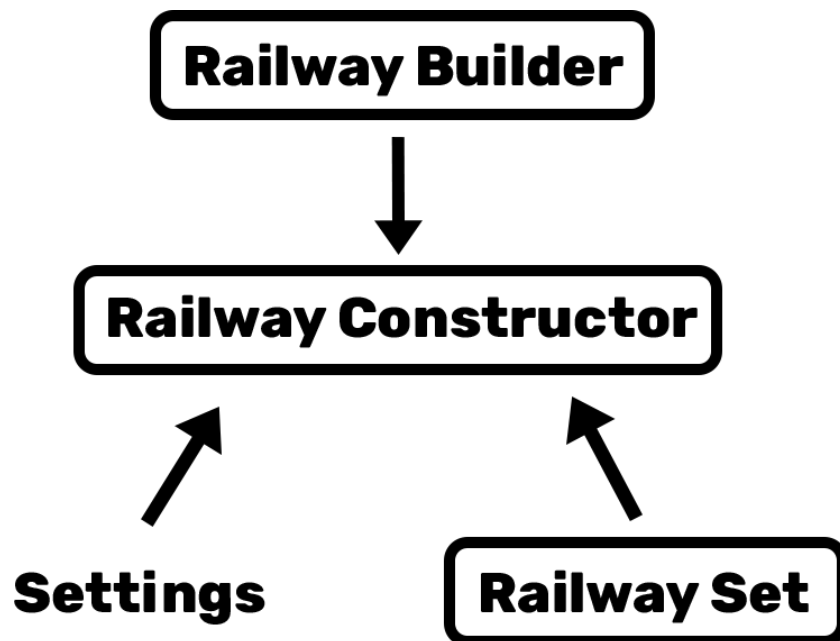
You may find the 'Terrain Storage' component helpful, as it allows you to serialize and reset the terrain data dynamically. You can find it in the hierarchy of both Railway Builder components.



HOW IT WORKS

General

The basic structure of Railway Constructor is as follows:



Railway Constructor: This component is the heart of the system, it holds both the construction settings and a reference to a Railway Set.

Settings: These are all the construction settings that can be applied by the user. The settings are configured and stored directly on the Railway Constructor component.

Railway Set: A scriptable object that stores the railways and other objects available for construction. A railway set is required for construction, where one set can be shared among multiple Railway Constructors.

Railway Builder: A custom script that accesses the Railway Constructor API. Two builders are included in the asset (Editor and Player), and you can also create your own if you need a custom solution.

Construction

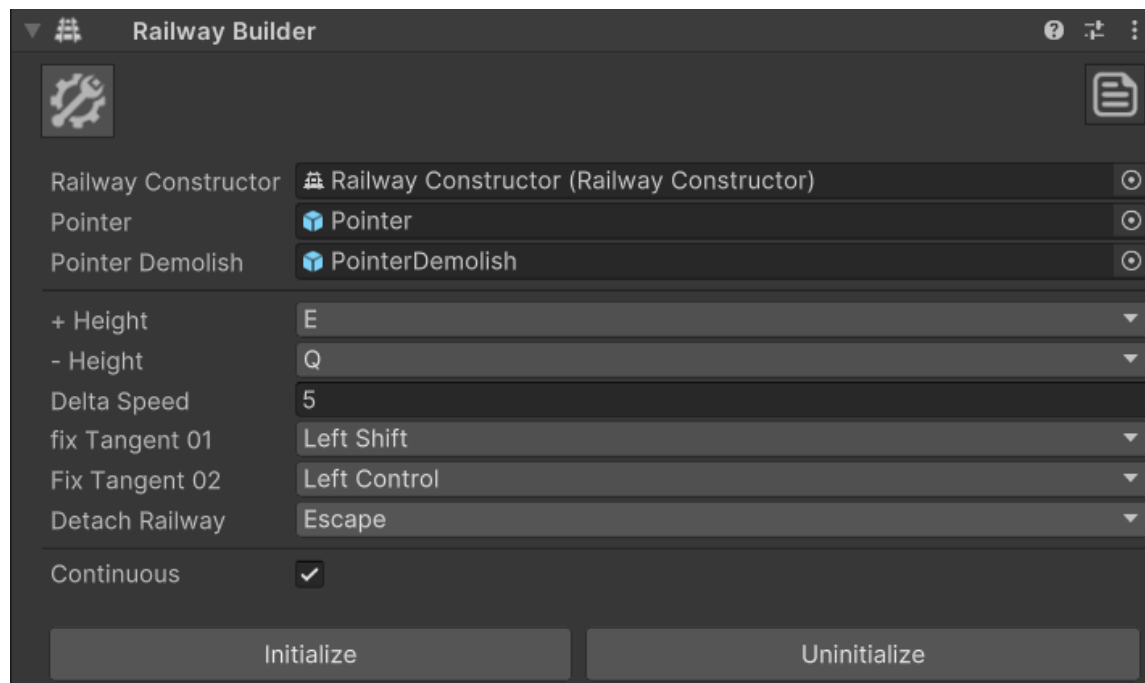
Both of the included railway builder components (player and editor) share the same principles regarding how to construct the railways.

You will need a raycastable ground (terrain, mesh) with a layer that matches the Ground Layer in the Railway Constructor settings. **For the Builder Editor, make sure the scene gizmos are enabled!**

After initializing the component, you can select a railway and start placing it into the scene.

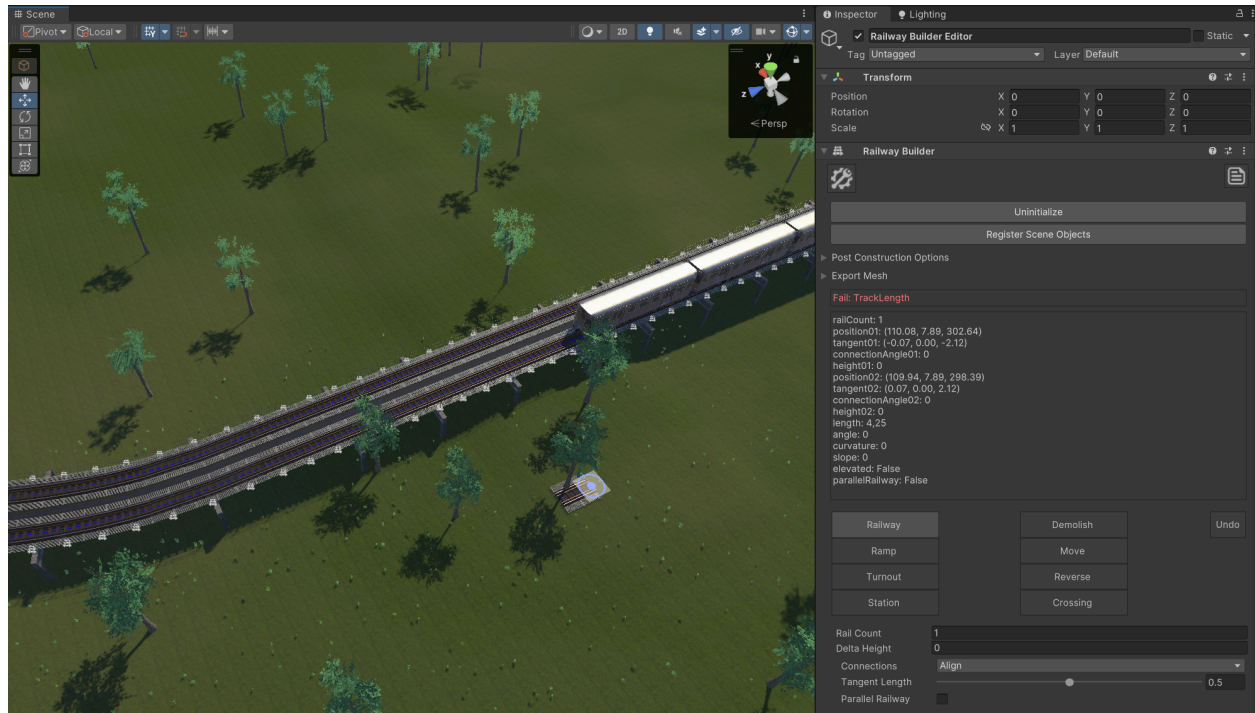
If you want to continue with objects already created from another session, you can register those objects. In the editor builder via the 'Register Scene Objects' button.

To display the different hotkeys available, click the settings icon in the top-left corner of the builder editor component.



When you place new railways onto existing ones, intersections are generated procedurally, taking into account the incoming curve angles, railway width, slopes, and other factors. If you elevate your railways, the tool will detect overlapping railways and intersections, adjusting the construction accordingly.

Both included builders display real-time construction data, such as informational details like angles and heights, as well as error data (in red), which can prevent construction based on conditions specified in the Railway Constructor component settings.

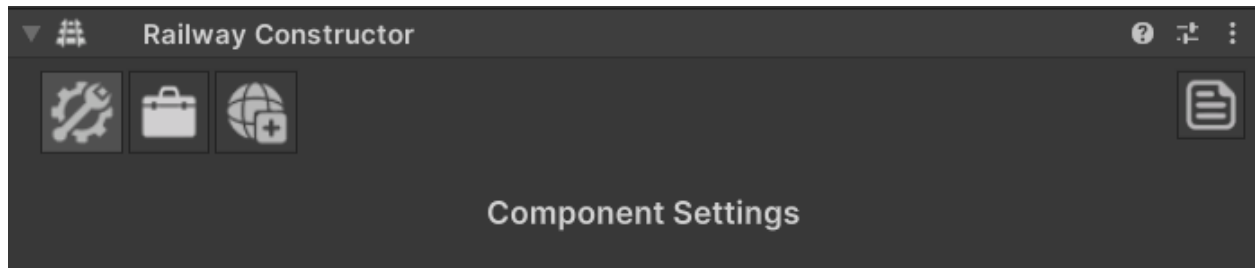


Remember that if you are working in the editor, you should always export your meshes once you are done with construction; otherwise, the mesh references on the railways may be lost once you export your scene.

Settings

These refer to the settings of the Railway Constructor component.

When you add a Railway Constructor component to a GameObject, you can access its settings by clicking the construction icon in the top-left corner of the component.



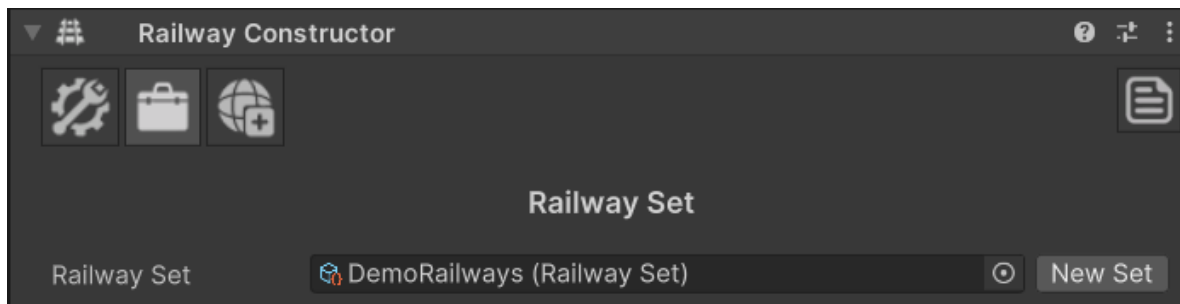
Each setting is a field that is serialized directly within the component itself, whether it's in the scene hierarchy or wrapped in a prefab.

Settings that are not self-explanatory should have tooltips to assist with understanding. In practice, it's always recommended to experiment with the settings during actual construction. This way, you'll quickly learn what best fits your needs.

Railway Set

A Railway Set is a ScriptableObject where railways are created and stored. ScriptableObjects serialize their data in the project folder, not in a scene hierarchy or prefab instance. A single railway set can be used by multiple Railway Constructor components.

To visualize a railway set, click the suitcase icon in the top-left corner of a Railway Constructor component, next to the settings icon.



Here, you can reference an existing set or create a new one using the dedicated button on the right.

After assigning a railway set, it will be displayed in the Inspector.



Of the three available menus, the icon on the far left represents the actual railways, while the other two icons only serve as presets. Once you become familiar with the railway creation process, you may find these presets to be a convenient way to share settings and save time.

You can start creating new railways using the + button on the right.

For further information on the process, please refer to the [RAILWAY CREATION](#) section.

Railway Builder

A 'Railway Builder' is simply a term used to describe a script that interacts with the Railway Constructor API.

A builder calls methods such as 'railwayConstructor.ConstructObjects' or 'railwayConstructor.Demolish,' and it manages the user interface for displaying the railways. It is kept completely separate from the main component, allowing for greater flexibility when creating your own solution.

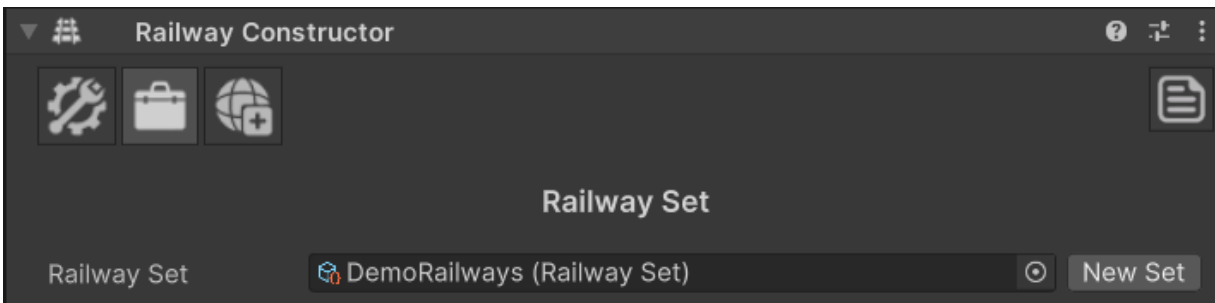
You can find example Railway Builders in the 'PampelGames/RailwayConstructor/Demo' folder. These builders are designed to automatically integrate custom railways.

If you intend to build your own builder or expand on the existing ones, you'll find the 'RailwayBuilderPlayer.cs' script in the 'Demo/Scripts' folder. It is well commented and provides a common practical example for railway building in strategy games.

RAILWAY CREATION

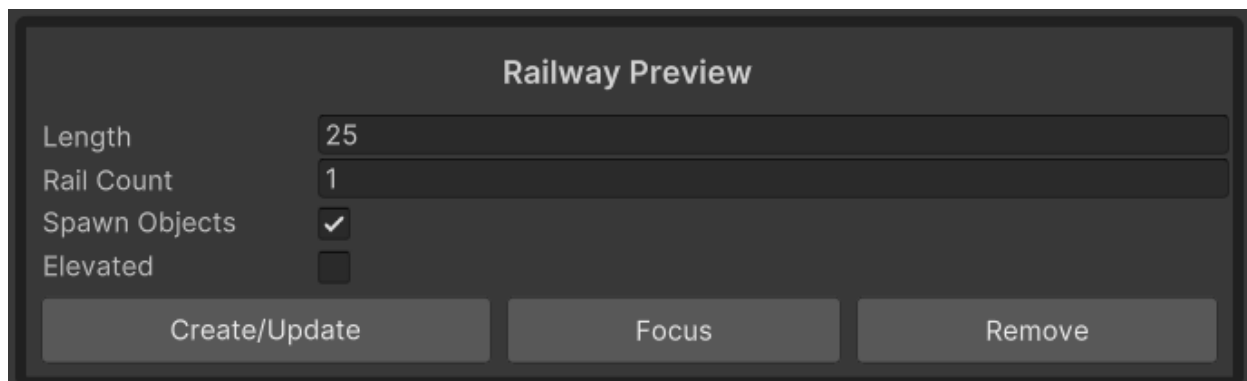
General

Railways are created, modified, and stored within Railway Sets, which are scriptable objects (data containers) in your project folder. After adding a Railway Constructor component to a GameObject, you can either reference an existing set or create a new one.

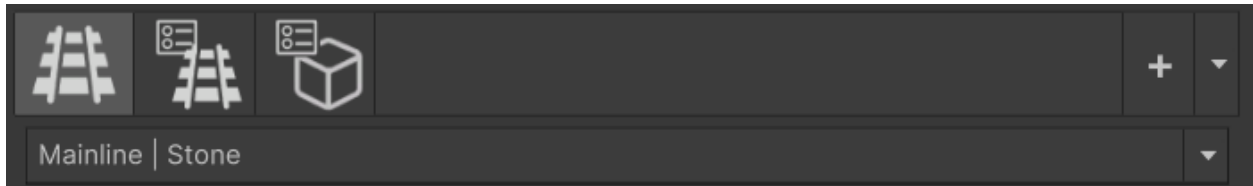


Railways can be previewed, providing a more convenient way than having to create the railways manually each time to test their appearance.

The Railway Preview foldout is located just below the Railway Set reference:



To preview railways, all you need to do is to ensure they are expanded within the Railway Setup tab and not collapsed before clicking the Create/Update button.

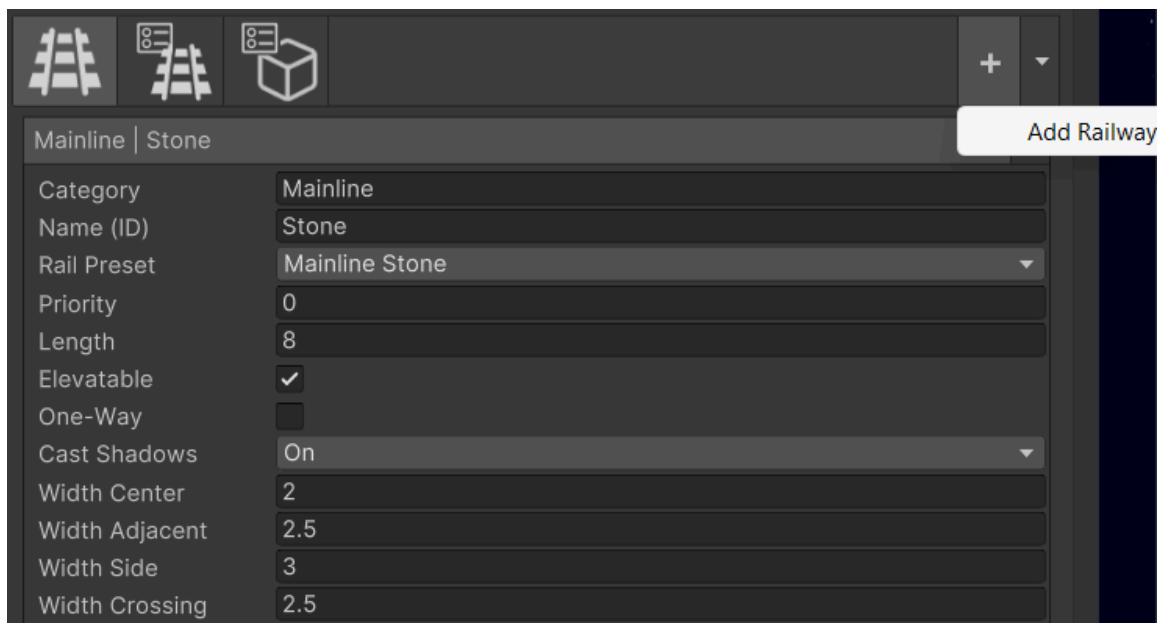


Tip:

For a smoother workflow, you can open a second Inspector by right-clicking the Railway Constructor component in the hierarchy and selecting “Properties...”.

You can then use the second Inspector solely for the “Create/Update” button.

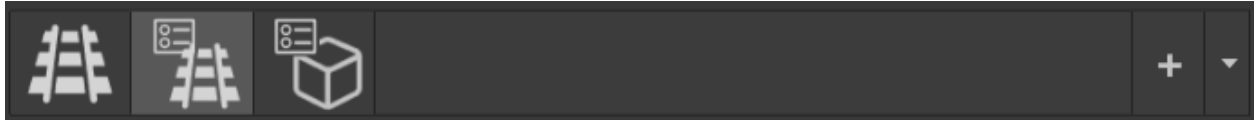
Of the three available menus, the one on the left contains the lanes for construction. New railways can be created using the '+' button on the right.



The category is used for rail and object presets, the name is required and must be unique within this railway set.

Rails and Sleepers

Rails and sleepers can only be created through presets. To create or modify rails, click the second item in the railway set. To add a new rail preset, you can use the plus button on the right side.



The available fields are explained in detail through tooltips. The demo railway set provides a good example of how a rail setup may look. The railway preview is also useful here.

Important: Ensure that each rail preset has a unique name.

Lanes

The 'Construction Lanes' list is where ballast lanes are added. After adding a new lane, you can hover over the different fields to see helpful information about what the different types are used for.

The Rail Preset allows you to select the set for the rails and sleepers.

The screenshot shows a configuration panel for a 'Mainline Stone' track. The panel has a dark theme with white text. At the top, there are three icons: a track, a track with a signal, and a track with a crossing. Below these is a header 'Mainline | Stone' with a dropdown arrow. The main section contains a list of settings: Category (Mainline), Name (ID) (Stone), Rail Preset (Mainline Stone), Priority (0), Length (8), Elevatable (checked), One-Way (unchecked), Cast Shadows (On), Width Center (2), Width Adjacent (2.5), Width Side (3), and Width Crossing (2.5). Below this is a section titled 'Construction Lanes' with a dropdown arrow. It contains a table with three columns: Type (Center), Material (Granite), and Height Curve (a green line graph).

Category	Mainline
Name (ID)	Stone
Rail Preset	Mainline Stone
Priority	0
Length	8
Elevatable	<input checked="" type="checkbox"/>
One-Way	<input type="checkbox"/>
Cast Shadows	On
Width Center	2
Width Adjacent	2.5
Width Side	3
Width Crossing	2.5
▼ Construction Lanes	
Type	Center
Material	Granite
Height Curve	

Again, you can use the preview window to visualize the current setup and to adjust rail counts and elevation as well.

The initial setup is configured for a single-lane railway. Multiple adjacent railways can be constructed dynamically using the builder's 'Rail Count' field.

The screenshot shows a railway builder interface. It has a dark theme with white text. At the top, there are four buttons: 'Railway', 'Ramp', 'Turnout', and 'Station'. To the right of these are three buttons: 'Demolish', 'Move', and 'Reverse'. At the bottom right is a button labeled 'Undo'. Below the buttons are two input fields: 'Rail Count' with the value '2' and 'Delta Height' with the value '0'.

Railway	Demolish	Undo
Ramp	Move	
Turnout	Reverse	
Station	Crossing	
Rail Count	2	
Delta Height	0	

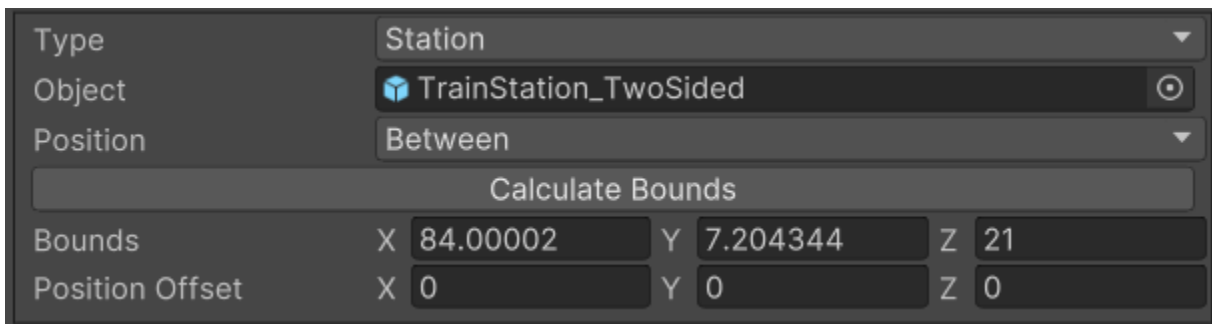
Object Presets

Object presets are a great way to save time and maintain a cleaner, more manageable railway set, especially for larger collections. These presets are optional.

Objects are independent GameObjects that are parented to the created railways. These can include traffic lights, railway signs, props, and more.

All the objects included with the asset can be found in the 'PampelGames/RailwayConstructor/Content/Objects/Prefabs' folder.

To spawn objects, you can add them through the 'Objects' list available on each railway. You can also create object presets, which function similarly to rail presets.



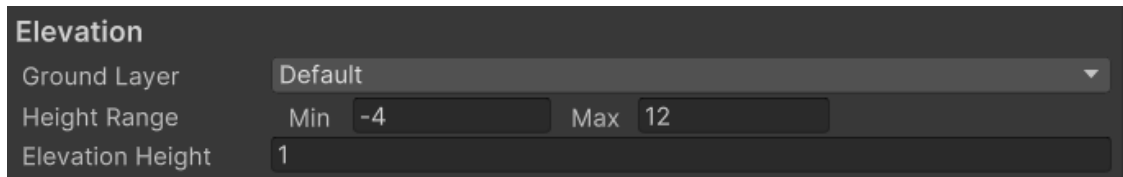
The screenshot shows a configuration panel for an object preset. It has a dark grey background with white text. The 'Type' dropdown is set to 'Station'. The 'Object' dropdown is set to 'TrainStation_TwoSided', which has a small blue cube icon and a circular arrow icon. The 'Position' dropdown is set to 'Between'. Below these is a 'Calculate Bounds' button. At the bottom, there are two rows of input fields: 'Bounds' and 'Position Offset'. Each row has three fields for X, Y, and Z coordinates. The 'Bounds' row has X: 84.00002, Y: 7.204344, and Z: 21. The 'Position Offset' row has X: 0, Y: 0, and Z: 0.

Type	Station		
Object	TrainStation_TwoSided		
Position	Between		
Calculate Bounds			
Bounds	X 84.00002	Y 7.204344	Z 21
Position Offset	X 0	Y 0	Z 0

For more detailed examples of how objects can be set up, please refer to the 'DemoRailways' railway set, located in the Demo folder.

Bridges

For the asset, bridges are simply elevated railways. A railway becomes elevated when its height above the ground exceeds the 'Elevation Height' setting within the Railway Constructor settings:



Elevation

Ground Layer: Default

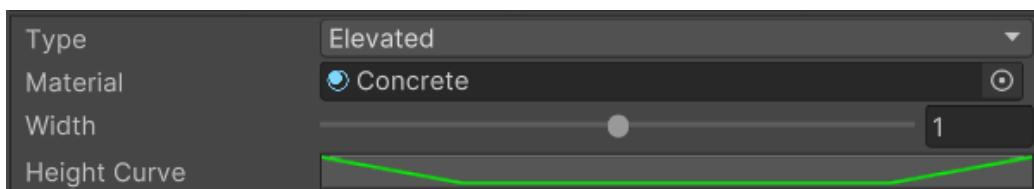
Height Range: Min -4 Max 12

Elevation Height: 1

A railway can consist of multiple elevated and non-elevated sections consecutively, such as on uneven terrain.

To make bridges (elevated railways) stand out from regular railways, there are two features you can use:

1. Construction lanes can be configured to apply only when the railway is elevated. This will for create geometry beneath the railways:



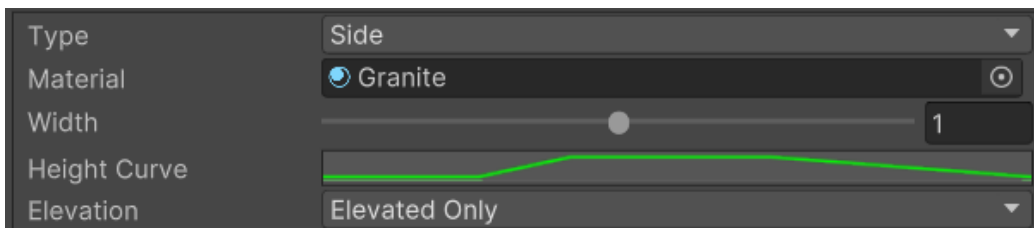
Type: Elevated

Material: Concrete

Width: 1

Height Curve: [Graph showing a slight dip in the center]

The Side lane also has an elevation setting:



Type: Side

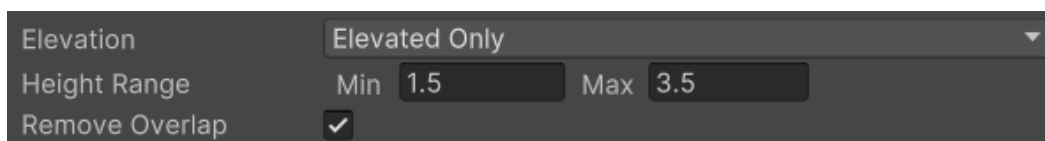
Material: Granite

Width: 1

Height Curve: [Graph showing a slight rise in the center]

Elevation: Elevated Only

2. Objects of type 'Custom' can be spawned based on the elevation as well. This allows you for example to place specific objects beneath elevated railways, such as support pillars.



Elevation: Elevated Only

Height Range: Min 1.5 Max 3.5

Remove Overlap: ☒

TRAFFIC SYSTEM

Traffic Component

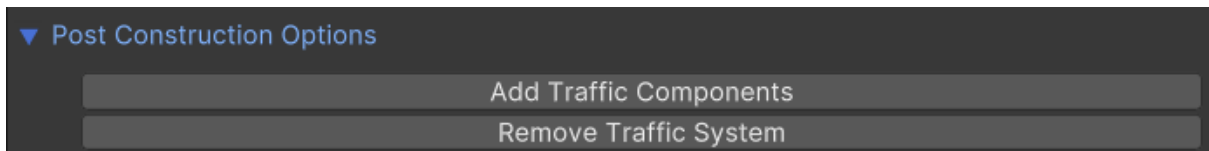
The 'Traffic' component can be added automatically to new railways by enabling the setting:



Traffic lanes are independent of construction lanes and contain separate splines for each rail, as well as additional data for connecting neighbours.

For more information, see the 'Traffic.cs' script, which also contains the Traffic Lane and neighbor details.

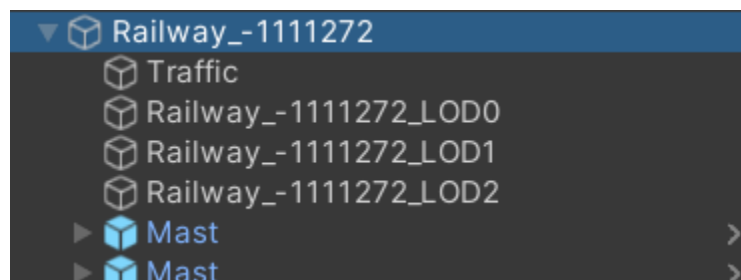
If you are working with the Railway Builder Editor, you can (re-)create them manually using the dedicated buttons:



Or you can create them via code, either for single railways or for the full system at once (please refer to the [API](#) for more details).

```
railwayConstructor.AddTrafficComponents();
```

Once created, you can find the traffic components added as child GameObjects on each railway and intersection.



Train Controller

The asset includes two trains (Mainland and Subway) along with a ready-made train controller (the 'TrainController.cs' script). To see them in action, open the 'RailwayDemoEditor' scene and press Play.

The trains move realistically along the rails and stop at StationObjects.

The controller generates its path once at Start(), which means it does not automatically detect newly added or removed railways by default.

The controller can be easily configured for custom trains, as shown in the demo scene. Settings such as acceleration, speed, and braking times are included by default, and with some coding experience you can expand its functionality further.

For example, if you want the controller to adjust dynamically to created or removed objects, you can use the Action delegates in the RailwayConstructor component:

```
public event Action<List<RailwayObject>, List<IntersectionObject>>>
OnRailwaysAdded = delegate { };
public event Action<List<RailwayObject>, List<IntersectionObject>>>
OnRailwaysRemoved = delegate { };
```

SAVE SYSTEM

The API provides convenient methods to retrieve easily serializable data, which you can use with custom save systems. Loading the data and recreating the railway system is just as straightforward.

The following methods are available to support the saving process (see [API](#) for details):

```
public List<SerializedSceneObject> GetSerializableRailwaySystem()  
  
public void AddSerializableRailwaySystem(List<SerializedSceneObject>  
serializedSceneObjects)  
  
public void SetSerializableRailwaySystem(List<SerializedSceneObject>  
serializedSceneObjects)
```

With the exception of the `UnityEngine.Splines.Spline` class and `UnityEngine.Splines.BezierKnot` struct, all data is stored as value types, incl. Lists of value types.

Please see the next page for examples.

Example 1

Here an example of how to save and load a road system using [EasySave](#) from the Unity Asset Store. This should work similarly to other generic saving solutions:

Saving

```
var railwaySystem =  
roadConstructor.GetSerializableRailwaySystem();  
ES3.Save(SaveKey, railwaySystem);
```

Loading

```
var railwaySystem = ES3.Load<List<SerializedSceneObject>>(SaveKey);  
roadConstructor.SetSerializableRailwaySystem(railwaySystem);
```

Example 2

You can also use the PGSaveSystem, which is included in the asset by default. To do this, you'll need an additional class to hold the road system. Mark this class as [Serializable], and mark the railway list as [SerializeReference].

```
[Serializable]  
public class RailwaySerialization  
{  
    [SerializeReference] public List<SerializedSceneObject>  
serializedSceneObjects;  
}
```

Saving

```
var serializedSceneObjects =  
roadConstructor.GetSerializableRailwaySystem();  
var railwaySerialization = new RailwaySerialization();  
railwaySerialization.serializedSceneObjects = serializedSceneObjects;  
PGSaveSystem.Save(railwaySerialization, "railways", 0);
```

Loading

```
var railwaySerialization =  
PGSaveSystem.Load<RailwaySerialization>("railways", 0);  
var serializedSceneObjects =  
railwaySerialization.serializedSceneObjects;  
roadConstructor.SetSerializableRailwaySystem(serializedSceneObject  
s);
```

INTEGRATIONS

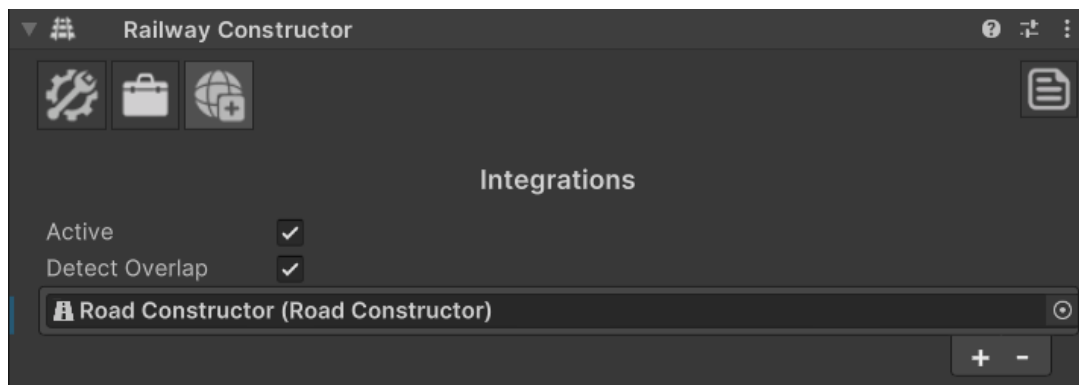
Road Constructor

Road Constructor can be used for both overlap detection and creating seamless crossings.

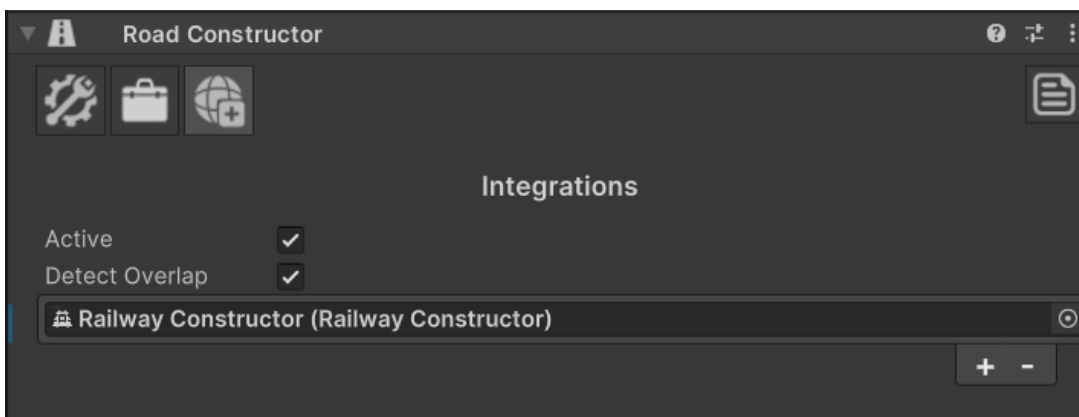
To set up the integration, add the RoadConstructor component to the RailwayConstructor integrations list, and also add the RailwayConstructor component to the RoadConstructor integrations list.

If you only want to create crossings and don't require overlap detection, you only need to do the second step - adding the RailwayConstructor component to the RoadConstructor integrations.

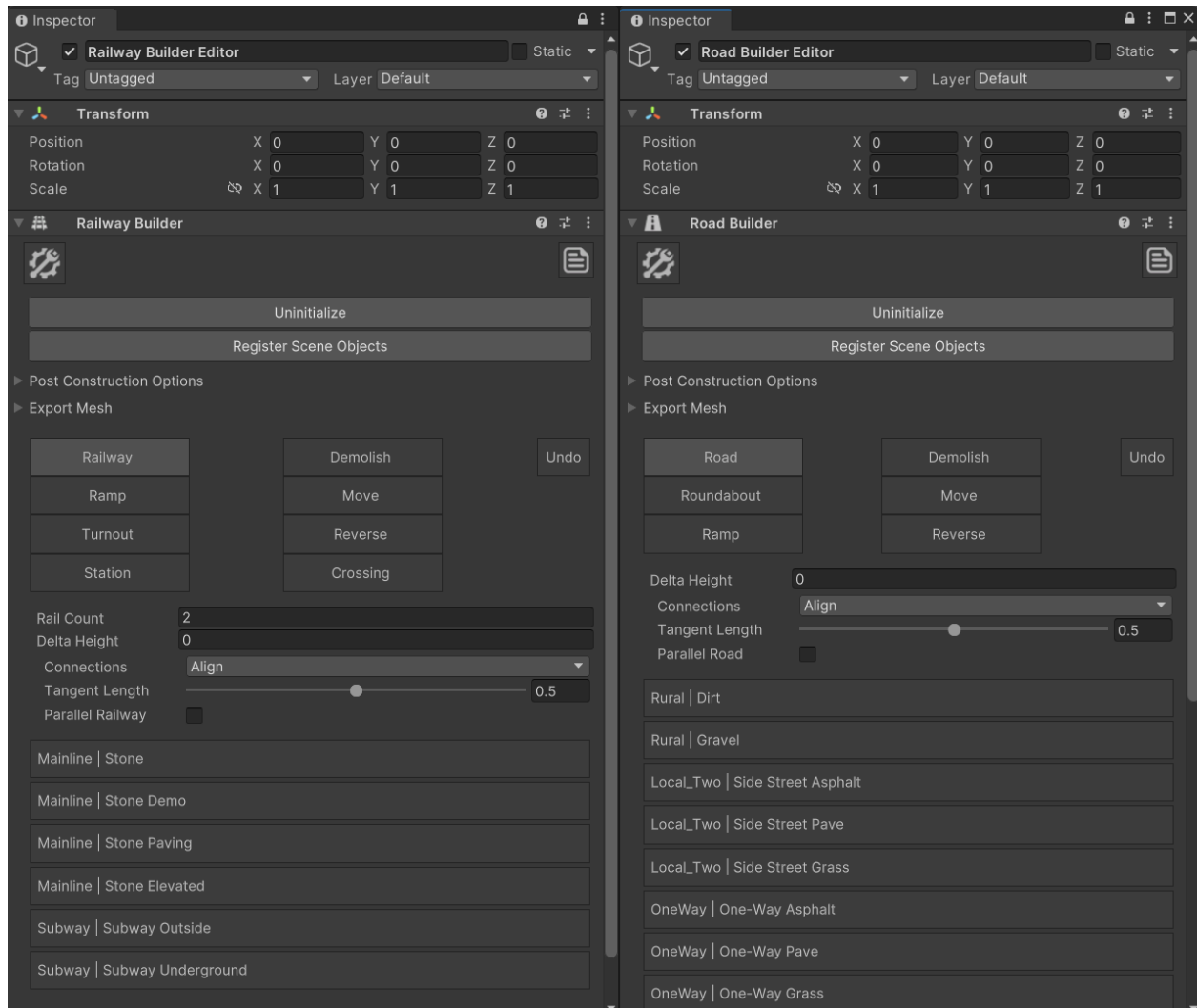
Step 1:



Step 2:



To build in the editor, you can either use a single inspector with both the Road Builder and Railway Builder components attached, or simply open a second, locked inspector (recommended).



Note: Due to a Unity Editor limitation, you cannot draw two RailwayBuilder or RoadBuilder components in separate inspectors at the same time.

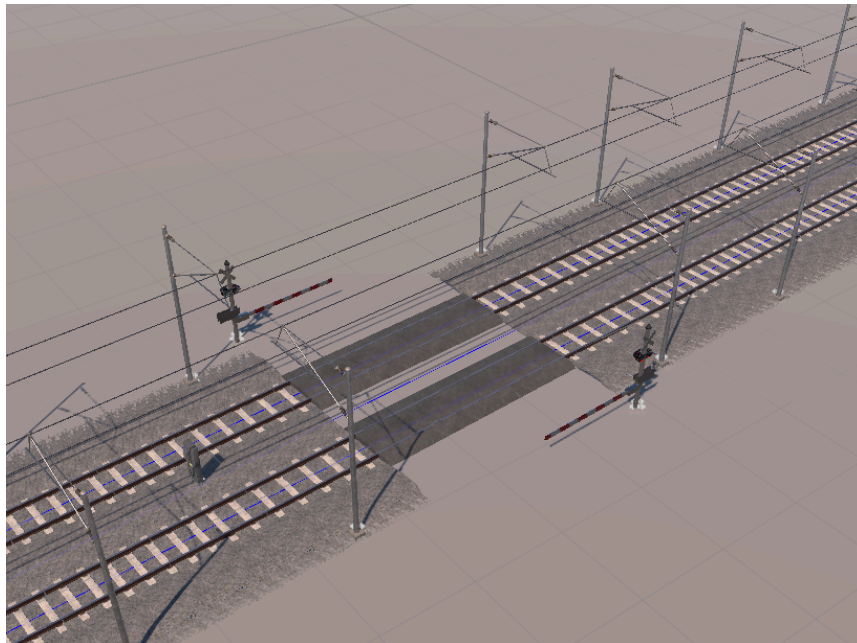
Creating Crossings

- 1) Ensure that both the Road Builder and Railway Builder are initialized and that constructed objects are registered. (If continuing from a previous construction, use the 'Register Scene Objects' buttons.)
- 2) If you haven't already, build a railway in the scene. For best results, make the section where the crossing will be created straight.
- 3) Activate the 'Crossing' tab and click on the existing railway in the scene to add a crossing.

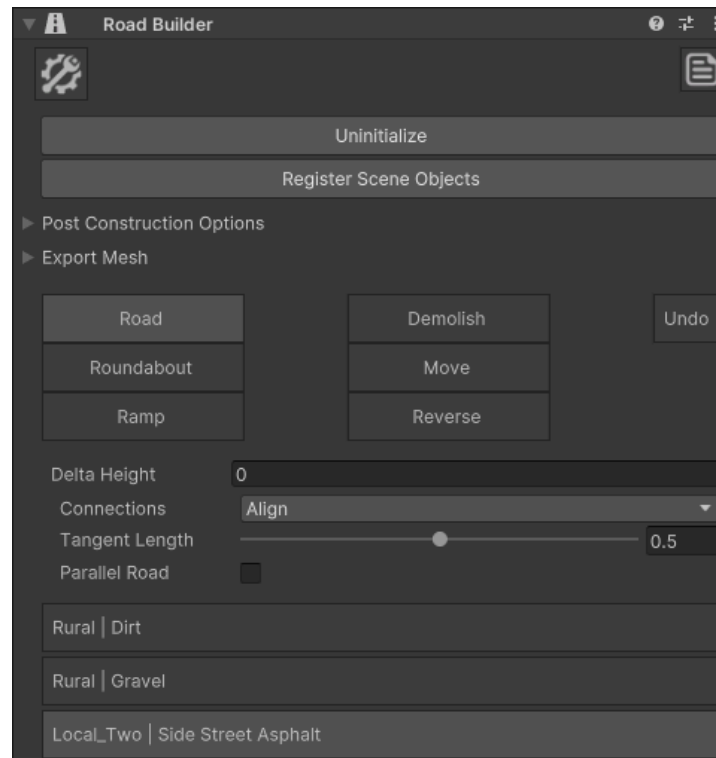
Railway	Demolish	Undo
Ramp	Move	
Turnout	Reverse	
Station	Crossing	

Select a position on a railway in the scene to insert a crossing

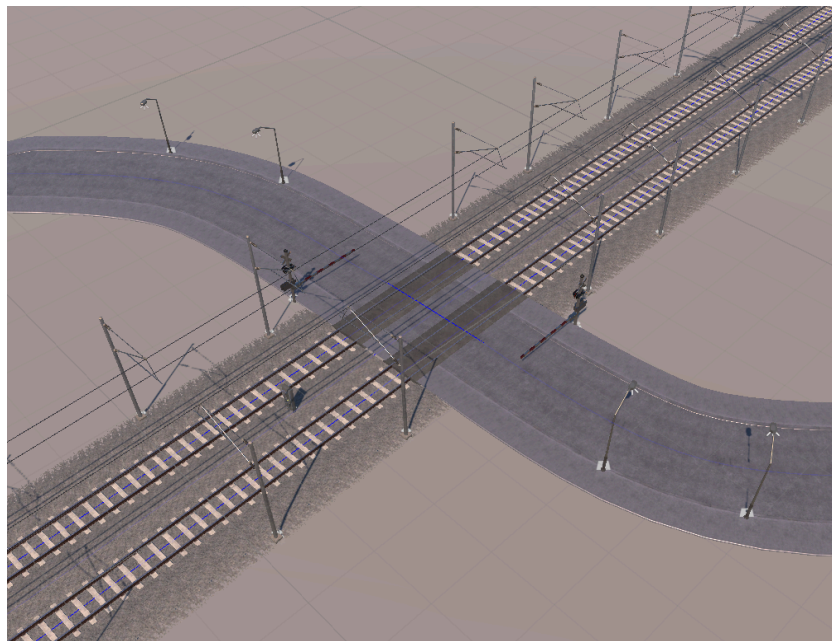
Rail Count	2
Delta Height	0
Crossing Length	10
Width (Add)	1



- 4) Deactivate the Crossing tab (or any other tabs in the Railway Builder) and select the Road tab in the Road Builder. Then choose the road you want to use for the crossing.



- 5) In the scene, ensure the mouse snaps correctly to one end of the crossing, then create the road as usual. That's it!



CUSTOMIZATION

Railway Builder

A 'Railway Builder' simply refers to a script that interacts with the Railway Constructor API. You don't have to create a 'Builder' at all; you can simply call the Railway Constructor methods from any location you prefer.

The included 'RailwayBuilder.cs' (editor) and 'RailwayBuilderPlayer.cs' (runtime) scripts both inherit from 'RailwayBuilderBase.cs' because they share certain functionalities.

The Railway Builder Editor should cover everything you need for railway creation in the editor. The logic is somewhat more complex because it runs in the 'OnSceneGUI' method within an editor script rather than simply in an update loop.

The Railway Builder Player is located in the demo folder, as it is intended to provide you with an idea of how to utilize the Railway Constructor in actual gameplay.

If you want to create your own game logic, you may want to at least take a quick look at the player script to see what methods are available in the Railway Constructor API and how you might want to call them. The script is well commented, and with basic C# knowledge, you should be able to familiarize yourself with it. The demo uses the UI Toolkit, which you are not required to use.

API

RailwayConstructor.cs

```
///      Initializes this <see cref="RailwayConstructor" /> component  
for constructing. In play-mode, this is done automatically in Awake.  
public void Initialize()
```

```
///      Deinitializes this <see cref="RailwayConstructor" />  
component, which includes clearing the history and removing helper  
objects.  
public void Uninitialize()
```

```
///      Returns true if this component has been initialized.  
public bool IsInitialized()
```

```
///      Registers existing scene objects for construction.  
///      The component has to be initialized first.  
///      The active construction set needs to contain the railway name  
for each object to be registered.  
public void RegisterSceneObjects()
```

```
///      Registers existing scene objects for construction.  
///      The component has to be initialized first.  
///      The active construction set needs to contain the railway name  
for each object to be registered.  
public void RegisterSceneObjects(out List<RailwayObject>  
sceneRailwayObjects, out List<IntersectionObject>  
sceneIntersectionObjects)
```

```
///      Gets a <see cref="Railway" /> by name.
public bool TryGetRailway(string railwayName, out Railway railway)

///      Gets a <see cref="RailwayDescr" /> by railway name.
public bool TryGetRailwayDescr(string railwayName, out RailwayDescr
railwayDescr)

///      Gets a list of all registered <see cref="RailwayObject" />s
from the scene.
public List<RailwayObject> GetRailways()

///      Gets a list of all registered <see cref="IntersectionObject"
/>s from the scene.
public List<IntersectionObject> GetIntersections()

///      Gets a list of all registered <see cref="RailwayObject" />s
and <see cref="IntersectionObject" />s from the scene.
public List<SceneObject> GetSceneObjects()

///      Gets the parent object holding all registered <see
cref="RailwayObject" />s and <see cref="IntersectionObject" />s from
the scene.
public Transform GetConstructionParent()

///      Clears all displayed objects and displayed demolish objects in
the scene hierarchy.
public void ClearAllDisplayObjects(bool activateSceneObjects = true)

///      Clears all displayed objects in the scene hierarchy.
public void ClearDisplayedObjects()
```

```

///      Clears all displayed demolish objects in the scene hierarchy.
public void ClearDisplayedDemolishObjects()

///      Clears all displayed move intersection objects in the scene
hierarchy.
public void ClearDisplayedMoveIntersectionObjects()

///      Snaps a position to the nearest railway, intersection or grid
using the width of the railway.
public Vector3 SnapPosition(string railwayName, Vector3 position, int
railCount, out Overlap overlap)

///      Snaps a position to the nearest railway, intersection or grid
using a custom radius.
public Vector3 SnapPosition(float radius, Vector3 position, out
Overlap overlap)

///      Undoes the previous construction operation and restores the
previous state. Requires objects within the undo storage.
public void UndoLastConstruction()

///      Removes all registered undo objects from the scene.
public void ClearUndoStorage()

///      Creates a railway object and two intersection objects (start
and end) for a segment between two positions. These objects are not
initialized in the system and should be deleted in the next frame via
the <see cref="ClearAllDisplayObjects" /> method.
<param name="railwayName">The name of the railway.</param>
<param name="position01">The first railway position.</param>
<param name="position02">The second railway position.</param>
<returns>The construction result, including new objects and detailed
construction failures.</returns>
public ConstructionResultRailway DisplayRailway(string railwayName,
float3 position01, float3 position02)

```

/// Creates a railway object and two intersection objects (start and end) for a segment between two positions. These objects are not initialized in the system and should be deleted in the next frame via the <see cref="ClearAllDisplayObjects" /> method.

*<param name="railwayName">The name of the railway.</param>
<param name="spline">Spline used for construction. Note that only the first and last knots are utilized - any others will be disregarded.
<returns>The construction result, including new objects and detailed construction failures.</returns>*

```
public ConstructionResultRailway DisplayRailway(string railwayName,  
Spline spline)
```

*/// Creates a railway object and two intersection objects (start and end) for a segment between two positions.
/// These objects are not initialized in the system and should be deleted in the next frame via the <see cref="ClearAllDisplayObjects" /> method.*

*<param name="railwayName">The name of the railway.</param>
<param name="position01">The first railway position.</param>
<param name="position02">The second railway position.</param>
<param name="railwaySettings">Optional settings that can be dynamically applied.</param>
<returns>The construction result, including new objects and detailed construction failures.</returns>*

```
public ConstructionResultRailway DisplayRailway(string railwayName,  
float3 position01, float3 position02, RailwaySettings  
railwaySettings)
```

Constructs new intersection and railway objects for a segment between two positions and registers them into the construction system.

*<param name="railwayName">The name of the railway.</param>
<param name="position01">The first railway position.</param>
<param name="position02">The second railway position.</param>
<returns>The construction result, including new objects and detailed construction failures.</returns>*

```
public ConstructionResultRailway ConstructRailway(string railwayName,  
float3 position01, float3 position02)
```

Constructs new intersection and railway objects for a segment between two positions and registers them into the construction system.

<param name="railwayName">The name of the railway.</param>

<param name="spline">Spline used for construction. Note that only the first and last knots are utilized - any others will be disregarded.</param>

<returns>The construction result, including new objects and detailed construction failures.</returns>

```
public ConstructionResultRailway ConstructRailway(string railwayName,
Spline spline, int railCount)
```

Constructs new intersection and railway objects for a segment between two positions and registers them into the construction system.

<param name="railwayName">The name of the railway.</param>

<param name="position01">The first railway position.</param>

<param name="position02">The second railway position.</param>

<param name="railwaySettings">Optional settings that can be dynamically applied.</param>

<returns>The construction result, including new objects and detailed construction failures.</returns>

```
public ConstructionResultRailway ConstructRailway(string railwayName,
float3 position01, float3 position02, RailwaySettings
railwaySettings)
```

Constructs a ramp between two specified positions. A ramp connects seamlessly to existing railways. Display objects are not initialized in the system and should be deleted in the next frame via the <see cref="ClearAllDisplayObjects" /> method.

<param name="railwayName">The name of the railway for the ramp.</param>

<param name="position01">The first position defining the ramp.</param>

<param name="position02">The second position defining the ramp.</param>

<param name="railwaySettings">Additional settings for constructing the railway ramp.</param>

<returns>A <see cref="ConstructionResultRamp" /> Result of the construction process.</returns>

```
public ConstructionResultRamp DisplayRamp(string railwayName, float3
position01, float3 position02, RailwaySettings railwaySettings)
```

Constructs a ramp between two specified positions. A ramp connects seamlessly to existing railways.

*<param name="railwayName">The name of the railway for the ramp.</param>
<param name="position01">The first position defining the ramp.</param>
<param name="position02">The second position defining the ramp.</param>
<param name="railwaySettings">Additional settings for constructing the railway ramp.</param>
<returns>A <see cref="ConstructionResultRamp" /> Result of the construction process.</returns>*

```
public ConstructionResultRamp ConstructRamp(string  
railwayName, float3 position01, float3 position02, RailwaySettings  
railwaySettings)
```

Constructs a railway turnout between two specified positions. A turnout is similar to a railway, with an additional turnout railway. These objects are not initialized in the system and should be deleted in the next frame via the <see cref="ClearAllDisplayObjects" /> method.

*<param name="railwayName">The name of the railway to be constructed.</param>
<param name="position01">The first position for the turnout.</param>
<param name="position02">The second position for the turnout.</param>
<param name="railIndex01">The rail index corresponding to the turnout start.</param>
<param name="railIndex02">The rail index corresponding to the turnout end.</param>
<param name="railwaySettings">The settings object that defines additional configuration for the railway.</param>
<returns> <see cref="ConstructionResultTurnout" /> Result of the construction process.</returns>*

```
public ConstructionResultTurnout DisplayTurnout(string  
railwayName, float3 position01, float3 position02,  
int railIndex01, int railIndex02, RailwaySettings  
railwaySettings)
```

Constructs a railway turnout between two specified positions. A turnout is similar to a railway, with an additional turnout railway.

<param name="railwayName">The name of the railway to be constructed.</param>

<param name="position01">The first position for the turnout.</param>

<param name="position02">The second position for the turnout.</param>

<param name="railIndex01">The rail index corresponding to the turnout start.</param>

<param name="railIndex02">The rail index corresponding to the turnout end.</param>

<param name="railwaySettings">The settings object that defines additional configuration for the railway.</param>

<returns> <see cref="ConstructionResultTurnout" /> Result of the construction process.</returns>

```

    public ConstructionResultTurnout ConstructTurnout(string
railwayName, float3 position01, float3 position02,
        int railIndex01, int railIndex02, RailwaySettings
railwaySettings)

```

Displays a railway station. These objects are not initialized in the system and should be deleted in the next frame via the <see cref="ClearAllDisplayObjects" /> method.

<param name="railwayName"> The name of the railway to associate with the station.</param>

<param name="position01">The starting position of the station, with y defining the flat height.</param>

<param name="position02">The direction of the station.</param>

<param name="tryUseObjectBounds">

If an object of type 'Station' is found for this railway, its bounds will be used for the platform length and spacing.

<param name="platformLength">The length of the platform (if no station object is used).</param>

<param name="platformSpacing">The spacing between platforms (if no station object is used).</param>

<param name="connectionLength">The length between a connecting railway and the platform.</param>

<param name="railwaySettings">Optional configuration settings for the railway construction process.</param>

```

public ConstructionResultStation DisplayStation(string railwayName,
float3 position01, float3 position02, bool tryUseObjectBounds,
    float platformLength, float platformSpacing, float
connectionLength, RailwaySettings railwaySettings)

```

Constructs a railway station.

<param name="railwayName"> The name of the railway to associate with the station.</param>

<param name="position01">The starting position of the station, with y defining the flat height.</param>

<param name="position02">The direction of the station.</param>

<param name="tryUseObjectBounds">

If an object of type 'Station' is found for this railway, its bounds will be used for the platform length and spacing.

<param name="platformLength">The length of the platform (if no station object is used).</param>

<param name="platformSpacing">The spacing between platforms (if no station object is used).</param>

<param name="connectionLength">The length between a connecting railway and the platform.</param>

<param name="railwaySettings">Optional configuration settings for the railway construction process.</param>

<returns> A <see cref="ConstructionResultStation" /> representing the result of the station construction operation.</returns>

```
public ConstructionResultStation ConstructStation(string railwayName,  
float3 position01, float3 position02, bool tryUseObjectBounds, float  
platformLength, float platformSpacing, float connectionLength,  
RailwaySettings railwaySettings)
```

Displays a crossing based on the snapped railway and parameters.

This method constructs a visual preview of a railway crossing using specified positions and settings.

<param name="position">The position of the crossing on an existing railway.</param>

<param name="searchRadius">The search radius within which the existing railway object will be found and snapped to.</param>

<param name="length">Length of the crossing to be inserted on an existing railway.</param>

<param name="widthAdd">The width is determined by the railway width. This value adds extra space.</param>

<returns>A <see cref="ConstructionResultCrossing" /> representing the result of the crossing display.</returns>

```
public ConstructionResultCrossing DisplayCrossing(Vector3 position,  
float searchRadius, float length, float widthAdd)
```


Constructs a railway crossing on the snapped railway using the given railway settings.

<param name="position">The position of the crossing on an existing railway.</param>

<param name="searchRadius">The search radius within which the existing railway object will be found and snapped to.</param>

<param name="length">Length of the crossing to be inserted on an existing railway.</param>

<param name="widthAdd">The width is determined by the railway width. This value adds extra space.</param>

<returns>A <see cref="ConstructionResultCrossing" /> representing the result of the construction process.</returns>

```
public ConstructionResultCrossing ConstructCrossing(Vector3 position,
float searchRadius, float length, float widthAdd)
```

Displays move intersection object.

<param name="currentPosition">The current position where the intersection object is displayed.</param>

<param name="searchRadius">The search radius within which the intersection object will be found.</param>

<param name="newPosition">The new position where the intersection object will be displayed.</param>

<param name="deactivateSceneObjects">A flag indicating whether to deactivate scene objects.</param>

<returns>A list of GameObjects representing the displayed move intersection objects.</returns>

```
public List<GameObject> DisplayMoveIntersection(Vector3
currentPosition, float searchRadius, Vector3 newPosition,
    bool deactivateSceneObjects = true)
```

Moves an existing <see cref="IntersectionObject" /> to a new position. Undo history will be lost after this method is called.

<param name="currentPosition">The position where to search for the intersection.</param>

<param name="searchRadius">The radius within which to search for the intersection.</param>

<param name="newPosition">The new position to move the intersection to.</param>

<returns>A <see cref="ConstructionResultMoveIntersection" /> representing the result of the operation.</returns>

```
public ConstructionResultMoveIntersection MoveIntersection(Vector3
currentPosition, float searchRadius, Vector3 newPosition)
```

Reverses the direction of a railway within the specified search radius from the given position. If a valid railway is found, it is inverted and replaced with a newly constructed railway.

<param name="position">The position at which to search for a railway.</param>
<param name="searchRadius">The radius within which to search for a railway.</param>
<param name="replacedRailway">Newly constructed railway.</param>
`public bool ReverseRailwayDirection(Vector3 position, float searchRadius, out RailwayObject replacedRailway)`

Reverses the direction of the specified railway.

<param name="railway">The railway object whose direction is to be reversed.</param>
<param name="replacedRailway">The new railway object created with the reversed direction.</param>
`public void ReverseRailwayDirection(RailwayObject railway, out RailwayObject replacedRailway)`

Creates display objects for demolishing for the specified position. These objects are not initialized in the system and should be deleted in the next frame via the *<see cref="ClearAllDisplayObjects" />* method.

<param name="position">The position from which to demolish objects.</param>
<param name="searchRadius">Search radius to find the demolishable objects.</param>
<param name="deactivateSceneObjects">Deactivates the scene objects.</param>
<returns></returns>
`public List<GameObject> DisplayDemolishObjects(Vector3 position, float searchRadius, bool deactivateSceneObjects = true)`

Demolishes objects within a specified radius from a given position.

<param name="position">The position from which to demolish objects.</param>
<param name="searchRadius">Search radius to find the demolishable objects.</param>
`public void Demolish(Vector3 position, float searchRadius)`

Demolishes the specified intersections and roads.

```
public void Demolish(List<IntersectionObject> intersections,  
List<RailwayObject> roads)
```

Updates the colliders of the specified scene objects based on the component settings.

<param name="sceneObjects">The list of scene objects to update colliders for.</param>

```
public void UpdateColliders<T>(List<T> sceneObjects) where T :  
SceneObject
```

Update the layers and tags of scene objects based on the component settings.

<param name="sceneObjects">List of scene objects to update layers and tags</param>

```
public void UpdateLayersAndTags<T>(List<T> sceneObjects) where T :  
SceneObject
```

Editor Only. Exports railway and intersection meshes into the project folder.

<param name="checkExistingMeshes">

Checks the project folder for each mesh before exporting.

If set to false, each mesh will be exported regardless of whether one already exists.

```
public void ExportMeshes(bool checkExistingMeshes)
```

Removes existing <see cref="Traffic" /> components and creates new ones for the full existing railway system.

```
public void AddTrafficComponents()
```

Removes existing <see cref="Traffic" /> components and creates new ones for the specified railways/intersections.

```
public void AddTrafficComponents(List<RailwayObject> railways,  
List<IntersectionObject> intersections)
```

Removes all traffic components from the system.

```
public void RemoveTrafficSystem()
```

For each railway marked as 'expanded' in the Railway Setup tab, a corresponding railway is created in the scene. These objects are parented to the 'Railway Preview' GameObject and not registered in the system.

```
public void ConstructPreviewRailways(Transform parent, int railCount, float railwayLength, bool spawnObjects, bool elevated)
```

Retrieves a list of serialized representations of all scene objects within the railway system. Each object is converted into simple, serializable fields for saving or transmission purposes.

<returns>A list of <see cref="SerializedSceneObject" /> representing the serialized railway system (incl. inheritors).</returns>

```
public List<SerializedSceneObject> GetSerializableRailwaySystem()
```

Reads serialized scene objects and adds them to the existing railway system.

<param name="serializedSceneObjects">A list of serialized scene objects containing data needed to restore the railway system.</param>

```
public void AddSerializableRailwaySystem(List<SerializedSceneObject> serializedSceneObjects)
```

Reads serialized scene objects and reconstructs the railway system.

<param name="serializedSceneObjects">A list of serialized scene objects containing data needed to restore the railway system.</param>

```
public void SetSerializableRailwaySystem(List<SerializedSceneObject> serializedSceneObjects)
```

Removes all registered objects from the railway system.

```
public void ClearRailwaySystem()
```