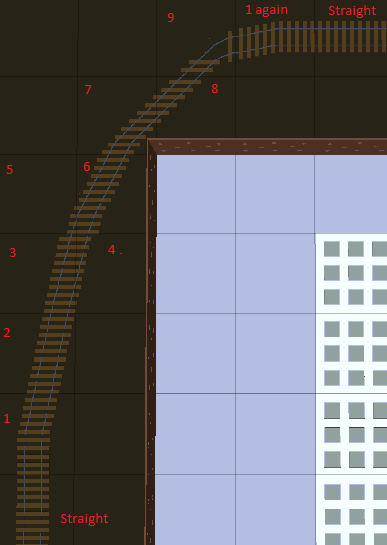
**Voxel City – Train Configuration**

Instructions for creating railroads curves:

For a straight path use the Railroad\_Straight prefab

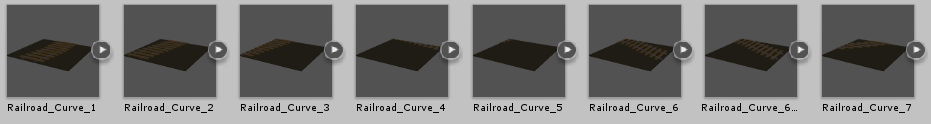


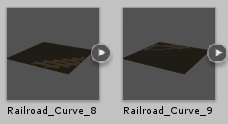
If you need a curve, there are a series of prefabs you will need to use. This was made this way to prevent train collisions against the walls. There are 9 prefabs used for curves and they are used this way:



From a Straight railroad, you start by using the Railroad\_Curve\_1, next one is Curve\_2 and then next ones will go side by side in pairs until you reach Curve\_9 (that means 3 and 4, then 5 and 6, then 7 and 8 as the picture shows).

Curve\_9 will go alone and then you will need to use Curve\_1 again to close the curve and keep going straight with the Straight railroads.





You may notice that there are two Railroad\_Curve\_6, the second one is called Railroad\_Curve\_Straighten. This particular curve can be used if you need to make a slight curve and then keep the road straight.

So follow the same pattern, begin with the Curve\_1, then 2, following by pairs 3 and 4, then 5 and 6 but this time on the #6 use the Straighten one. From there, you’ll be able to continue the path using Railroad\_Straight prefabs. You won’t be making a full curve but will use it to move the train at one side of the grid.

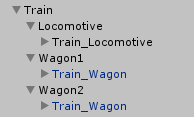
The Train:

You have prefabs for 1 Locomotive and a Wagon. You can repeat as many Wagons as you want to make the train longer.

The recommendation for using those prefabs is to create an Empty Gameobject on the scene for each prefab. Rotate locally the Empty Gameobject in order to have the Z Axis pointing to the front of the path since the Train will move on that axis.



Then put each prefab inside of each Empty Gameobject.

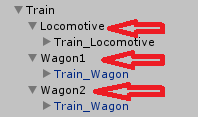


As you can see in the example above, an Empty Gameobject called Train was created to regroup all prefabs. Then 3 Empty Gameobjects were made called Locomotive, Wagon1 and Wagon2 respectively. Inside those Empty Gameobjects are the actual prefabs. Add a Rigidbody Component with the IsKinematic checkbox checked just to the locomotive (not the Wagons).

The Scripts:

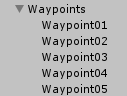
This package provides 3 scripts. Locomotive.cs, TrainWaypoints.cs and Barrier.cs

You’ll need to use TrainWaypoints.cs on all Empty Gameobjects you’re using for the train, locomotive and wagons as well. Then the Locomotive.cs will go ONLY in the locomotive Empty Gameobject.

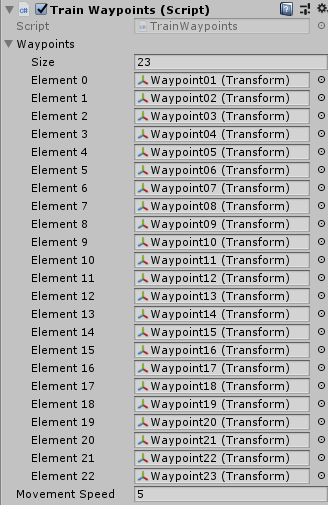


TrainWaypoints includes a List of objects that will mark the way to the train. You will need to create Empty Gameobjects in the scene and position them where you want your train to go.

In the demo scene provided with this package you’ll see an Empty Gameobject called Waypoints and there you’ll see 23 Empty Gamobjects marking train’s way.



Once you have your Empty Gameobjects created along the way, go to the Locomotive TrainWaypoints script and set the list size number with the number of your gameobjects and assign them one by one in the list, as the following screen shows



You can also set the train movement speed from here.

TIP: Once you do this on the locomotive, go to the settings button of the script in the upper right part and select Copy Component, then go to each wagon Empty Gameobject and Paste as New Component, so you’ll avoid doing all of this on each wagon.

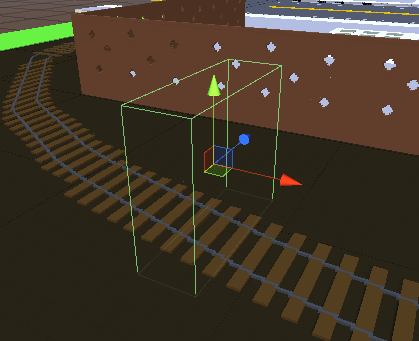
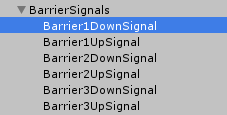
The third script is Barrier.cs and this will allow each barrier to go down when the train is approaching and go up when the train goes away.

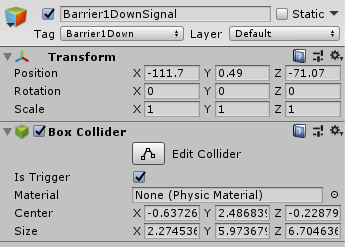
For this to be accomplished you’ll need to create a series of tags and assign them. Go to the Tags & Layers menu and set tags for barriers up and down as the next image shows:



The example sets 3 barriers that were included on the Demo Scene.

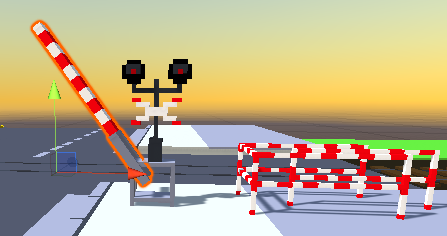
You’ll also need to create Empty GameObjects at the point where you want your next barrier to go down and when your barrier need to go up and then assign them the corresponding tag (i.e. Barrier1Down and Barrier1Up respectively). Then add a Box Collider to those Empty Gameobjects and stretch them enough for train to collide with them. Also set the Box Collider as Trigger.





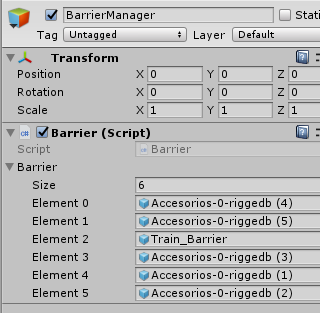
Once you have created all your Up and Down points, create a new Empty Gameobject called for example BarrierManager and assign the Barrier.cs script to it.

This scripts contains an Array of GameObjects that you’ll have to populate with all Barrier prefab Gameobjects you are using on the scene.





So in the script, set the number of barriers to associate and populate the array.



In this example I’m using 6 elements corresponding to barriers. The script will work in pairs, as you have one barrier in front of the other on each side of the street, so element 0 and 1 will work together. 2 will work with 3 and 4 with 5.

You can have as many barriers as you want, just make sure to populate the array with the remaining elements.

The script is prepared to work with 6 elements, but if you have more, then edit the scripts Locomotive.cs and Barrier.cs.

Modifying Locomotive.cs

At the first part you have declared public bool variables for each Up and Down states of the barriers.

public bool sendBarrier1DownSignal = false;

public bool sendBarrier1UpSignal = false;

Just add more variables as you see fit with their values on false.

On the second part you’ll find a switch statement inside the OnTriggerEnter method.

switch (collider.tag)

{

case "Barrier1Down":

sendBarrier1DownSignal = true;

break;

case "Barrier1Up":

sendBarrier1UpSignal = true;

break;

Here you need to add more cases. Note that the switch is validating the collider tag. Remember that you’ve created tags before. Just put the names of those tags here. Finally there are some setters to modify the value of the Booleans from other script.

public void SetDown1Signal(bool value)

{

sendBarrier1DownSignal = value;

}

public void SetUp1Signal(bool value)

{

sendBarrier1UpSignal = value;

}

Just add more setters for new signals you create.

Modifying Barrier.cs

You will see a method called BarrierManager() with some IF and ELSE IF statements, like the following:

if (locomotive.sendBarrier1DownSignal)

{

barrier[0].GetComponentInChildren<Animator>().Play("BarrierDown");

barrier[1].GetComponentInChildren<Animator>().Play("BarrierDown");

locomotive.SetDown1Signal(false);

}

else if (locomotive.sendBarrier1UpSignal)

{

barrier[0].GetComponentInChildren<Animator>().Play("BarrierUp");

barrier[1].GetComponentInChildren<Animator>().Play("BarrierUp");

locomotive.SetUp1Signal(false);

}

You just need to add more statements like this one pointing to the proper position in the Array (replace barrier[0] and barrier[1] with the number of the proper elements in the array).

You also need to change the condition of the IF statement, pointing to the signal that needs to be activated:

Replace if (locomotive.sendBarrier1DownSignal) with the proper variable name created in Locomotive.cs script

Once you have all set, you’ll be ready to go.

If you have any doubts, contact [hypernaturalgames@gmail.com](mailto:hypernaturalgames@gmail.com). I’ll try to respond as soon as I can.