
DOTS Traffic City

Release 1.2.1c

604Spirit

Nov 08, 2024

CONTENTS

1 Getting Started	2
1.1 Project Requirements	2
1.2 Limitations	2
1.3 Package Installation	3
1.4 City Creation	5
1.5 Structure	10
2 Road	18
2.1 Installation	18
2.2 Road Segment	23
2.3 Road Segment Creator	29
2.4 Baking Info	70
2.5 Workflow	71
2.6 Traffic Node	72
2.7 Path Creator	79
2.8 Path	85
2.9 Traffic Light	106
2.10 Pedestrian Node	120
2.11 Traffic Area	140
2.12 Traffic Public Route	144
2.13 Runtime Road	149
2.14 Road Configs	160
3 Traffic	162
3.1 Traffic Car	162
3.2 Traffic Public	184
3.3 Car Prefab Creator	186
3.4 Vehicle Collection	200
3.5 Presets	204
3.6 Traffic Configs	209
3.7 Traffic Test Scene	228
4 Pedestrian	229
4.1 Pedestrian	229
4.2 Animation Baker	258
4.3 Pedestrian Test Scene	271
5 Player	282
5.1 Built-in Solution	282
5.2 User Custom Solution	292

5.3	Pure Traffic Simulation	294
6	Npc	295
6.1	Configs	295
7	Streaming	297
7.1	Cullpoint Info	297
7.2	Scene Streaming	298
7.3	Configs	303
8	Common Info	305
8.1	Common Info	305
8.2	Common Configs	308
8.3	Common Test Scenes	313
8.4	Layer Info	314
9	Scene Handle	316
9.1	Bootstrap	316
9.2	Scene Unload	317
10	Test Scenes	318
10.1	Traffic Test Scene	318
11	Sound	335
11.1	Built-in	335
11.2	FMOD	337
11.3	Data	347
11.4	Code Examples	350
12	3rd Party	353
12.1	EasyRoads3D	353
12.2	Integration Cities	358
13	Debug	364
13.1	Traffic Debug	364
13.2	Traffic Node Debug	371
13.3	Pedestrian Debug	379
13.4	Pedestrian Node Debug	383
13.5	Path Debug	386
13.6	Traffic Object Finder	392
13.7	Common Debug	394
14	Glossary	398
14.1	Common Terms	398
14.2	Traffic	398
15	Upgrade Guide	399
15.1	v1.0.x to v1.1.0	399
15.2	HDRP	400
15.3	Cinemachine v3 Upgrade	402
15.4	Cinemachine v2	403
16	Known Issues	405
16.1	Unity.Physics package spikes by BuildPhysicsWorld & SolveAndIntegrate systems on mobile devices.	405
16.2	[FMOD] assert : assertion: ‘level >= 0.0f’ failed	405
16.3	Leak Detected : Persistent allocates 3 individual allocations	405

17 FAQ	406
17.1 Common Questions	406
17.2 Issues	406
17.3 Errors	406
18 Change Log	408
18.1 [1.2.1c] - 08-11-2024	408
18.2 [1.2.1b] - 06-11-2024	408
18.3 [1.2.1] - 04-11-2024	409
18.4 [1.2.0] - 28-10-2024	409
18.5 [1.1.0g] - 19-09-2024	410
18.6 [1.1.0f] - 10-09-2024	411
18.7 [1.1.0e] - 16-08-2024	411
18.8 [1.1.0d] - 12-08-2024	412
18.9 [1.1.0c] - 09-08-2024	412
18.10 [1.1.0b] - 06-08-2024	413
18.11 [1.1.0] - 05-08-2024	413
18.12 [1.0.7d] - 06-06-2024	415
18.13 [1.0.7c] - 31-05-2024	415
18.14 [1.0.7b] - 29-05-2024	415
18.15 [1.0.7] - 24-05-2024	416
18.16 [1.0.6] - 22-04-2024	416
18.17 [1.0.5] - 15-04-2024	417
18.18 [1.0.4] - 04-04-2024	417
18.19 [1.0.3b] - 01-04-2024	418
18.20 [1.0.3] - 29-03-2024	418
18.21 [1.0.2] - 25-03-2024	418
18.22 [1.0.1b] - 22-03-2024	419
18.23 [1.0.1] - 20-03-2024	419
18.24 [1.0.0] - 19-03-2024	419

DOTS Traffic City is a tool for quickly creating performant city traffic.

You can buy it from *Unity Asset Store*: [DOTS Traffic City](#)

[Youtube tutorial playlist](#).

Check out the [*Getting Started*](#) section for more information, including how to install the project.

GETTING STARTED

1.1 Project Requirements

Minimum Unity version:

- 2022.3.21+

Required packages:

- Entities 1.2.0
- Entities.Graphics 1.2.0
- Unity.Physics 1.2.0
- Custom Physics Authoring
- Unity.Collections 2.4.0
- Burst 1.8.12

Asset store packages:

- Zenject. [optional, but recommended].
- FMOD plugin for the sounds [optional].

1.2 Limitations

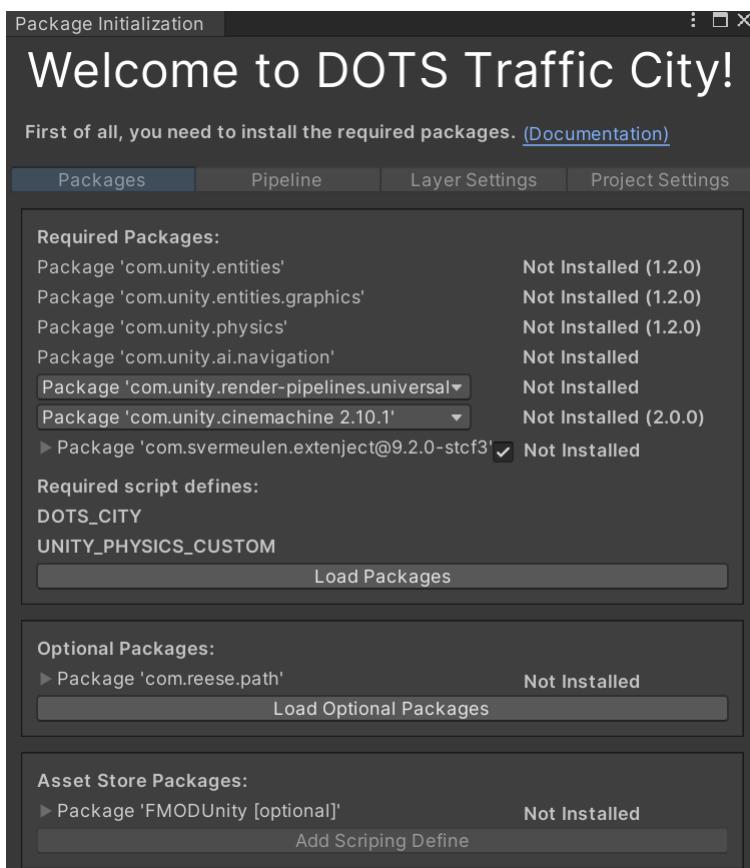
- WebGL not supported.
- Built-in RP not supported.
- Vehicles with trailers or wagons are not currently supported for *NoPhysics*.
- *Animator* with skeletal bone animation in pure **DOTS** space currently not available (available only *hybrid* entities with Animator approach or *pure* entities with **GPU** animations).
- NavMesh surface obstacles only calculated with **NavMeshObstacle**.
- *Ragdoll* currently only collides with **default** colliders.

1.3 Package Installation

Youtube tutorial.

1.3.1 Steps

1. Download & import from the [Unity Asset Store](#).
2. Make sure that you have **Unity 2022.3.21+** (except **2022.3.40 - 2022.3.50** which are [broken](#), download **Unity 2022.3.51+** or **Unity 6.0.23+** instead).
3. If the script first compilation [hangs](#) for more than 5 minutes, end the task in your OS's task manager & restart **Unity**.
4. First time initialization window will appear automatically or you can open it manually from the toolbar **604Spirit/CityEditor/Window/Package Initialization**.



5. Click *Load Packages* to start downloading the packages required for this tool.
6. If you get the error ‘No git executable was found’, install the [git](#) on your PC ([how to install](#)) or download these packages from the store.

Note:

Required custom packages [is optional from version v1.1.0, but recommended]:

- **Extenject** (*com.svermeulen.extenject*) - library for injecting dependencies ([Extenject](#)).

Note:

Script define symbols required for the project:

- DOTS_CITY
 - UNITY_PHYSICS_CUSTOM
-

7. After the packages have been downloaded & installed, if the console has *nunit.framework* error, restart *Unity*.

8. Click *Load Optional Packages* to start downloading the optional packages (*optional package, git required*).

Note:

Optional packages:

- Reese's DOTS Navigation (*com.reese.path*) - Reese's DOTS navigation package for *navigating* on the NavMesh (original git) (the project uses the 604spirit's fork version).
-

Note:

Script define symbols required for the project:

- REESE_PATH
-

Warning: If you get the error 'No git executable was found', read [this](#).

9. Download the optional assets from the *Asset Store* [*from version v1.1.0, steps 9-11 are optional, a built-in audio engine is available by default*]:

Note:

Optional asset store packages:

- FMOD - asset store plugin for *game sounds* FMOD
-

Note:

Script define symbols required for the project:

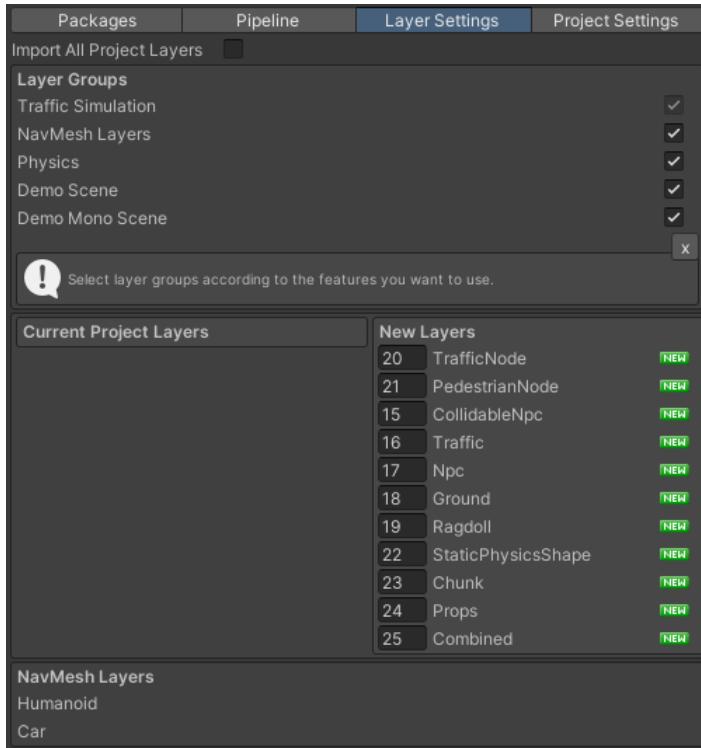
- FMOD
-

10. After that, press the *Add Scripting Define* button.

11. Install the *FMOD sound* settings.

12. If the project is created from scratch, *Pipeline*, *Layer settings*, *Project settings* are automatically installed & go to the last step, if not, follow the next steps.

13. Open the *Pipeline* tab, press the *Import Graphics* button if you want to use the demo pipelines (optional step), otherwise set *Rendering path* to *Forward+* in your pipeline settings.
14. Open *Layer settings* tab & select the layers to import according to your use case & press *Apply* button.



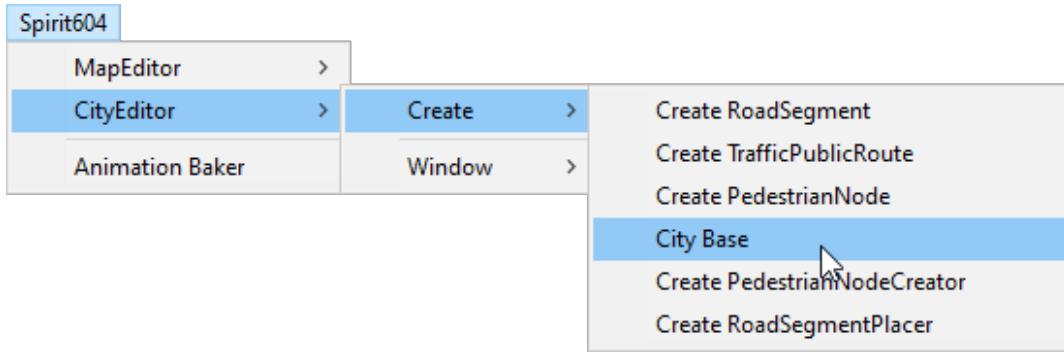
15. **TrafficNode** & **PedestrianNode** layers are **required**, others are optional, read more about project layers [here](#).
16. Open *Project settings* tab & press *Add all scenes to build* if you want to add demo scenes to your project.
17. The next step is [to set up the new scene](#) or launch the existing *Demo* or *Demo Mono* scene.

1.4 City Creation

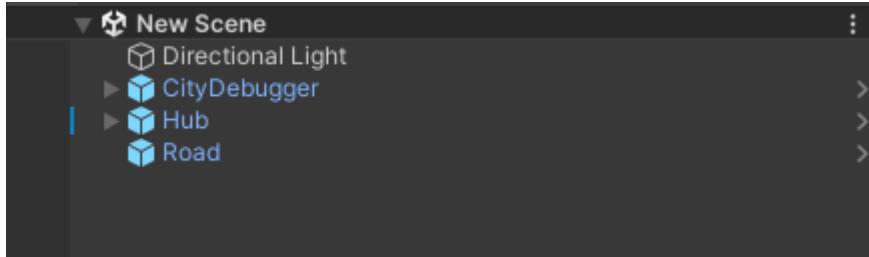
1.4.1 New Scene

1. Create new *scene*.
2. In the *Unity* toolbar open:

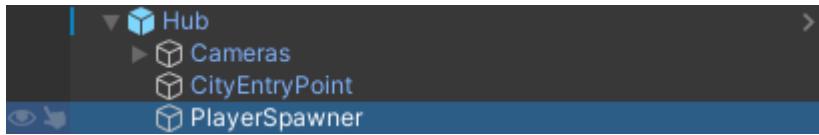
Spirit604/CityEditor/Create/City Base



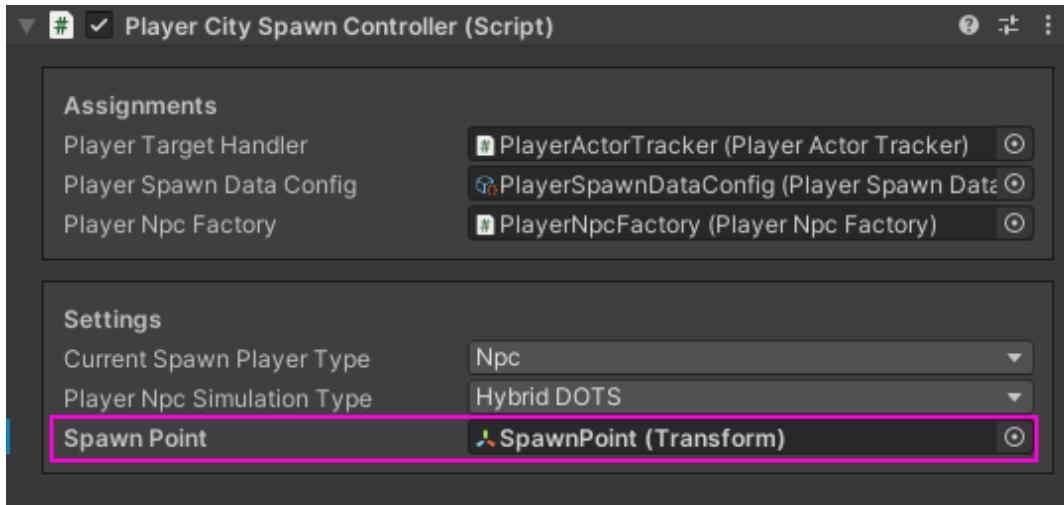
3. Initial scene example:



4. Create a *SpawnPoint object* (any transform) in the scene where you want the player to spawn & assign it to *PlayerSpawner*.

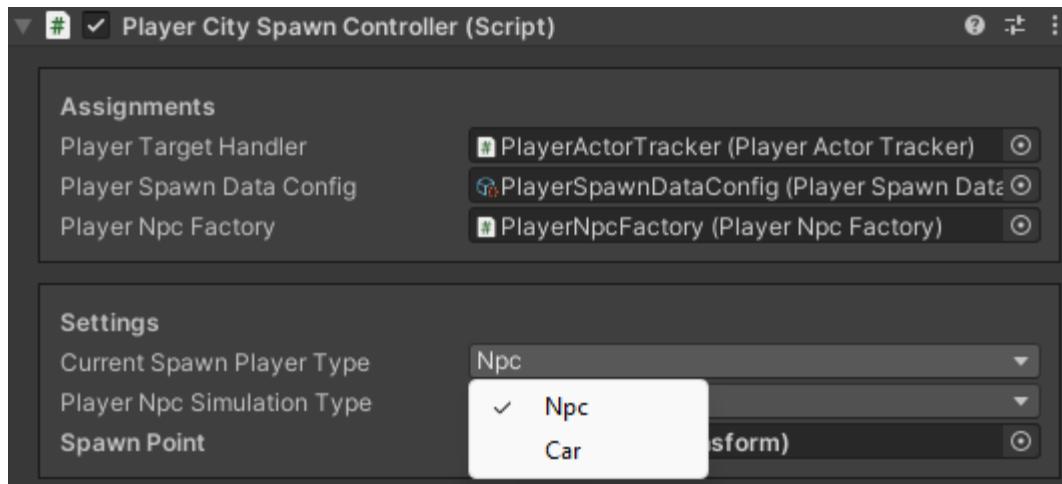


Scene hierarchy example.

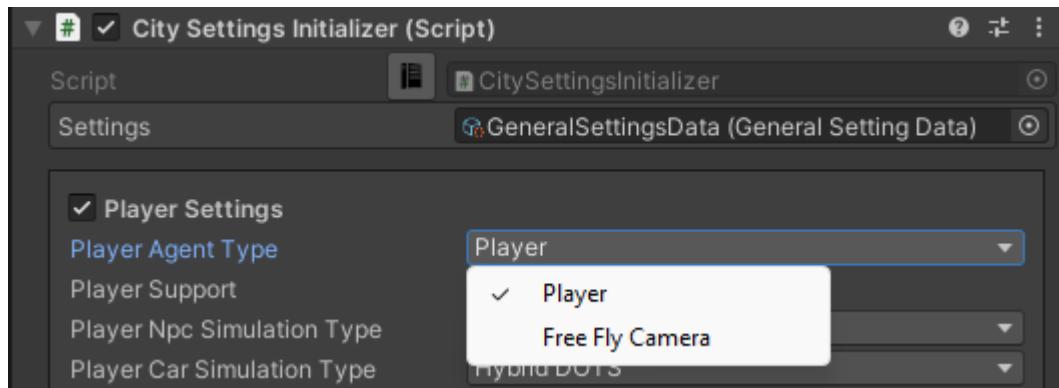


Component example.

5. In the *PlayerSpawner* select which type of player you want to spawn first, NPC or Car.



6. Select the *Player agent type* in the *General Settings* to spawn the *Player* or the *Free fly camera*.



7. If you want to add your own player, [read here](#), otherwise you can use the temporary built-in *Player NPC* & *Player car* [optional step].
8. Add [Road segments](#) to the scene.
9. In the *Road Parent* press *Force connect segments* button.
10. Create & connect [Pedestrian nodes](#) using the *Pedestrian Node Creator*
11. In the *Road Parent* press *Bake Path Data* button (should be done after each road edit & before starting the scene) & select *Hub* object on the scene & generate a *subscene*.
12. For more information on how to create a road, read the [road installation](#).
13. Create ground, if missing (*GlobalSurfaceCollider* example prefab) & set the layer for your ground surfaces to [Ground \(18\)](#) & layer for your static objects to [StaticPhysicsShape \(22\)](#) (read more about [PhysicsShapeTransfer](#) service, if you are going to use *DOTS* only).
14. Set desired local position of *Cull point* & Culling distances at which road objects, traffic, pedestrians etc. will be activated.

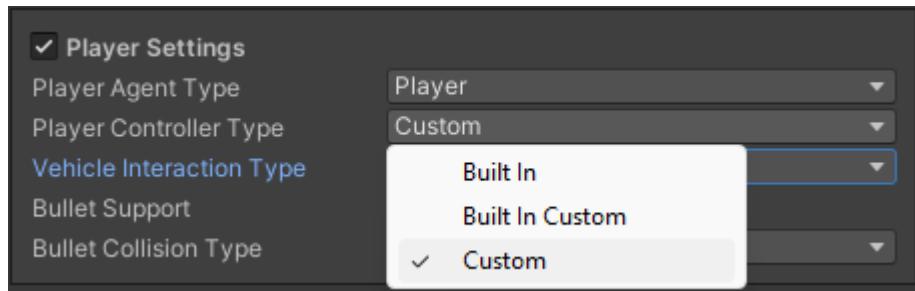
15. By default, the cull point is the child in the *Main Camera City*, but if you want to use your own *player & camera*: [optional step]



- Select *CitySettingsInitializer* on the scene:



- Set the *Player controller type* to *Custom* in the *General Settings* config.



- Disable the *Main Camera City*.



- Create a new gameobject, add a *CullPointRuntimeAuthoring* component & add this object by child to your camera (set local position to zero) or add it to the scene not too far from the roads, if the camera is not yet created.

16. Create *traffic vehicles*.

17. Create *pedestrians*.

18. Add & customize *game sounds* [optional step].

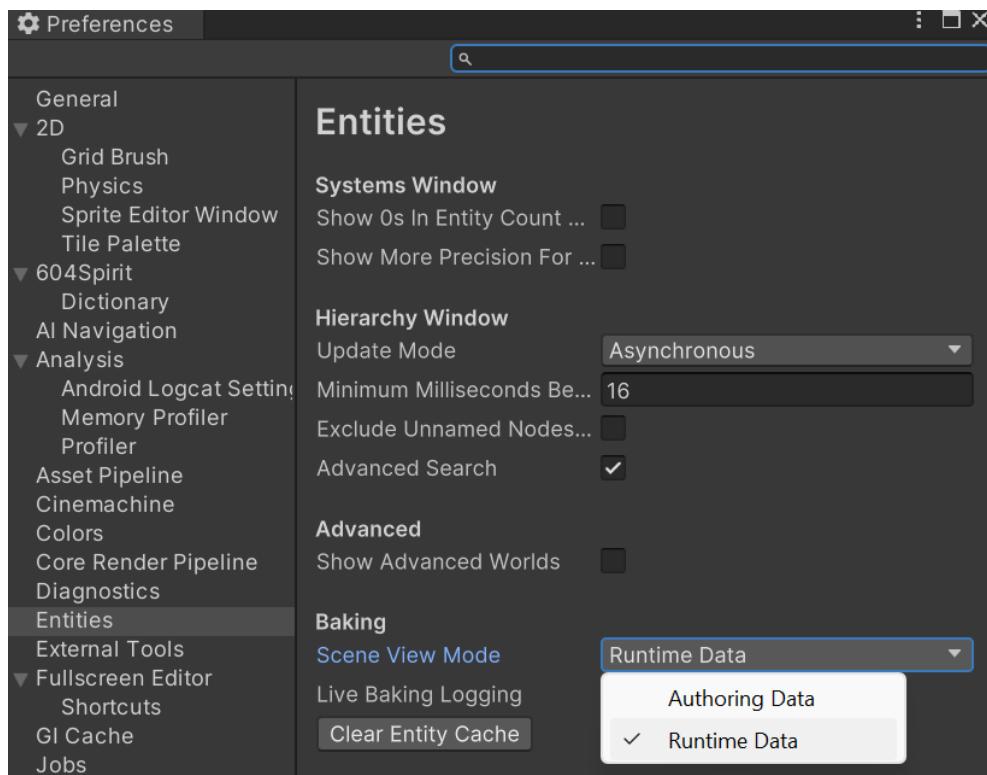
19. **By default, the `Unity.Entities` is not rendered on the *Scenerview*, to fix this follow these steps:**

1. In the *Unity editor* toolbar select:

Edit/Preferences

2. Select the *Entities* tab.

3. Set *Scene view mode* to *Runtime Data*.



20. Launch the scene.

1.4.2 Demo Scene

1. In the *Project Folder* view, select the following scene:
DotsCity/Scenes/Demo
2. Press *Play* button.
3. Read more about *Project Scenes & Scene Structure*.

1.4.3 Demo Mono Scene

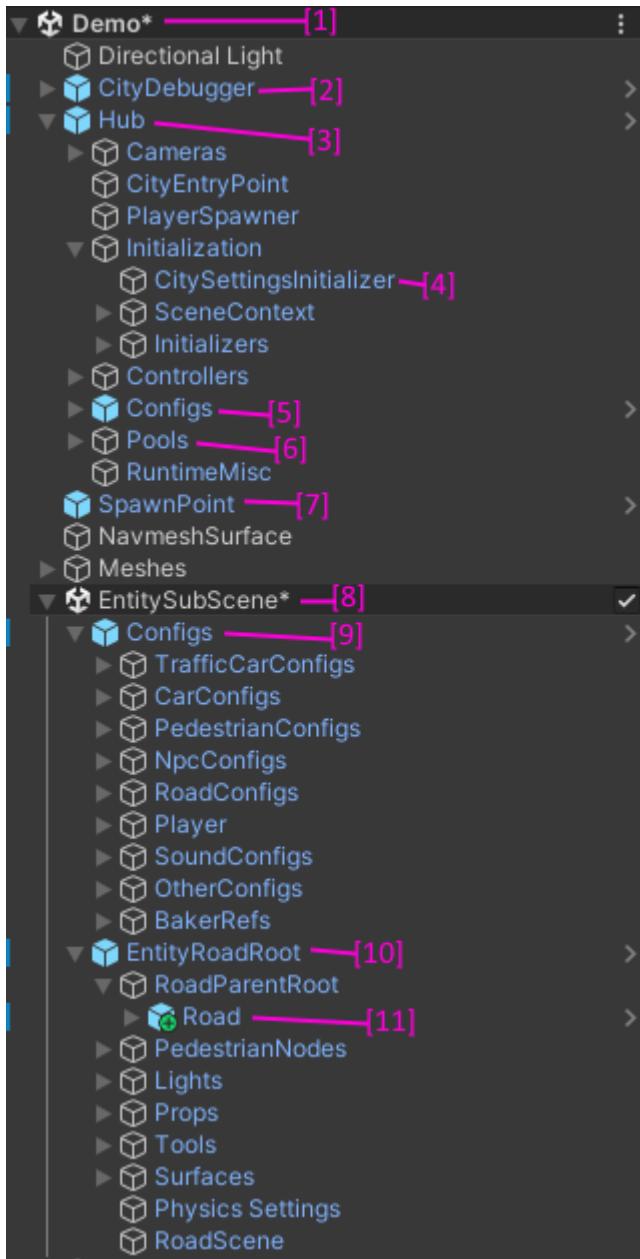
1. In the *Project Folder* view, select the following scene:
DotsCity/Scenes/Demo Mono
2. Press *Play* button.
3. Read more about *Project Scenes & Scene Structure*.

1.5 Structure

1.5.1 Project Scenes

1. **Demo** : the main optimized demo scene of the project (read more about scene structure [here](#)).
2. **Demo Mono** : the new demo scene of the project, which contains an example of traffic & NPCs fully interacting with monobehaviour scripts (v1.1.0+).
3. **Demo source** : same as *Demo* scene, but contains raw 3D models without optimization, also scene *streaming objects* are disabled, including *static physics colliders*.
4. **RuntimeTile Road Demo** : sample scene showing how to create a city builder based on tiles (runtime spline coming soon). (**new**)
5. **Runtime ChunkRoad Demo** : sample scene demonstrating the road chunks added at runtime. (**new**)
6. **Custom Train Demo** : an example of how a 3rd party train solution can be integrated. (**new**)
7. **Traffic test scene** : *traffic test scene* where all parameters can be tested with a set of most *traffic* situations.
8. **Pedestrian test scene** : *pedestrian test scene* where the workability of entities can be tested.
9. **Props test scene** : *props test scene* where the workability of entities can be tested.
10. **City stress scene** : stress scene that combines thousands of *vehicles* and *pedestrians*.
11. **City stress scene mobile** : mobile version of *City stress scene*.
12. **Traffic stress scene** : stressed scene is extremely crowded with *vehicles*.
13. **Traffic stress scene mobile** : mobile version of *Traffic stress scene*.
14. **Pedestrian stress scene** : stressed scene is extremely crowded with *pedestrians*.
15. **Pedestrian stress scene mobile** : mobile version of *Pedestrian stress scene*.
16. **Pedestrian animation stress scene** : performance scene comparison between *Unity Animator* animations and *GPU animations*.
17. **Vehicle physics stress scene** : stress scene for the thousands cars with *Custom vehicle controller*.
18. **Vehicle custom physics test scene** : scene for testing various parameters of the *Custom vehicle controller*.

1.5.2 Scene Hierarchy



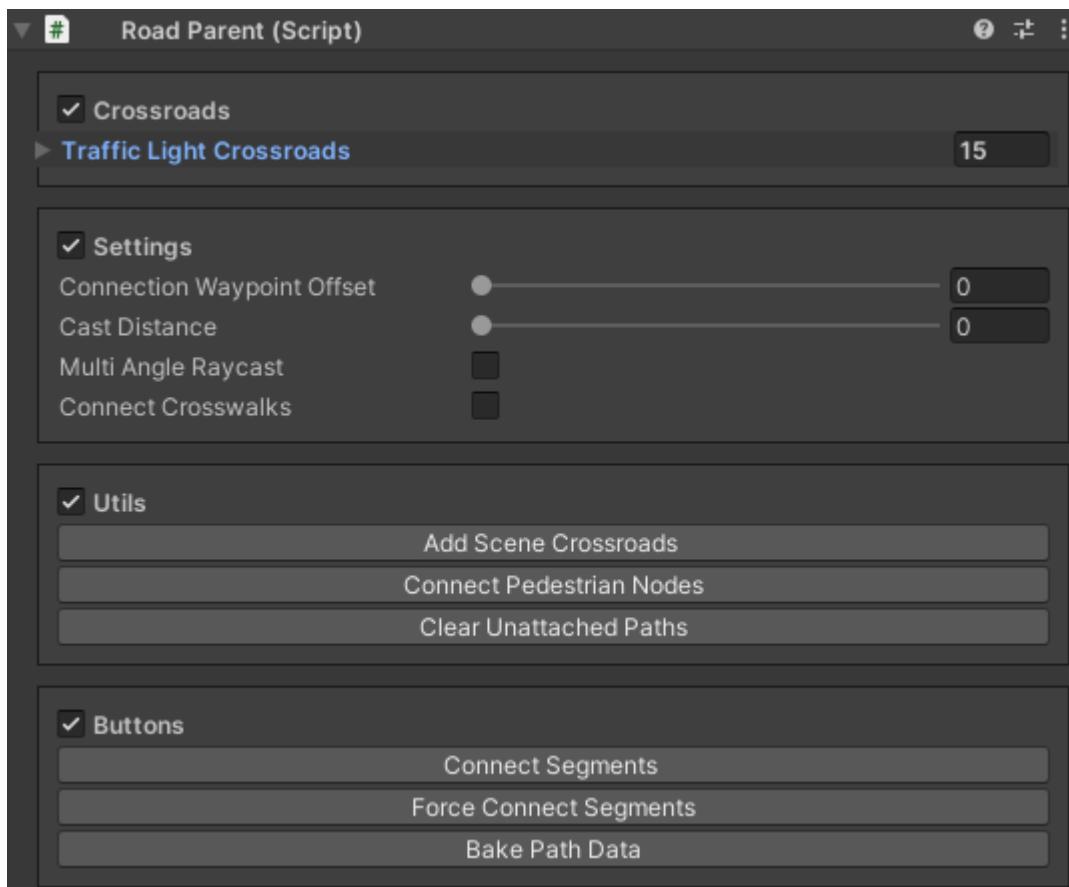
1. **Main scene.**
2. **City debugger** : contains all the *debuggers* for the city.
3. **Hub** : contains the *Entity Subscene Generator* (*read more* on how to create it).
4. **City Settings Initializer** : contains the *General Settings* of the DOTS city [moved to the *Config* tab].
5. **Main scene configs** (*read more* about the *config editing* workflow).
6. **Pools** : contains all the entity presets.
7. **Spawnpoint** : spawnpoint of the player (assigned in the *PlayerSpawner*).

8. **Subscene** (*EntitySubScene*) : subscene into which all of the entities are converted (generated by the *Entity Subscene Generator*).
9. **Subscene configs** (read more about the *config editing* workflow).
10. **EntityRoadRoot** : object root generated by the *Entity Subscene Generator*.
11. **Road** : *road parent* (read more about the *road editing* workflow).

1.5.3 Scene Components

Road Parent

The root of all crossroads in the scene.



How To Use

You read more [here](#).

Settings

Connection waypoint offset : automatically adds a *waypoint* at each selected offset to the *automatically* created *paths* (if the value is greater than zero).

Cast distance : raycast connection distance between *Traffic nodes*, if zero, then infinite value.

Multi angle raycast : multi-angle raycasting along Z-axis.

Connect crosswalks : auto-connect *Pedestrian node* crosswalks

Utils

Add crossroads : adds scene crossroad if missing.

Connect pedestrian nodes : automatically connects *pedestrian nodes* that configured *auto-connection*.

Clear unattached paths : delete the *paths* that are not connected to any a *TrafficNode*.

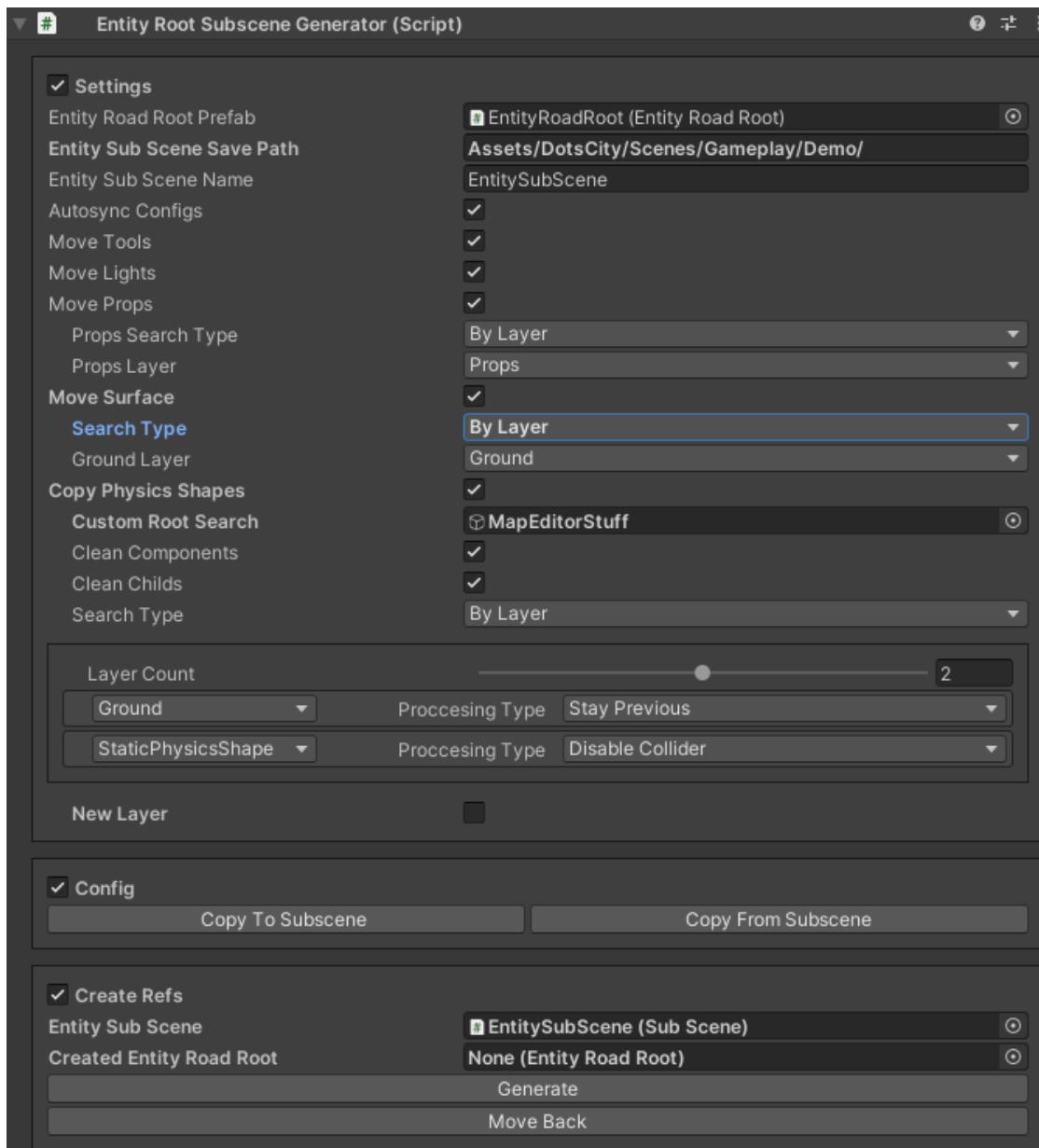
Buttons

Connect segments : creates the *automatically generated paths* for missing paths of *external* nodes.

Force connect segments : creates the *automatically generated paths* for all paths of *external* nodes (if a path was previously created & autopath lock is off for the *Traffic node*, then the path will be overridden).

Bake path data : *bake* road data.

Entity Subscene Generator



Where To Find

1. Create *city base* [if missing, optional step].
2. Select *Hub* in the scene.

How To Use

You read more [here](#).

Settings

Entity subscene save path : save path of *subscene*.

Entity subscene name : name of *subscene*.

Autosync configs : on/off auto-sync config on *main scene & subscene*.

Move tools : on/off moving of following tools: *PedestrianNode Creator*, RoadSegmentPlacer.

Move lights : on/off moving of *subscene*.

Move props : on/off moving of *props*.

Move surface : on/off moving of the selected physics surface.

Copy physics shapes : on/off feature of physics shape *cloning*.

Config

Copy to subscene : *subscene* configs will be synchronised with the *main scene*.

Copy from subscene : *main scene* configs will be synchronised with the *subscene*.

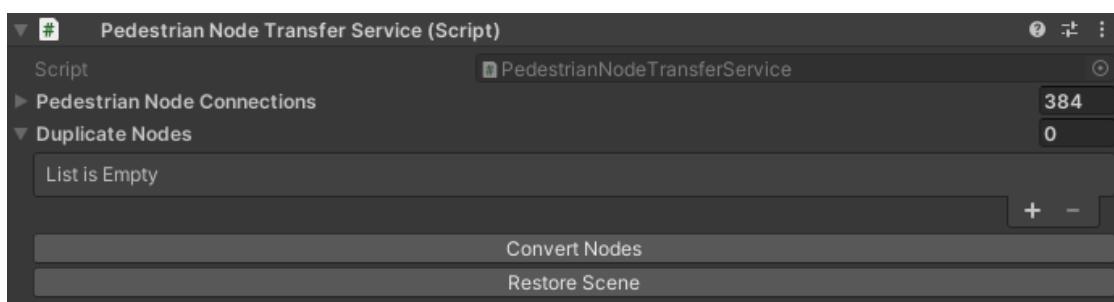
Buttons

Generate : generates *subscene*.

Move back : move road from *subscene* to the *main scene* (can be useful for editing roads in the *main scene* due to *Editor* performance).

PedestrianNode Transfer Service

Tool for cloning *Pedestrian nodes* that are part of the prefab from the *main scene* to the *subscene* .



Note: For example, can be useful to separate the *Building prefab* asset and its attached *Pedestrian nodes*.

How To Use

Automatically used by *Entity Subscene Generator*.

PhysicsShape Transfer Service

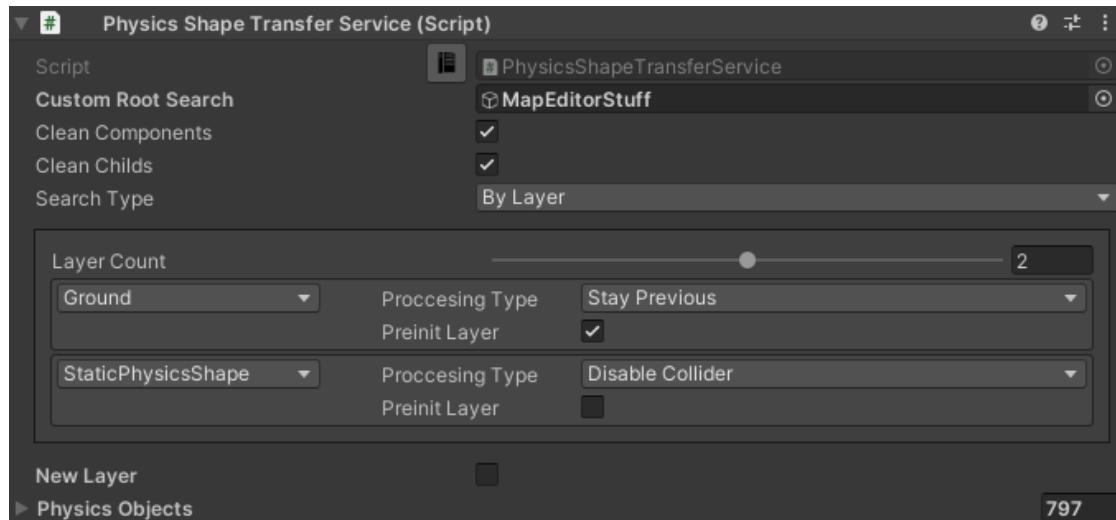
- This tool only works for *DOTS* if the *DOTS* simulation type is selected in the *General Settings*.
- Tool for cloning physical shapes from the *main scene* to the *subscene*.
- There is also a tool exists to maintain both the *default physical world* and the *DOTS* physical world at the same time, so that *default colliders* and *DOTS* colliders exist at the same time if you need to use *default colliders* in *MonoBehaviour* classes.

Current project use cases:

- Keep *default colliders* to work with *Legacy ragdoll*.
- Cloning of physical shapes from the *main scene* to be split them into subscene pieces by using *SubSceneChunk Creator* or cloned into the *main subscene* by using *Entity Subscene Generator*.

Note: The tool can only use one tool at a time, either an *Entity Subscene Generator* or a *SubSceneChunk Creator*.

Settings



Clean components : removes all components from the physics shape.

Clean childs : removes all childs from the physics shape.

Search type : searching shapes by layer or tag.

Proccesing type :

- **Stay previous** : cloning found PhysicsShape or Collider to *subscene*, useful to keep default collider and Unity.Physics collider running at the same time.
- **Disable collider** : cloning found PhysicsShape or Collider to *subscene* and disable it in the main scene.

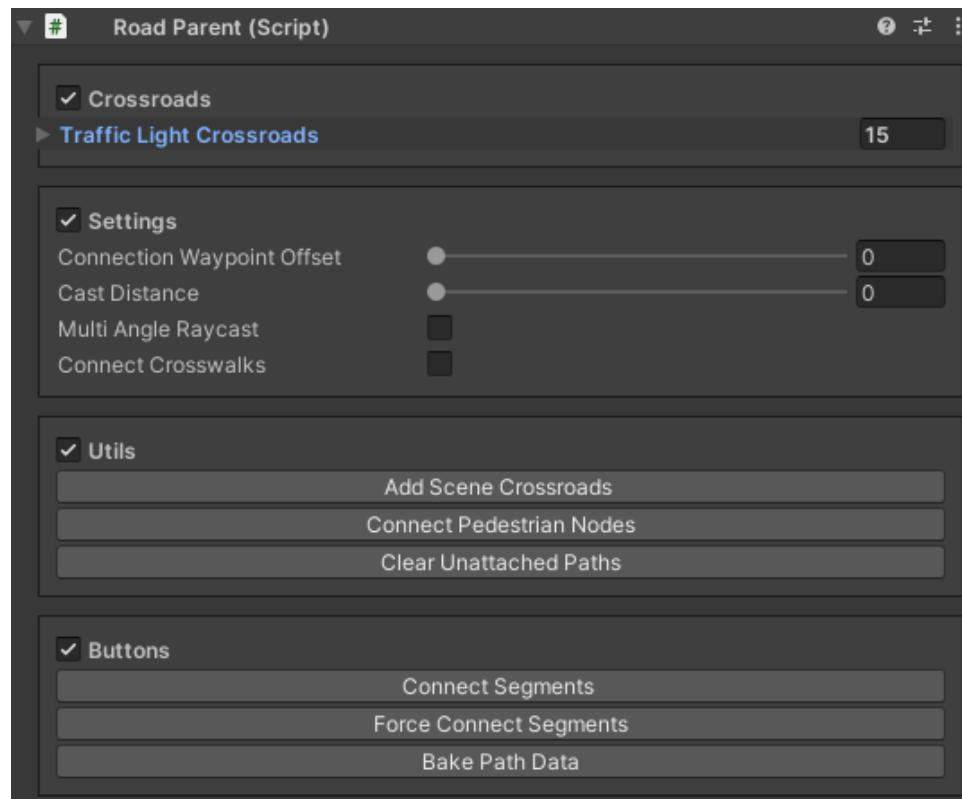
Preinit layer : enable *pre-init* cull state for physics objects.

New layer : assigns new layer for cloned shape.

2.1 Installation

2.1.1 Traffic Road Installation

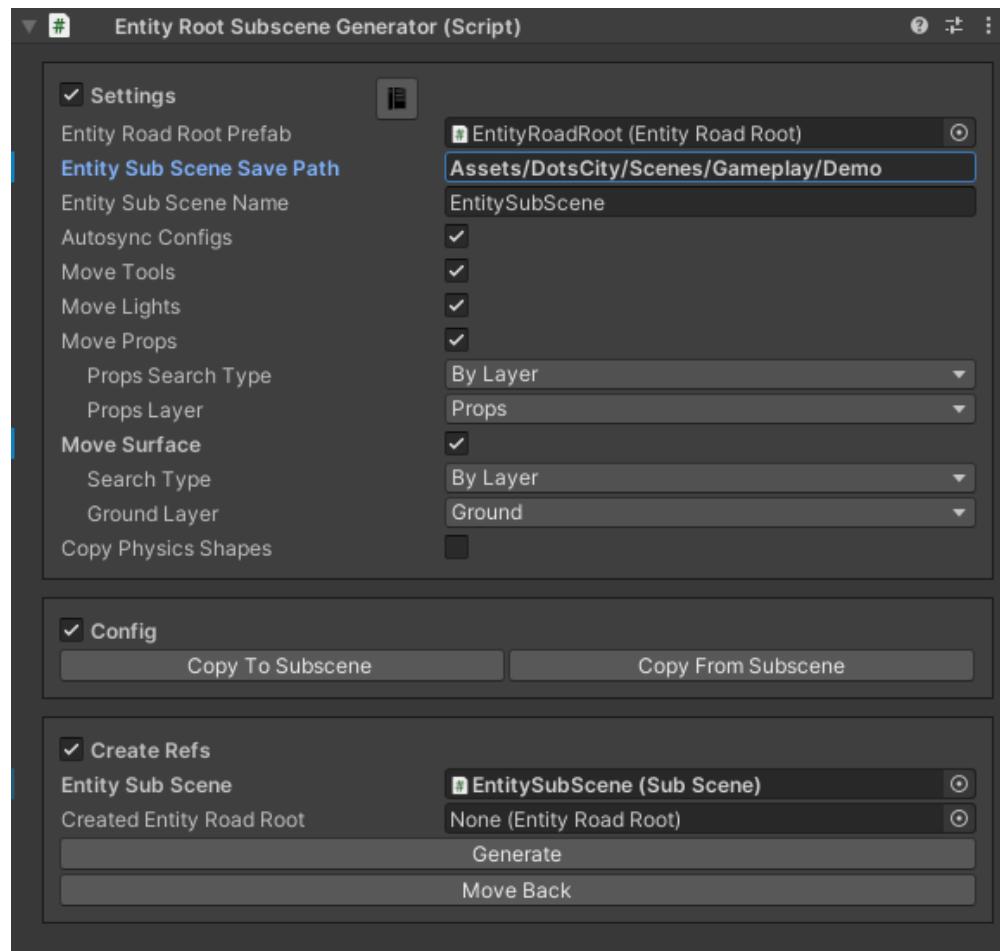
1. Create initial *scene* objects.
2. Create the required *RoadSegments*. And add them as children to the *RoadParent*.
3. Create the *necessary paths*.
4. Configure the *paths*.
5. Create the necessary *PedestrianNode* using the *PedestrianNodeCreator*.
6. Adjust the *Traffic lights*.
7. Create the *Traffic Areas*, for example if you have a congested car park **[optional step]**.
8. Create the *Public Routes* if you have *Public transport* with a given route **[optional step]**.
9. Open the *Road Parent*.



10. Press *Force connect* button to connect segments (make sure all segments are *on one line*).
11. Press the *Bake* button (should be done after each road edit & before starting the scene, for more info the *bake info*).
12. Create the *subscene* (**one-time procedure**).
13. For further changes to roads and configs, read the *Road* editing workflow & to sync both scenes, read the *Config* editing workflow.

2.1.2 Entity Subscene Creation

From DOTS 1.0 onwards, all entity conversions must be done using subscenes. It's necessary to create a separate *subscene* for roads.



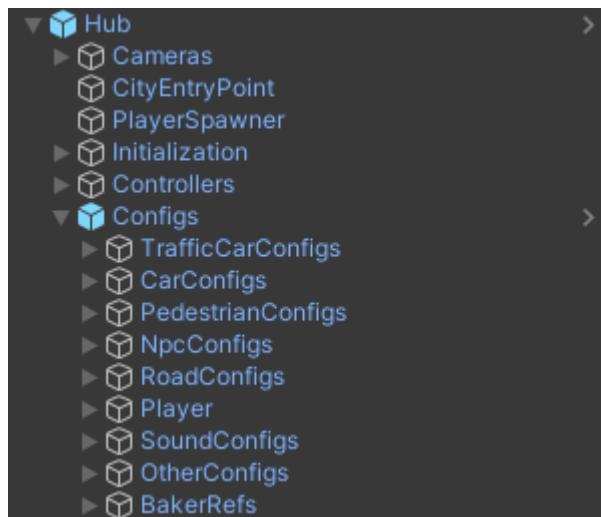
Steps:

1. Select *Hub* in the scene.
2. Select *Entity subscene path* the path to create a *subscene*.
3. Enter the *Entity subscene name* or use the default name.
4. On/off autosync configs (before migrating the configs to the *subscene*, they will be synchronized with the configs that are in the *Hub*).
5. On/off copy physics shapes feature (read more about *physics shape transferring*) [required if you plan to use DOTS physics].
6. Press the *Generate* button.
7. All created *RoadSegments* and *PedestrianNodes* will automatically be moved to the *subscene*.

2.1.3 Config Editing Workflow

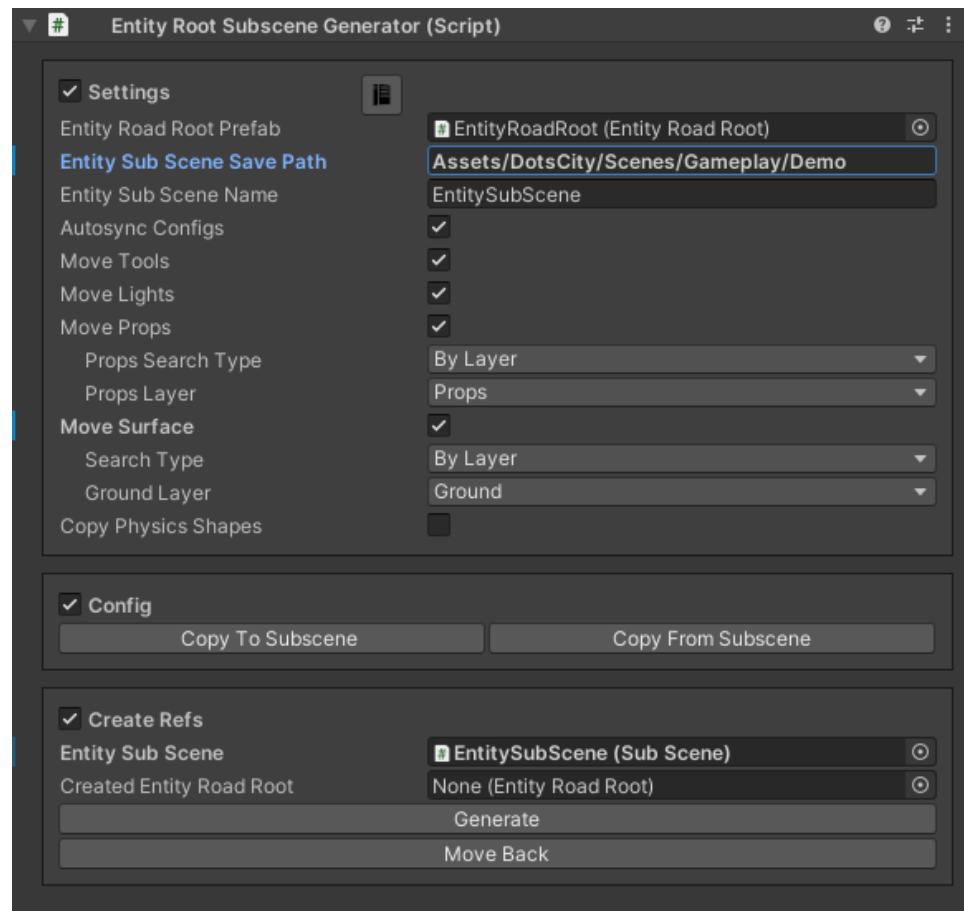
There are 2 variants to edit configs:

Main Scene Editing

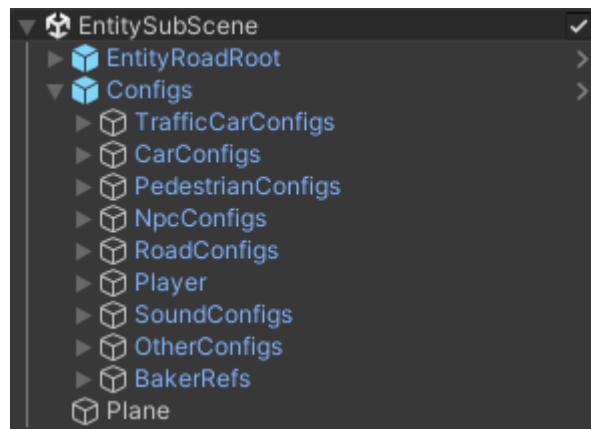


Steps

1. Select *Hub* in the scene.
2. After editing any config in the main scene *Hub* press the *Copy To Subscene* button or if the config is a non-scriptable object, apply the prefab to the selected config row.



Directional Editing



Steps

1. Open the *EntitySubScene* *subscene*.
2. Edit any config.
3. After editing any config in the subscene, in the *Hub* press the *Copy From Subscene* button or if the config is a non-scriptable object, apply the prefab to the selected config row in the subscene.
4. Save & close *subscene*.

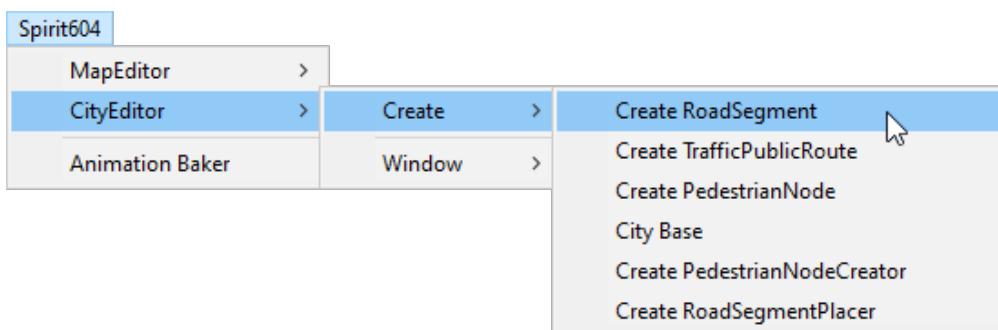
2.2 Road Segment

Road Segment is the main component of the road that contains *traffic nodes*, *paths*, and *pedestrian nodes*.

Youtube tutorial.

2.2.1 How To Create

Select from the *Unity* toolbar:



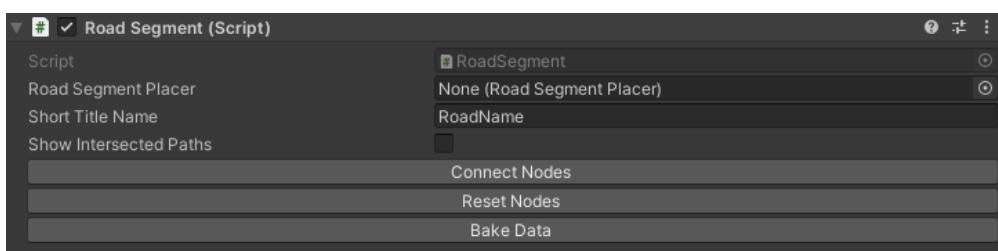
2.2.2 How To Customize

By default, *RoadSegment* contains *RoadSegmentCreator* component that can be used to customize a segment.

2.2.3 Main Components

Road Segment

Component for connection to other road segments.



Variables

Road segment placer : reference to the RoadSegmentPlacer.

Short title name : short name for RoadSegmentPlacer.

Show intersected paths : on/off *intersection points* in the scene.

Buttons

Connect nodes : *auto-connect Traffic Nodes*.

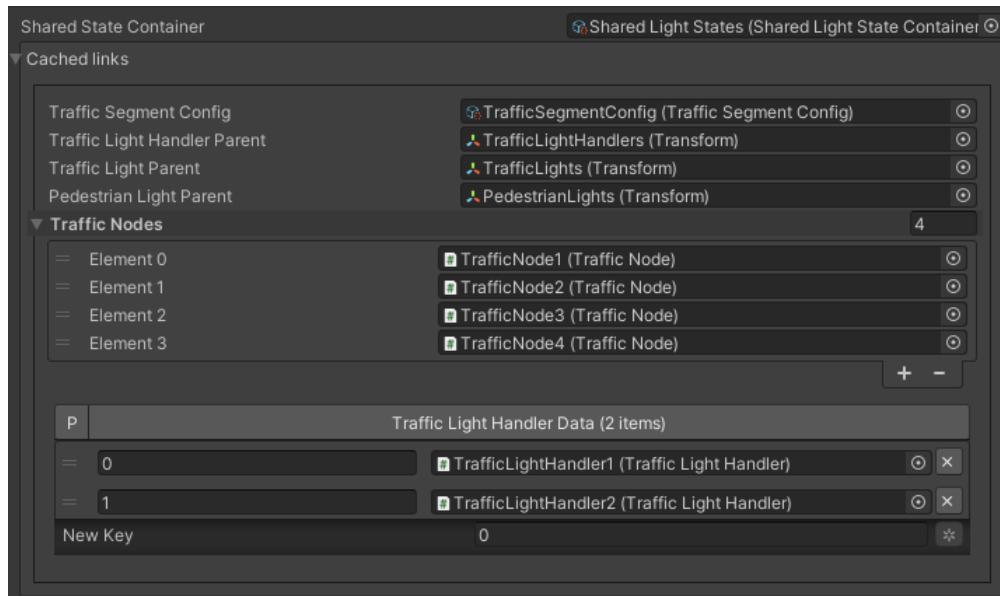
Reset nodes : reset already *auto-connected* paths of *Traffic Nodes* (except *locked* nodes).

Bake data : *bake* all *paths* data (path's length, intersections).

TrafficLightCrossroad

Component for handling traffic lights at crossroad. For a quick look at timelines of city crossroads and traffic light connections, *see here*.

Cached



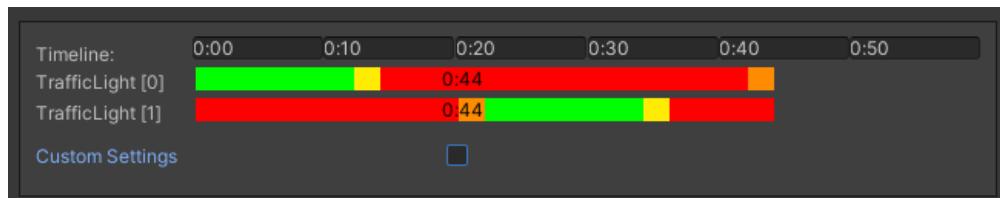
Shared state container : *shared light state container*, that contain common traffic light timings that are shared with other light crossroads.

Traffic nodes : all *Traffic Nodes* of *RoadSegment*.

Traffic light handler data : light index and light handlers that are linked to the *TrafficLightCrossroad*.

Timeline common

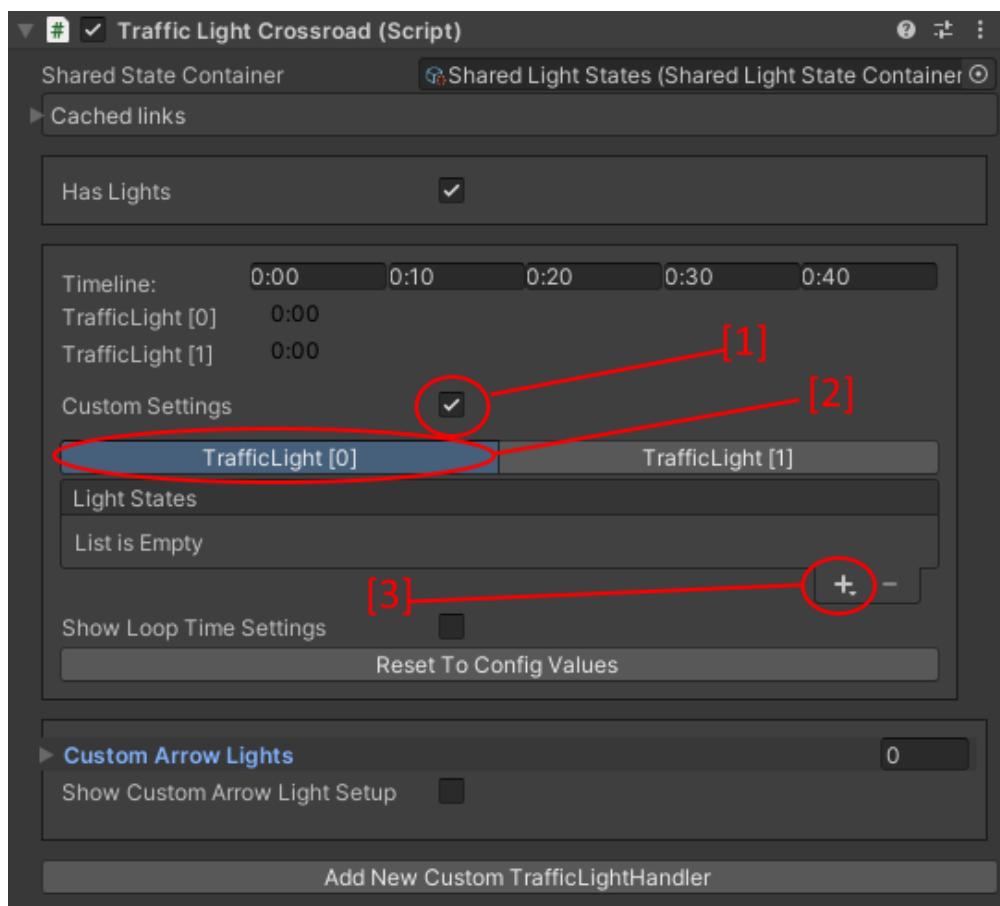
Timeline common uses the timeline from the *Shared state container*.



Note: You can easily replace the *shared state container* for all crossroads using the *Global Light Settings* tool.

Timeline custom

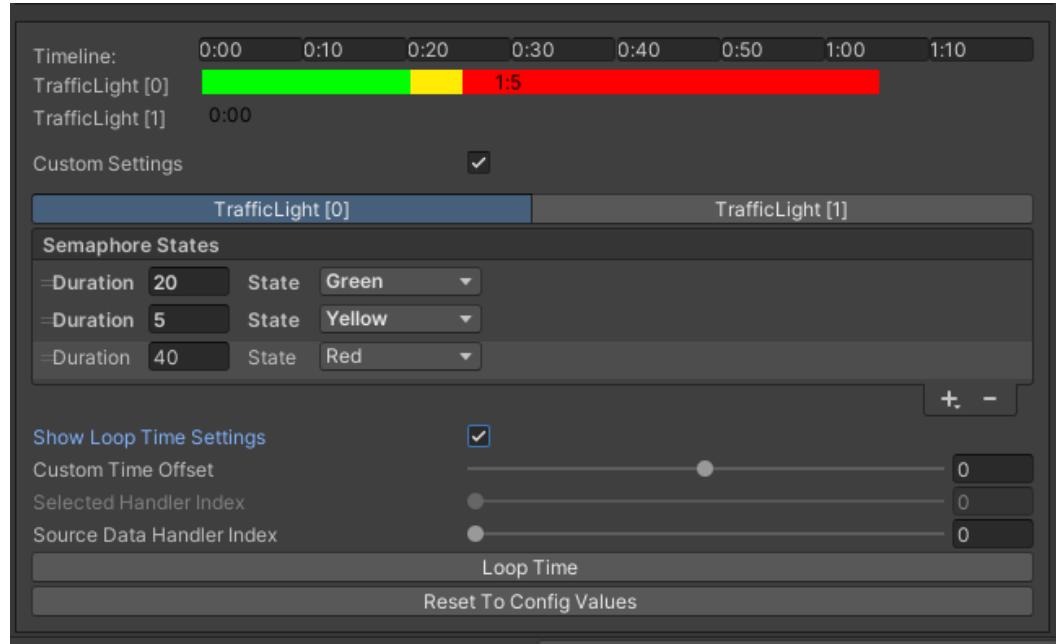
Custom timeline is designed for custom timings of the traffic light segment



How to add states:

1. Enable *custom settings*.
2. Select the desired *TrafficLightHandler*.

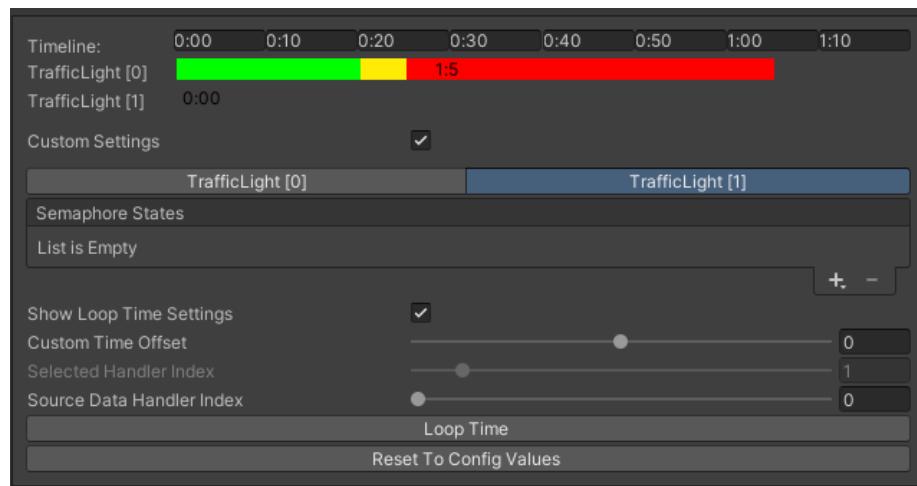
3. Press the + button.
4. Add desired *states*.
5. Enter the duration of the *state*.



Once you have set up 1 *TrafficLightHandler*, you can loop to the 2nd *TrafficLightHandler*.

How to loop timeline:

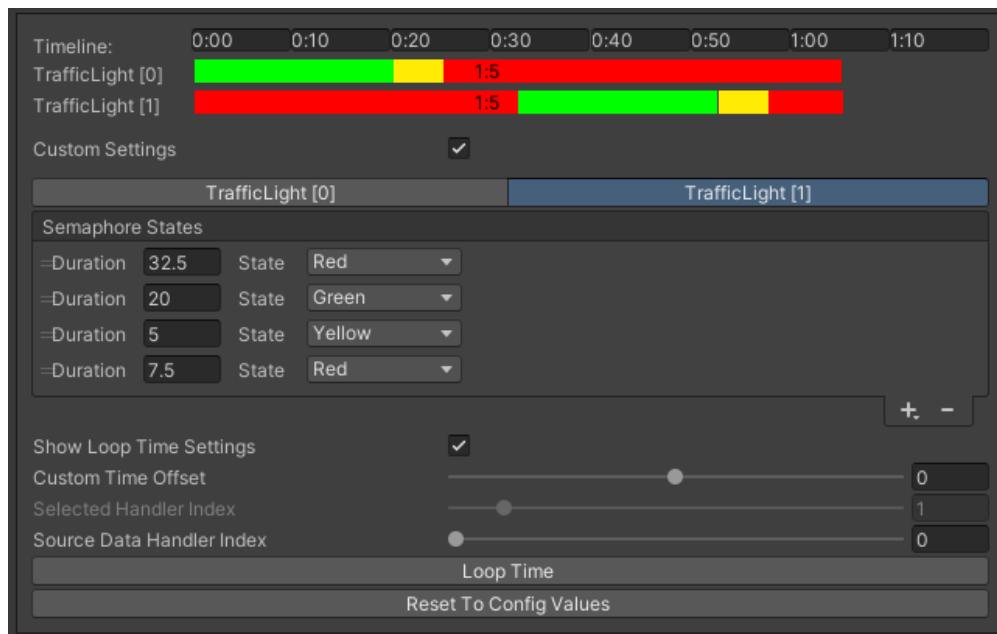
1. Select the *TrafficLightHandler* to be looped.
2. Enter the *Source Data Handler Index* parameter based on which to loop.



Settings example.

3. Click the *Loop Time* button.

Loop result:

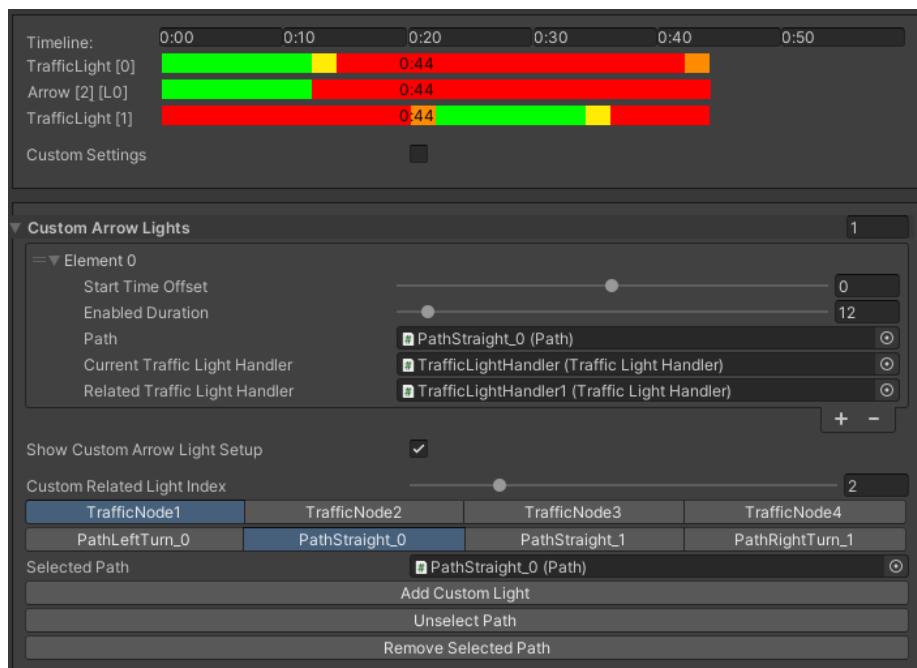


Custom arrow lights

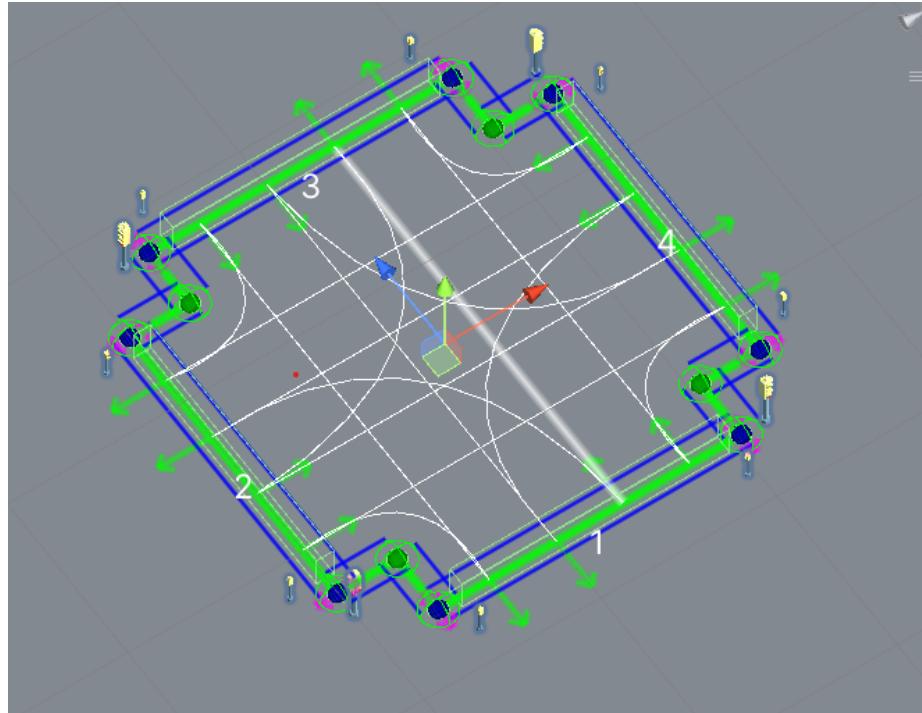
Arrows are used for the custom traffic light for the selected *path*.

How to create arrows:

1. Click *Show Custom Arrow Light Setup*.
2. Select *Custom Related Light Index*.
3. Select related *TrafficNode* from the toolbar.



4. Select related *path* from the toolbar.



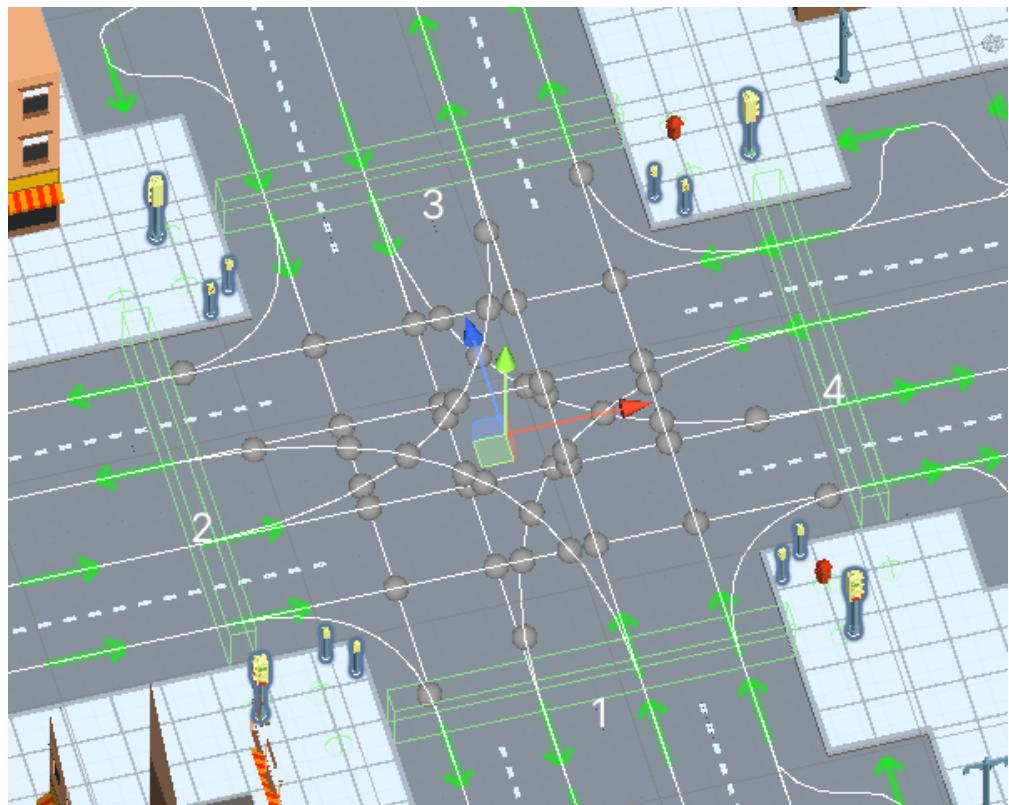
Selected path example.

5. Click the *Add Custom Light* button.

Note: To remove the light arrow, select the appropriate *TrafficNode* and *Path* and press the *Remove Selected Path* button.

2.2.4 Baking info

The intersection of *paths* is only baked in those *paths* that are in the segment. How to *bake*.



Intersection points example.

2.3 Road Segment Creator

Road Segment Creator is a tool for creating and customizing a *RoadSegment*

- *How To Use*
- *Standard Shapes*
 - *Default Crossroad*
 - *Turn Road*
 - *Straight Road*
 - *Merge Crossroad*
 - *Merge Straight Road*
 - *Merge Crossroad To Oneway Road*
 - *Oneway Straight*
 - *Oneway Turn*
- *Custom Straight Road*
 - *How To Use*
 - *Settings*

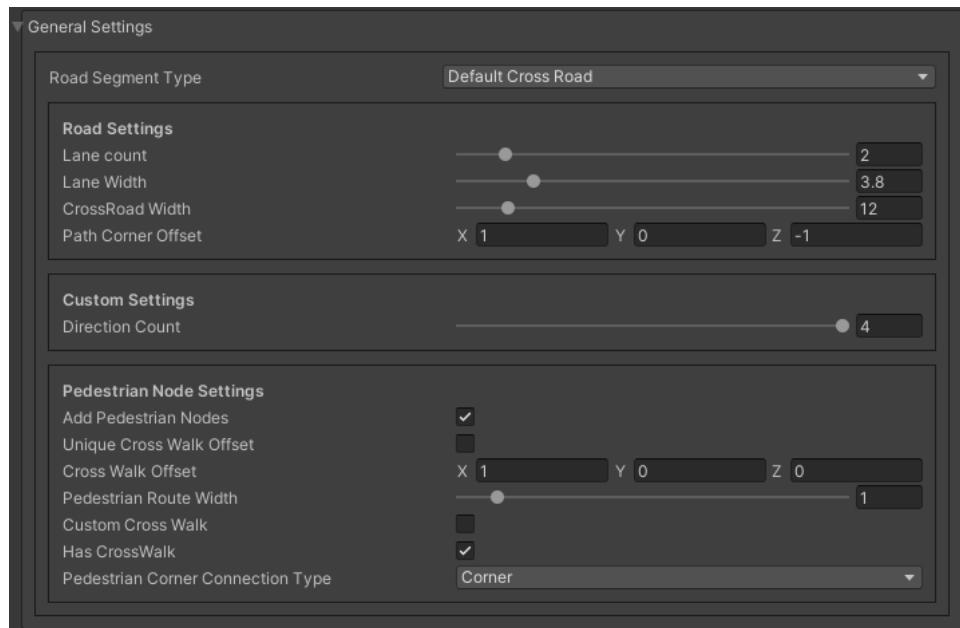
- * *Custom Settings*
- * *Snap Node Settings*
- * *Snap Surface Settings*
- * *Snap Line Settings*
- * *Path Settings*
- * *Examples*
- *Custom Segment*
 - *How To Use*
 - *Settings*
 - * *New Node Settings*
 - * *Custom Path Settings*
 - *Additional Settings*
 - * *Extrude Lane*
 - * *Parking Builder*
 - * *Custom Settings*
 - * *Snap Node Settings*
 - * *Custom TrafficNode Editor Window*
 - * *Examples*
 - *Settings Description*
 - * *Snap Node Settings*
- *Components*
 - *General settings*
 - *Custom settings*
 - *Pedestrian node settings*
 - *Light settings*
 - * *How To Use*
 - * *Traffic lights*
 - * *Pedestrian lights*
 - *Path settings*
 - * *Node selection panel*
 - * *Road settings*
 - * *Scene settings*
 - * *Turn connection settings*
 - *Segment handler settings*
 - *Other settings*

- * Buttons
- Hotkeys
- Parking Builder
 - How To Use
 - Settings
 - * Common
 - * Path
 - * Node
 - * Pedestrian
- Auto Crossroad
 - How To Use

2.3.1 How To Use

Youtube tutorial.

1. Create a *RoadSegment*.
2. Place the segment at the desired position.
3. By default, *RoadSegment* prefab contains *RoadSegmentCreator* component.
4. Select the *Road segment type* depending on the *shape* of your intersection.



5. In the other tab, you can convert any *Standard shape* segment to a *Custom segment* for more flexibility (also check out the new *Auto-Crossroads* feature for automatic crossroad generation).
6. Adjust *general settings*.
7. Adjust *custom settings*.

8. Customize *pedestrian node settings*.
9. Customize *light settings*.
10. Customize *path settings*.
11. Add *RoadSegment* to the *RoadParent* as children.

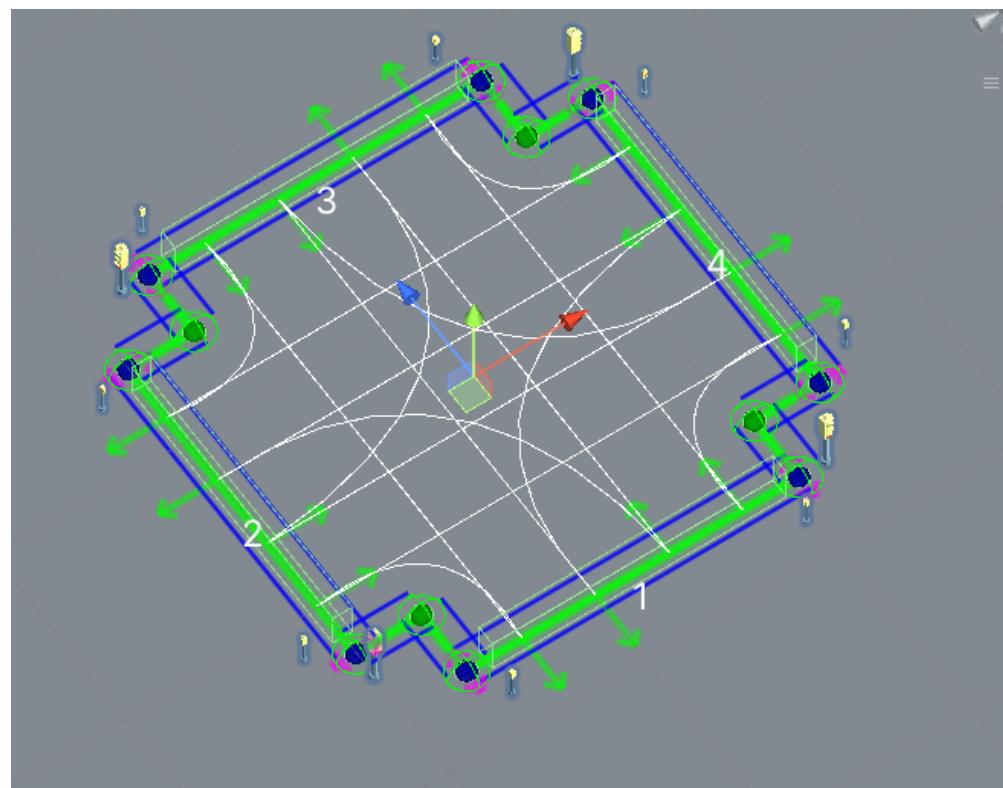
2.3.2 Standard Shapes

Youtube tutorial.

Default Crossroad

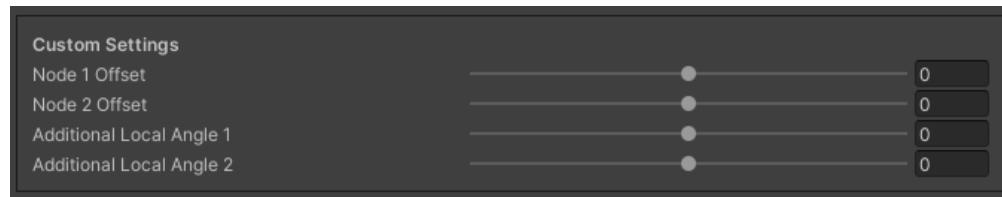


Direction count : number of sides of the crossroad.



Example.

Turn Road

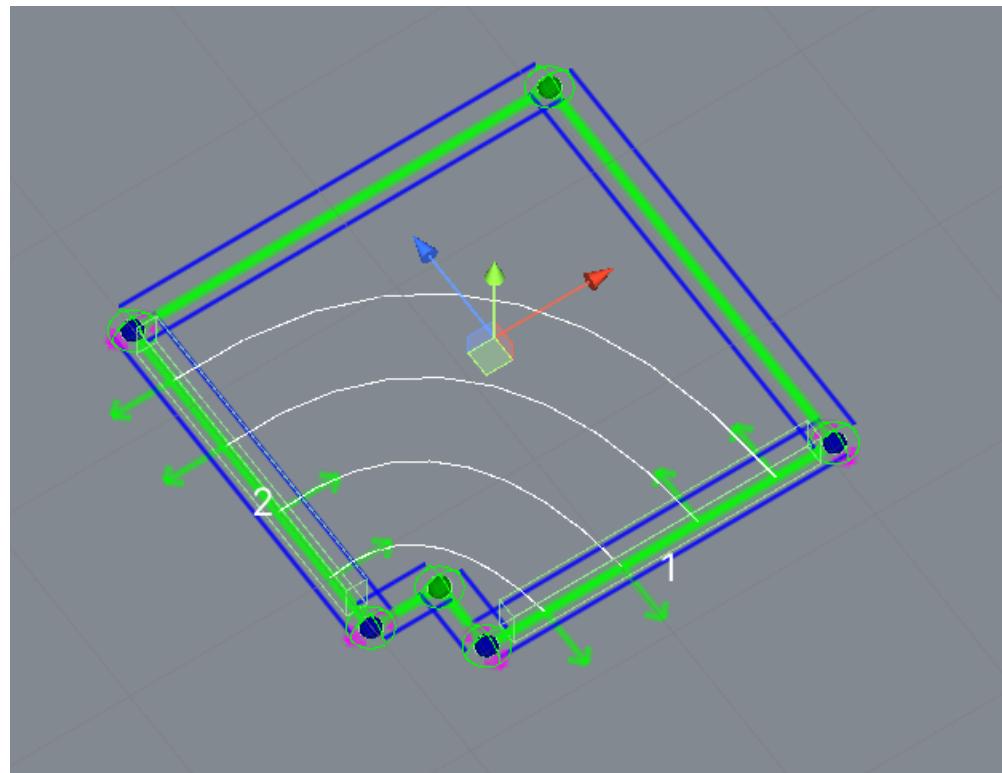


Node 1 offset : node 1 offset on the X-axis.

Node 2 offset : node 2 offset on the X-axis.

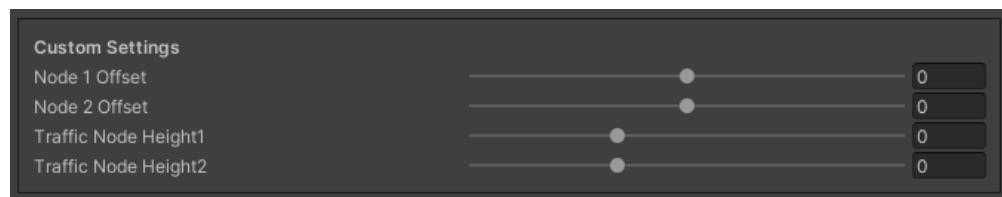
Additional local angle 1 : additional node 1 rotation angle.

Additional local angle 2 : additional node 2 rotation angle.



Example.

Straight Road

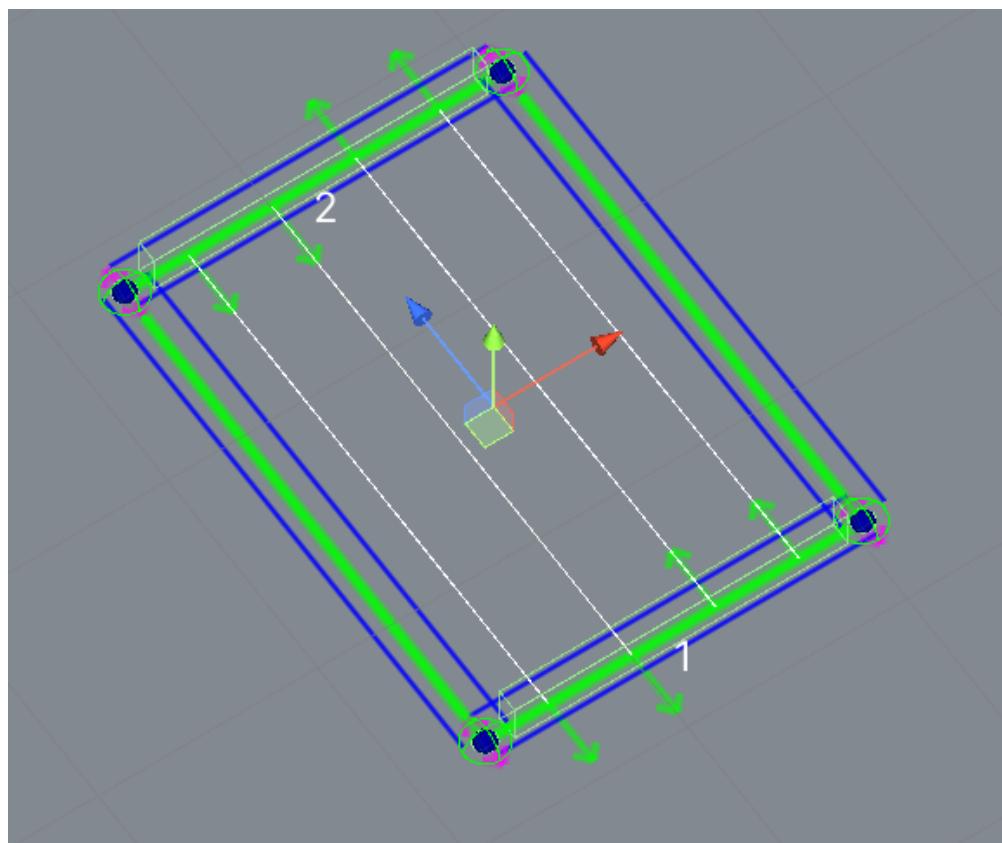


Node 1 offset : node 1 offset on the X-axis.

Node 2 offset : node 2 offset on the X-axis.

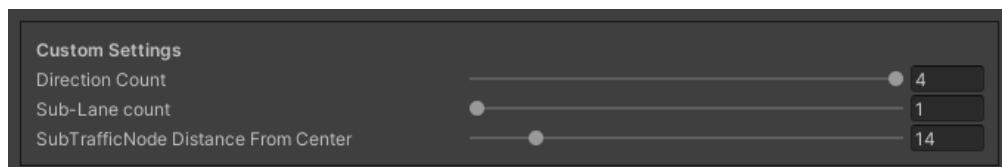
Traffic node height 1 : node 1 offset on the Y-axis.

Traffic node height 2 : node 2 offset on the Y-axis.



Example.

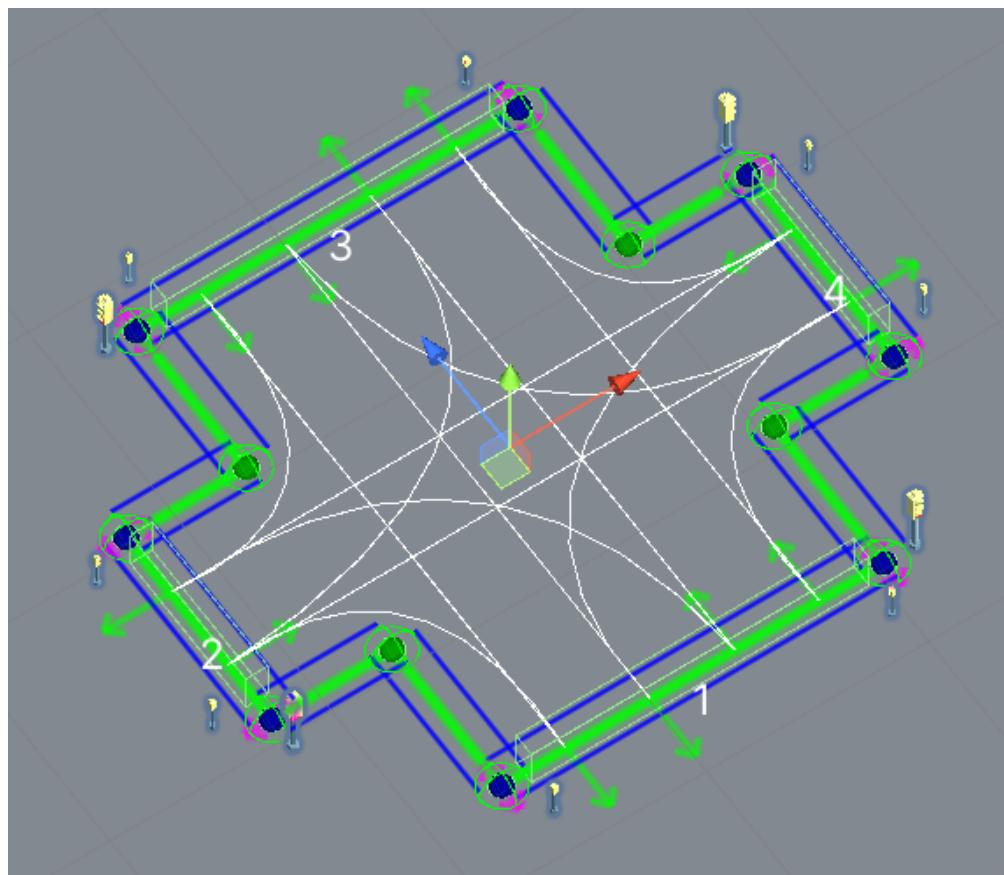
Merge Crossroad



Direction count : number of sides of the crossroad.

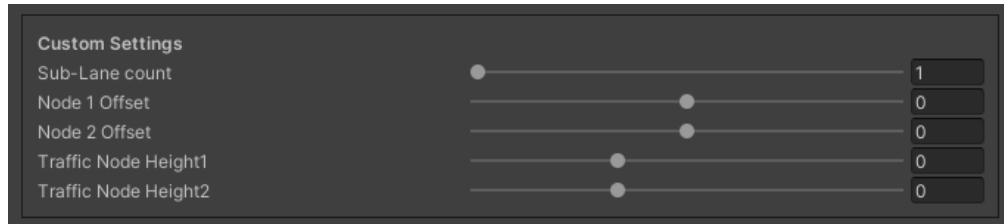
Sub-lane count : number of sub-lanes (a sub-lane is a lane with a different number of lanes than the main lane).

SubTrafficNode distance from center : distance between the *SubTrafficNode* (node that contains a sub-lane) and the center of the segment.



Example.

Merge Straight Road



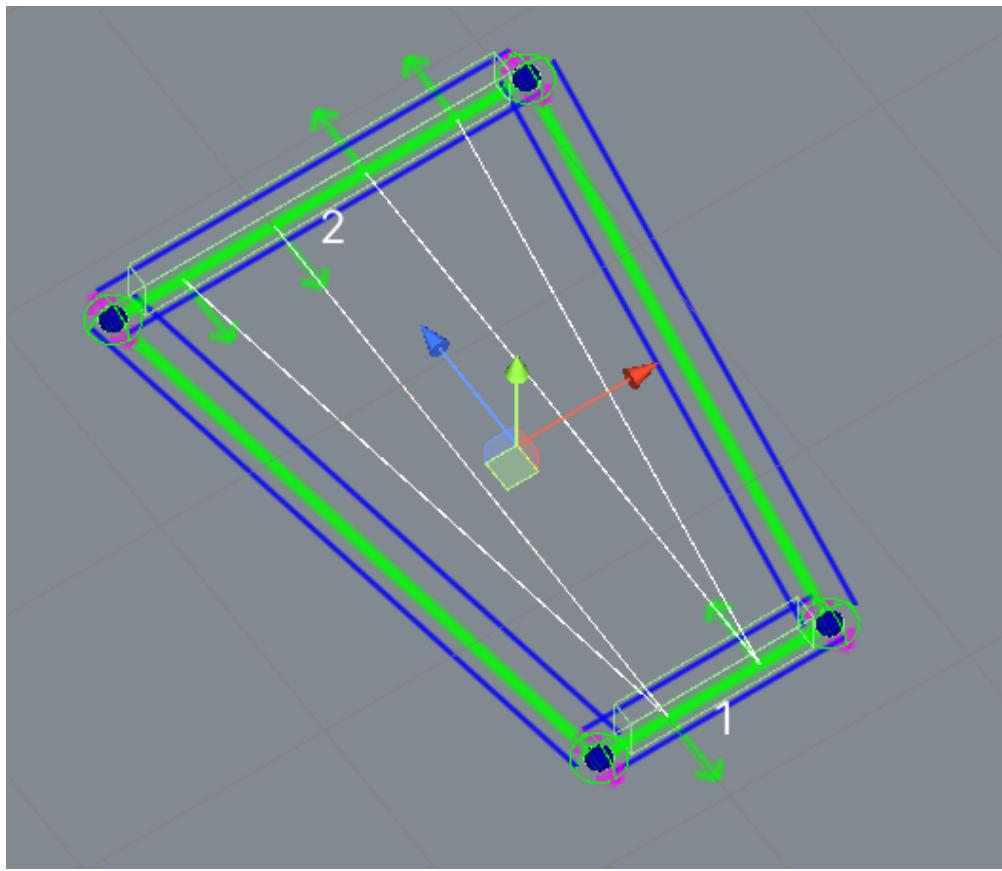
Sub-lane count : number of sub-lanes (a sub-lane is a lane with a different number of lanes than the main lane).

Node 1 offset : node 1 offset on the X-axis.

Node 2 offset : node 2 offset on the X-axis.

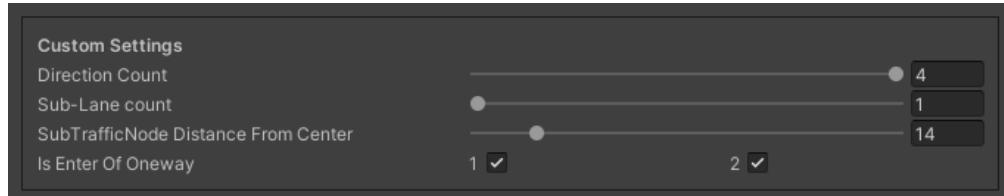
Traffic node height 1 : node 1 offset on the Y-axis.

Traffic node height 2 : node 2 offset on the Y-axis.



Example.

Merge Crossroad To Oneway Road

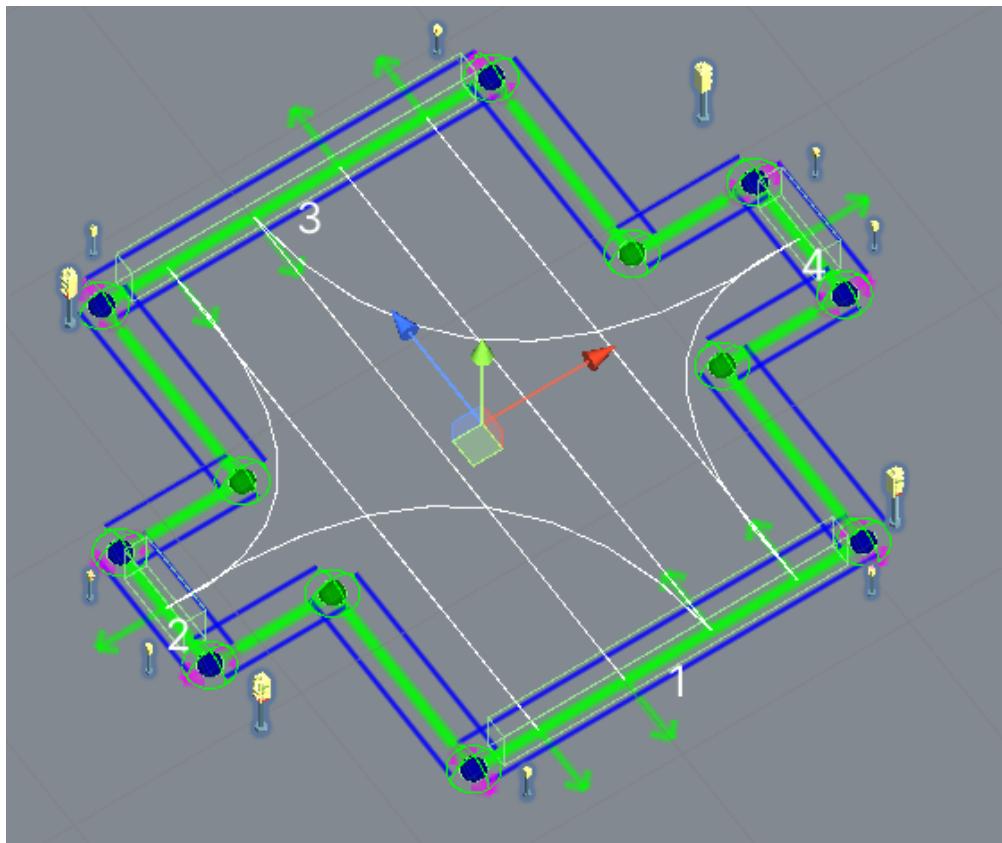


Direction count : number of sides of the crossroad.

Sub-lane count : number of sub-lanes (a sub-lane is a lane with a different number of lanes than the main lane).

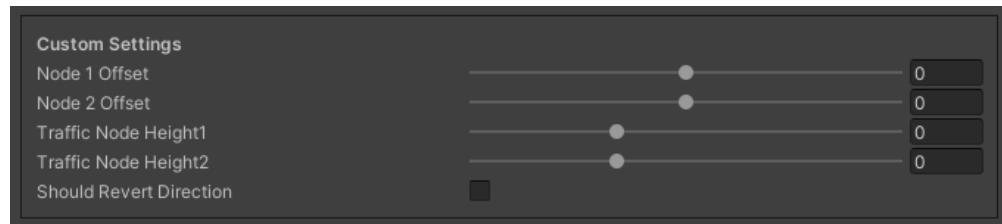
SubTrafficNode distance from center : distance between the *SubTrafficNode* (node that contains a sub-lane) and the center of the segment.

Is enter of oneway : if it is on, it is the start of one-way traffic, if it is off, it is the end of one-way traffic.



Example.

Oneway Straight



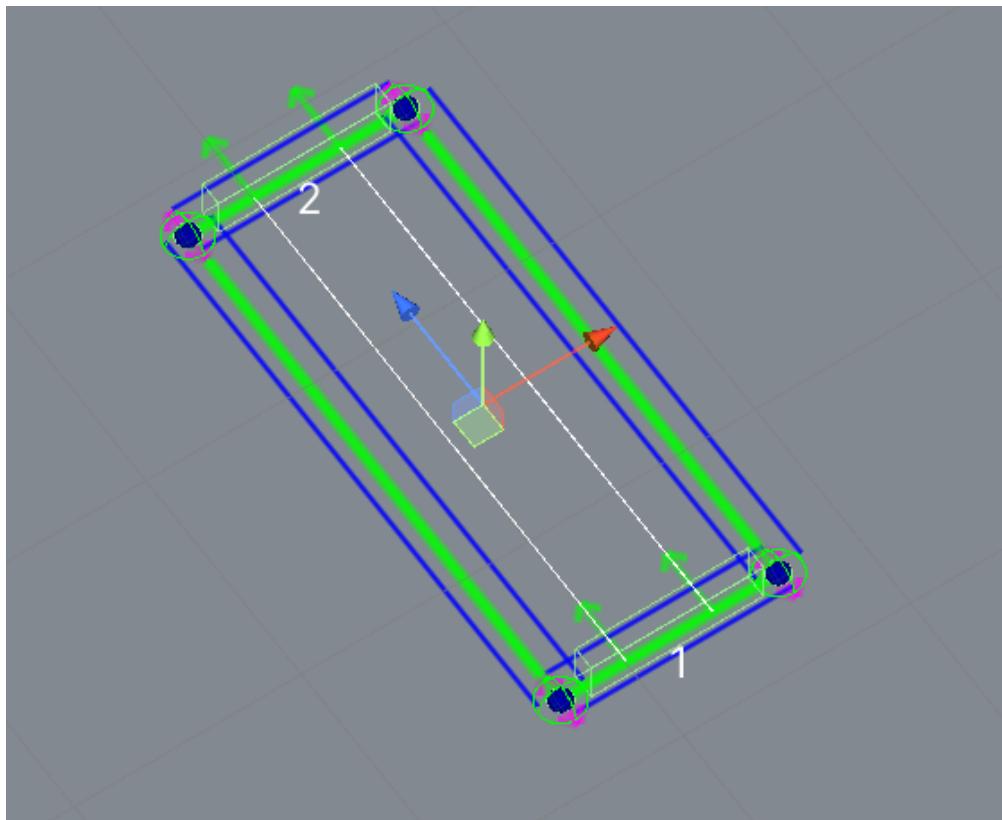
Node 1 offset : node 1 offset on the X-axis.

Node 2 offset : node 2 offset on the X-axis.

Traffic node height 1 : node 1 offset on the Y-axis.

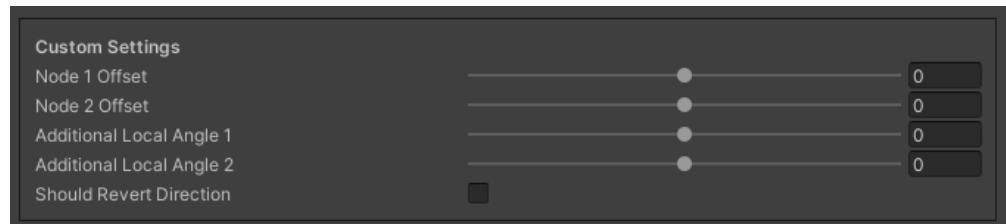
Traffic node height 2 : node 2 offset on the Y-axis.

Should revert direction : direction of the crossroad lanes will be reversed.



Example.

Oneway Turn



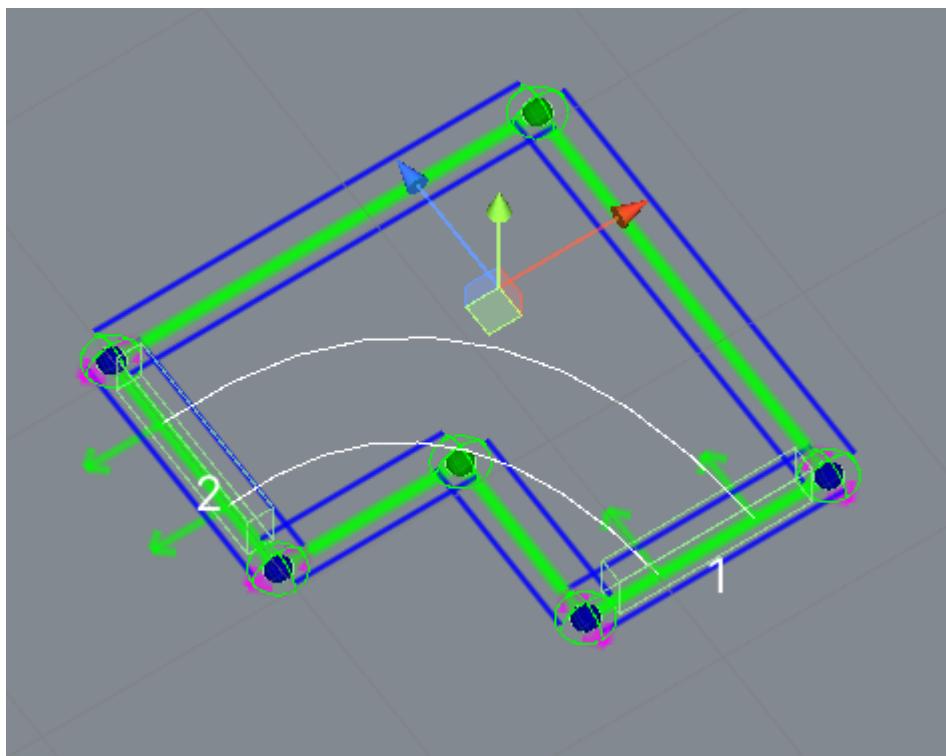
Node 1 offset : node 1 offset on the X-axis.

Node 2 offset : node 2 offset on the X-axis.

Additional local angle 1 : additional node 1 rotation angle.

Additional local angle 2 : additional node 2 rotation angle.

Should revert direction : direction of the crossroad lanes will be reversed.



Example.

2.3.3 Custom Straight Road

Creator for creating straight roads of any shape.

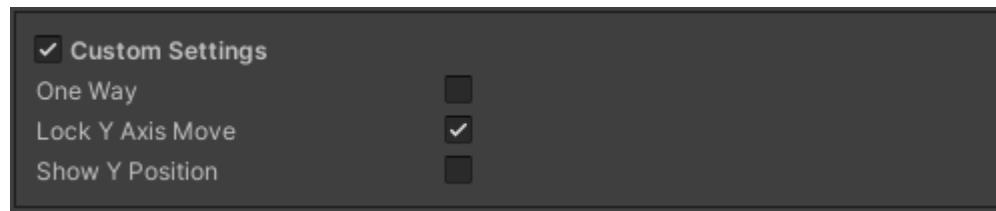
[Youtube tutorial.](#)

How To Use

1. Place the custom straight segment where you want it.
2. Place the *traffic nodes* at the start and the end of the path (or expand the road by holding *left-shift* key and clicking the *left-mouse* button).
3. Rotate the *TrafficNodes* in the direction of the route (make sure that the *rotation of the nodes* is set correctly).
4. Adjust the number of lanes and the speed limit of the segment.
5. If necessary, add more additional nodes to the paths (by pressing + in the scene) [**optional step**].
6. Rotate the nodes of the paths according to the direction of the path [**optional step**].
7. *Snap TrafficNodes* to the surface by pressing the *Snap To Surface* button if necessary [**optional step**].
8. Complete all the *default steps*.

Settings

Custom Settings

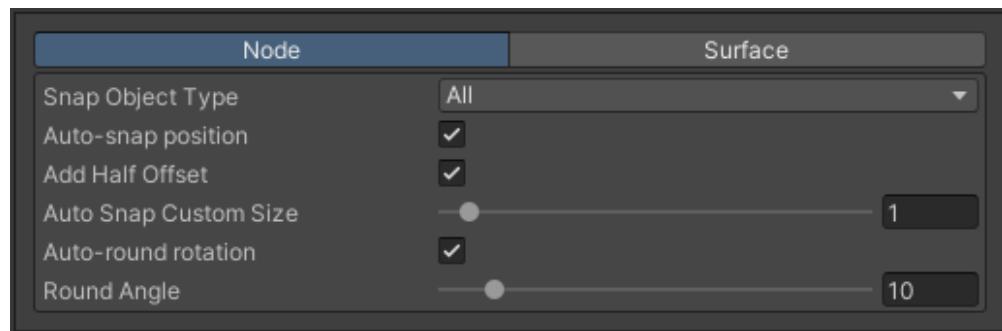


One way : segment contains only one-way paths.

Lock Y axis move : lock the Y axis to move the nodes.

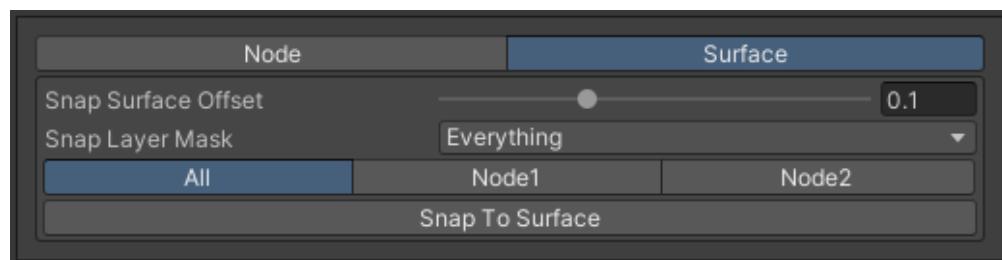
Show Y position : show Y position of the nodes.

Snap Node Settings



Info.

Snap Surface Settings



Snap surface offset : offset between snap point and the node (Y axis).

Node Buttons

[which node you want to snap to.]

- All

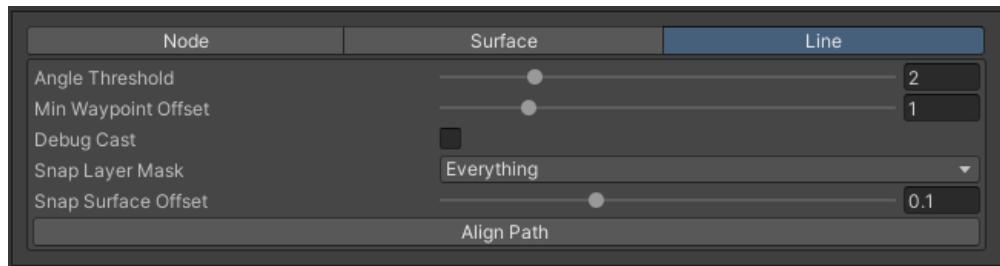
- Node1
- Node2

Buttons:

- Snap to surface: snap selected nodes to the surface.

Snap Line Settings

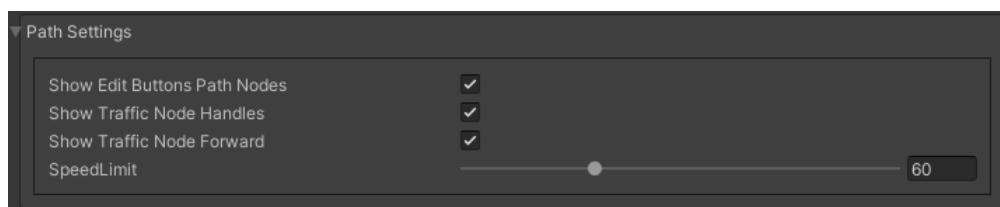
Creates additional *path nodes* along the curved meshes of the collider to make the *path* follow the shape of the collider (**v 1.0.4+**).



Angle threshold : minimum angle between normal faces to create new *path nodes*.

Min waypoint offset : min offset between generated *path nodes*.

Snap surface offset : offset between snap point and the node (Y axis).

Path Settings

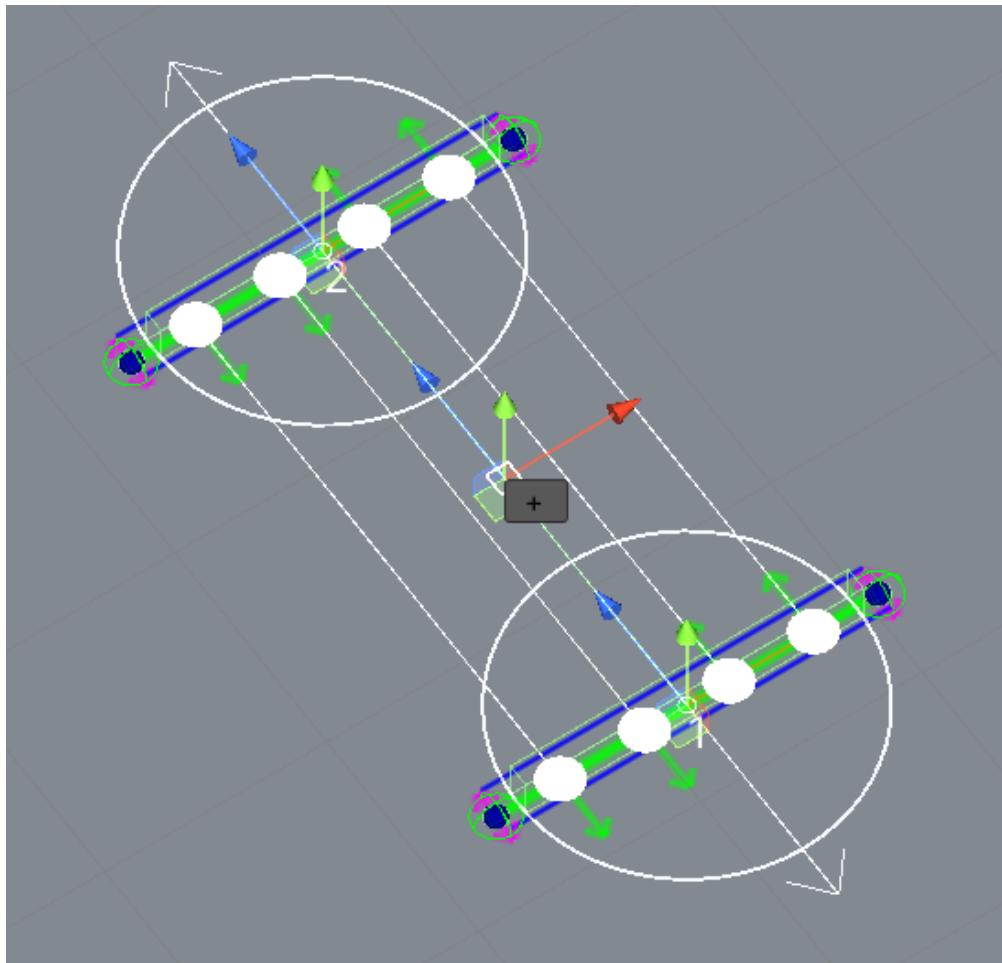
Show edit buttons path nodes : on/off edit (add & remove) button paths of node.

Show traffic node handles : on/off traffic node position handles.

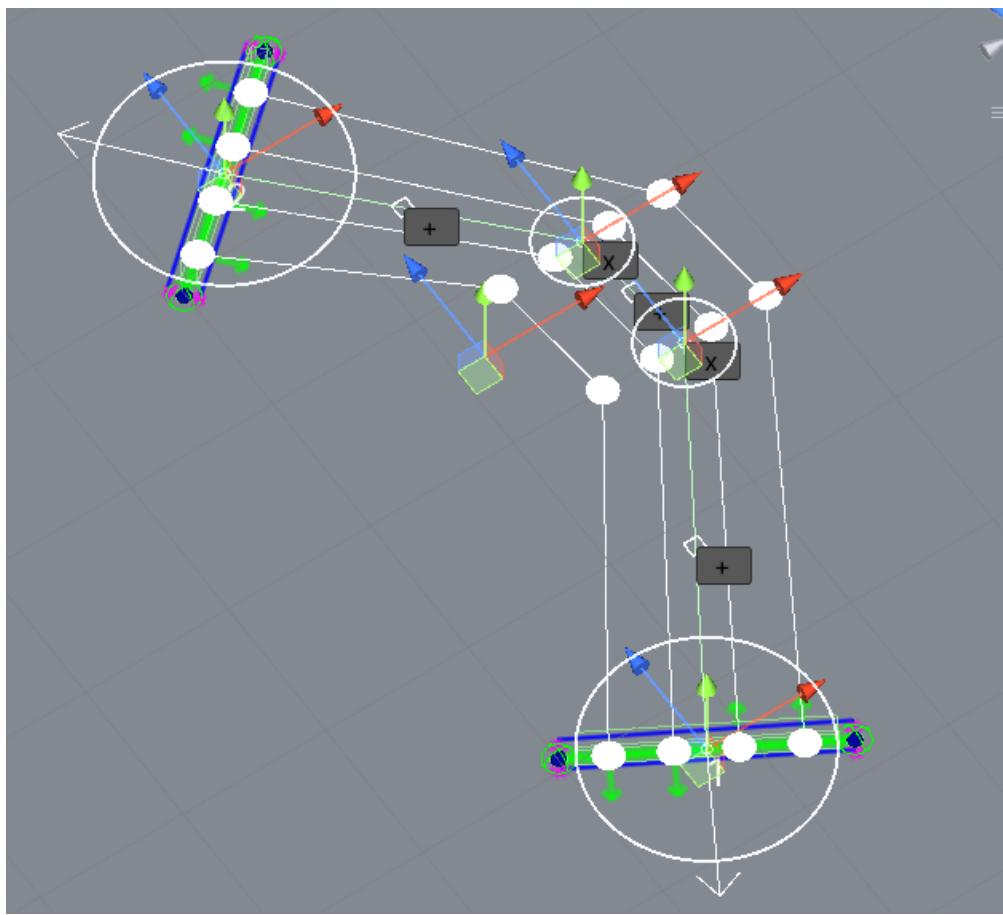
Show traffic node forward : on/off display of node's forward direction.

Speedlimit : speed limit for all paths of the segment.

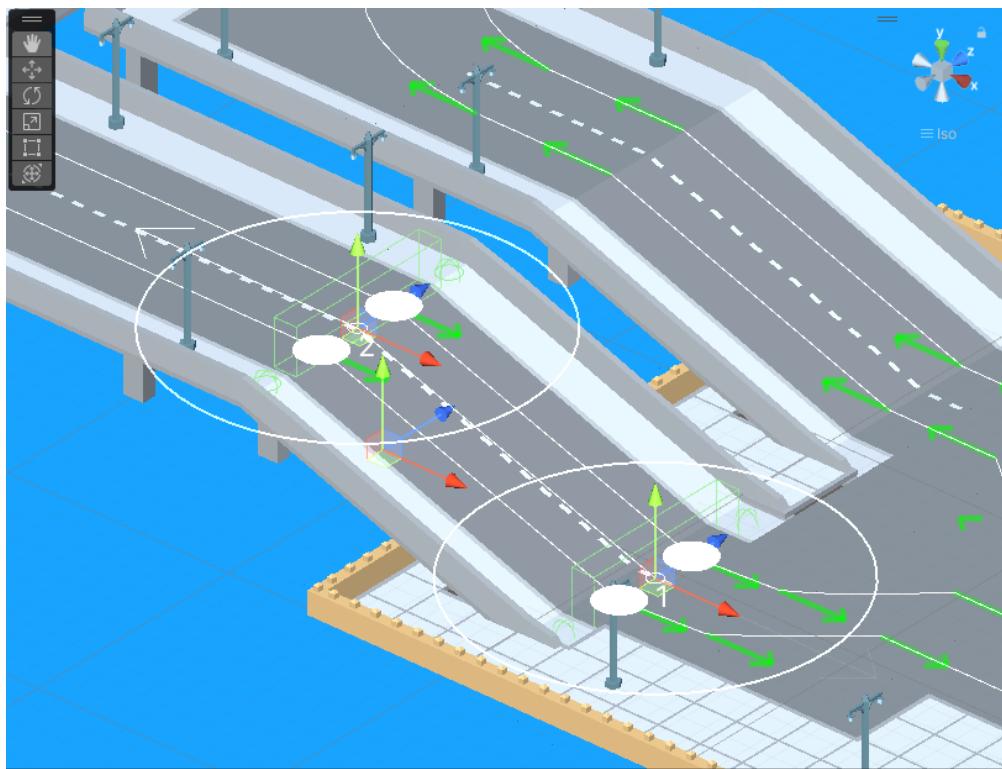
Examples



Source segment example.



Complex shape example.



Surface snapping example.

2.3.4 Custom Segment

Creator for creating segments of any shape and complexity.

Youtube tutorial.

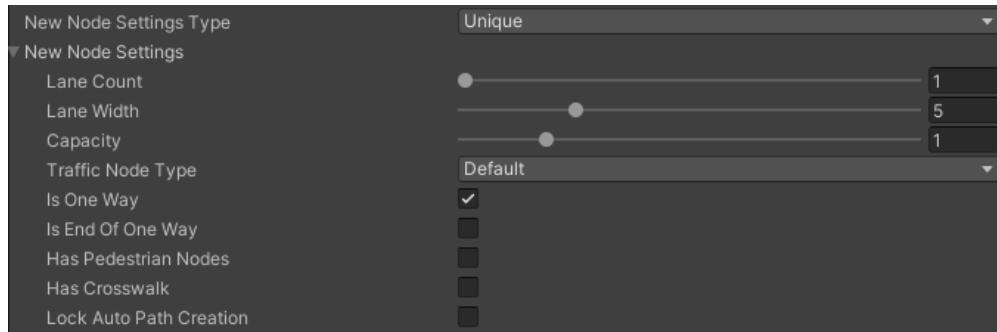
How To Use

1. Place the custom segment where you want it.
2. Toggle on *Custom settings* parameter.
3. Select the *New node settings type* & create a new *TrafficNode* by pressing the *Add Traffic Node* button [optional step].
4. *Place* & rotate all created *TrafficNode* according to your needs (make sure that the *rotation of the nodes* is set correctly).
5. *Snap TrafficNodes* to the surface by pressing the *Snap To Surface* button if required [optional step].
6. Open the *PathCreator tool* to quickly create *paths* between *nodes*.
7. Complete all the *default steps*.

Note: You can convert any *default template* to *Custom Segment* in the *Other settings* tab.

Settings

New Node Settings

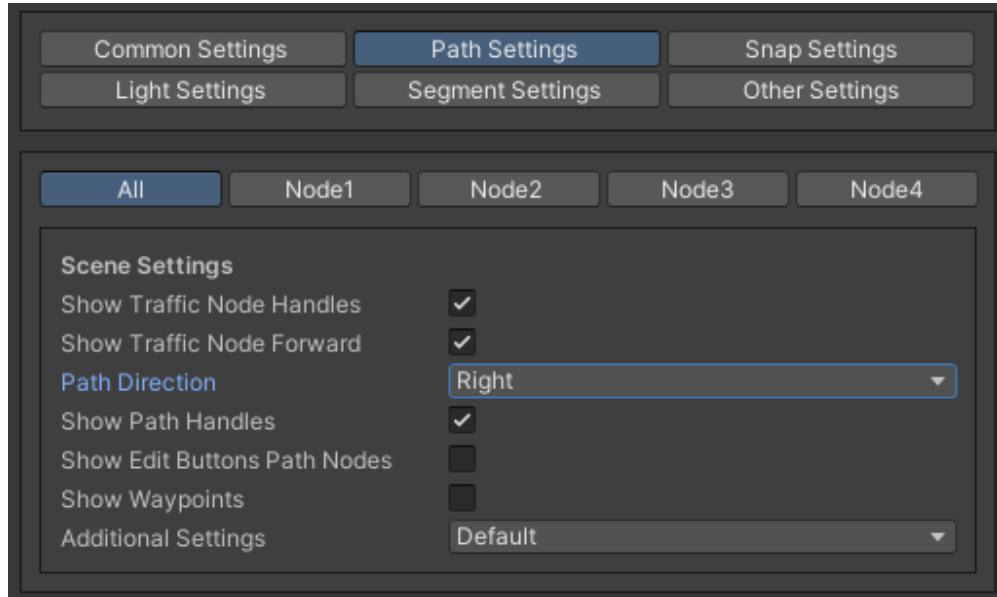


Custom settings : on/off custom settings for advanced node customization.

New node settings type [custom settings enabled] new *TrafficNode* will be created like:

- **Prefab** : new prefab.
- **Unique** : created with unique defined *settings*.
- **Copy last** : will be created with the settings of the last created node.
- **Copy selected**
[will be created with the settings of the selected node.]
– **Copy node index**

Custom Path Settings

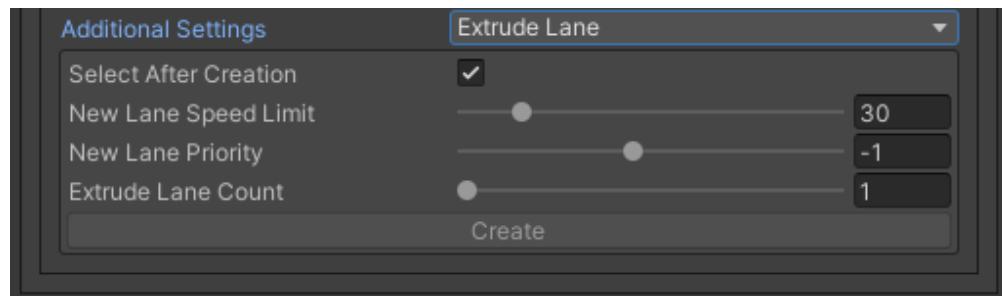


Show traffic node handles : on/off handles of *TrafficNodes*

Show traffic node forward : on/off display of *TrafficNode* forwading.

Additional Settings

Extrude Lane



How to use:

1. Drag the green sphere from where you want the new lane to start.
2. Drop the cursor where you want the lane to end.
3. Adjust the position handle of the new path.
4. Press *E* key or press *Create* button in the inspector to create new lane.

Parking Builder

Parking Builder info.

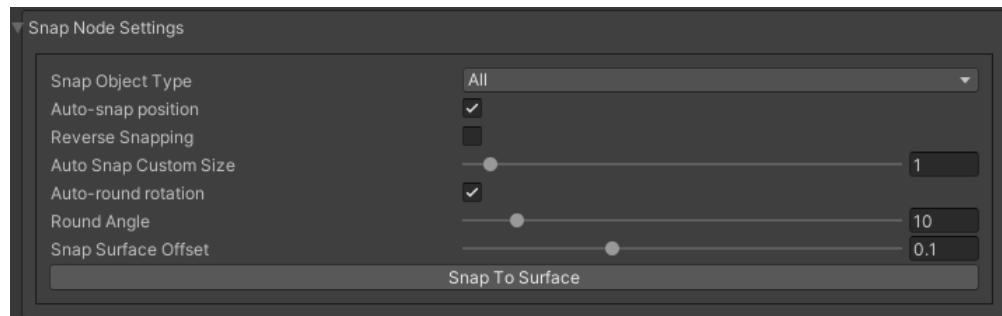
Custom Settings



Lock Y axis move : lock the Y axis to move the nodes.

Show Y position : show Y position of the nodes.

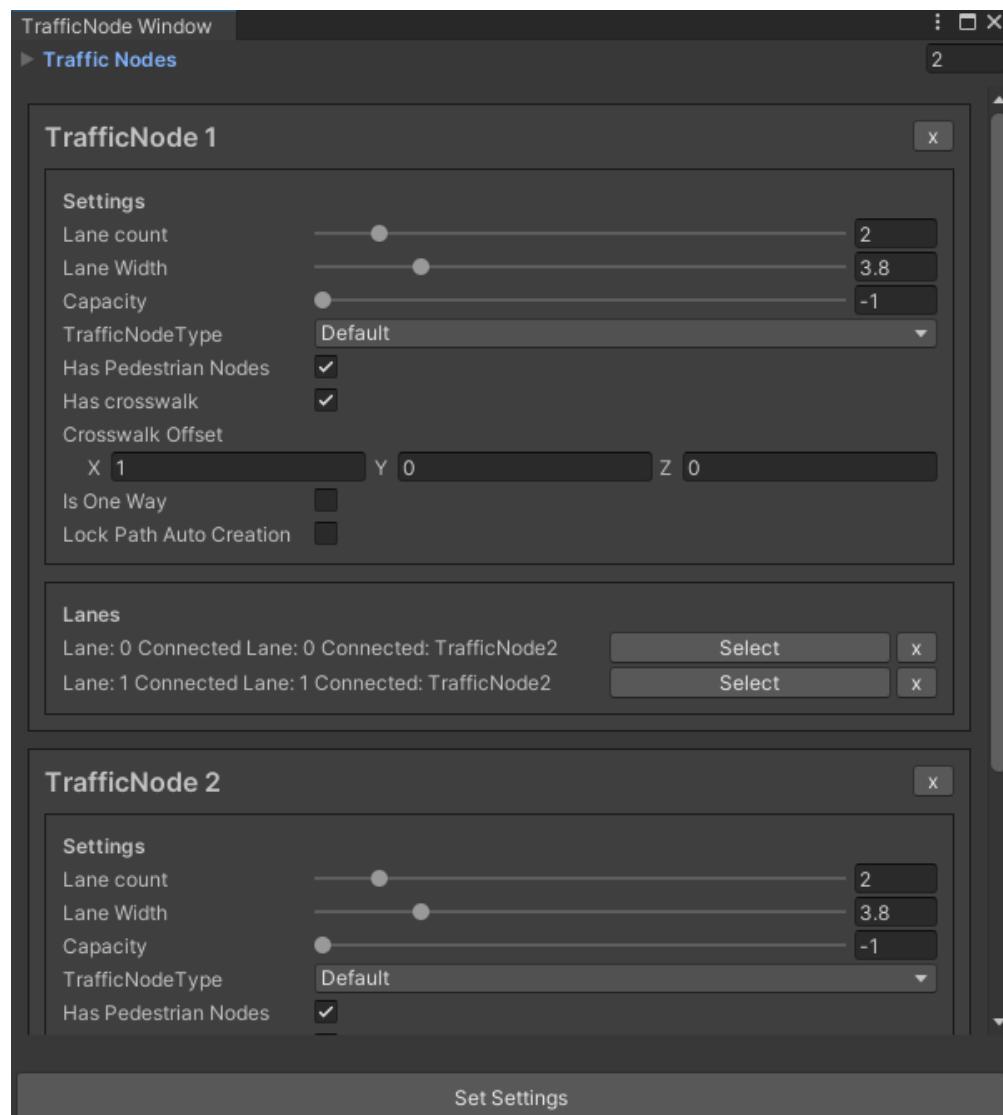
Snap Node Settings



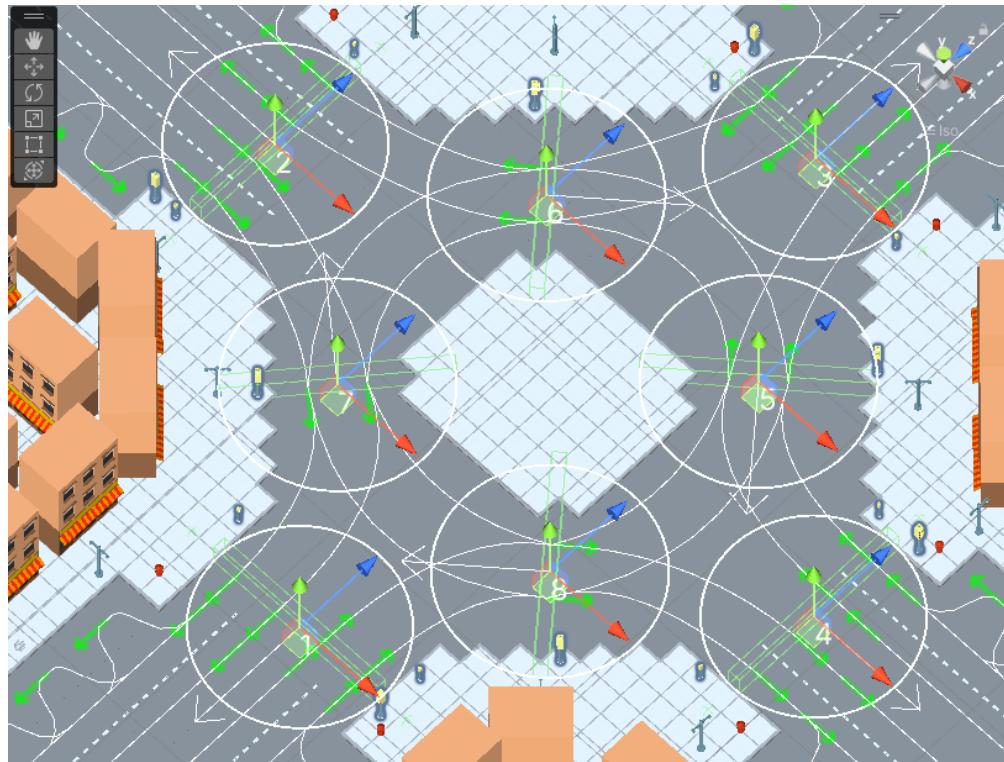
Info.

Custom TrafficNode Editor Window

Window that you can configure each *TrafficNode settings*. *Custom settings* should be enabled.



Examples



Example.

Settings Description

Snap Node Settings

Snap object type:

- **All** : snap *TrafficNode & Path node*.
- **Traffic node** : only *TrafficNode*.
- **Path node** : only *Path node*.

Auto-snap position on/off position snapping.

- **Add half offset** : the snapped object is shifted by half of the set snapping size.

Auto snap custom size : snapping value.

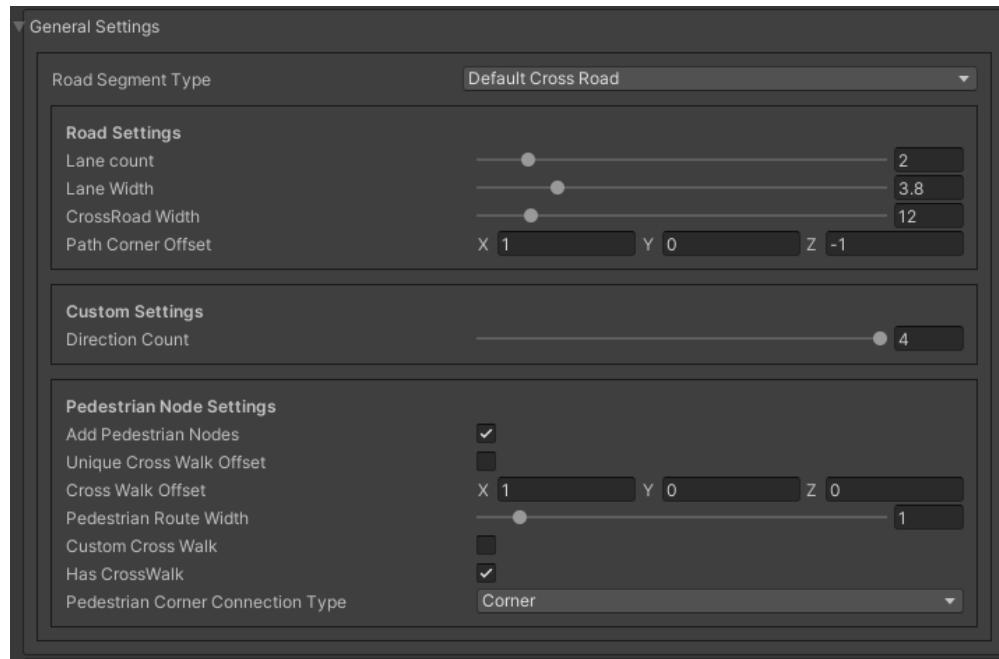
Auto round rotation:

[on/off rotation snapping.]

- **Round angle** : snapping angle value.

2.3.5 Components

General settings



Lane count : number of lanes.

Lane width : lane width.

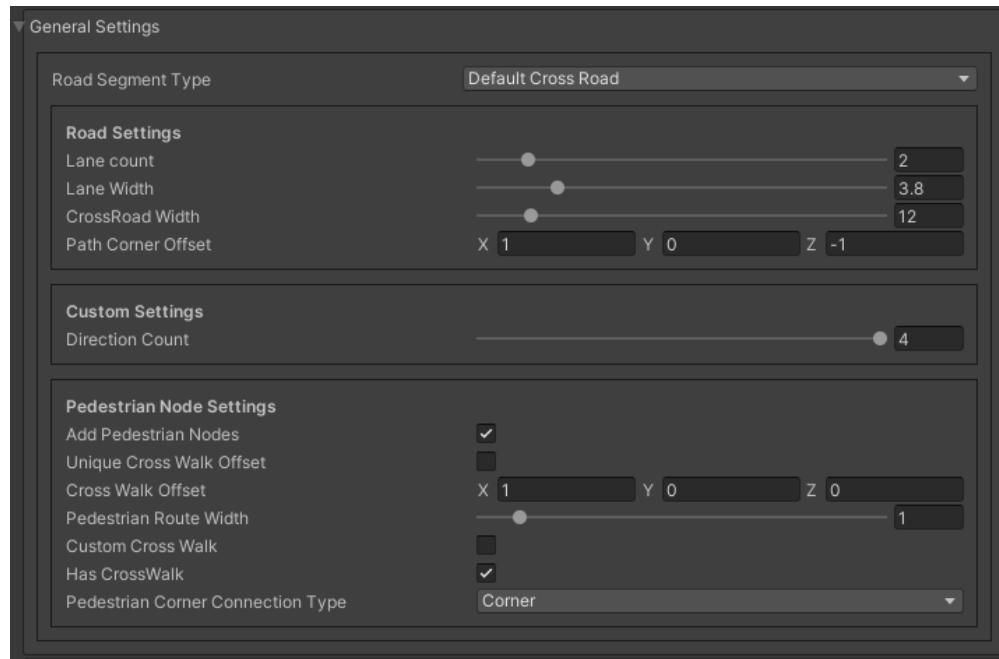
Crossroad width : distance between *traffic nodes*.

Path corner offset : offset to change the rotation angle of curved paths.

Custom settings

Custom settings.

Pedestrian node settings



Add pedestrian nodes : add a *pedestrian nodes* to the segment.

Unique crosswalk offset : set up a unique offset for the selected crosswalk.

Crosswalk offset : set up a common offset for the crosswalks.

Pedestrian route width : *pedestrian route width*.

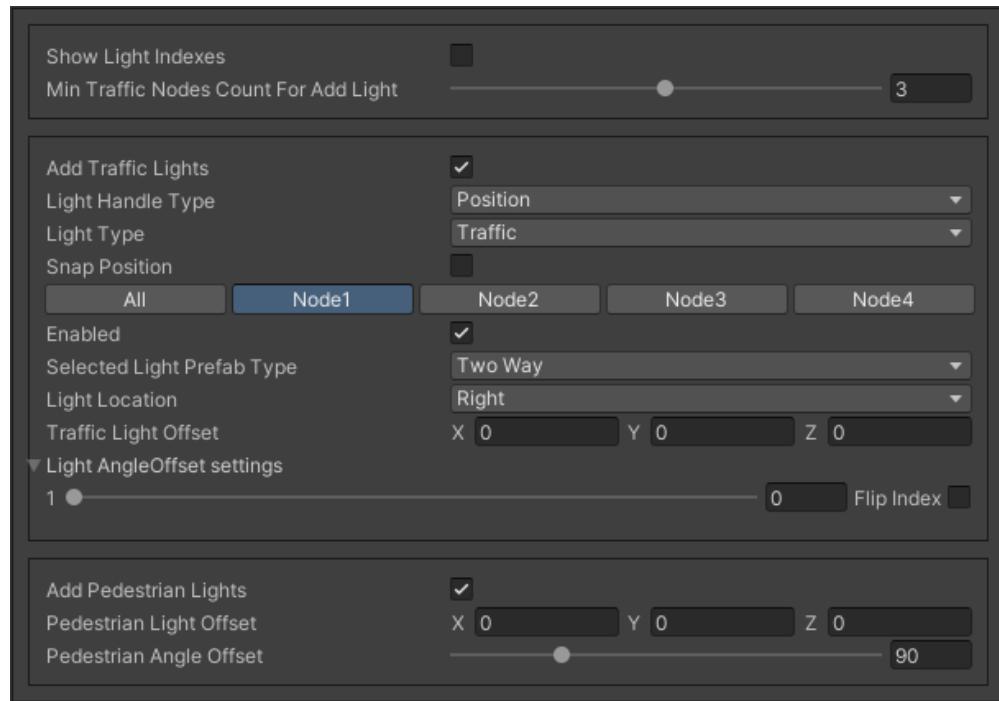
Custom crosswalk : on/off selected crosswalk.

Has crosswalk : on/off *crosswalk* for pedestrians.

Pedestrian corner connection type:

- **Disabled**
- **Corner** : will be created corner *pedestrian node* to connect crosswalks.
- **Straight** : crosswalks will be connected directly.

Light settings



Youtube tutorial.

How To Use

1. Turn on traffic light option.
2. Select *Light prefab type*.
3. Set the traffic light offset or enable *Light handle type*.
4. If you want to configure the traffic lights individually, select the *Node* button with the appropriate index.

Traffic lights

Show light indexes: on/off display light *TrafficLightHandler* index around *traffic nodes* and traffic lights in the scene.

Min TrafficNodes count for add light: minimum number of *traffic nodes* in the segment to add traffic light.

Add traffic light: add traffic light to the segment.

Light handle type:

- **None**
- **Position**: enable position handle for traffic lights.
- **Rotation**: enable rotation handle for traffic lights.

Selected light prefab type

[prefab of the traffic light to be added [can be changed in creator settings].]

- **Oneway**
- **Two way**
- **Four way**

Light location :

- **Right** : will be added to the right of the *traffic nodes*.
- **Left** : will be added to the left of the *traffic nodes*.
- **Right left** : will be added on both sides of the *traffic node*.

Traffic lights offset : local traffic light offset relative to *traffic node*.

Light angle offset settings:

- **Angle offset** : local rotation angle of the traffic light.
- **Flip index** : switches to the opposite *light index* in the traffic light.

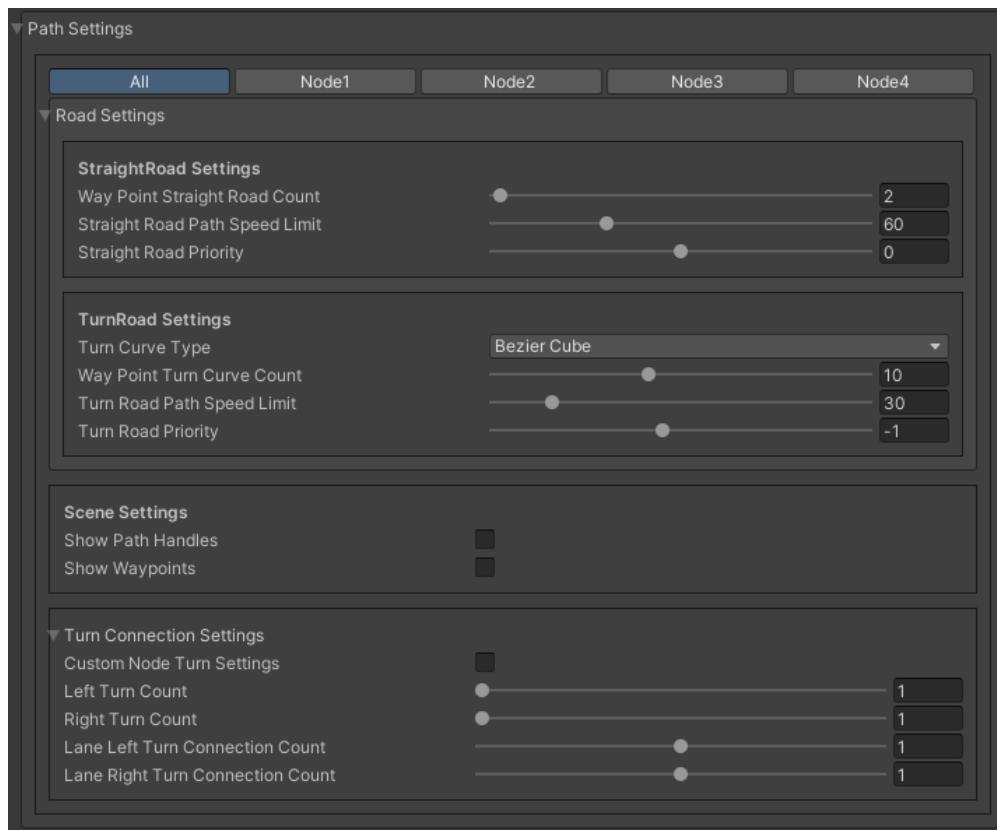
Pedestrian lights

Add pedestrian lights : add pedestrian light to the segment.

Pedestrian light offset : local pedestrian light offset relative to *traffic node*

Pedestrian angle offset : pedestrian light rotation angle offset.

Path settings



Node selection panel

How to customize path:

1. Select *TrafficNode* on the inspector panel.
2. Select desired *path* on the inspector panel (it will be highlighted in the scene).
3. Adjust the position of the path nodes (make sure *path handles* is enabled).
4. Press *Open Path Settings* button to customize *Path settings window*.

Road settings

StraightRoad settings: [settings for straight paths of the segment]

- Waypoint Straightroad count
- Straight road path speed limit
- Straight road priority

TurnRoad settings: [settings for turn paths of the segment]

- Turn curve type
- Waypoint turn curve count

- Turnroad path speed limit
- Turnroad priority

Scene settings

Show path handles

[on/off position handles in the scene.]

- **Show edit buttons path nodes** : on/off *add & remove* buttons nodes in the scene.

Show waypoints

[on/off visual circle position of the waypoint in the scene.]

- **Show waypoints info** : on/off info of waypoints (local index, speedlimit).

Turn connection settings

Custom node turn settings : on/off the turn settings for each *traffic node*.

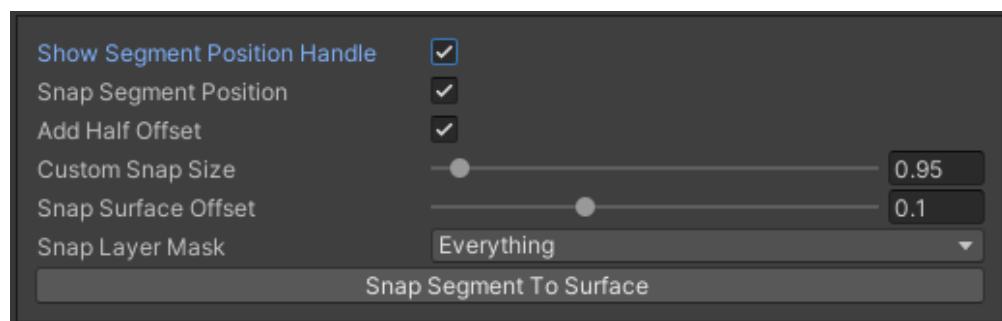
Left turn count : number of left turns from the *traffic node*.

Right turn count : number of right turns from the *traffic node*.

Lane left turn connection count : number of connections to the left from the lane traffic lane.

Lane right turn connection count : number of connections to the right from the lane traffic lane.

Segment handler settings



Show segment position handle : on/off position handle for segment.

Snap segment position : on/off snap segmant position.

Add half offset : the snapped object is shifted by half of the set snapping size.

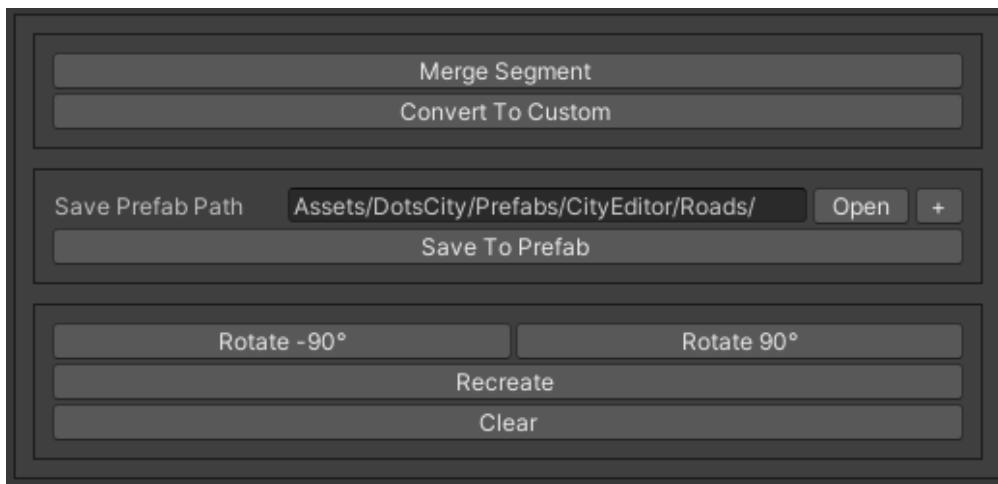
Custom snap size : snapping value.

Snap surface offset : snap surface offset.

Snap layer mask : snap layermask.

Snap segment to surface : snap the segment to the surface.

Other settings



Merge segment : opens the merge segment tool.

Convert to custom : converts the current segment to *custom segment*.

Save prefab paths : segment save prefab path.

Save to prefab : save segment to prefab.

Buttons

Rotate -90°/90° : rotate segment by 90° degree.

Recreate : recreate segment.

Clear : clear segment.

Hotkeys

Capslock : rotate segment by 90° degree.

2.3.6 Parking Builder

A tool to quickly create a parking space. Is part of the *RoadSegmentCreator* and can only be enabled in the *custom segment*.

Youtube tutorial.

How To Use

1. Position a *custom segment* on the road where the parking spaces will be.



2. Set the size of the parking slot (*settings*).

3. Enable position handle



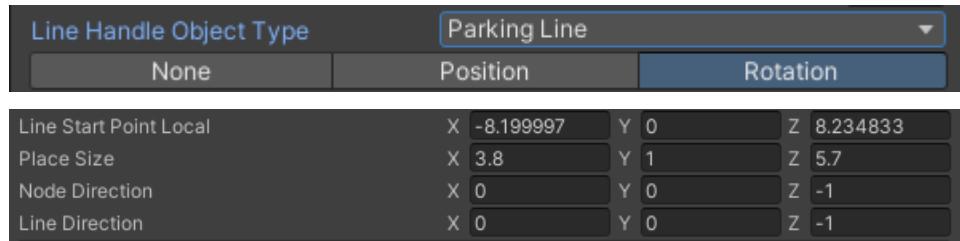
4. Position the parking pointer where you want the line to start.



5. Enable rotation handle and set the rotation of the parking slot by dragging a circle in the scene.



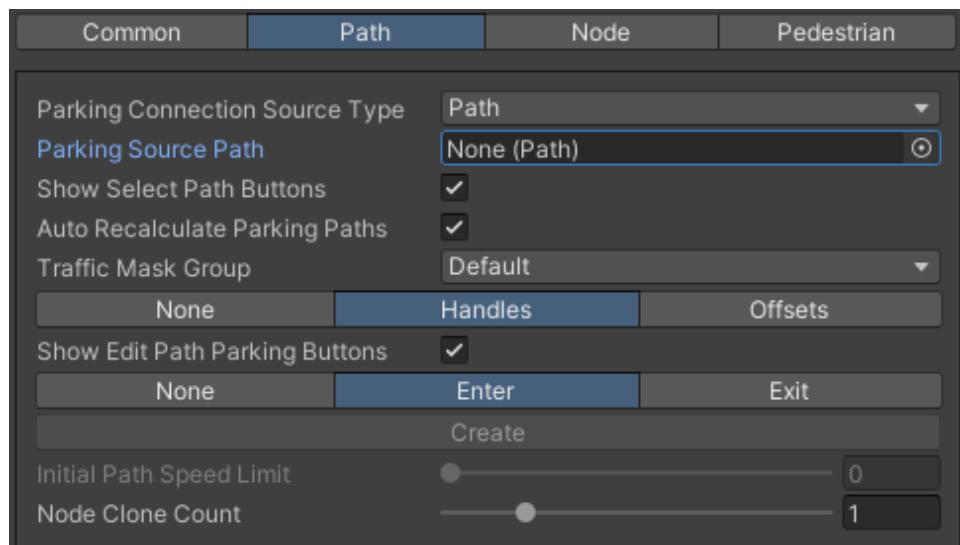
- Set the object parking line to *parking line* and rotate the direction of the parking line by dragging a circle in the scene.



- Enter the *number of parking slots*.

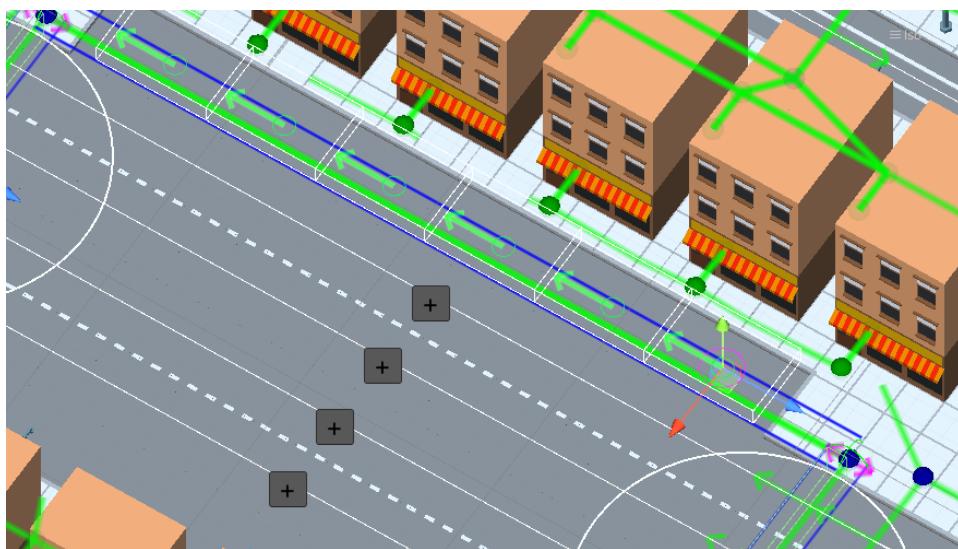


- Open the *Path* tab.

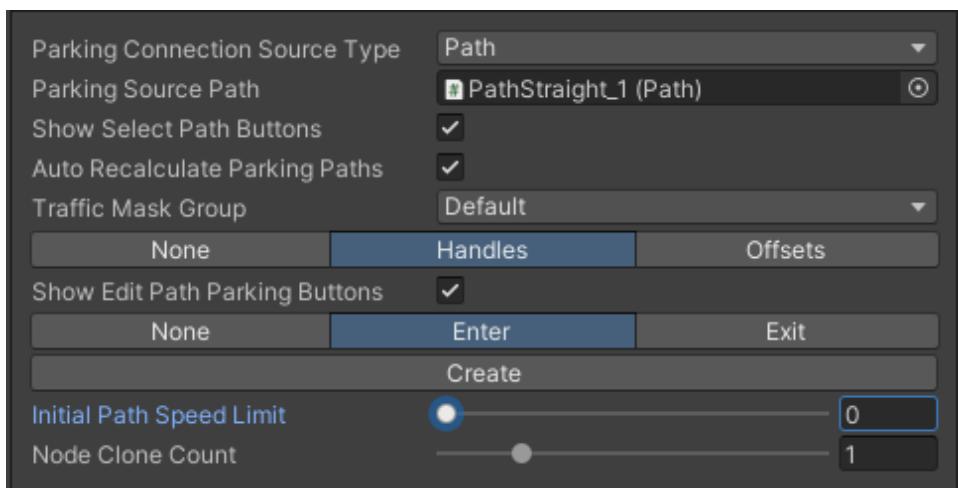


- Toggle on *Show select path buttons* option.

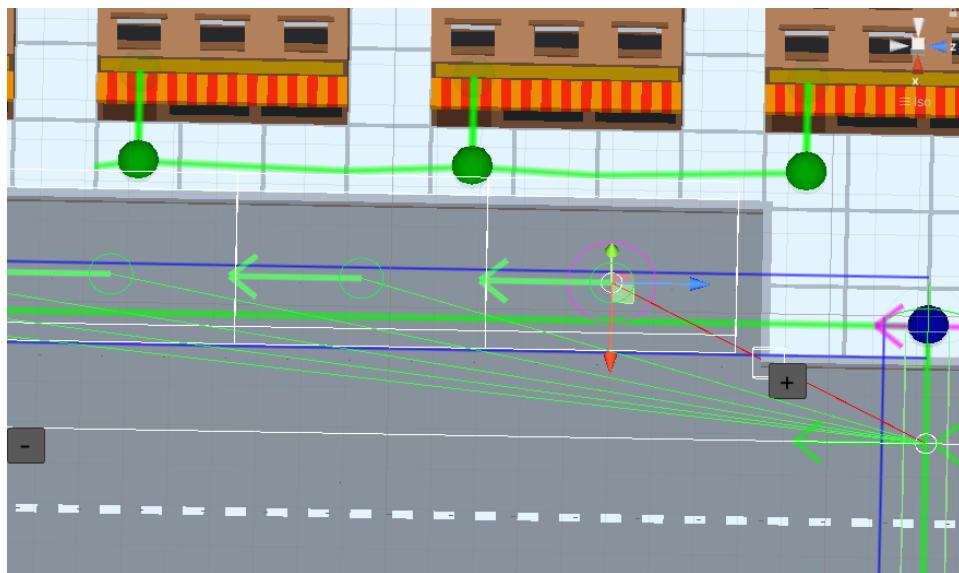
10. Select the source path in the scene.



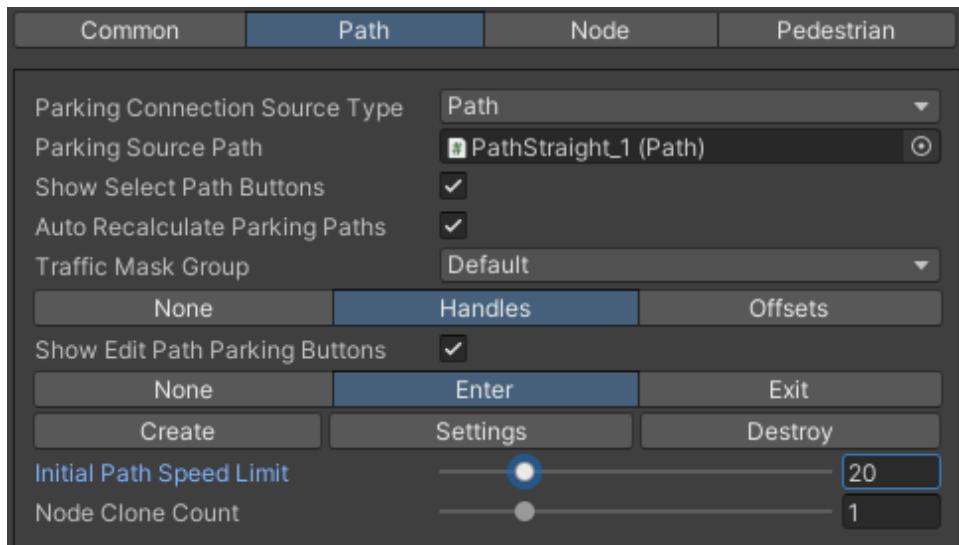
11. Select the *Enter* tab and press the *Create* button.

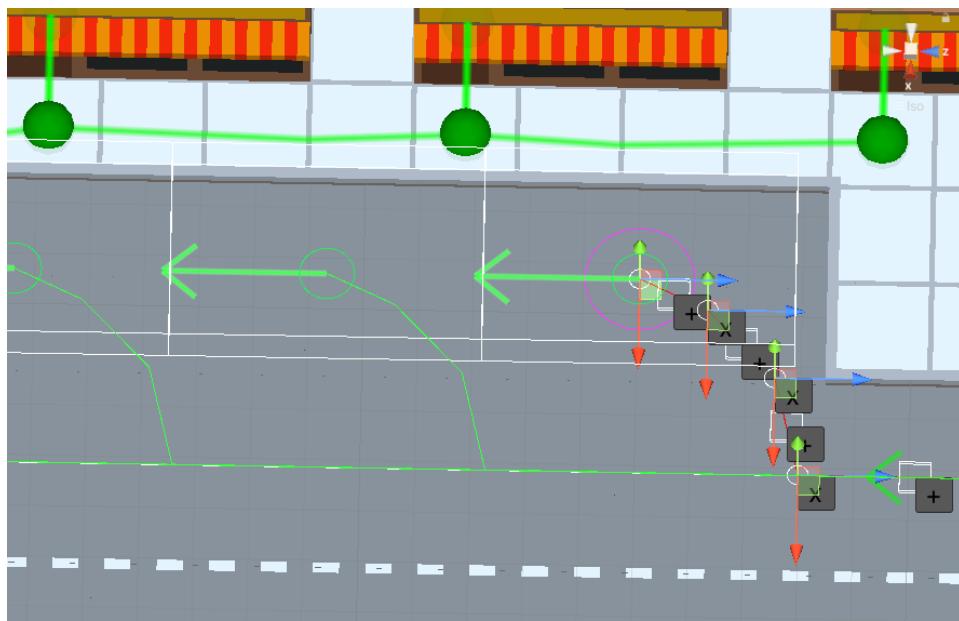


12. In the created path create additional waypoint nodes by pressing + in the scene.

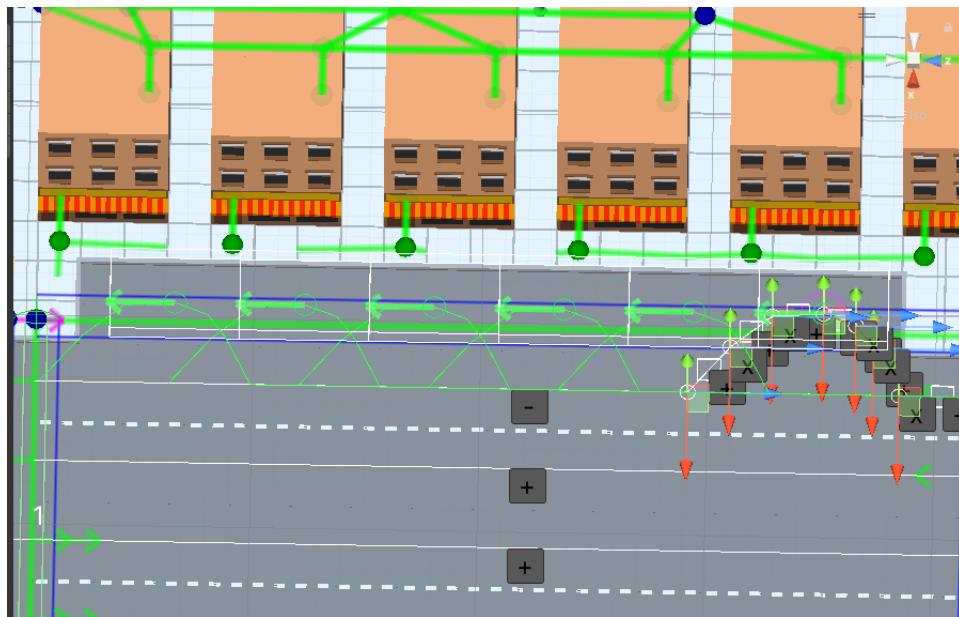


13. Customize *Traffic Group*, *Initial speed limit* and *Node Clone Count* parameters.

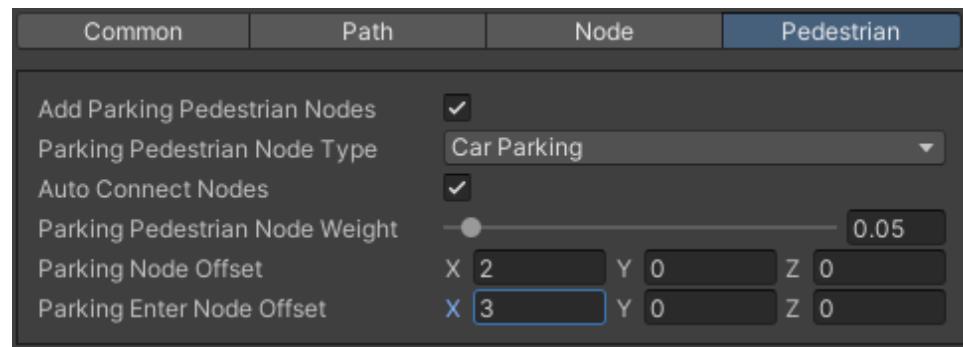




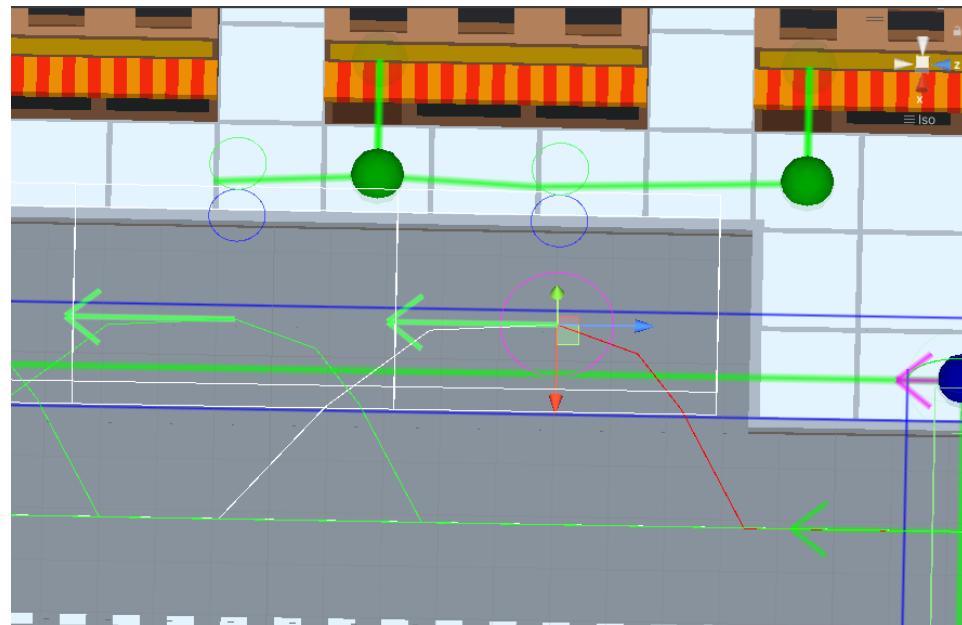
14. Open the *Offsets* tab and adjust the position handle for each node individually if required [optional].
15. Repeat the same steps (11 - 14) for the *exit path*.



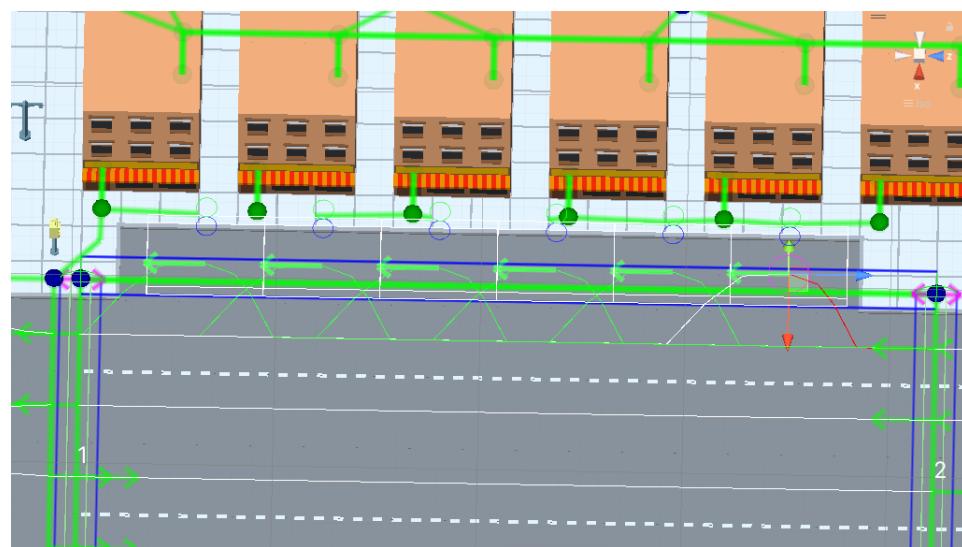
16. Open *Pedestrian* tab.



17. Customize *Weight*, *Parking node offset* and *Parking enter node offset*

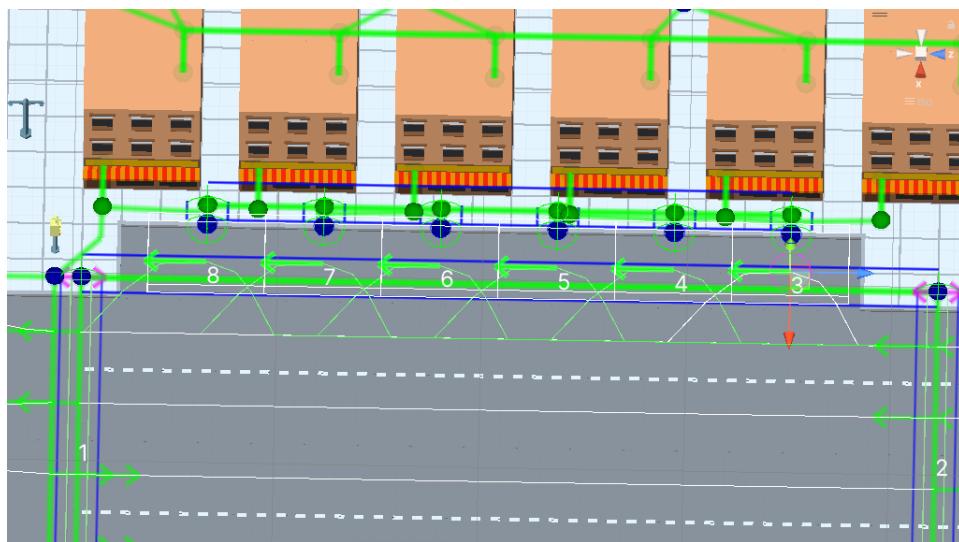


Blue circle - enter parking car PedestrianNode. Green circle - default PedestrianNode linked to the parking PedestrianNode.



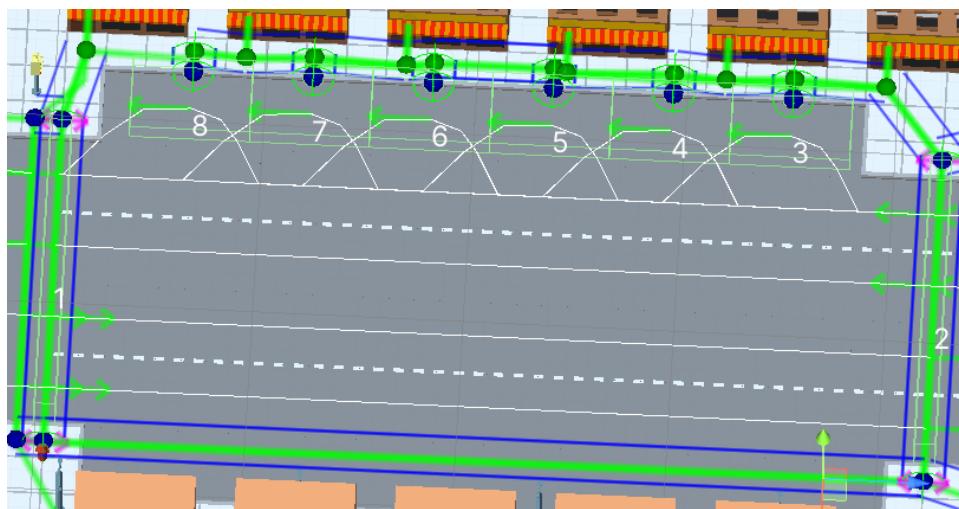
Preview parking line result.

18. Press *Create Line* button.



Create line result.

19. Connect the *pedestrian nodes* to the *pedestrian nodes* of the city.



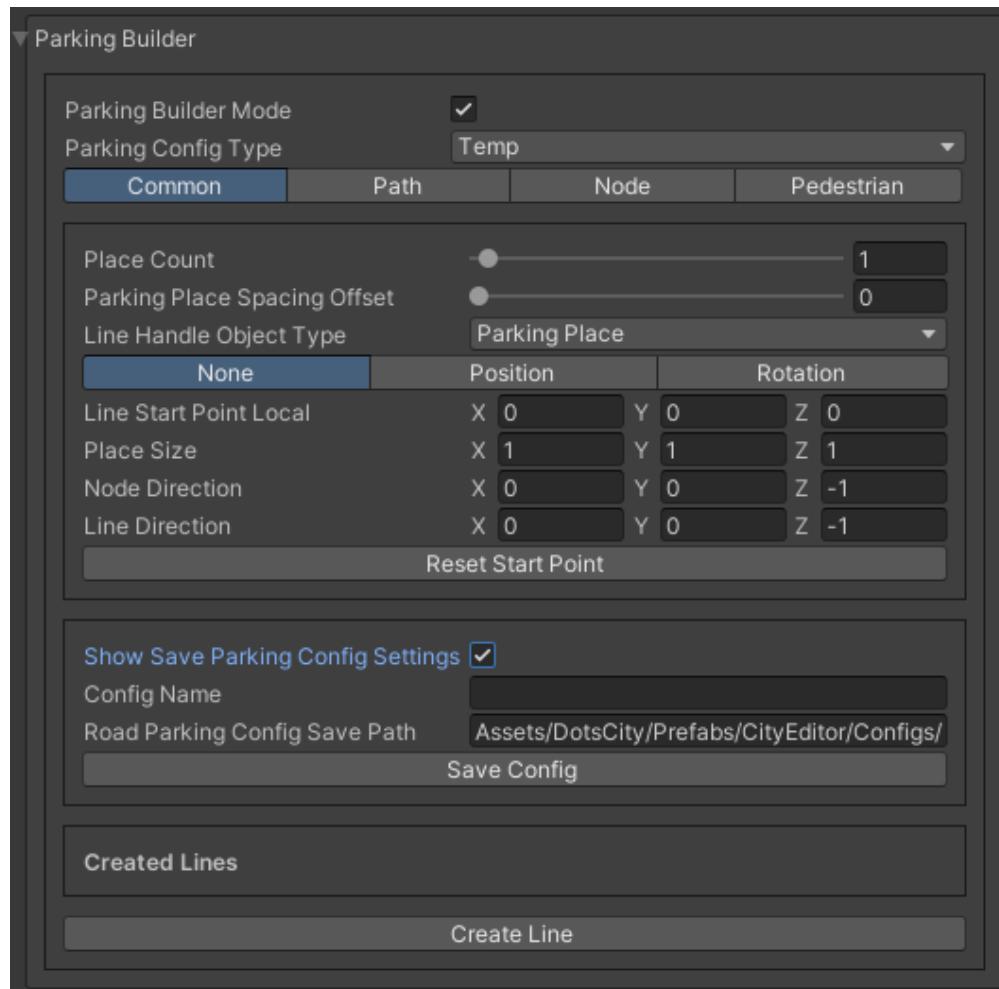
Note:

Created lines can be edited or deleted in the *Created lines* tab.



Settings

Common



Place count : number of parking slots.

Parking place spacing offset : distance between parking slots.

Line object type:

- **Parking place** : handle parking place.
- **Parking line** : handle parking line.

Handles:

- **None** : no handles.
- **Position** : enabled position handle for the place or line.
- **Rotation** : enabled rotation handle for the place or line.

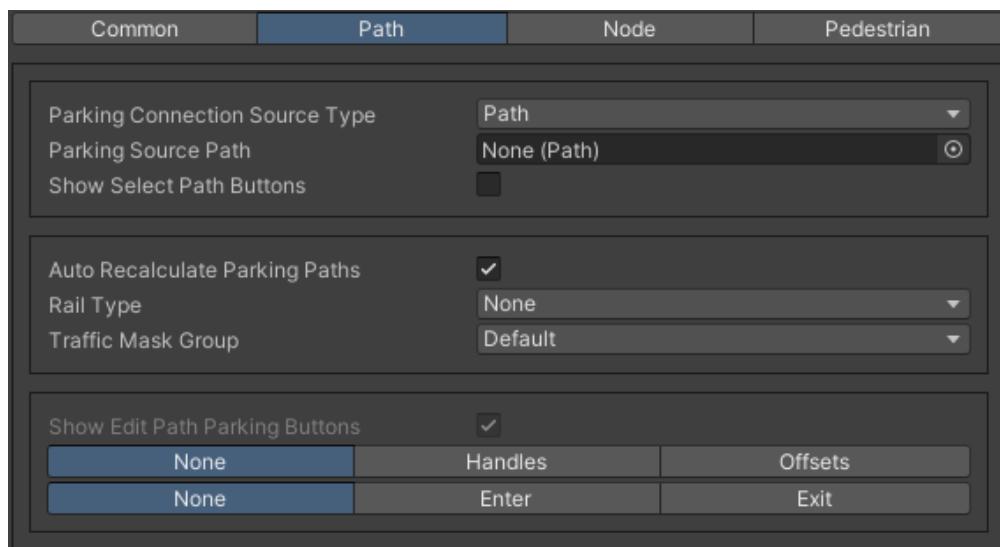
Line start point local : local parking line start position.

Place size : parking lot size.

Node direction : local direction of the *TrafficNode* in the parking place.

Line direction : local direction of the parking line.

Path



Parking connection source type :

- **Path** [paths will be connected to the *Parking source path (PathPoint connection)*]
 - **Parking source path** : path from which the created parking slot paths will start and end.
 - **Show select path buttons** : on/off display exist paths of the segment to add a parking source path.
- **Node** [paths will be connected to the selected *TrafficNodes (TrafficNode connection)*]
 - **Source TrafficNode** : node from which the created parking slot paths will start.
 - **Target TrafficNode** : node to which the paths connected from the parking place.
- **Single node** [paths will be connected to the selected single *TrafficNode* (same node for enter & exit paths)]
 - **Source TrafficNode** : enter & exit *TrafficNode* for parking *paths* are the same.

Auto recalculate parking paths : paths ends will be recalculated when changing the position of the parking line.

Rail type:

- **None** : *Rail Movement* is disabled.
- **Enter only** : the vehicles entering the car park have a *Rail Movement*.
- **Exit only** : the vehicles leaving the car park have a *Rail Movement*.
- **Enter & exit** : enter & exit paths have a *Rail Movement*.

Traffic mask group : *group* of the vehicles that allowed on the parking.

Show edit path parking buttons : on/off edit (add & remove) buttons of the path.

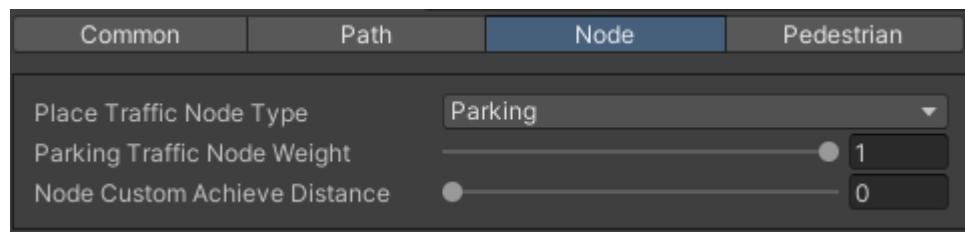
Handles Panel:

- **None** : handles disabled.
- **Handles** : position handles of the path enabled for first parking place.
- **Offsets** : position handles for all parking places.

Path Selection Panel:

- **None** : displayed *Enter & Exit* paths.
- **Enter**
[displayed only *Enter* paths.]
 - **Initial path speed limit** : initial speed limit of *Enter* paths.
 - **Node clone count** : number of nodes in the next paths that are will clone position from source path.
- **Exit**
[displayed only *Exit* paths]
 - **Initial path speed limit** : initial speed limit of exit paths.
 - **Node skip last count** : number of last nodes in the next paths that are will clone position the last nodes from source path.

Node

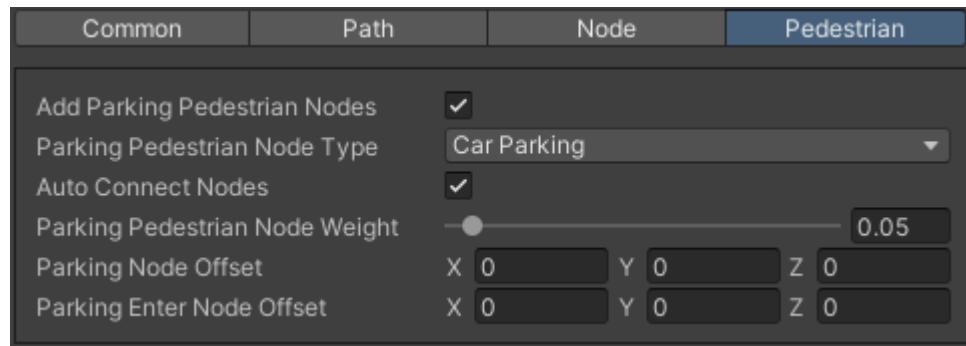


Place TrafficNode type : *TrafficNode type*.

Parking TrafficNode weight : *TrafficNode weight*.

Node custom achieve distance : custom distance to achieve a node (if 0 value default value will be taken).

Pedestrian



Add parking pedestrian nodes : add an *entry parking node* and a *node* linking it.

Parking pedestrian node type : *parking node type*.

Auto connect nodes : auto connect created entry parking node and nearby created node.

Parking pedestrian node weight : *weight* entry parking node.

Parking node offset : *entry parking node* offset relative to *traffic nodes*.

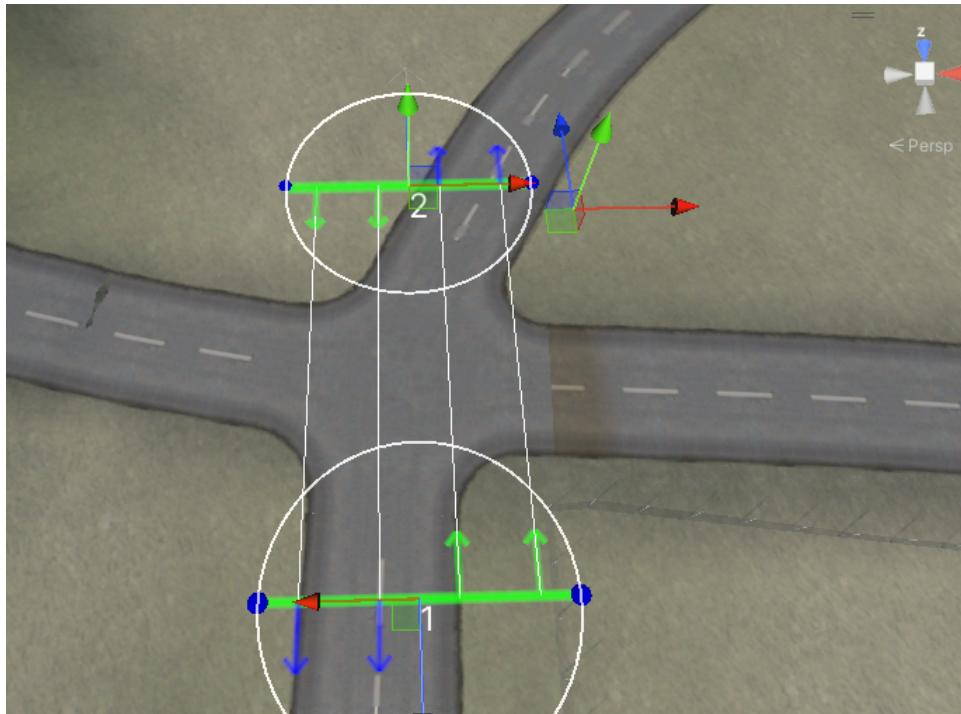
Parking enter node offset : *node* that connected to *entry parking node* relative to *traffic nodes*.

2.3.7 Auto Crossroad

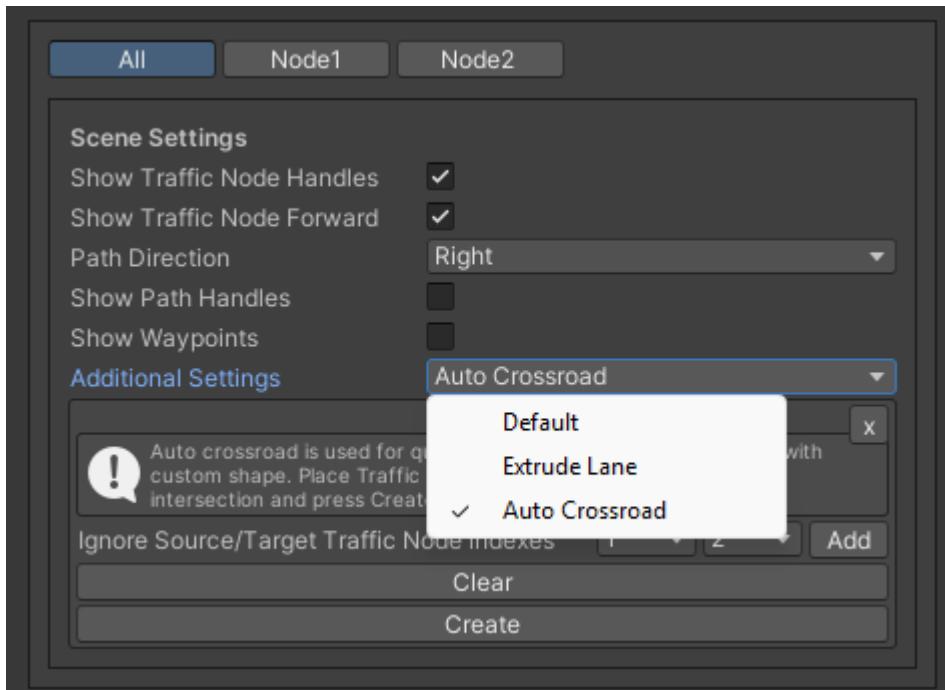
For automatic generation of custom crossroads, use this feature.

How To Use

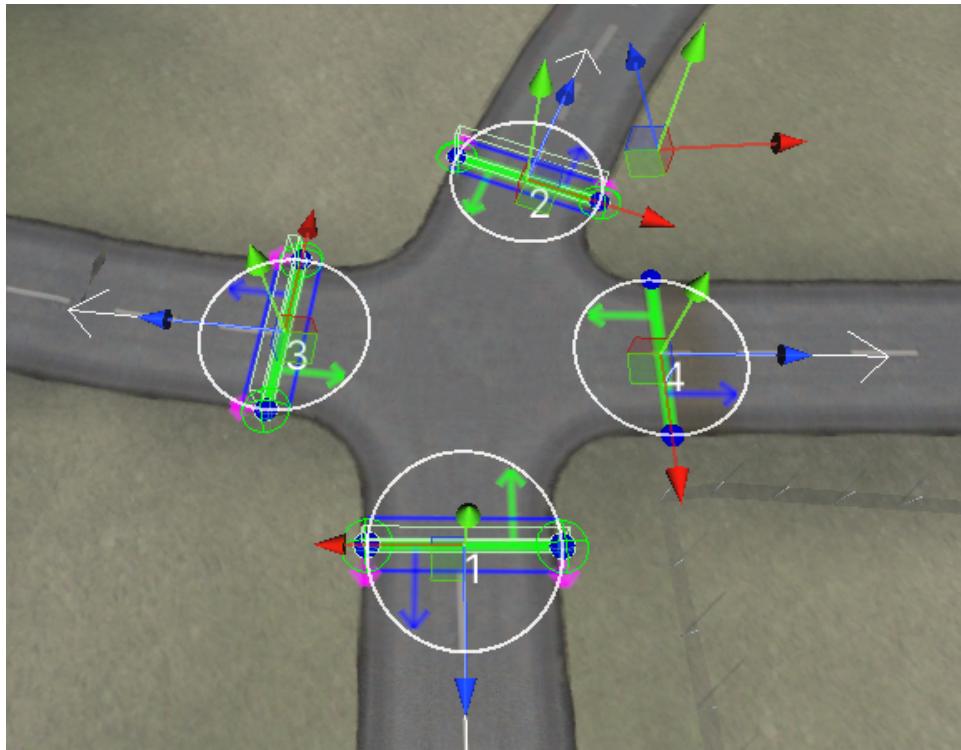
1. Create a *Custom road segment*.



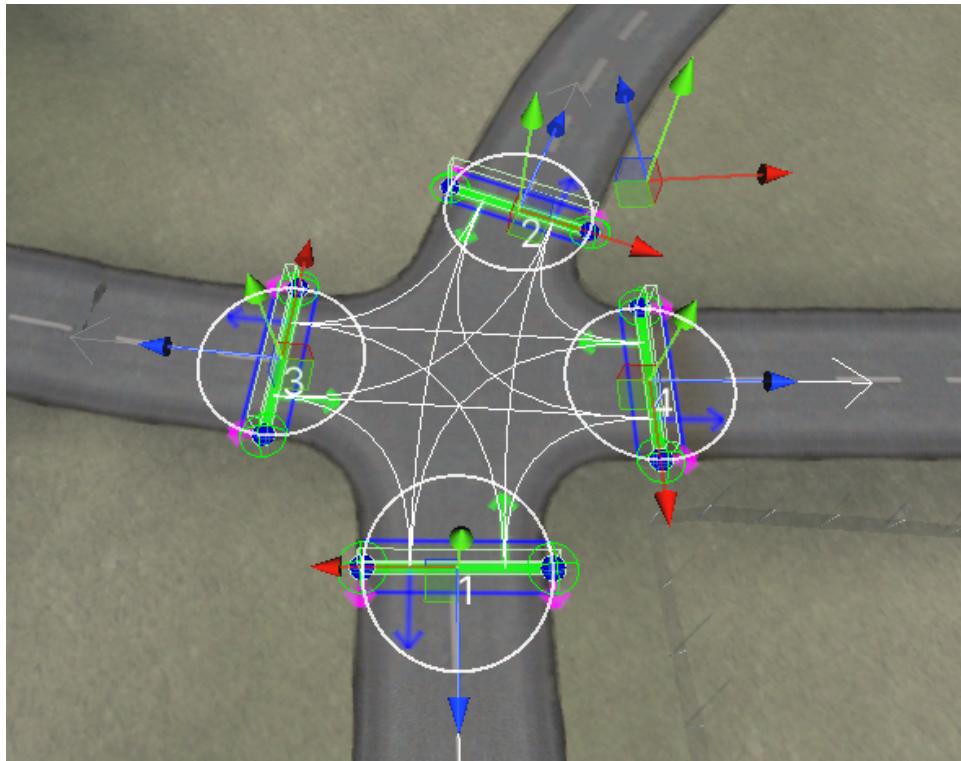
2. Open the *Path Settings* tab & enable the *Auto-Crossroads* option in the *Additional Settings* dialog.



3. Press the *Clear* button to delete existing *paths*.
4. Place *Traffic nodes* at the entrances/exits of the intersection.



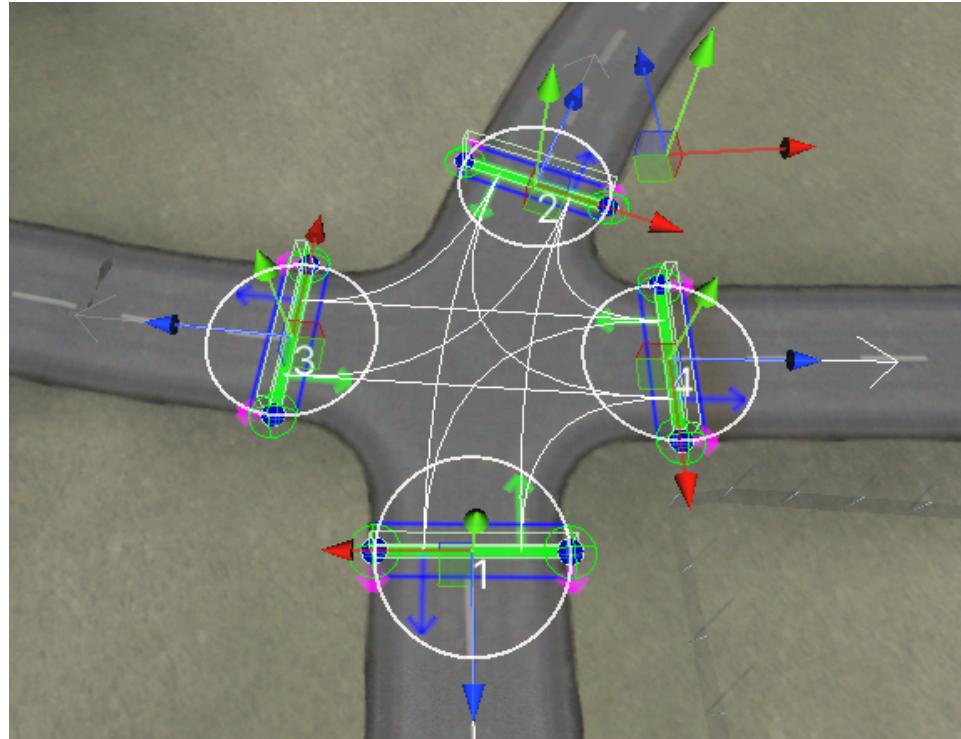
5. Press the *Create* button.



6. If you want to avoid connection for certain *Traffic nodes* select indexes according to scene indexes & press *Add* button, then press *Create* button again.
7. **For example 1-3 & 3-1 nodes.**



8. 1-3 & 3-1 nodes no longer connected.



Example result.

2.4 Baking Info

- Baking data is required to store calculated data to avoid heavy calculations each time. This should be done after each *road change* and before the launch of the scene.
- To validate that all traffic objects are properly configured (use the *TrafficObjectFinder tool* to find the traffic objects with the errors by *InstanceID* shown in the console).

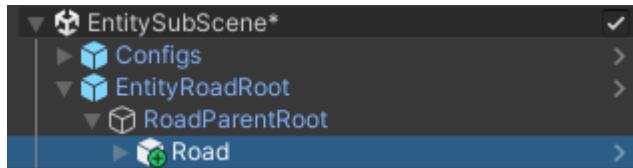
Youtube tutorial.

Baked data:

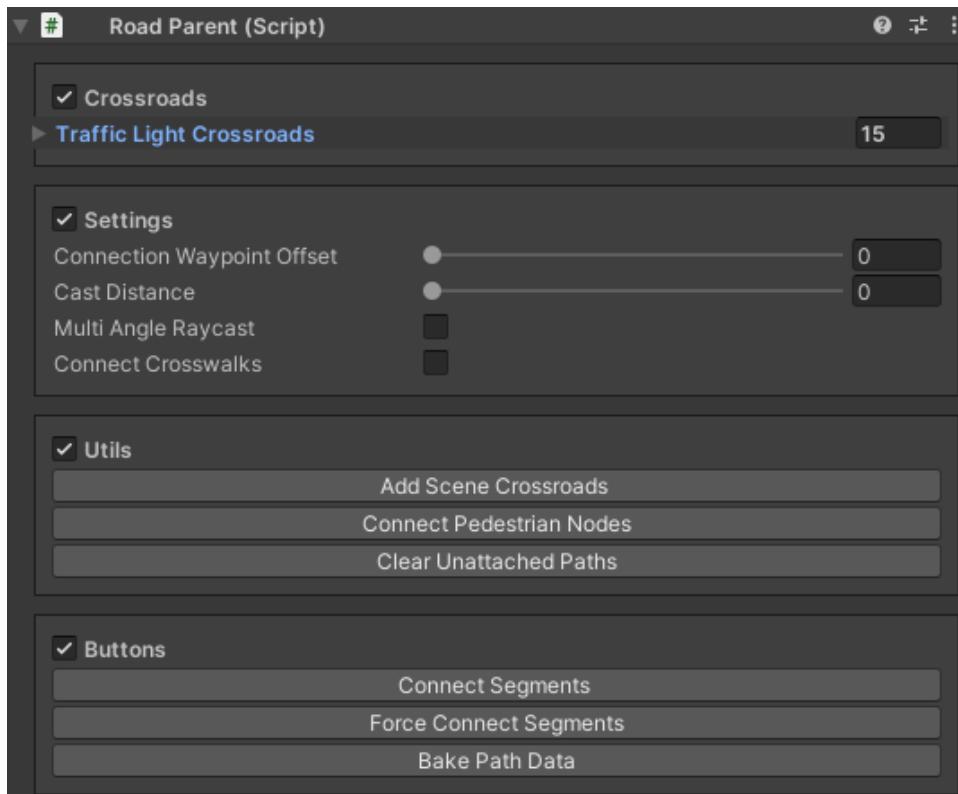
- Road segment data.*
- Path data.*

2.4.1 How To Bake

1. Open *RoadParent*.



Subscene example.



2. Press the *Force connect Segments* button to connect segments (if you are using *auto-connection*, make sure all segments are *on one line*). [optional]
3. Press the *Bake Path Data* button.

2.5 Workflow

2.5.1 Road Editing

Any change of road can take a while if it is done in a *subscene*, so make changes to the road in the *main scene* for *Editor* performance reasons:

Steps

1. Open the *Hub* in the scene.
2. Select the *Entity Subscene Generator*.
3. Click *Move Back* button to move road from subscene to main scene.
4. Make any change in the road.
5. In the *Road Parent*, press the *Force connect* button if you have created *new segments* (make sure all segments are *on one line*) or create paths between segments with *Path creator*.
6. Then, in the *Road Parent*, press the *Bake path data* button to bake & validate the road.
7. If the console shows any errors related to the road, use the *TrafficObjectFinder* tool to find broken road objects & fix them according to the message.
8. Click *Generate* button in the *Entity Subscene Generator*.

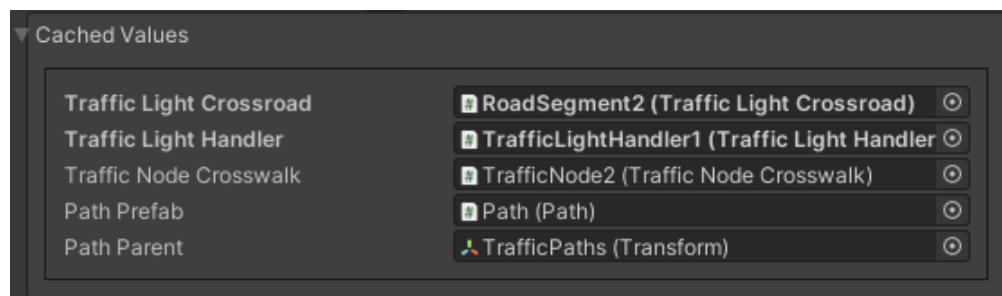
Note: By default, the config from the *main scene* is copied to the *subscene* when *subscene* generation is repeated.

2.6 Traffic Node

Traffic node is a set of traffic node entities that are connected to other traffic node entities by a *path*

2.6.1 Settings

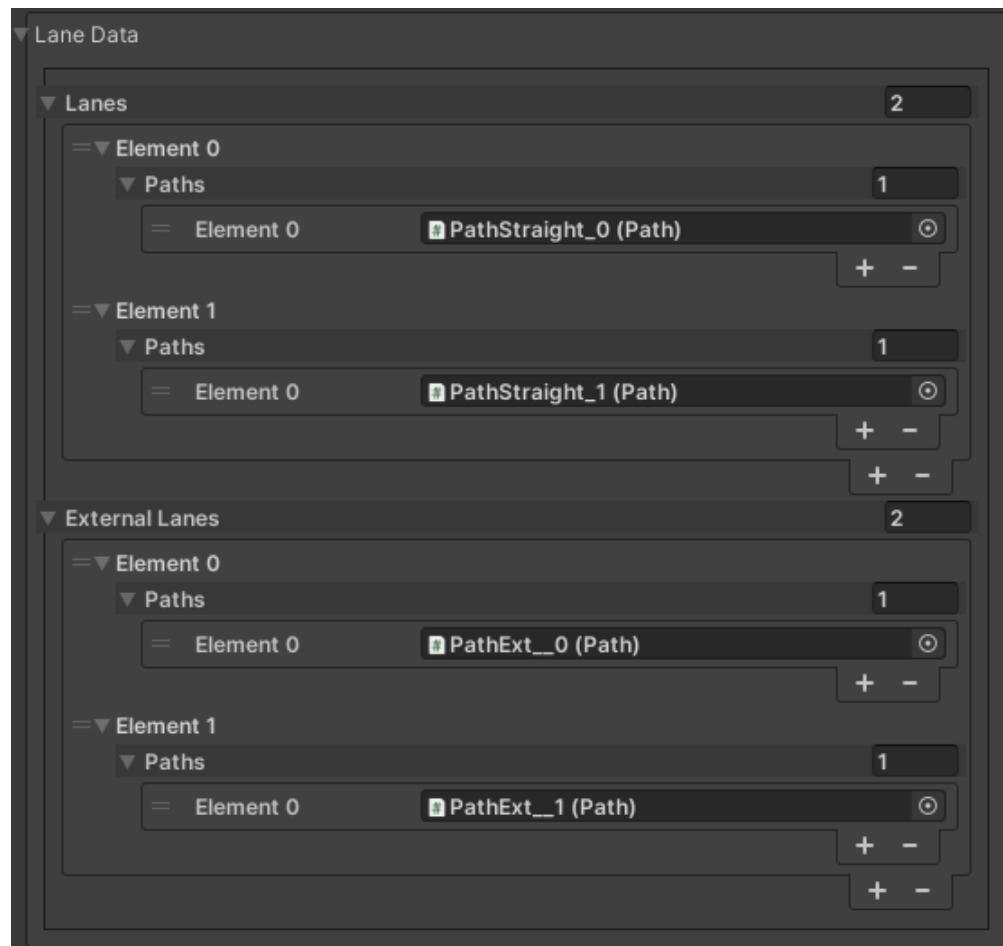
Cached



Traffic light crossroad : the *crossroad* to which the node belongs.

Traffic light handler : traffic light that the traffic node is linked (*TrafficLightHandler*).

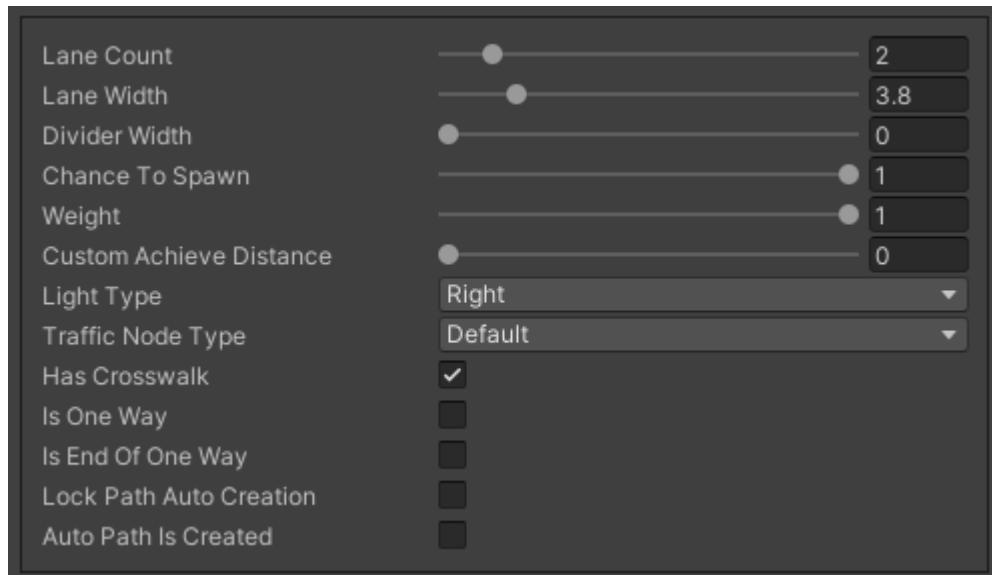
Lane Data



Lanes : *right side lanes* (to connect *TrafficNodes* within a *RoadSegment*).

External lanes : *left side lanes* (to connect nodes in *external RoadSegments*) (additonal info).

Settings



Lane count : number of lanes.

Lane width : lane width.

Divider width : divider line width.

Chance to spawn : chance of the vehicle spawning in the node.

Weight : weight of the node for route selection by traffic.

Custom achieve distance : custom distance to achieve a node (if 0 value default value will be taken).

Capacity : capacity of the nodes (used for parking, bus station, etc...).

Light type:

- **Right** : right-hand lanes have traffic lights.
- **Left** : left-hand lanes have traffic lights.
- **Right and left** : right and left lanes have traffic lights.

Traffic node type:

- **Default**
- **Parking** : node where cars are *parked* (read more *parking states*).
- **Traffic public stop** : node where *public traffic* stops to pick up passengers.
- **Destroy vehicle** : node where the vehicle entity is destroyed (useful for nodes outside the map).
- **Traffic area** : *TrafficArea node*.
- **Idle** : node where the vehicle is idling.

Has crosswalk : quick on/off crosswalk option for pedestrians.

Is one way : all lanes are one-way traffic lanes (*more info*).

Is end of one way : node ends one-way traffic for this *RoadSegment* (*more info*).

Lock path auto creation : on/off prevent auto path creation (*more info*).

Auto path is created : auto path is created ([more info](#)).

Buttons

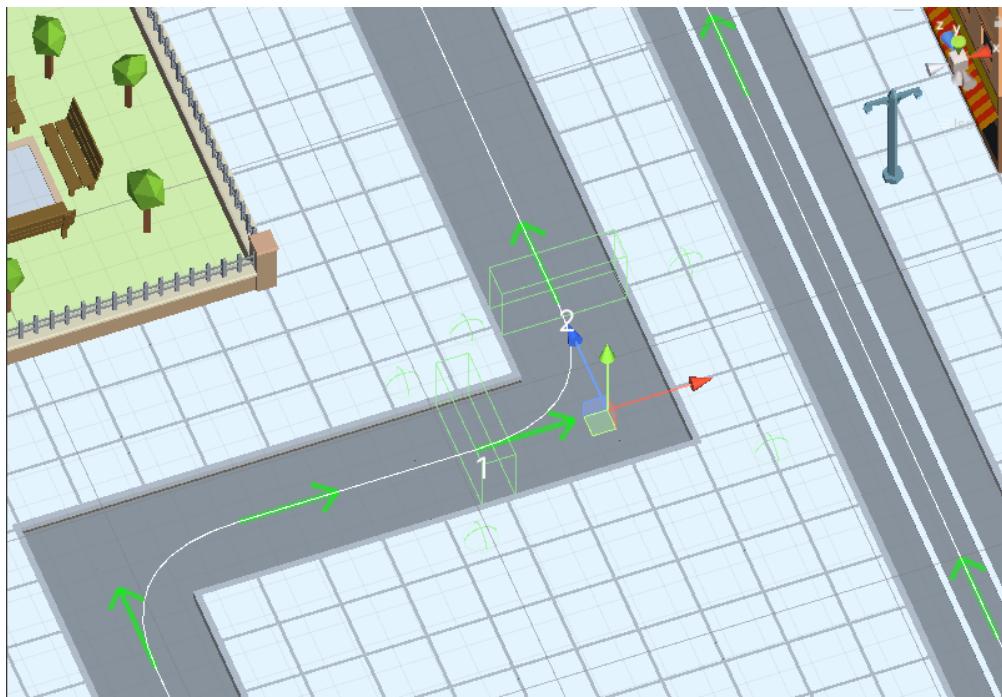
Connect : node will try to *connect* to other nodes if no external paths are created yet.

Force connect : node will try to *connect* to other nodes whether it is *connected* now or not (except *Lock path auto creation* option is enabled).

Resize : resize *collider* of node.

2.6.2 OneWay Node Info

Oneway node description example:



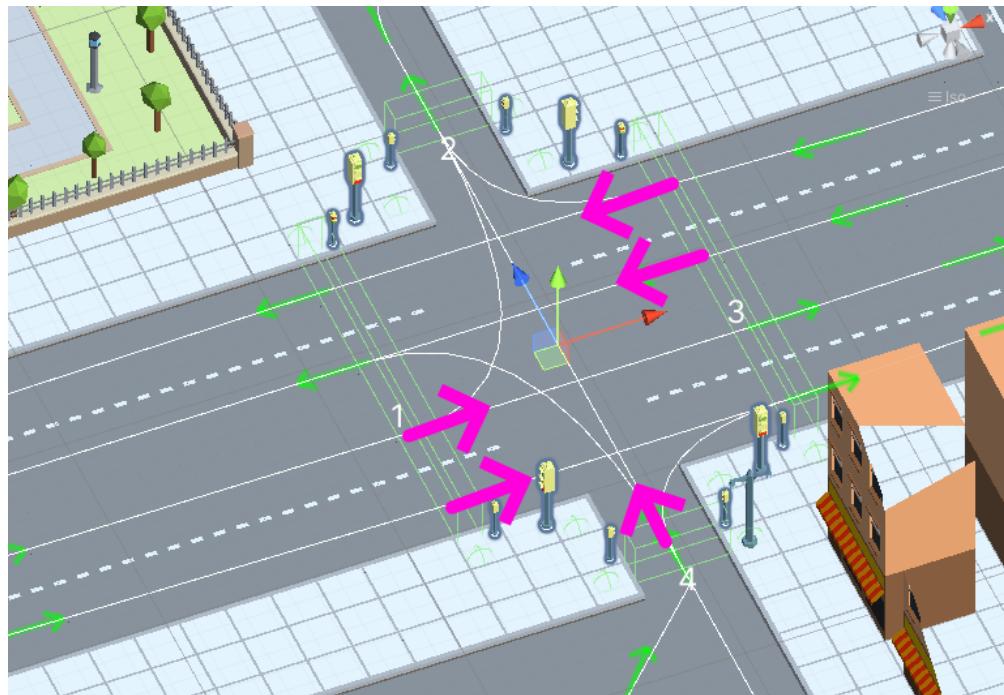
Node example key features:

- **Node 1:**
 - Is one way [**enabled**]
 - Source path is in the : [**Lanes**]
 - External Lanes [**Always empty**]
- **Node 2:**
 - Is one way [**enabled**]
 - Is end one way [**enabled**]
 - Source path is in the : [**External Lanes**]
 - Lanes [**Always empty**]

2.6.3 Direction Connection Info

Rightside Lanes

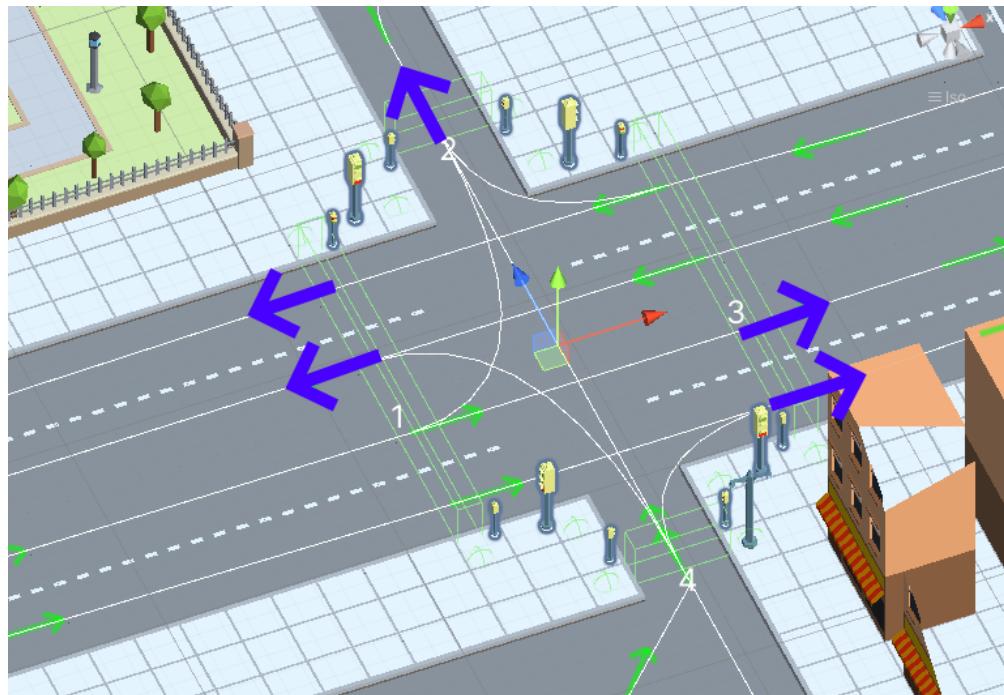
Rightside lanes (default lanes) connect *TrafficNodes* within a *RoadSegment*.



Rightside lanes example.

Leftside Lanes

Leftside lanes (external lanes) connect *TrafficNodes* in external *RoadSegments* (*external connection example*).

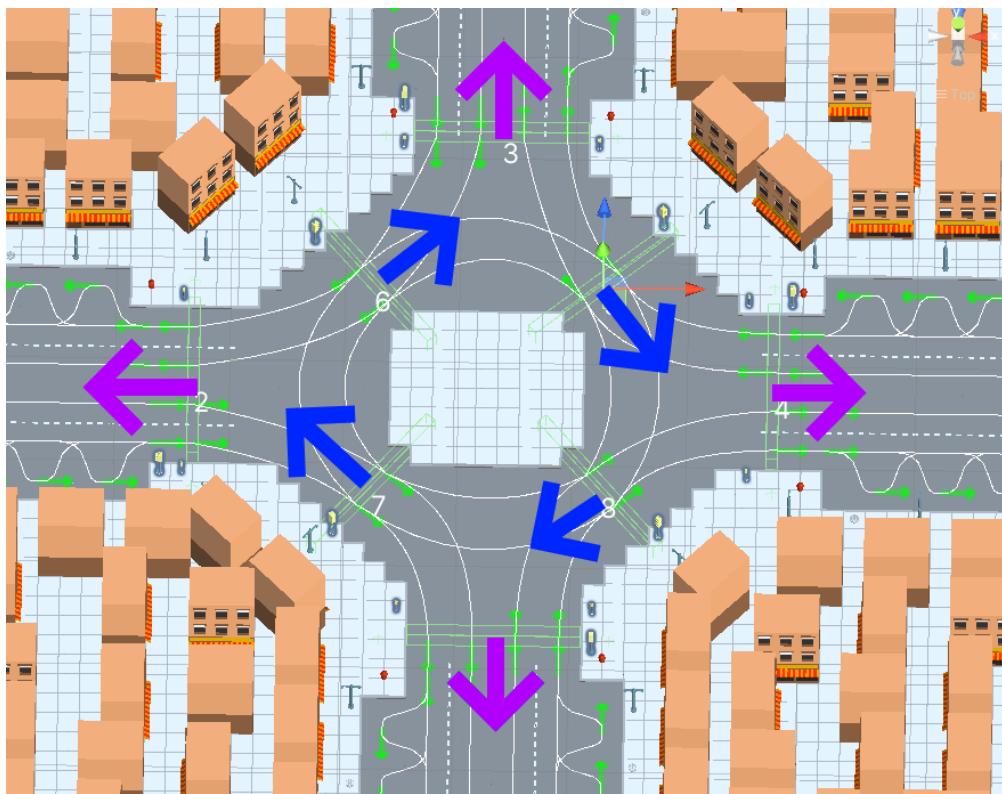


Leftside lanes example.

Warning: Intersecting *External paths* should be replaced by a separate *segment* to *bake* the intersection of the paths.

Node Rotation

Direction of each *TrafficNode* must be opposite to the center of the segment



Example description:

- The arrow represents the forward rotation of the *node*.
- Purple arrows the direction of the outer *nodes* of the *segment*.
- Blue arrows the direction of the internal *segment oneway nodes*.

2.6.4 Auto-path Connection

- To quickly create connections between *RoadSegments* on the same line, the *Auto-Path* connection is used.
- If the *segments* are not on the same line you should to create another *Custom straight road segment* or *Custom segment* between them and do the same connection.
- You can also manually create paths between *segments* using the *PathCreator tool*.

How To Use

- To activate auto-connection paths for all nodes you can in *RoadParent* by pressing *Connect* button.
- Each time you create a new *RoadSegment*, press *Force connect* in *RoadParent*, then *Bake Path Data (baking info)*.

Note:

- To prevent auto-path connection for the selected *TrafficNode* enable **Lock path auto creation** in the *settings* of the node.

- Each *TrafficNode* has a *box collider* whose size is calculated based on the number of lanes, their width, and the type of lanes (*one-way* or not).
- Make sure that the *direction of the node* is set correctly.



Auto path connection example.

2.6.5 CullState Info

States

- **Culled** : entity not available for spawning.
- **CloseToCamera** : entity available for spawn.
- **InVisionOfCamera** : entity available for spawn only during the initial scene start.

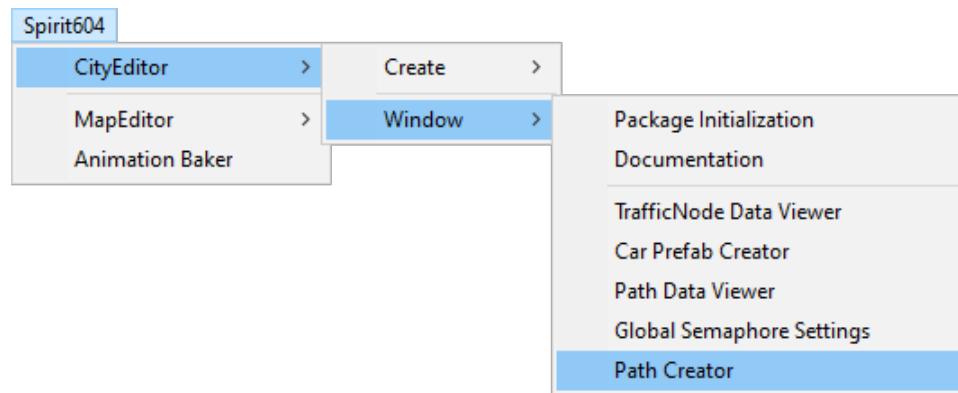
2.7 Path Creator

Path Creator is a tool for quickly creating *paths* between *traffic nodes*.

2.7.1 How To Use

1. Open the tool from the *Unity* toolbar:

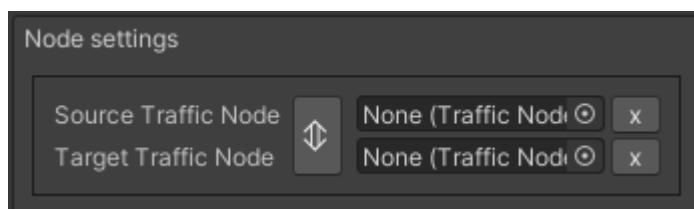
Spirit604/CityEditor/Window/Path Creator



2. Select the *source traffic node* and *target traffic node* in the scene (*example*).
3. Customize new *path settings*.
4. Select the desired *Connection Type* (*connection settings*).
5. Customize *Source & Target connection side*, so that the path is positioned correctly (*example*) (*connection settings*).
6. Click the *Create* button.
7. *Customize* the created *paths*.

2.7.2 Settings

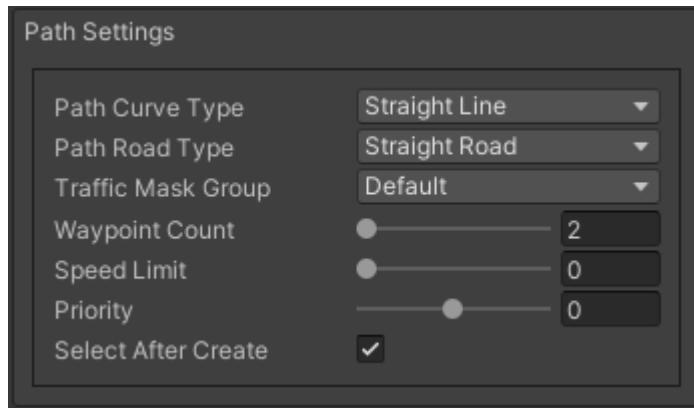
Node Settings



Source Traffic Node : source *traffic node*.

Target Traffic Node : target *traffic node*.

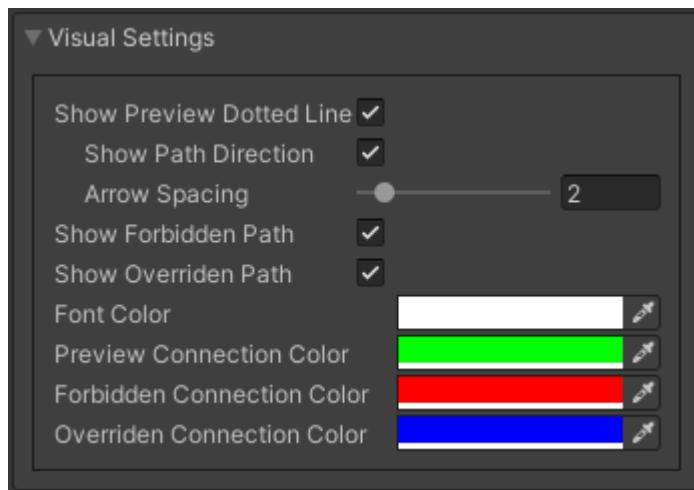
Path Settings



Path settings.

Select after create : the path will be selected in the inspector after creation.

Visual Settings



Show preview dotted line: on/off connection line in the scene.

- **Show path direction** : on/off arrows of the connection line.
- **Arrow spacing** : arrow spacing.

Show forbidden path : on/off display of forbidden connection line.

Show overridden path : on/off display of overridden connection line (if disabled preview color will be taken).

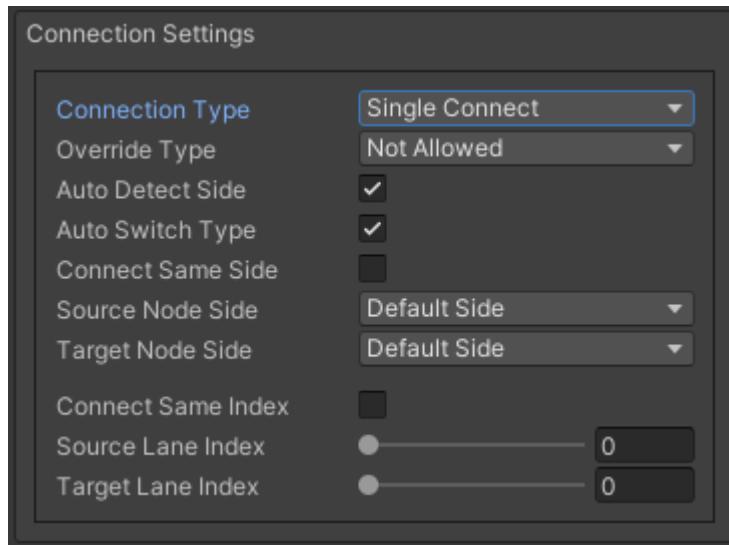
Font color : font color of traffic node index gizmos.

Preview connection color : preview connection line color.

Forbidden connection color : forbidden connection line color.

Overridden connection color : overridden connection line color.

Connection Settings



Connection type:

- **Single connect** : only 1 *path* is created.
- **One direction connect** : *paths* of all lanes are created for one side.
- **Two direction connect** : *paths* of all lanes are created for two sides [New].

Override type:

- **Not allowed** : *path* will be created only if the *path* has not been created before.
- **Allowed** : *path* will be overwritten if created earlier.

Auto detect side : when selecting nodes, the selected *sides* will be automatically detected.

Auto switch type : automatically switch connection type after selecting nodes depending on connection sides.

Connect same side : target *side* will be the same as source *side*.

Source node side :

- **Default side** : selected *right side* point in the source *traffic node*.
- **External side** : selected *left side* point in the source *traffic node*.

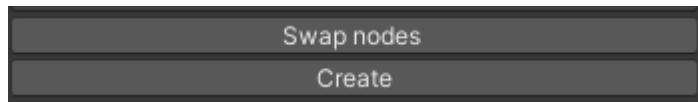
Target node side :

- **Default side** : selected *right side* point in the target *traffic node*.
- **External side** : selected *left side* point in the target *traffic node*.

Single connect setting :

- **Connect same index** : target index will be the same as source index.
- **Source lane index** : source lane index.
- **Target lane index** : connected lane index.

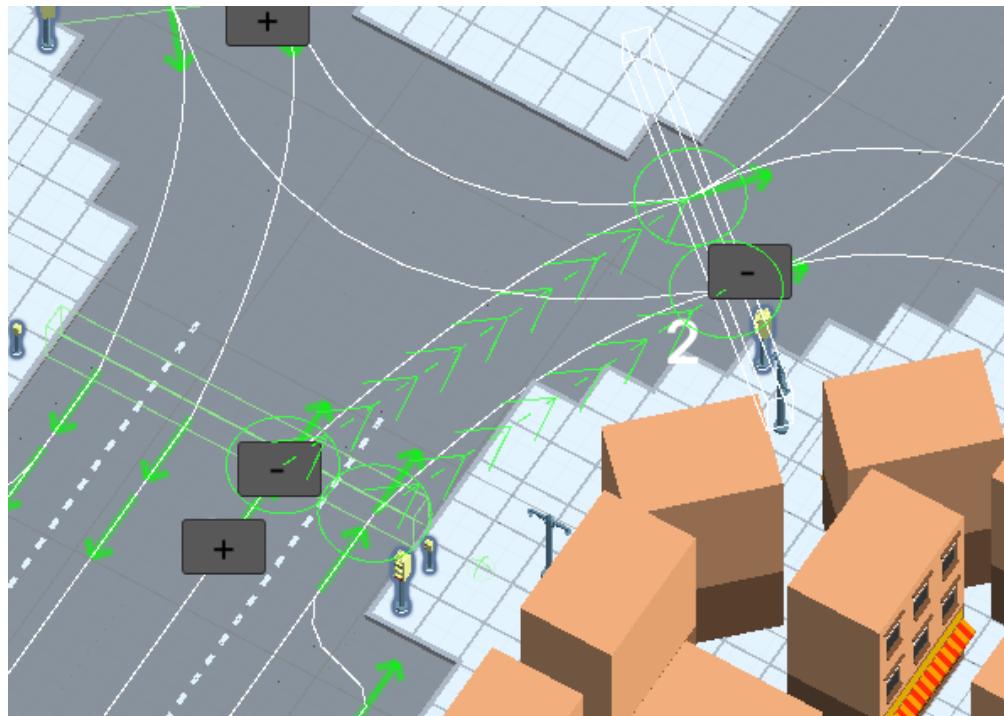
Buttons



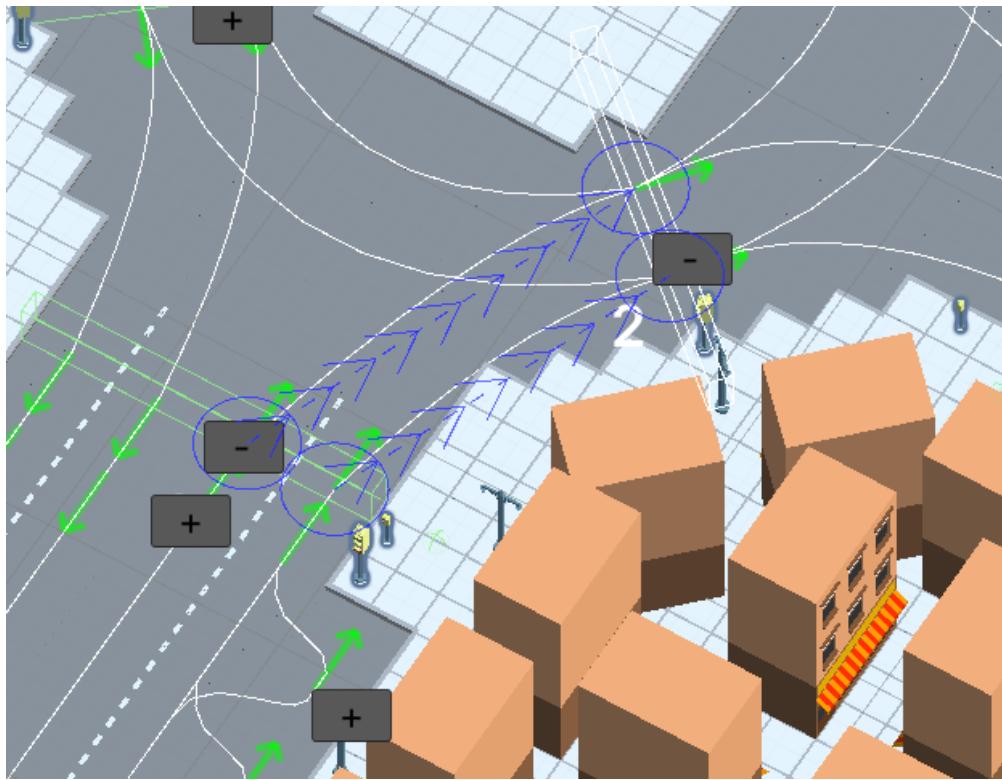
Swap nodes : swap source and target node.

Create : create available paths.

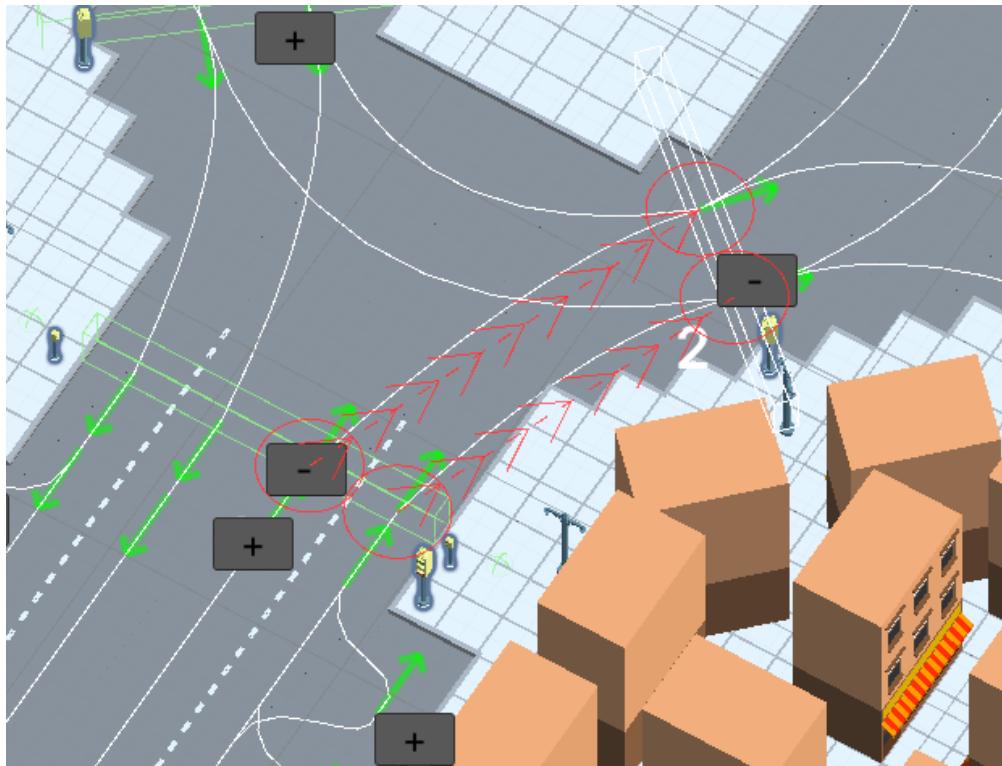
2.7.3 Examples



Connection available example (allow override path enabled, show overriden path disabled).



Connection available example (allow override path enabled, show overridden path enabled, paths already created for these nodes & overwritable).



Connection forbidden example (path already created for these nodes & can't be overwritten).

2.8 Path

Path is a set of *waypoints* along which *vehicles* travel and is a connection between *traffic nodes*.

2.8.1 How To Create

Paths are generated in *RoadSegment* using *RoadSegmentCreator* or you can add them to existing *Road Segments* using the *Path Creator tool*

2.8.2 How To Customize

Settings

There are two ways to edit:

1. Directly in the *Inspector*.
2. Using the *Advanced Settings Window*.

Nodes

How To Move

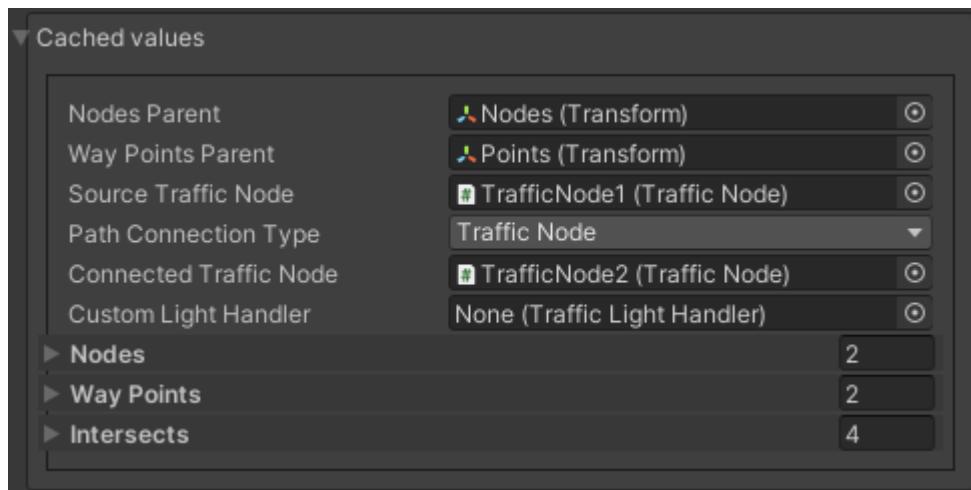
1. Enable *Show handles* option.
2. Drag the desired path handles.

How Add & Remove

1. Enable *Show handles* option.
2. Enable *Show edit buttons* option.
3. Press the + button to add or press the x to remove node.

2.8.3 Path Settings

Cached values



Source traffic node : source node traffic from which the path starts.

Path connection type:

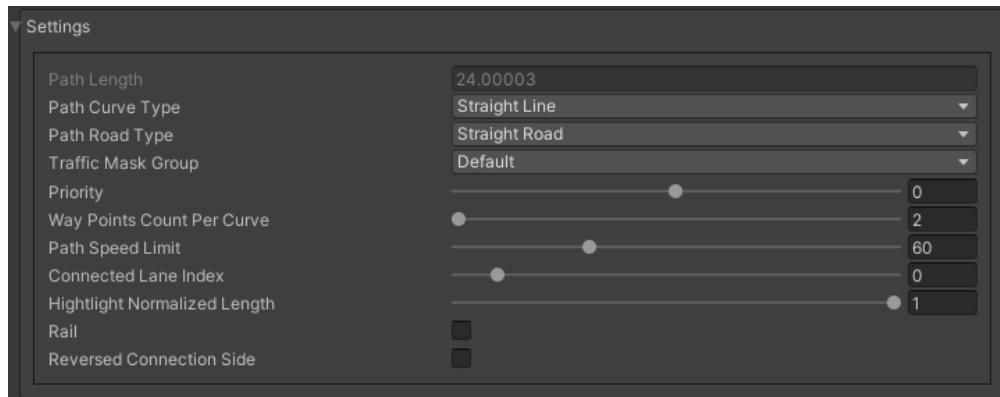
- **Traffic node :**
 - **Connected traffic node** : *connected traffic node*.
- **Path point :**
 - **Connected path** : connected path in the *custom point*.

Nodes : key nodes for creating curves (Bezier).

Waypoints : *waypoints* of path.

Intersects : intersection points with other paths (*baked data*) (*example*).

Settings



Path length : path length (*baked value*).

Path curve type:

- **Straight line** : default point to point line.
- **Bezier cube** : bezier cube curved line.
- **Bezier quad** : bezier quad curved line.

Path road type:

- **Straight road** : is used to automatically calculate lane changing by traffic.
- **Turn road**

Traffic mask group: *group types* of traffic vehicles that can go on this path.

Priority : order of crossing intersected paths (vehicle with the higher priority gets through first).

Waypoints count per curve : number of waypoints in the curve segment.

Path speed limit : speed limit for the entire route

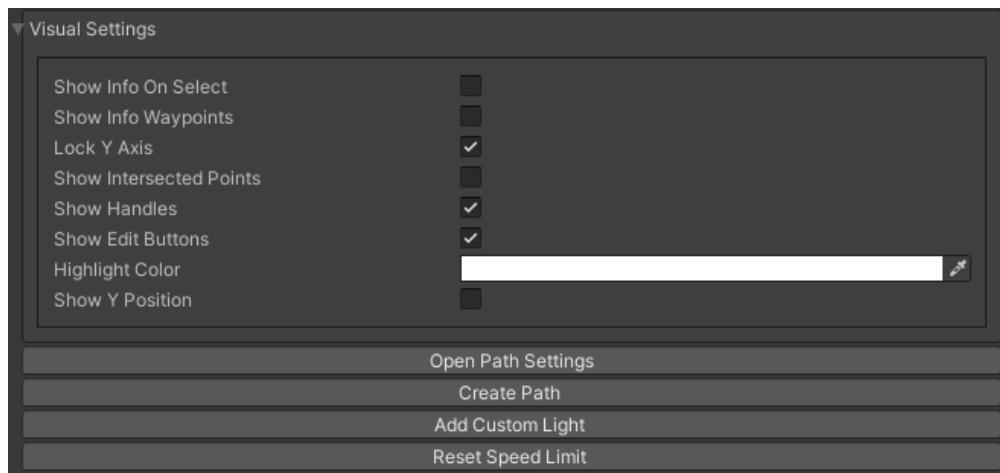
Connected lane index : connected lane index.

Highlight normalized length : normalized length of the highlighted path (editor only).

Rail : cars entering this path will move with a *rail movement*.

Reversed connection side : path will be connected to the *opposite side of the node*.

Visual Settings



Common settings

Show info on select : shared parameter between paths instances that automatically enables *Show info waypoints* on selecting new path.

Show info waypoints : show info of waypoints in the scene.

Lock Y axis : lock Y-axis for position handles of nodes.

Show intersected points : show intersected points in the scene.

Show handles : show position handles for nodes.

Show edit buttons : show edit buttons for path (add/remove nodes).

Highlight color : highlight color of the path.

Show Y position : show Y-position of nodes.

Curved settings

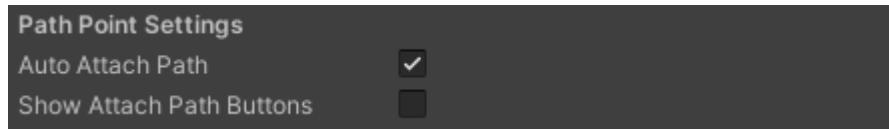


Draw tangent : on/off tangents in the scene.

Clamp tangent : two position handles of tangent will move together.

Convert to StraightLine : converts a *Curved line* into a *Straight line*.

Path point settings



Auto attach path : automatically attach the last node of the source to the connected path.

Show attach path buttons : on/off connect path buttons in the scene.

Buttons

Open path settings : open *Path Settings Window*.

Create path : generation and positioning of waypoints based on the position of the nodes and the selected curve.

Add custom light : custom *TrafficLightHandler* will be added to the path.

Reset speed limit : each waypoint will be assigned a common speed limit of path.

Traffic Node connection

Default connection between *traffic nodes* in *road segment*, used in most cases.

Path Point connection

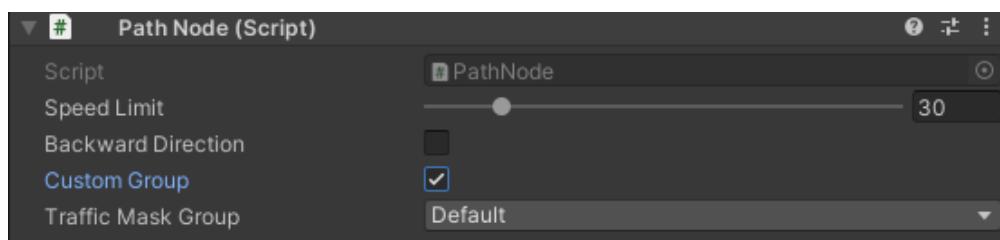
Is used to connect one path to another in a path in a custom point (generated by parking builder *example*).

How To Use

1. Select source path in the scene.
2. Select *Path connection type* to *Path Point* in the inspector.
3. Enable *Show attach path buttons*.
4. Select desired path.
5. Customize position handles of source path nodes.

2.8.4 Waypoint Info

The path is made up of these waypoints, which direct each *vehicle*.



Speed limit : the maximum speed of the vehicle when it reaches this waypoint.

Backward direction : when this option is activated, the vehicle will reverse (*test scene*).

Custom group : override *traffic group* for the current path node.

Note: You can debug the group nodes [here](#).

2.8.5 Traffic Group Info

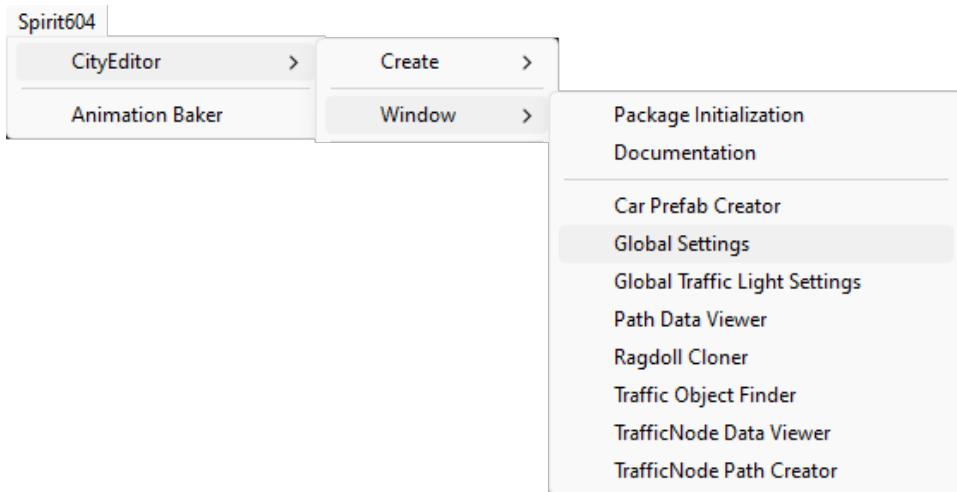
- Each path contains a mask which, depending on the *mask type*, contains a group of the selected *TrafficGroupType*.
- A vehicle that has the correct *TrafficGroupType* can be driven or spawned on this path.
- If the vehicle enters the forbidden *path* or *path node*, it will automatically try to change the current lane (*test scene*).

Note:

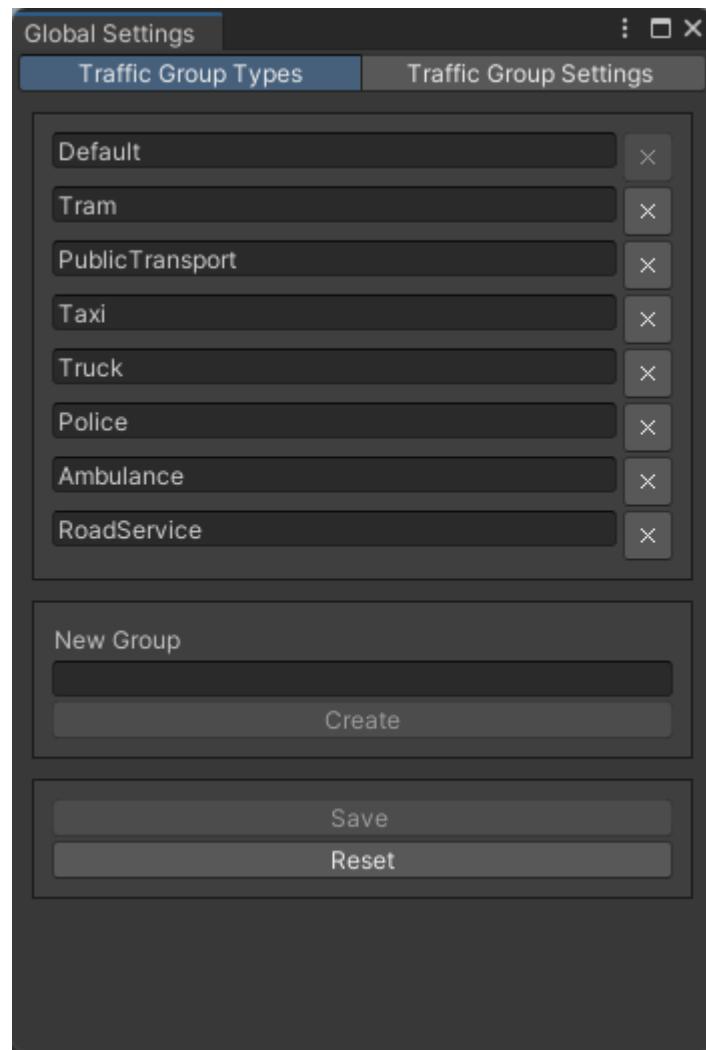
- You can change the available types in the *Global Settings* or in the enum file:
* `TrafficGroupType.cs`
 - You can debug the path group [here](#).
-

How To Change

1. Open the *Global Settings* window.



2. Select the *Traffic Group Types* tab.

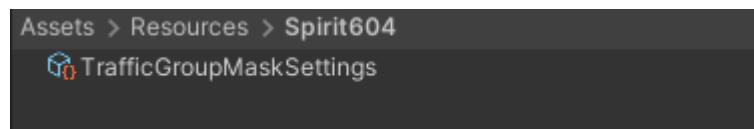


3. Rename, remove, add groups as you wish.
4. After making any changes, click on the *Save* button.

Traffic Group Settings

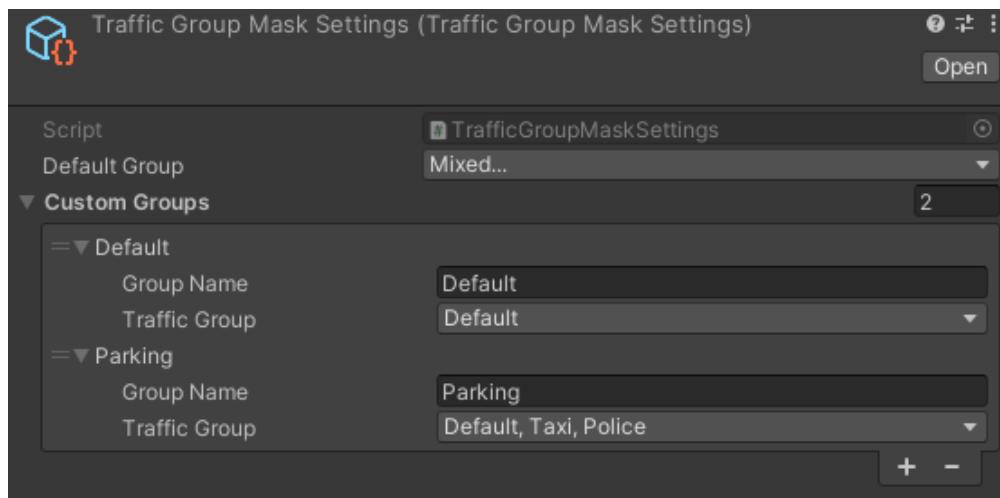
How To Create

Settings automatically created if missing in project path *Assets/Resources/Spirit604/TrafficGroupMaskSettings.asset*.



Settings

Contains the types of default group and user-created groups.



Mask Types

Default

Default group of the *TrafficGroupType* that defined in the *settings*.

Allowed

User-selected *TrafficGroupType* types that can go through the path.

Forbidden

User-selected *TrafficGroupType* types that are forbidden the path (not selected in the list is allowed).

Custom Group

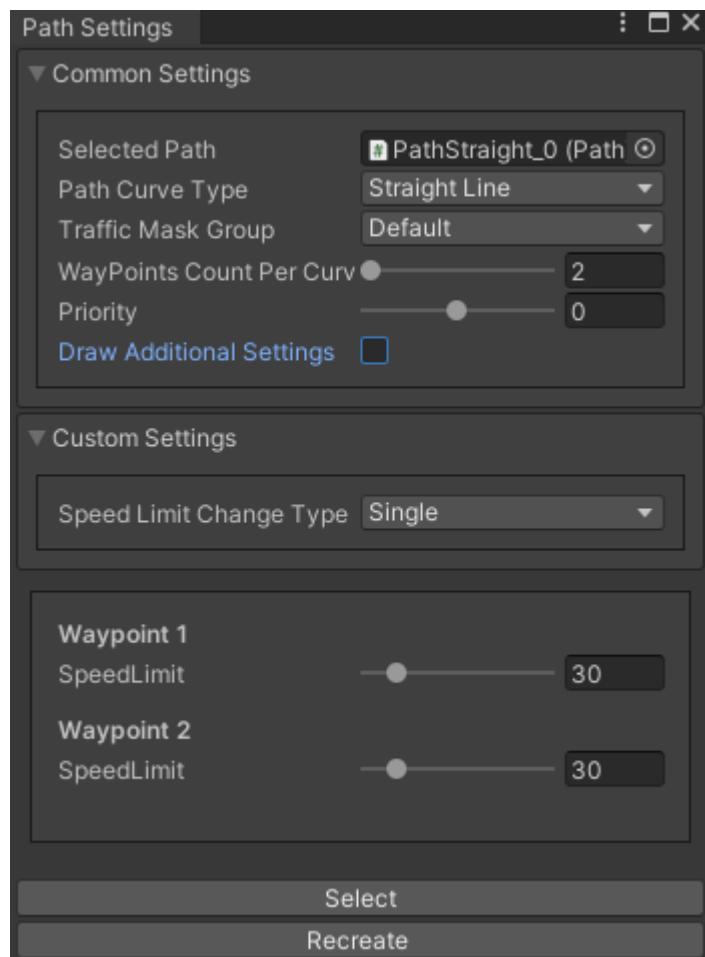
Custom group of *TrafficGroupType* that defined in the *settings*.

2.8.6 Advanced Settings Window

How To Open

1. Select the *Path*.
2. Press *Open Path Settings* button in the inspector.

Settings



Common settings

Path curve type.

Traffic mask group.

Waypoints count per curve.

Priority.

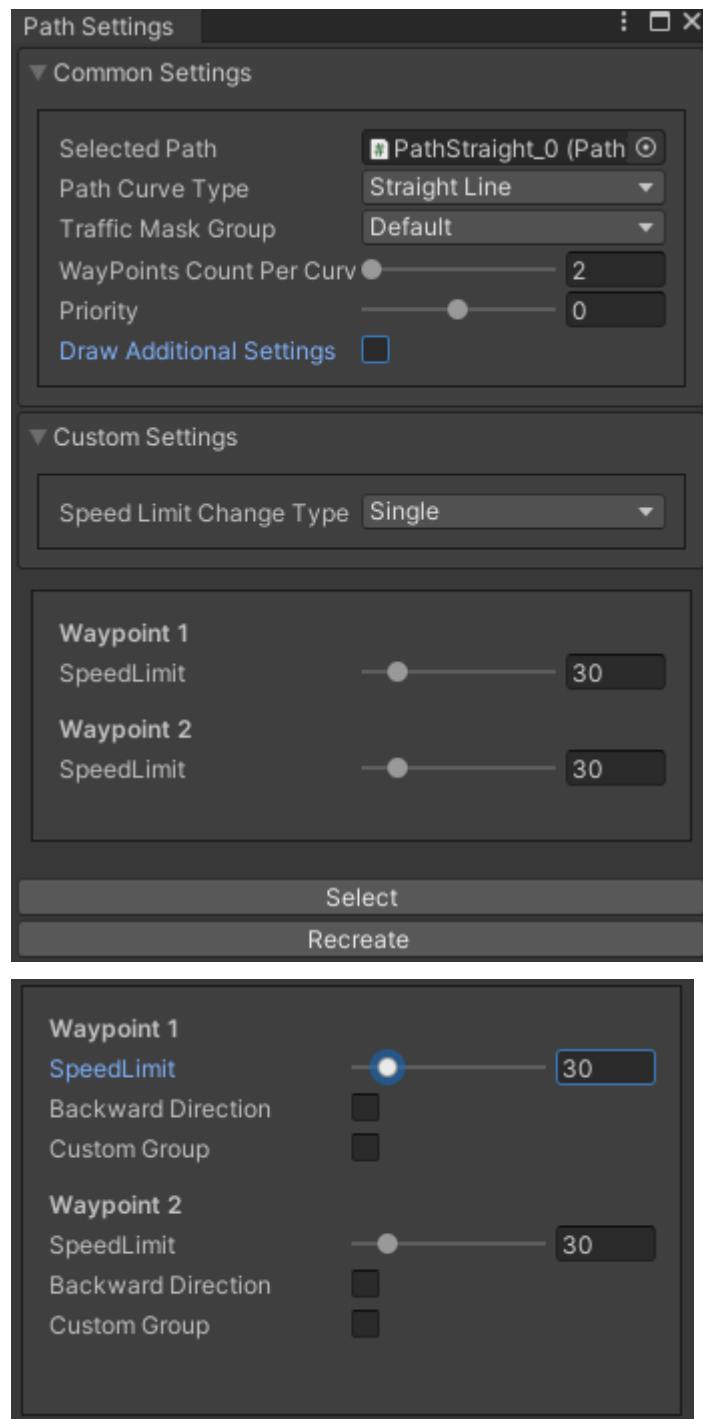
Draw additional settings : displays additional settings for each waypoint (*Backward Movement, Traffic Group*).

Custom settings

Speedlimit change type :

Single

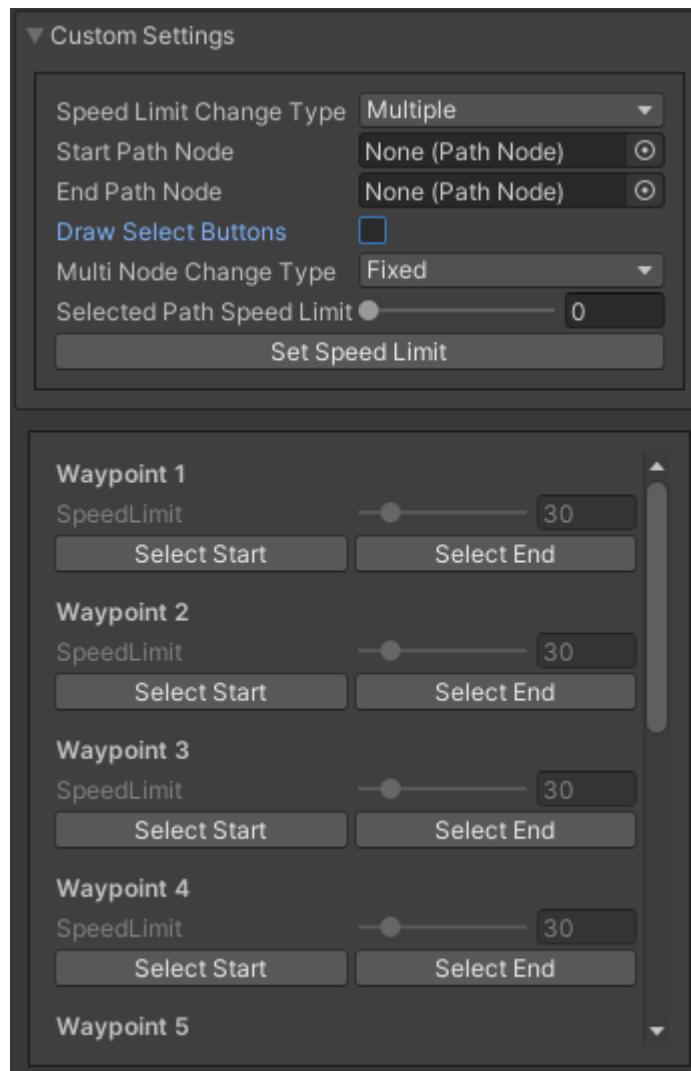
Single - change each waypoint one at a time.



Draw additional settings enabled.

Multiple

Multiple - the speed limit will be changed on the selected section.

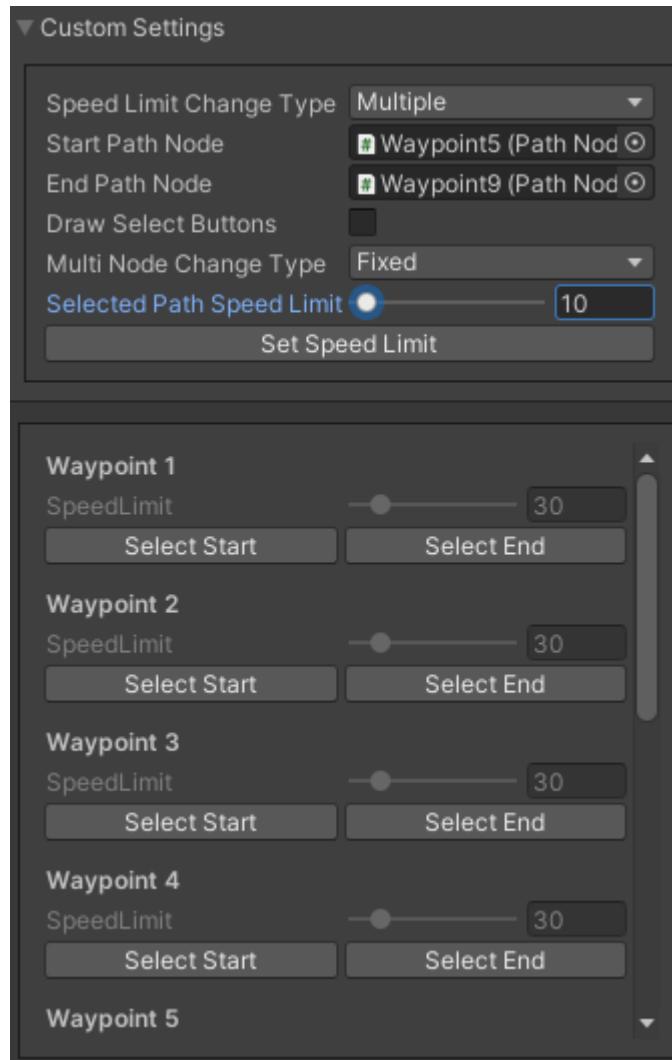


Multiple node change type:

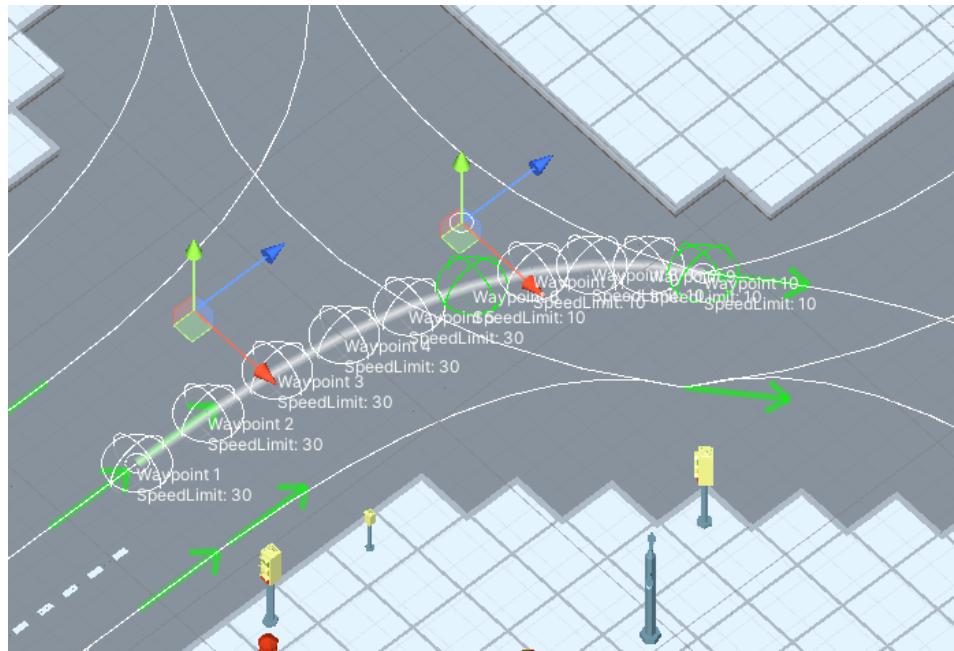
- **Fixed** : all waypoints change speed limit.
- **Interpolate**
 - [speed will be interpolated from the beginning of the section to the end.]
 - **Interpolate type :**
 - * **Node index** : speed is interpolated relative to the waypoint index.
 - * **Distance** : speed is interpolated relative to the position of the waypoint.
 - **Start speed limit** : initial speed limit of the section.
 - **End speed limit** : end speed limit of the section.

How to use:

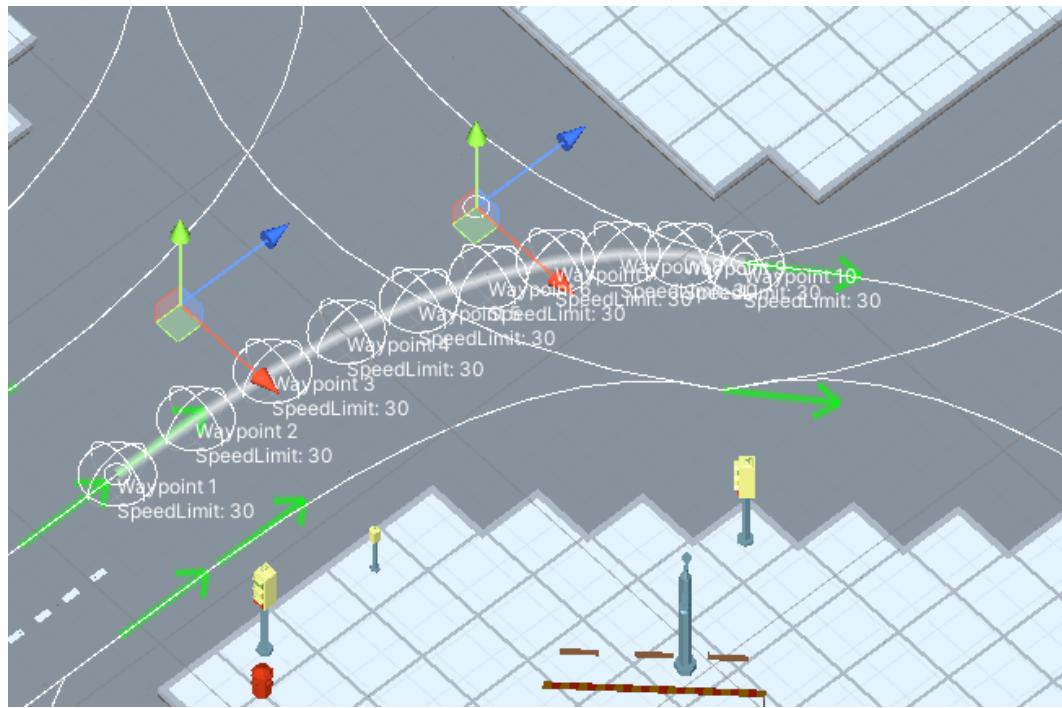
- Select the start and end of the section in the window or switch on the *Draw Select Buttons* and select the start (*S*) and end (*E*) in the scene.
- Set the parameter **Selected Path Speed Limit** to the value you need.



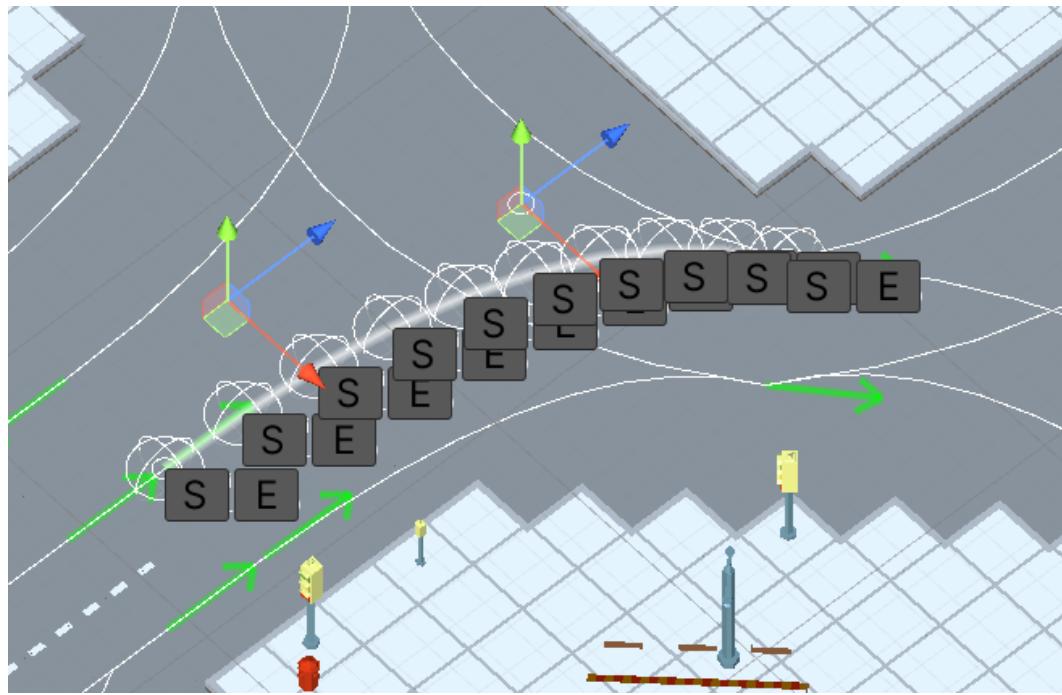
- Click **Set Speed Limit**.



Result.



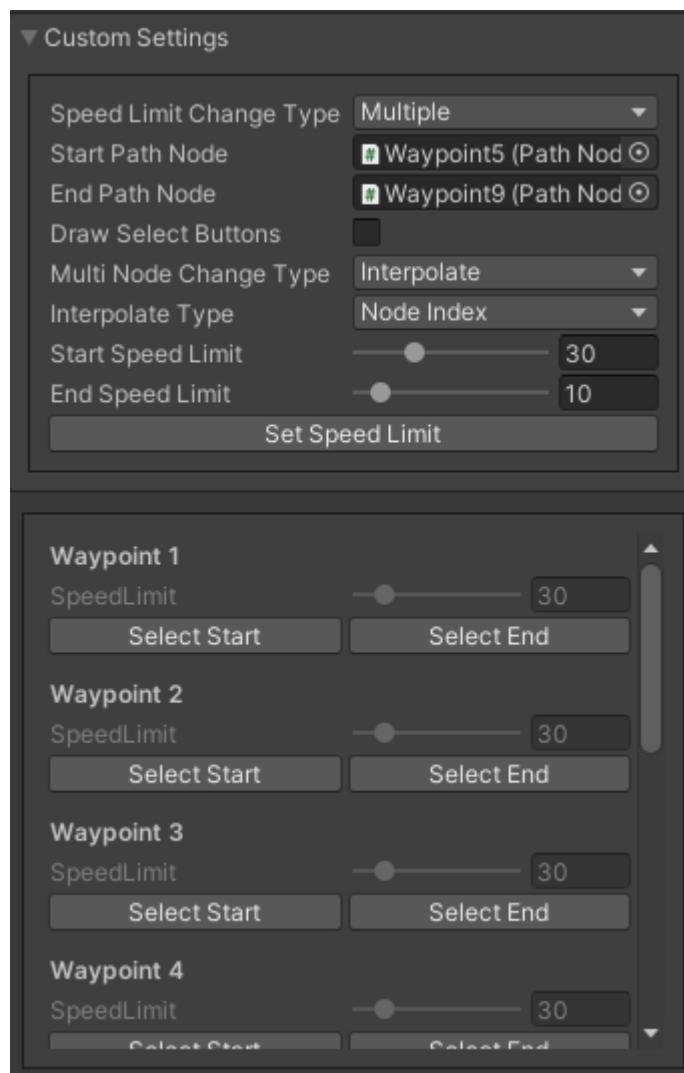
Source path example.



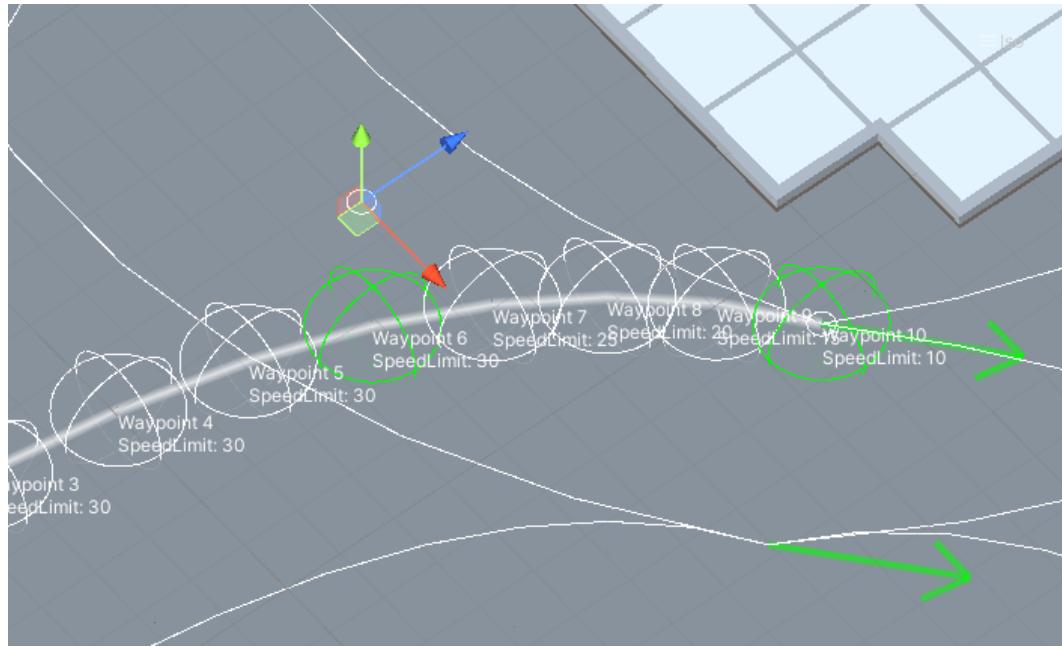
Draw Select Buttons enabled "S" (start) "E" (End) example.



Path section selected (green circles start & end of section) example.



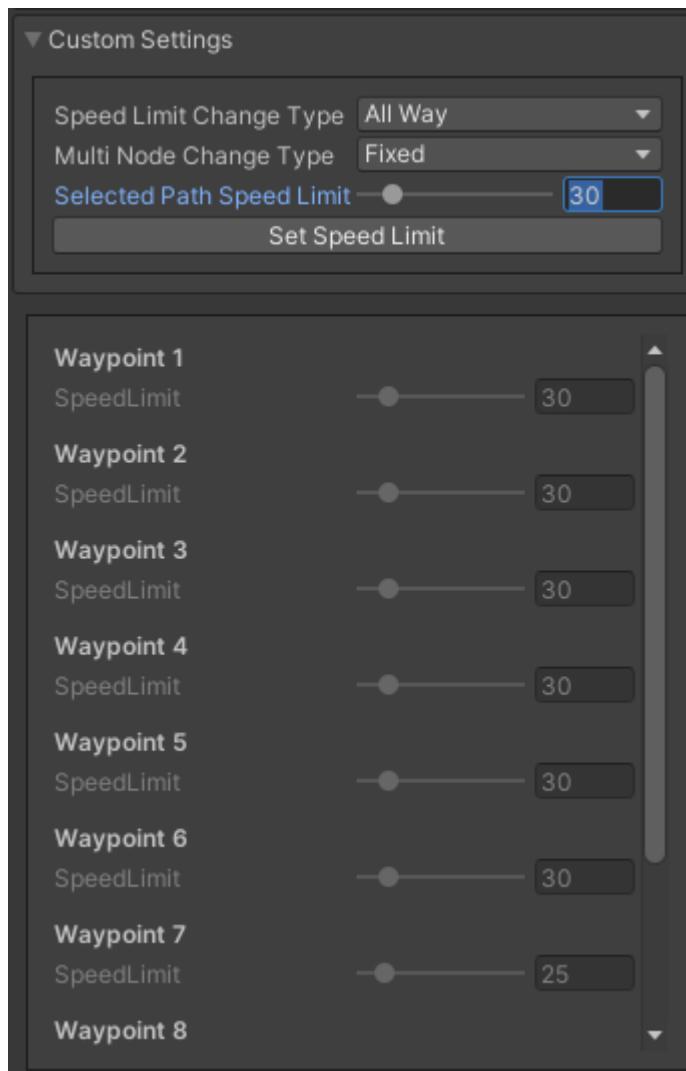
Interpolating settings example.



Interpolating result.

All way

All way - all path waypoints will change the speed limit according to the set options.



Multiple node change type:

- **Fixed** : all waypoints change speed limit.
- **Interpolate**
 - [speed will be interpolated from the beginning of the section to the end.]
 - **Interpolate type :**
 - * **Node index** : speed is interpolated regarding to the waypoint index.
 - * **Distance** : speed is interpolated regarding the position of the waypoint.
 - **Start speed limit** : initial speed limit of the section.
 - **End speed limit** : end speed limit of the section.

How to use:

- Set the parameter **Selected Path Speed Limit** to the value you need.



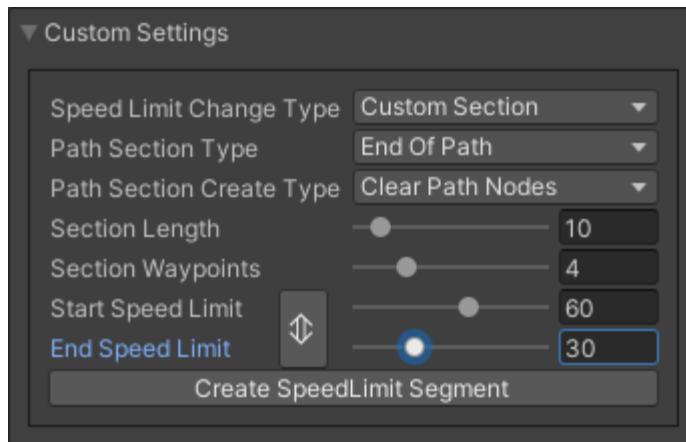
- Click the *Set Speed Limit* button.



Result.

Custom section

Custom section - the section with the custom speed is automatically generated depending on the parameters.



Path section type:

- **Start of path** : section will be created at the beginning of the path.
- **End of path** : section will be created at the end of the path.
- **All path** : section will be generated all along the path.

Path section create type:

- **Clear path nodes** : waypoints will be generated anew each time a section is created.
- **Use exist nodes** : existing waypoints will be used for the section.

Section length : length of the created section.

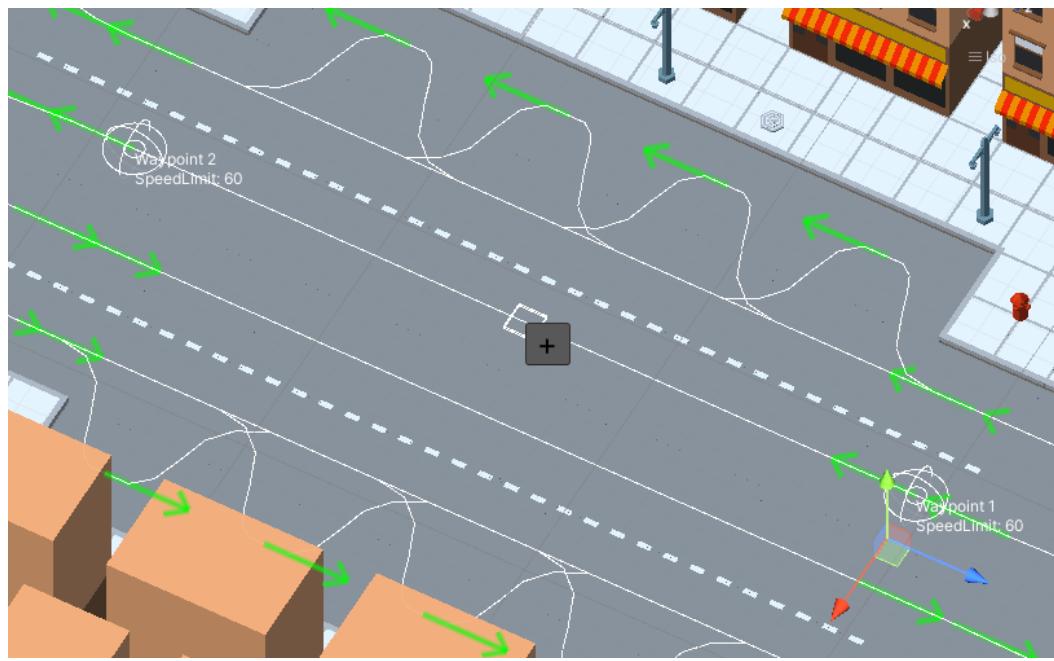
Section waypoints : number of waypoints of the created section.

Start speed limit : initial speed of the section.

End speed limit : end speed of the section.

How to use:

- Set all parameters.
- Click *Create SpeedLimit Segment*.



Source path.



Result.

2.8.7 Baking Info

Each *path* bakes the data to speed up the entity conversion. How to *bake*.

Baked Data:

- *Path Length* (is used to calculate obstacles on the path by *TrafficCarObstacleSystem*).
- *Intersection data (segment baking info)* (used by *TrafficCarObstacleSystem* to order the intersection of intersecting paths).

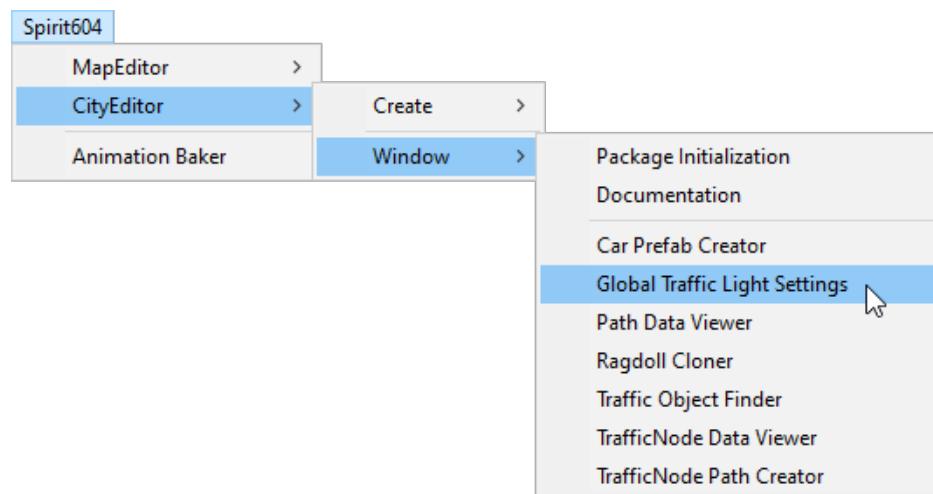
2.9 Traffic Light

Youtube tutorial.

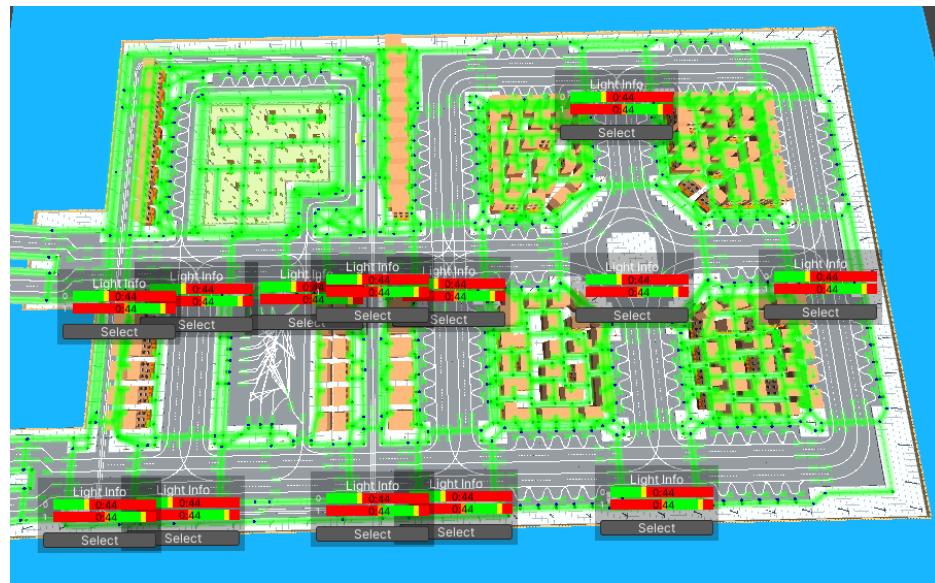
2.9.1 How To Customize City Crossroads

1. Open *Global Light Settings*.

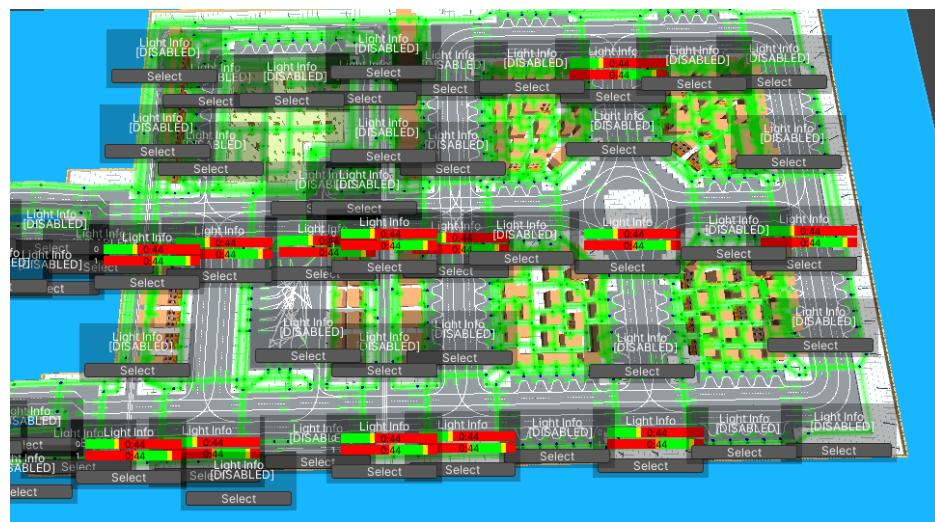
Spirit604/CityEditor/Window/Global Traffic Light Settings



2. Enable *Show world info*.



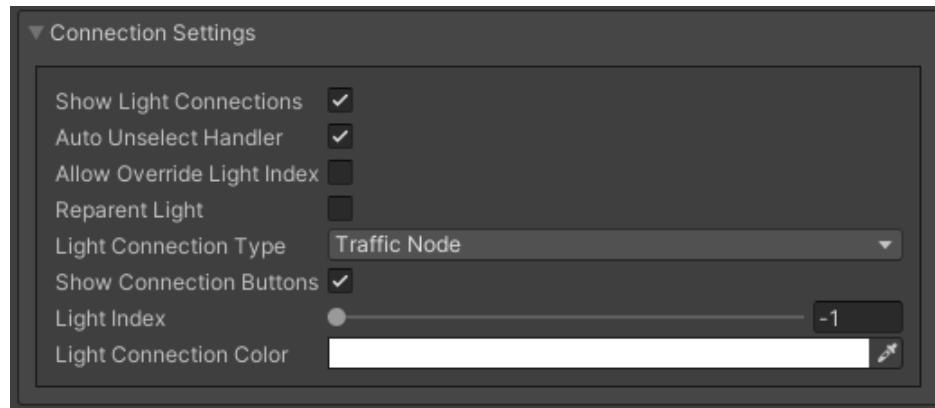
3. You can now quickly view and adjust the timelines of all the crossroads.
4. Enable *Show disabled Lights* to see crossroads with traffic lights switched off.



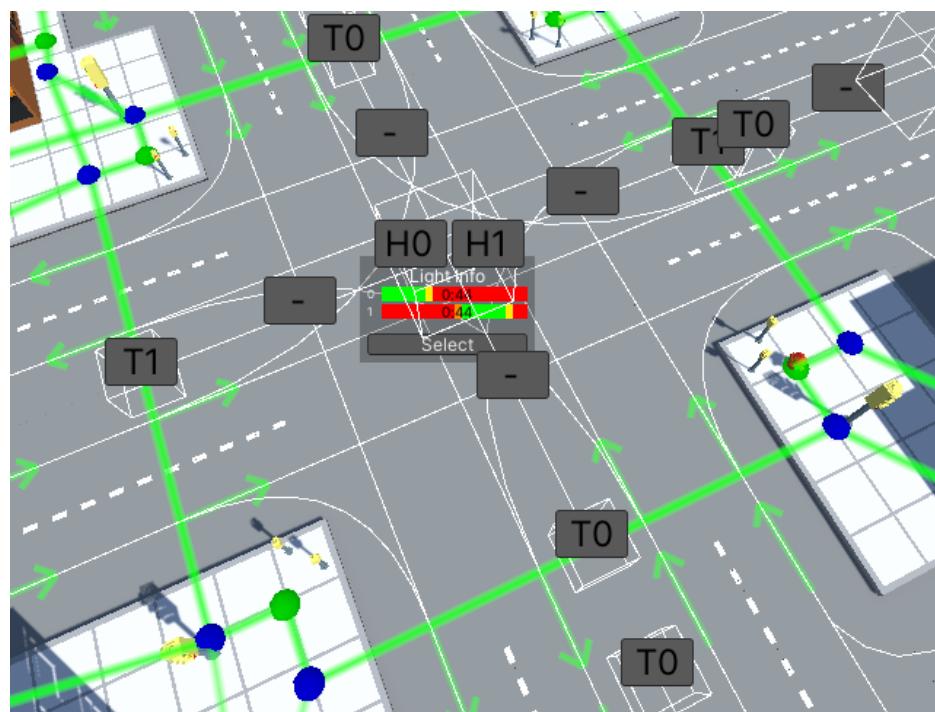
5. Select the desired crossroad and press *Select*.
6. In the *TrafficLightCrossroad* component, you can now set the timings.

2.9.2 How To Assign Light

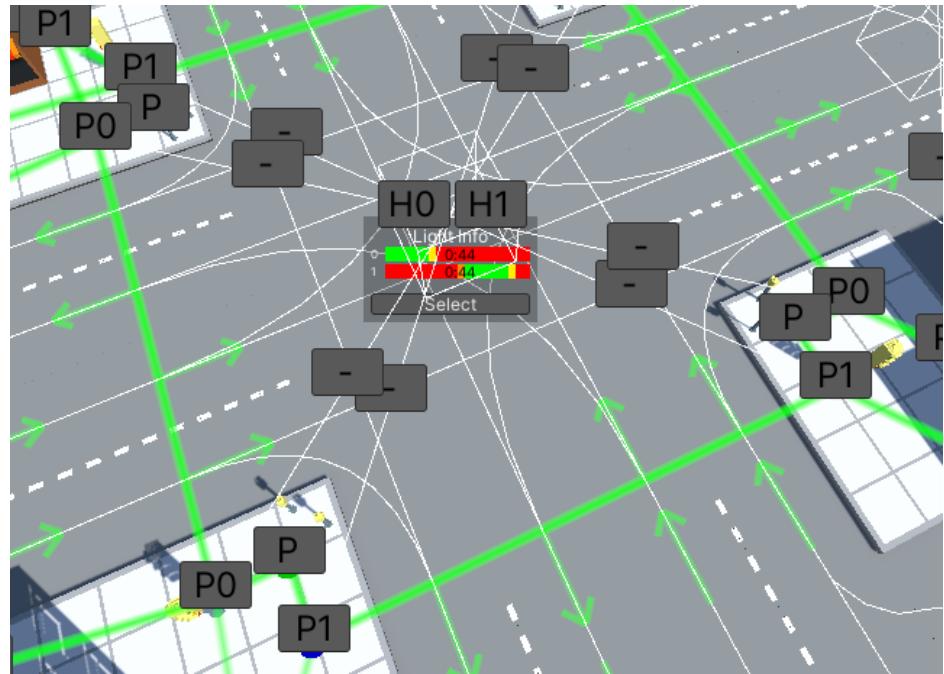
1. Open *Global Light Settings*.
2. Enable Show light connections.



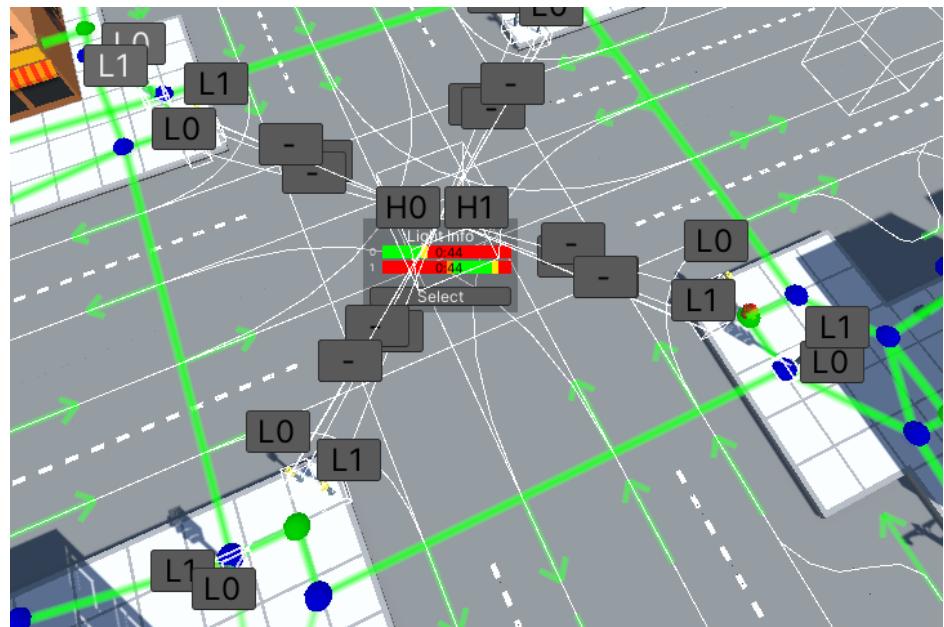
3. Select Light connection type for example *Traffic node*.



4. Select H0 or H1 depending on the desired light index.
5. Next, select the desired T (*TrafficNode*).
6. Now, the selected *TrafficNode* will have the selected *TrafficLightHandler*.
7. In the same way, you can assign *PedestrianNodes* and *Light objects* by changing the Light connection type.



Pedestrian node connection example.



Light object connection example.

2.9.3 Global Lights Settings

Window for quick display of crossroad timings and for linking the traffic lights to different entities.

How To Use

Read more [here](#).

Settings

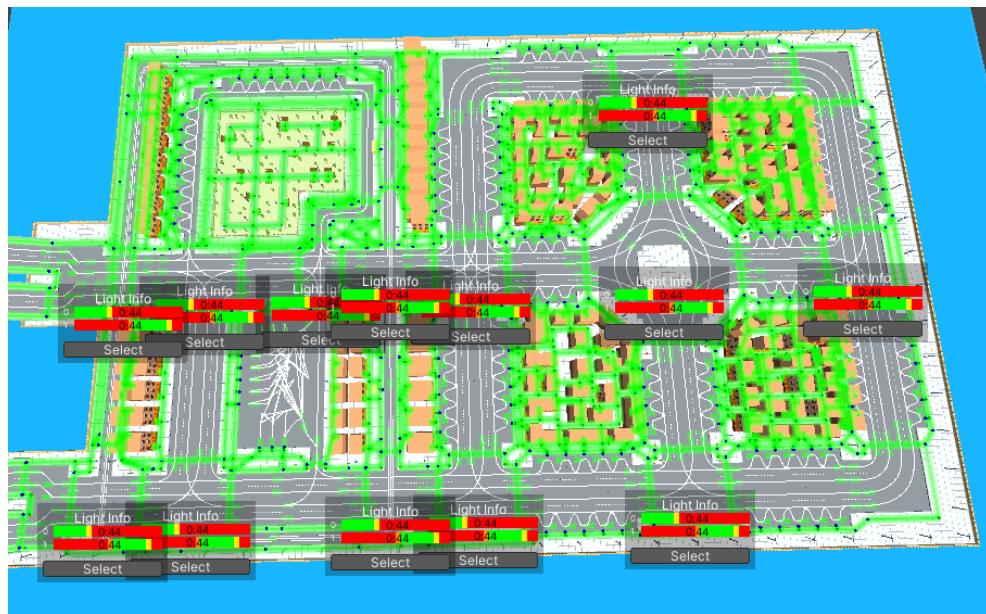


Common Settings

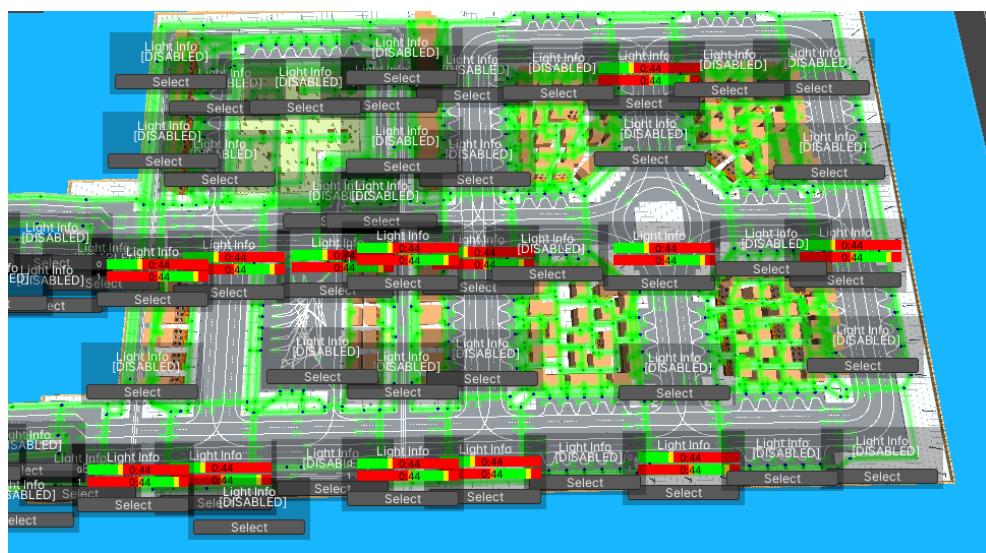
Focus on select : move the *SceneView* camera to the selected traffic light crossroad when you select.

Show world info : show enabled traffic light data in the scene (*example*).

Show disabled lights : show all traffic light data (include disabled) in the scene (*example*).

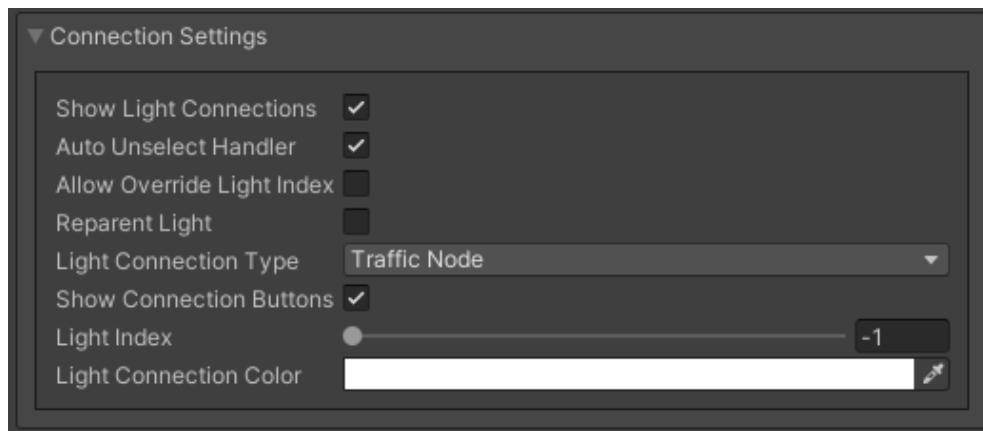


Scene light info example.



Scene light info (include disabled) example.

Connection Settings



Show light connections: on/off light connections in the scene.

Auto unselect handler: auto unselect *TrafficLightHandler* when connecting *TrafficLightHandler* traffic lights to any object.

Allow override light index: allow index traffic light overrides in traffic *light objects*.

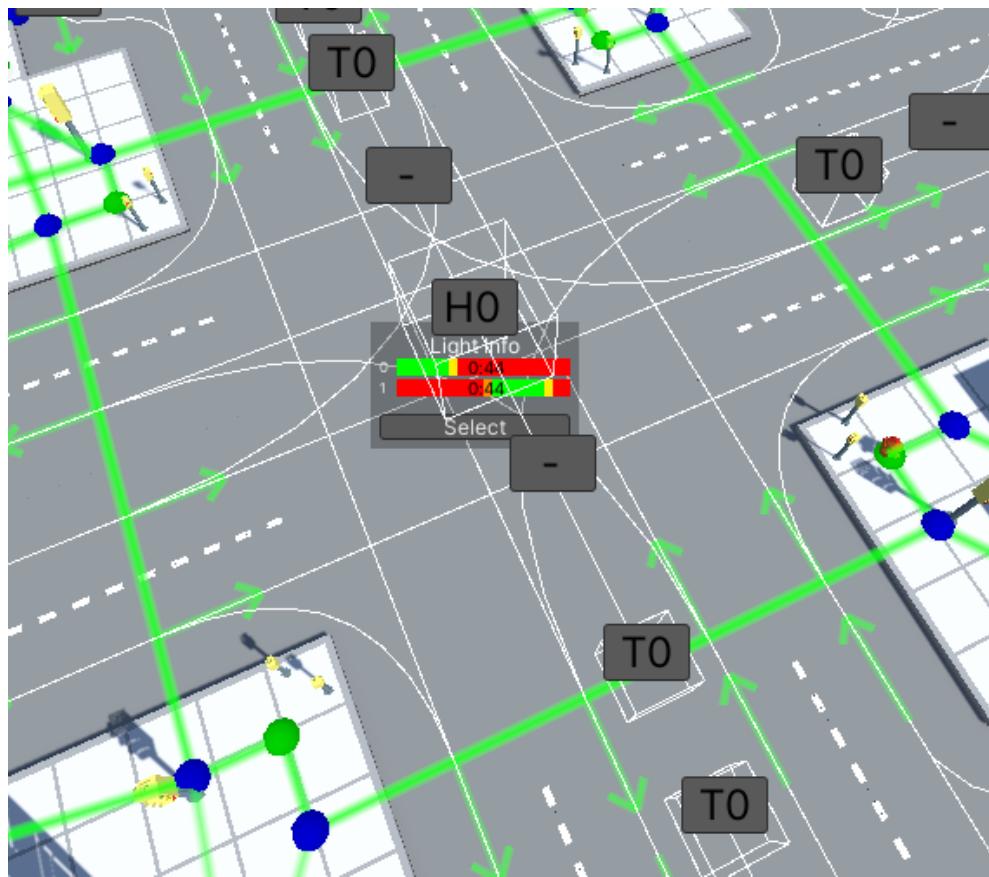
Reparent light: traffic *light object* will be a child of the connected crossroad.

Light connection type :

- **All** : show all connection types.
- **Traffic node** : show *traffic node* connection only.
- **Pedestrian node** : show *pedestrian node* connection only.
- **Light** : show light object connection only.

Show connection buttons : show connection buttons for selected *Light connection type*.

Lights index : objects with a selected *light index* are displayed (-1 value - all indexes are displayed).



Selected Light connection type : [TrafficNode] and Lights index : [0] example.

World Lights

Custom settings : on/off custom timeline settings for selected crossroad.

Timeline: shows *light states* of crossroad and total duration.

- **TrafficLight [0]** : *TrafficLightHandler* with *light index* 0.
- **TrafficLight [1]** : *TrafficLightHandler* with *light index* 1.

SceneView Light Objects Description

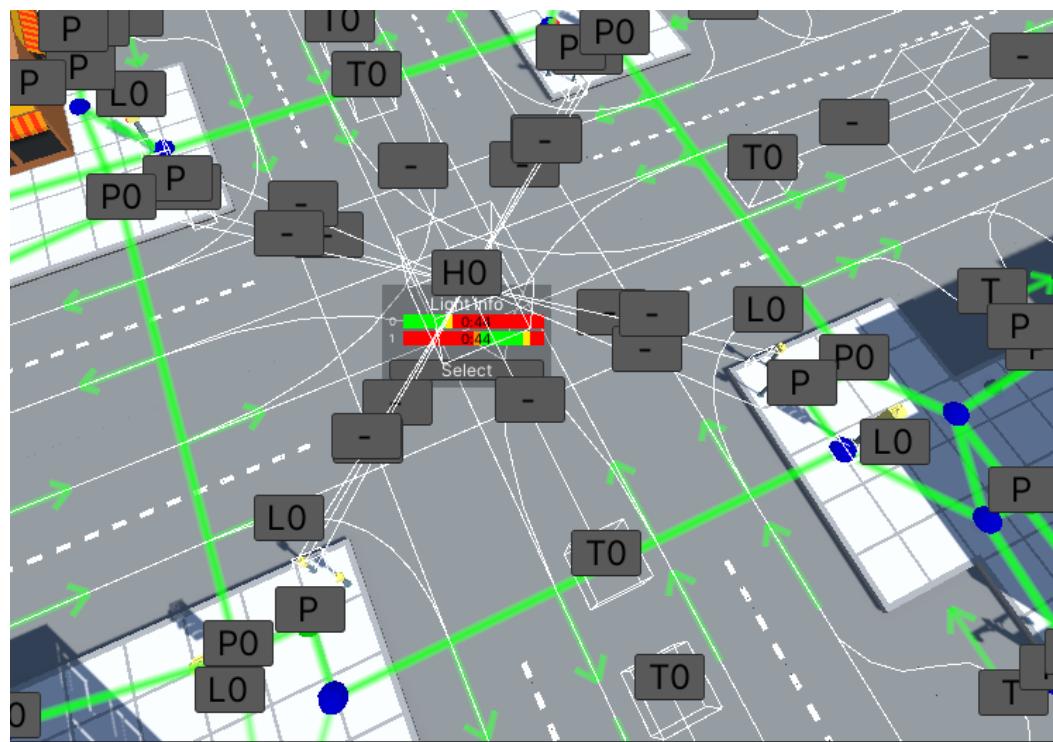
Select:

- H0/H1 : *TrafficLightHandler* (index 0, index 1).
- T0/T1/T : *TrafficNode* (index 0, index 1, no index).
- P0/P1/P : *PedestrianNode* (index 0, index 1, no index).
- L0/L1/L : *Light object* (index 0, index 1, no index).

Unselect:

- H- : unselect *TrafficLightHandler*.
- T- : unselect *TrafficNode*.

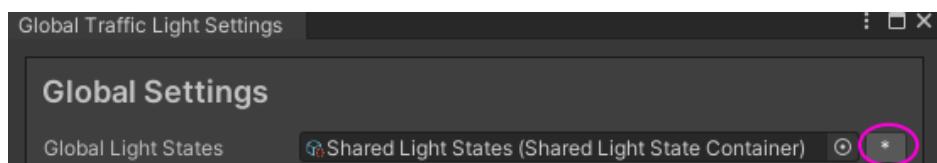
- P- : unselect *PedestrianNode*.
- L- : unselect *Light object*.



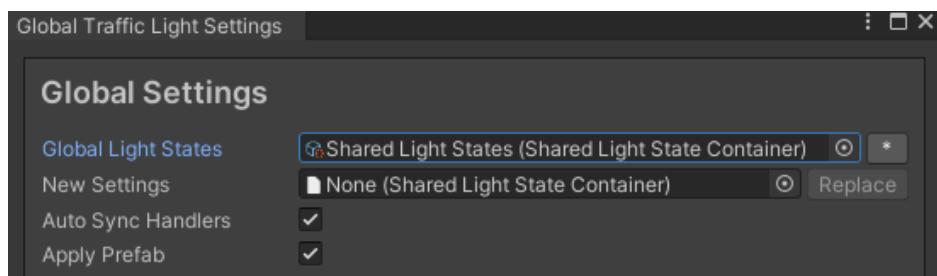
All connection types and -1 light index are enabled example.

How To Replace Global Light States

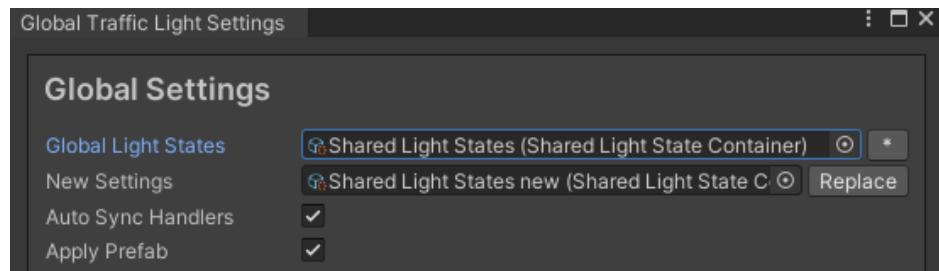
1. Open *Global Light Settings* window.
2. Click the * button to expand the *Replace settings*.



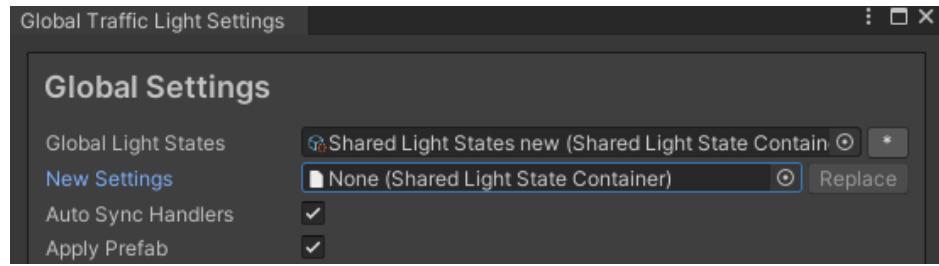
3. Select the source *state container* you wish to replace.



4. Select your new desired *State container*.



5. Click the *Replace* button.
6. As a result, all the source *State containers* are replaced.

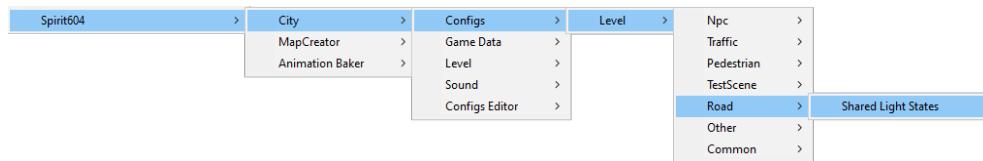


2.9.4 Shared Light State Container

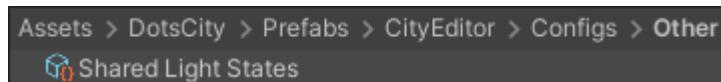
Contains common timings of *light states* that are shared between *traffic light crossroads*. You can easily replace shared containers using the *Global Light Settings* tool.

How To Create

from the project context :

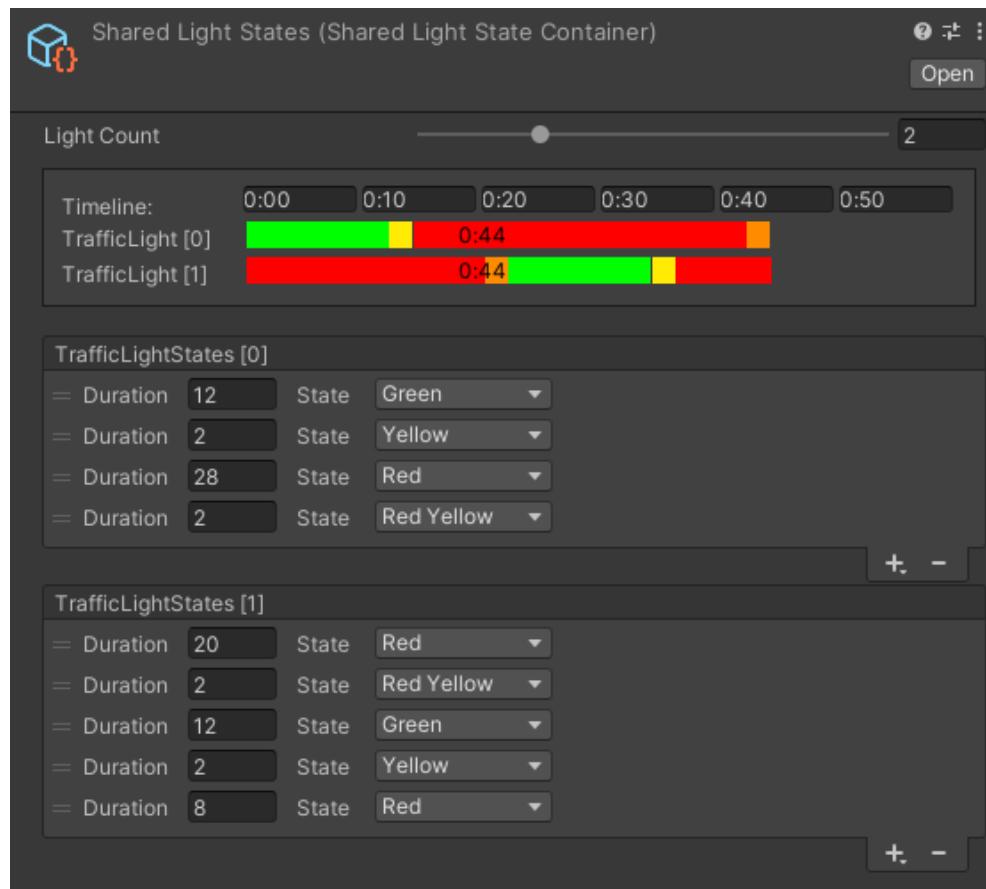


Default Container Path



Project path example.

Settings



Example.

2.9.5 Light States

- Green : car only drives on a green lights.
- Red
- Yellow
- Red Yellow : red and yellow lights at the same time, shown as orange in the inspector.

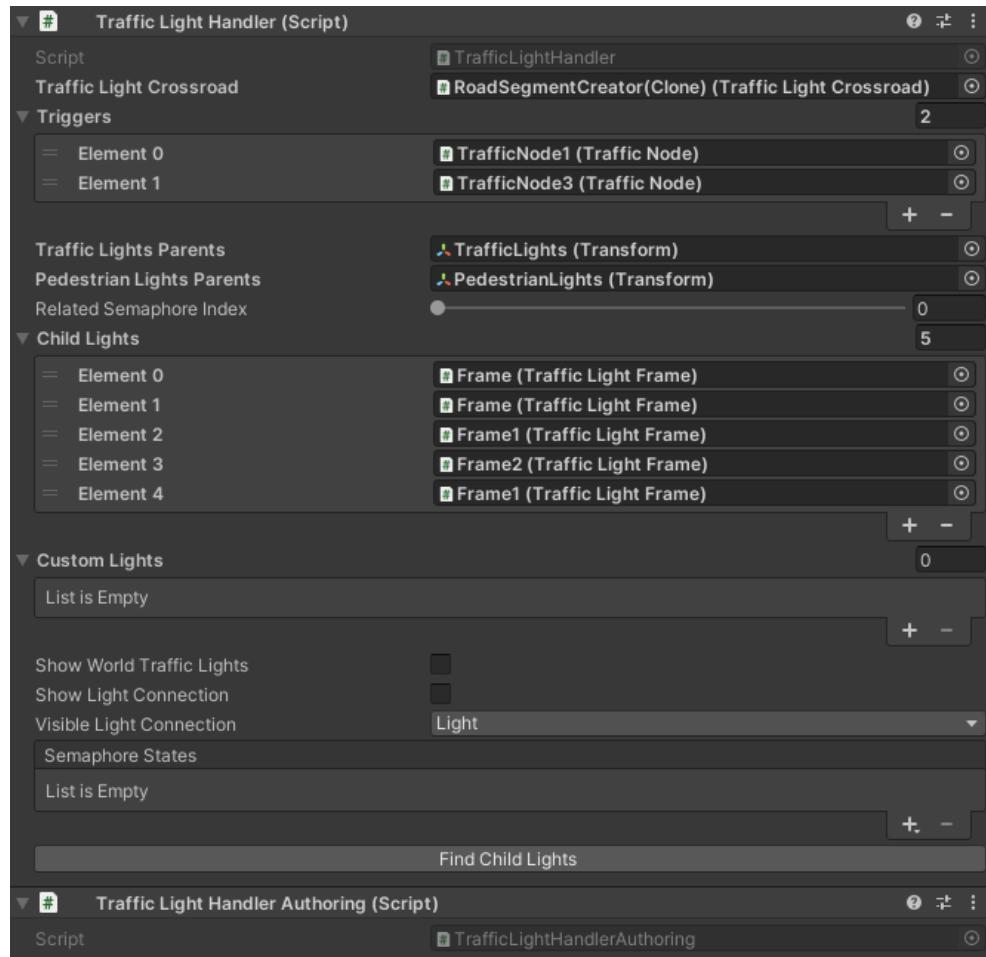
2.9.6 Light Index

Unique traffic light ID in `TrafficLightCrossroad` defined in `TrafficLightHandler` used to link `TrafficLightHandlers` and `traffic lights` by index.

2.9.7 Traffic Light Handler

Traffic Light Handler is an entity for handling the state of a traffic light. Is part of *TrafficLightCrossroad*.

Settings



Traffic light crossroad : reference to the *TrafficLightCrossroad*.

Triggers : nodes that relate to the handler.

Traffic light parent : parent to which the *light objects* will be added.

Pedestrian light parent : parent to which the *light objects* will be added.

Related light index : linked traffic *light traffic index*.

Child lights : list of attached child *light objects*.

Custom lights : list of attached custom *light objects*.

Light states : *light state of handler*.

Components

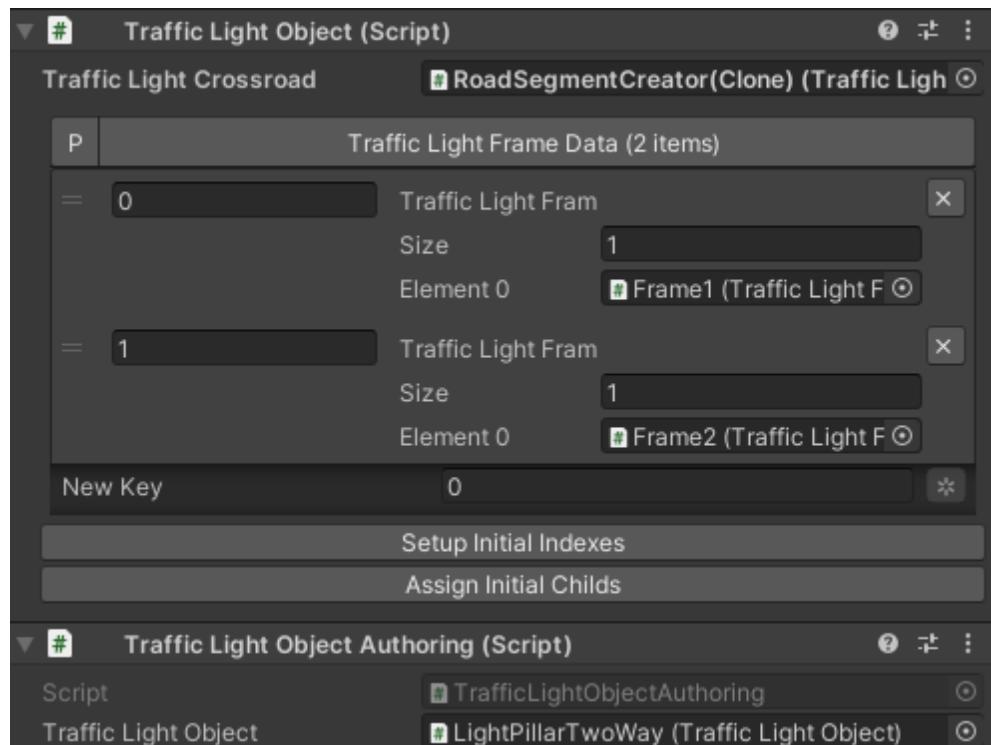
Authoring

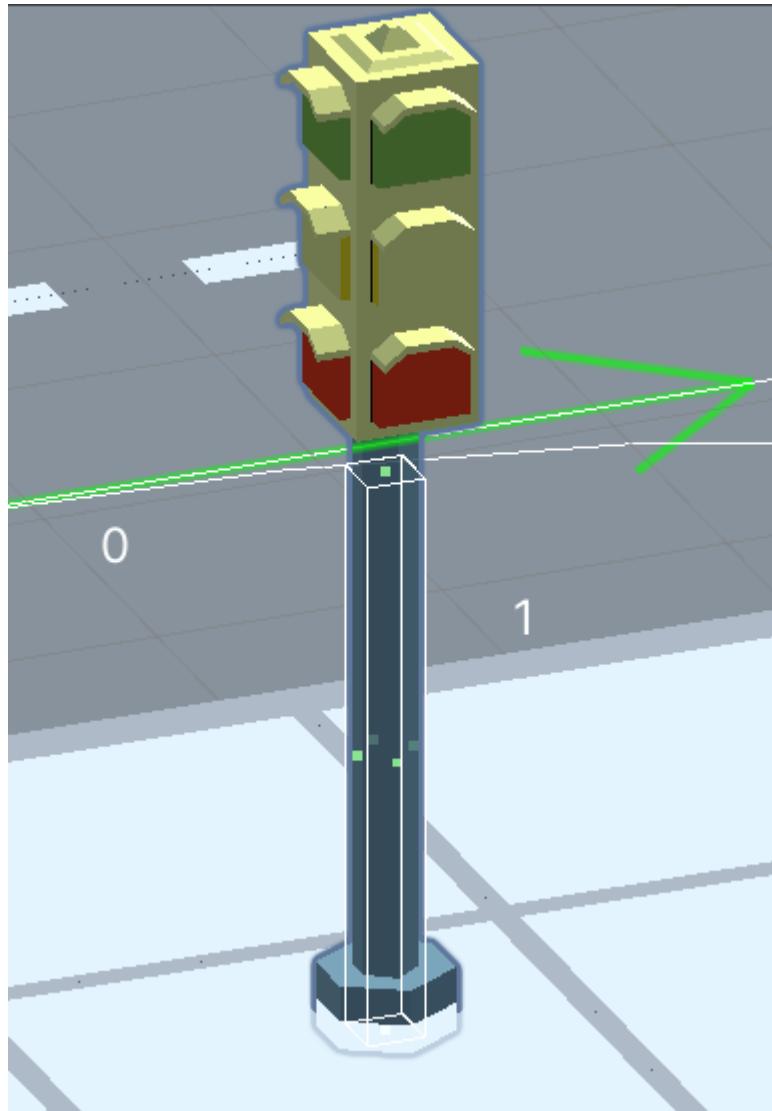
2.9.8 Traffic Light Object

Main Component

Traffic light object in the scene (parent component of the traffic light).

Contains data on the *light frames* and linked *light indexes*.

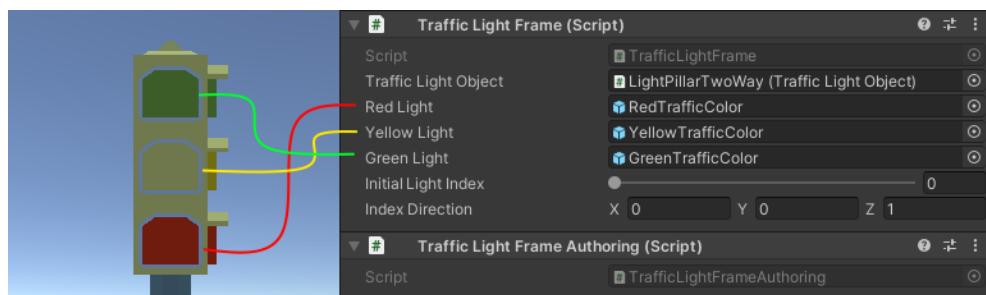




Traffic light object example.

Light Frame

A child component that contains the data for the traffic light indicators.



Traffic light object : reference to [light frames](#).

Red light : red light *state* entity.

Yellow light : yellow light *state* entity.

Green light : green light *state* entity.

Initial light index : initial *light index*.

Index direction : direction in which the *light index* is displayed in the scene.

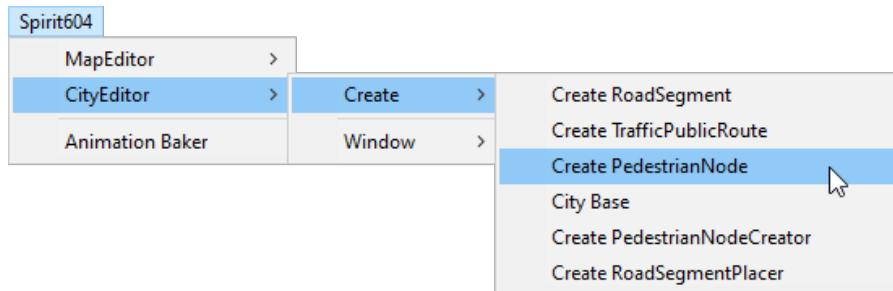
2.10 Pedestrian Node

2.10.1 How To Create

There are 2 ways:

1. Create node by yourself in the *Unity* toolbar:

Spirit604/Create/PedestrianNode

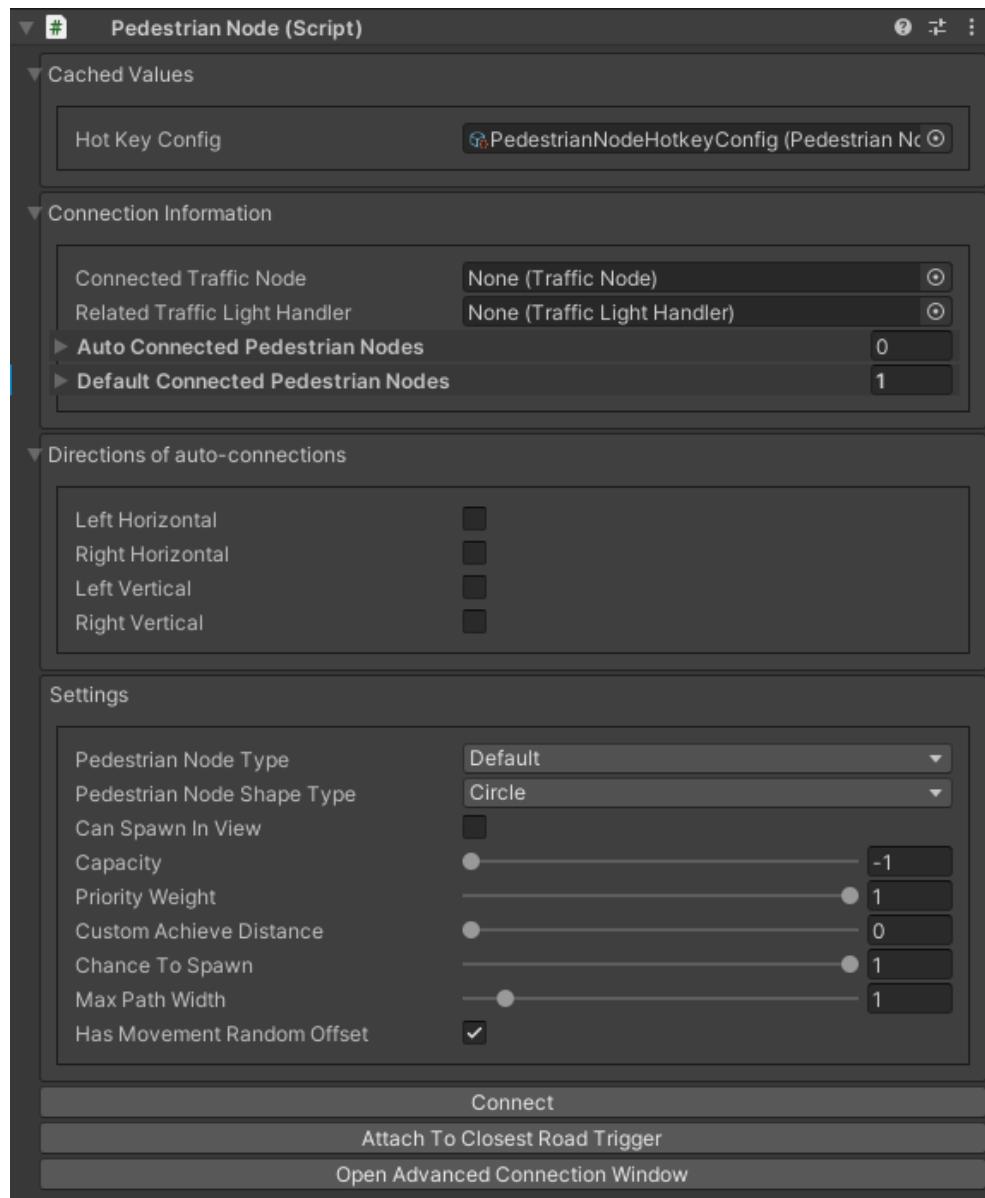


2. Create node with *PedestrianNodeCreator tool*.

2.10.2 Pedestrian Node

Pedestrian node is a node for creating *pedestrian* route.

Youtube tutorial.



Connection Data

Connected traffic node : connected *traffic node* (for parking).

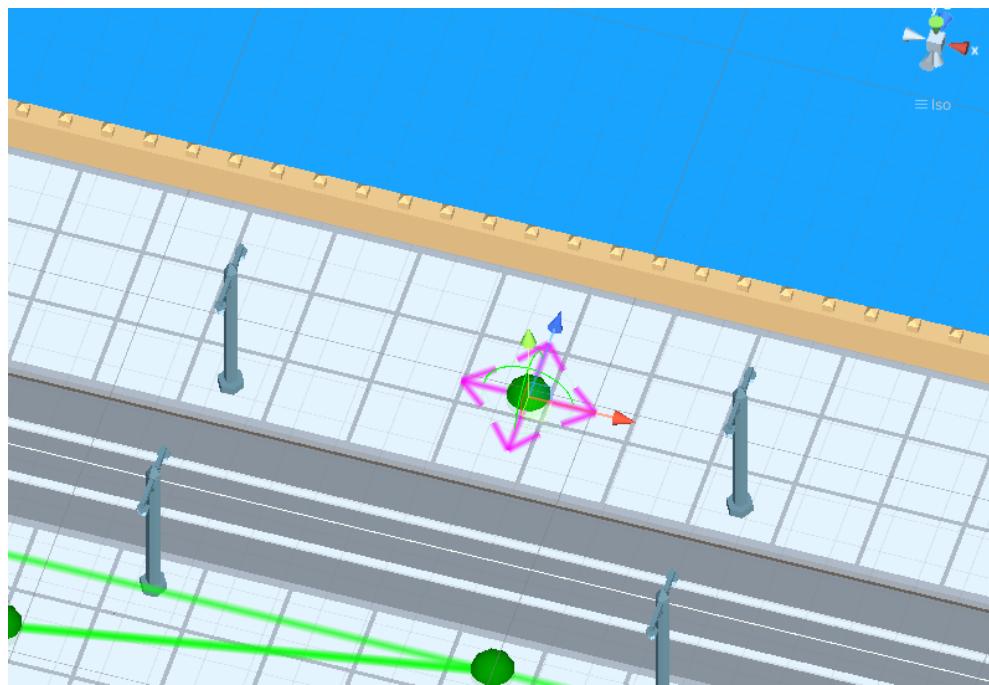
Related traffic light handler : connected traffic light.

Auto connected pedestrian nodes : connected other nodes along *selected direction* raycasts.

Default connected pedestrian nodes : connected other nodes by the user.

Directions of auto-connections

Direction of raycasts to connect with other nodes (can be useful for generating squared scenes).



All connection directions enabled example.

Note: Used by *road parent*.

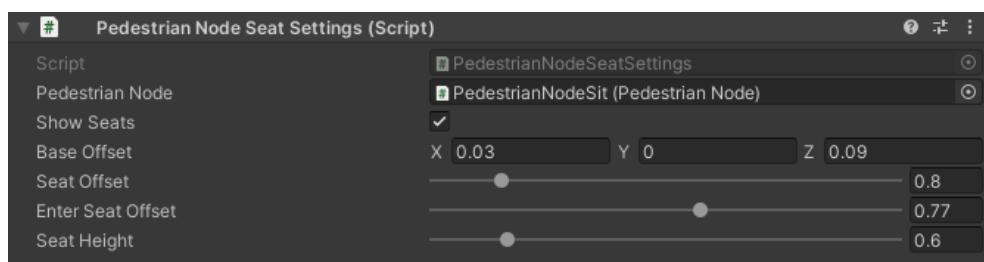
Custom Node Settings

Default

Default node for the route.

Sit

Node for benches, seats, etc (*test scene*).



Show seats : enable editor display of seats.

Base offset : offset between the center of the seats and the center of the node.

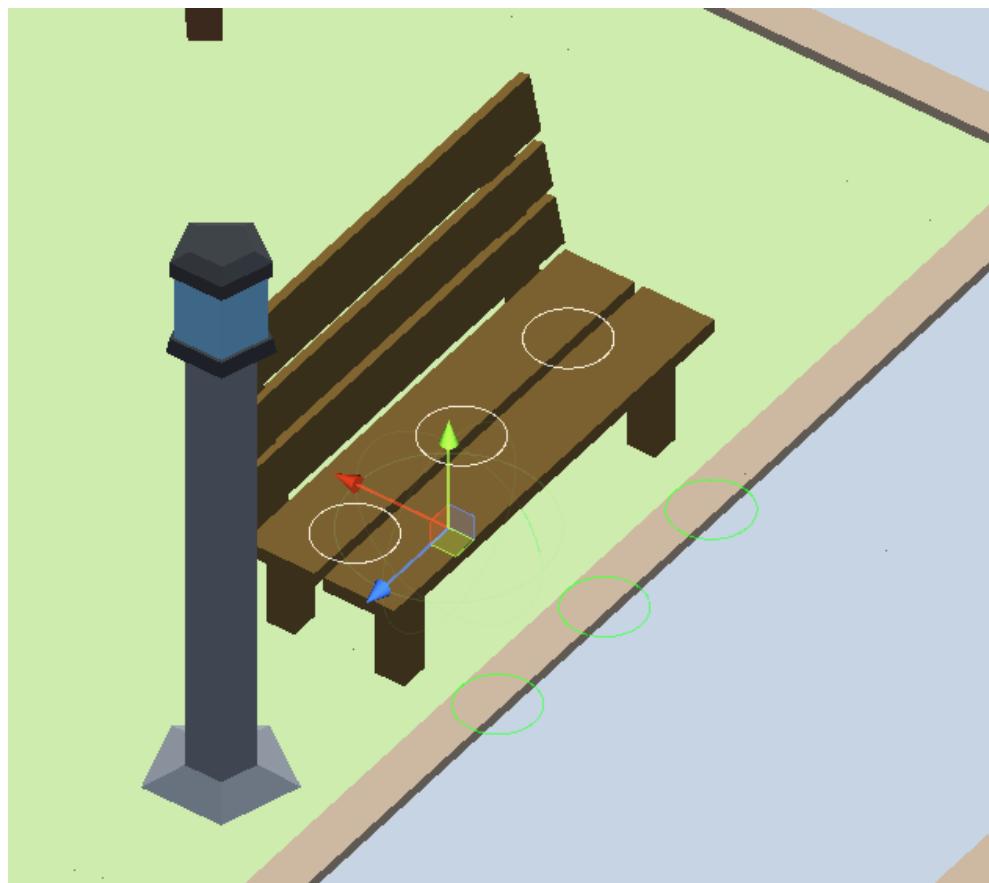
Seat offset : offset between the seats.

Enter seat offset : offset between the animation start position and the seat.

Seat height : seat height.

Capacity : number of seats (adjustable in the *settings* of the node).

Note: Required *PedestrianNodeSeatSettings* component.



Bench example.

House

Node for entry/exit to the house (*test scene*).

Tip:

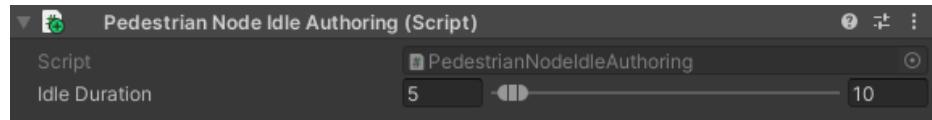
- Enable *Can spawn in view* in the pedestrian node *settings* to simulate leaving from the house.
 - Pedestrians who have entered the house node will be destroyed.
-

Idle

Node for temporary idling pedestrians (*test scene*).

Note:

- To set a custom idle time at the node, add the *PedestrianNodeIdleSettings* component.



Car parking

Node to enter/exit a parked car (generated by *ParkingBuilder*). Read more about *parking states*.



Parking node example.

Talk area

Node for crowd conversations of pedestrians (*test scene*).

Area shape type: type of area shape.

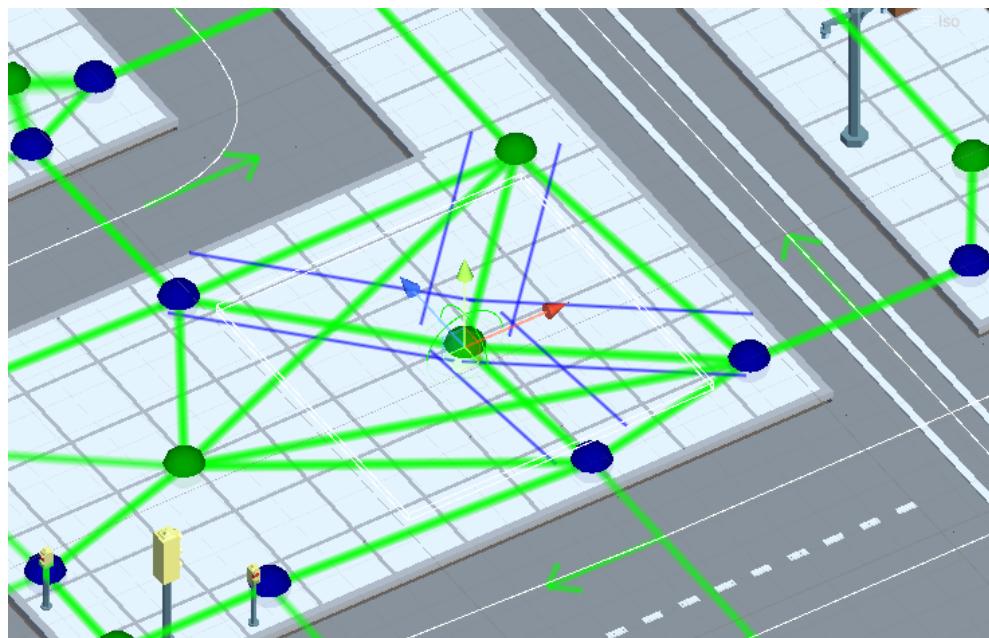
- Square
- Circle

Area size : area size.

Min/Max spawn count : min/max number of pedestrians that the area can contain.

Unlimited talk time : on/off infinite conversation for pedestrians in the talk area.

Show bounds : show bounds of area.

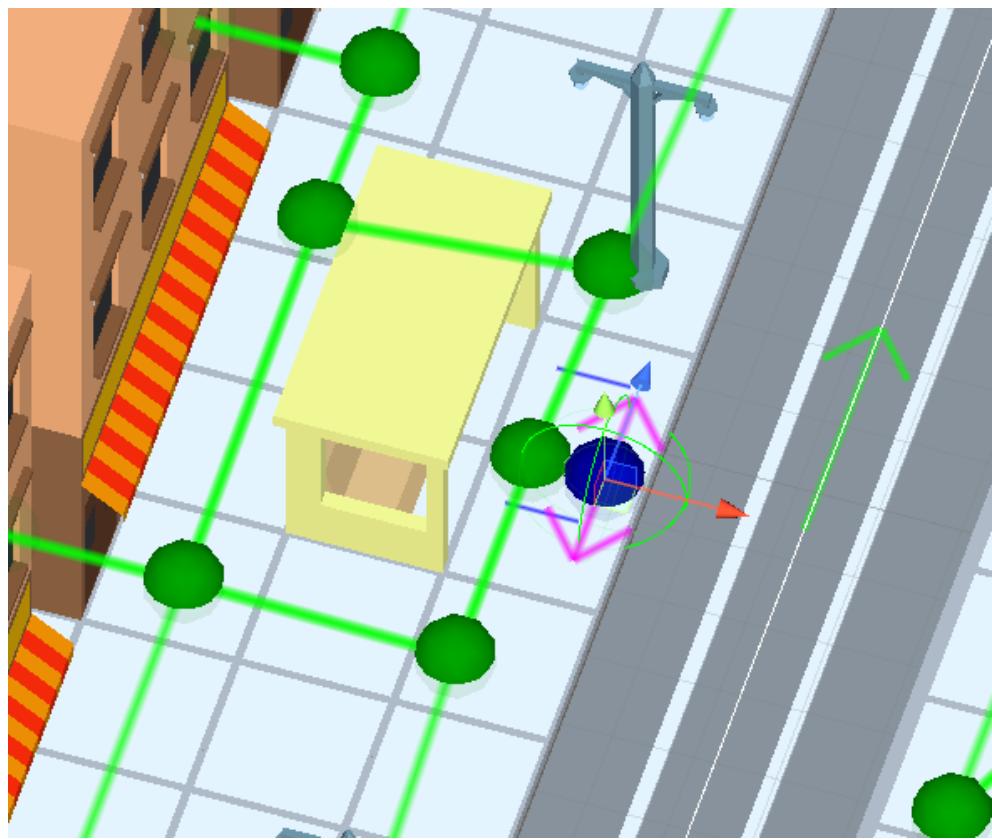


Talk area example.

Traffic public stop station

Node for waiting for *public transport*.

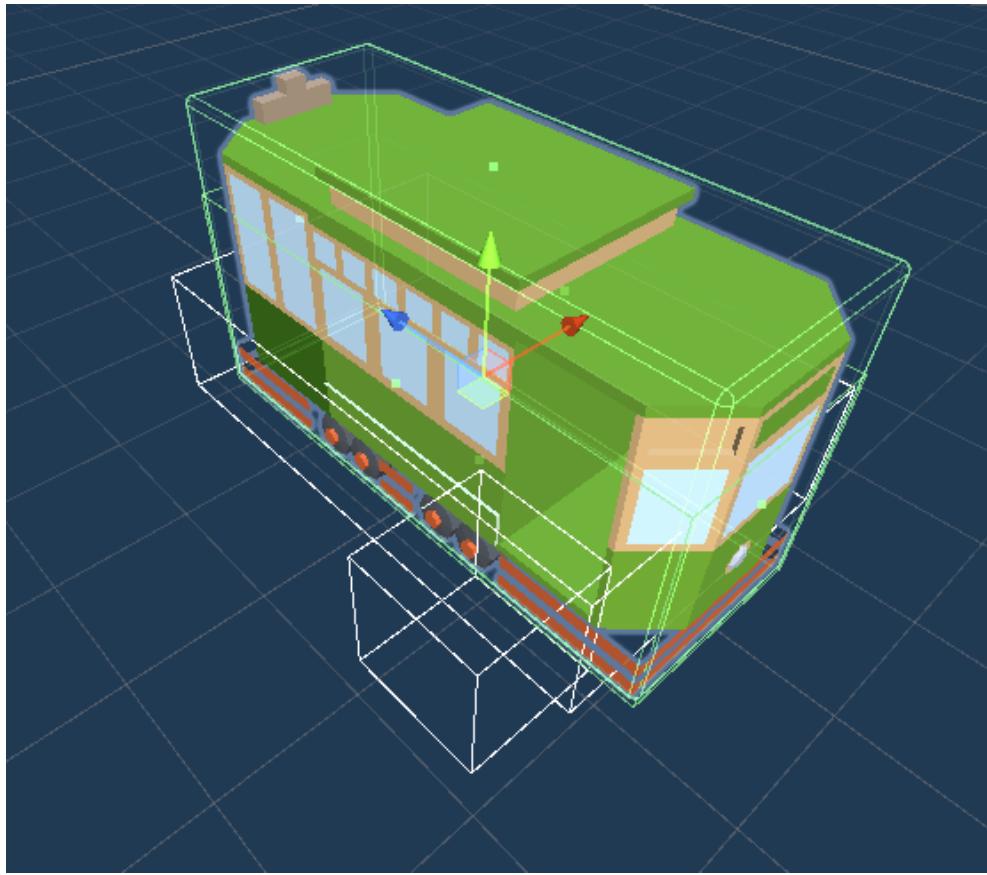
Note: To set the maximum number of waiting vehicle passengers, adjust the *capacity* parameter.



Stop station example.

Traffic public entry

Node for entering *public transport*.



Public entry example (white box).

Warning: Entry node should be any *GameObject* with the component *VehicleEntryAuthoring* which adds a node with *Traffic public entry* type.

Common Settings

Shape type

[shape of the area for randomization of *pedestrian* targets.]

- Circle
- Square
- Rectangle

Can spawn in view : can spawn in view of camera or not.

Capacity : -1 value is unlimited; Capacity for objects like benches, houses, public stop stations etc...

Priority weight : weight for choosing random node by *pedestrian*.

Custom achieve distance : custom achieve distance for *pedestrian*. If 0 then default value is taken.

Chance to spawn : chance to spawn *pedestrian* at node [0 = 0%, 1 = 100%].

Max path width : maximum width of the route around the node.

Height : maximum height size of the node area (square and rectangle shapes only).

Has movement random offset : are supposed to randomize the position around a node.

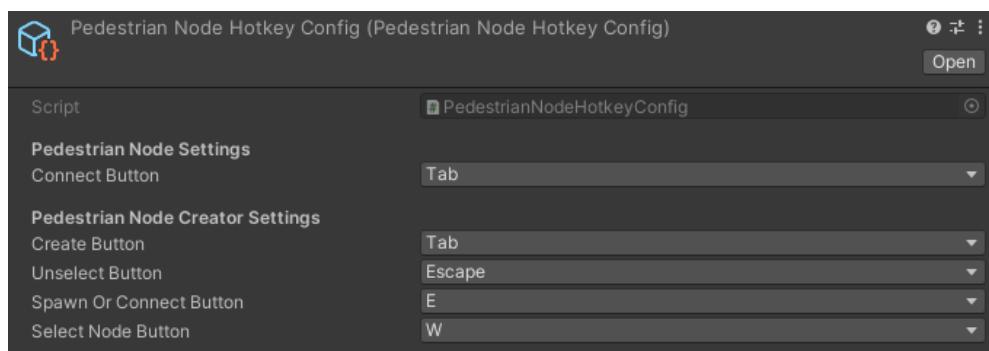
Buttons

Connect : node will make raycasts to the *selected directions* to connect other nodes.

Attach to closest traffic node : trying to connect to a nearby *TrafficNode*.

Open advanced connection window : open *Advanced connection window*.

Hotkeys



2.10.3 CullState Info

States

- **Culled** : entity not available for spawning.
- **CloseToCamera** : entity available for spawn.
- **InVisionOfCamera** : entity available for spawn only during the initial scene start (unless the *Can spawn in view* option is enabled).

2.10.4 Pedestrian Node Creator

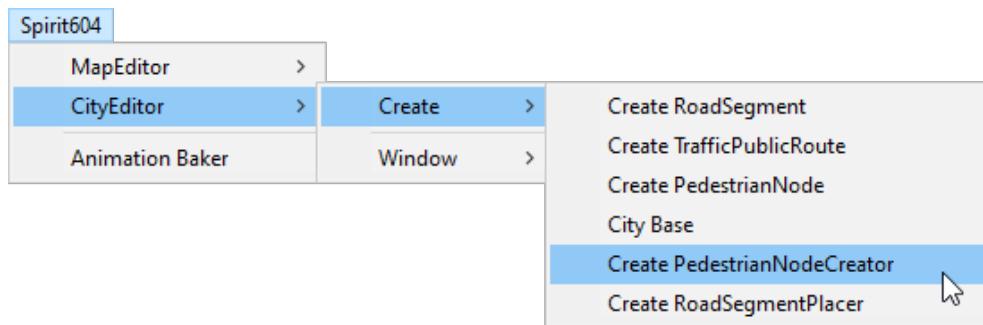
Pedestrian Node Creator is a tool to quickly create and connect *pedestrian nodes*.

Youtube tutorial.

How To Create

Select in the *Unity* toolbar:

Spirit604/Create/PedestrianNodeCreator



How To

Create Node

1. Press the *Tab* button on the keyboard to create preview *PedestrianNode*.
2. Place the preview *PedestrianNode* at the desired position.
3. If you need to attach the *PedestrianNode* to a custom shape surface, enable *Auto Attach To Surface* in the *Inspector*.
4. Press *E* button on the keyboard to create the final *PedestrianNode*.

Tip: You can change the *hotkeys* to your taste.

Select Node

1. Choose *Selection mode*.
2. Click *W* above the node to select *PedestrianNode*.

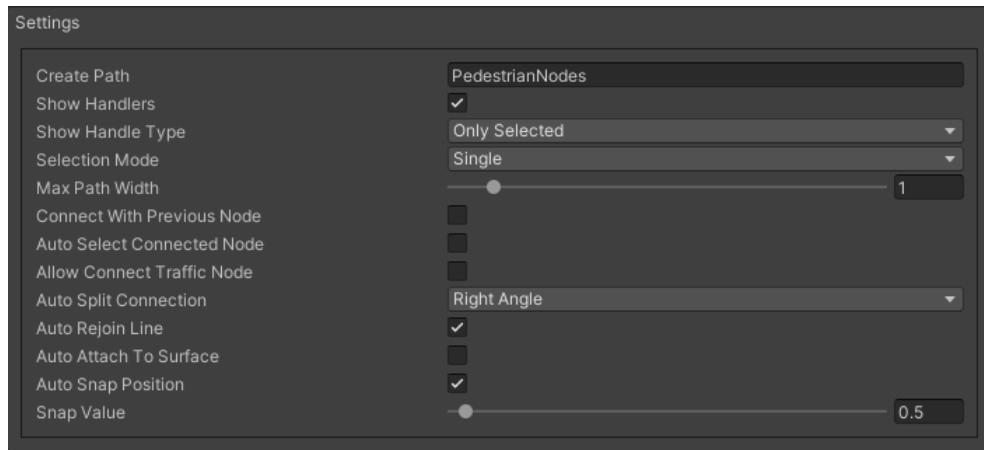
Connect Node

1. *Select the desired node*.
2. Click *E* over the target *PedestrianNode* to connect (*Single selection mode* only).

Locate Node

1. Choose *Selection mode*.
2. *Select the desired nodes*.
3. Drag the position handle to the desired position.

Settings



Show handlers : on/off position handles for nodes.

Show handle type:

- **Only created** : only the created nodes will have handles shown
- **Only selected** : only the selected nodes will have handles shown.
- **All** : all nodes will have handles shown

Selection mode:

- **Single** : only 1 node is selected.
- **Multiple**
 - [multiple nodes can be selected.]
 - **Multiple handle type:**
 - * **Single** : node has a position handle each individually.
 - * **All** : all nodes have the same position handle.
 - **Unselect selected** : if you try to select an already selected node, it will be unselected.

Max path width : global width of routes for all nodes (enable preview [here](#), save global width [here](#)).

Connect with previous node : currently created node will be connected to the previously created node.

Auto select connected node : node will be selected after it is connected to the source node.

Allow connect traffic node : on/off feature to connect to the *TrafficNode*.

Auto split connection

[if a node is located between a connection of existing nodes, the connection will be reconnected between them (made with a *Raycast*).]

- **Disabled**
- **Right angle** : 90° angle.
- **Custom angle** : user custom angle.

Auto rejoin line : if there are other nodes on the connection line, they will automatically be connected to each other in one row.

Auto attach to surface

[auto attach created node to surface.]

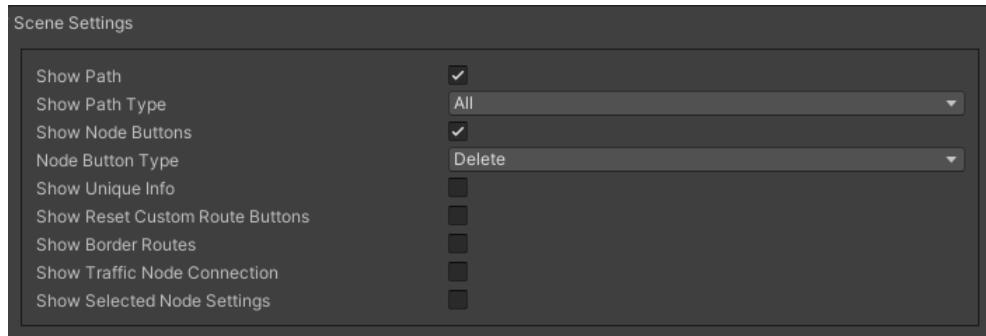
- **Surface mask** : layer mask to attach.
- **Attach type:**
 - **Collider** : attach to collider.
 - **Mesh** : attach to mesh.

Auto snap position

[auto snap node position during creation.]

- **Snap value** : snapping value.

Scene Settings



Show path : show pedestrian node routes.

Show path type:

- **All** : all the nodes will be shown.
- **Only created** : only the nodes created by the creator will be shown.

Show node buttons

[on/off display custom buttons of selected nodes.]

- **Node button type:**
 - **Delete** : node will be deleted by clicking.
 - **Unselect** node will be unselected by clicking.

Show unique info : unique information of the node will be displayed (different from the original prefab).

Show reset custom route buttons : for nodes with a custom route width, the reset buttons will be displayed.

Show border routes :

- **Current** : route will be displayed with the assigned width of the nodes.
- **Selected** : route will be displayed with the selected route width in the *creator settings*.

Show traffic node connection : on/off display the connection to the *TrafficNode*.

Show selected node settings : shows *node settings* in the inspector.

Buttons

Create node : create preview node.

Add all scene pedestrian nodes : all nodes will be added to the creator.

Add all scene custom pedestrian nodes : only nodes with custom widths will be added to the creator.

Save global path width : hange the width of the route for all nodes.

Reset all custom path width : for all nodes with custom widths will be assigned the default value.

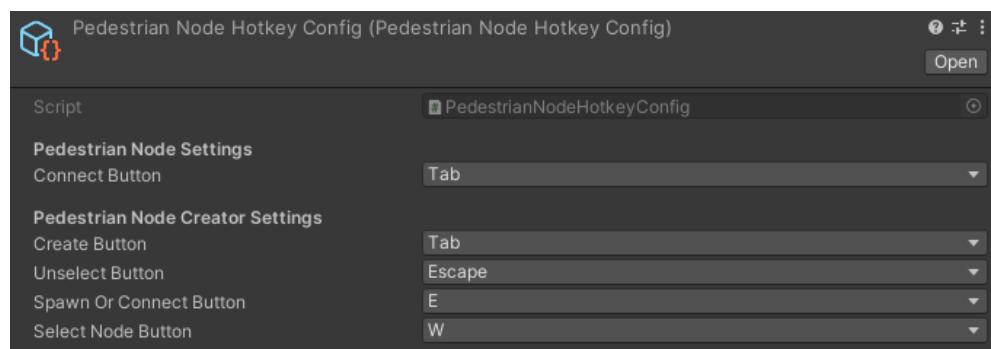
Clear created nodes info : clear the list of nodes created by the creator.

Clear selection : clear selected nodes [multiple selection mode only].

Snap to grid : snap selected node position [for *selected node* only, *auto snap* should be enabled].

Open advanced connection window : open *Advanced connection window* [for *selected node* only].

Hotkeys

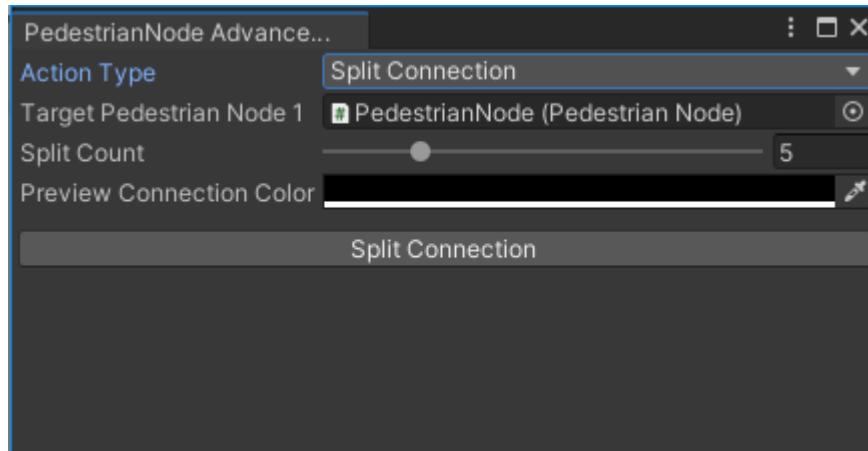


2.10.5 Advanced Connection Window

Help window for advanced node connection settings.

Split Connection

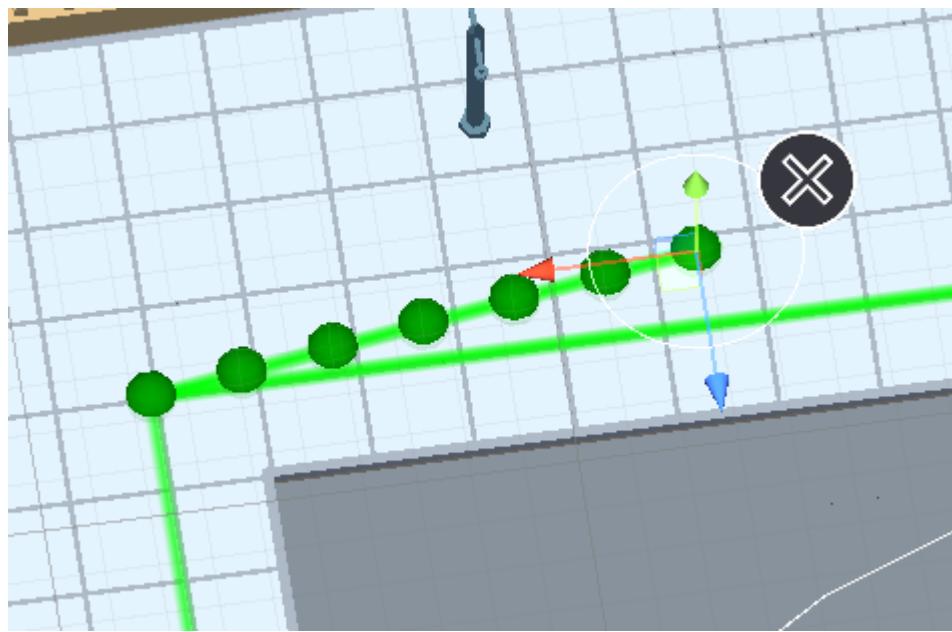
Split the existing connection into several nodes.



Target pedestrian node : selected node where the split connections will be.

Split count : number of new nodes created between the selected two.



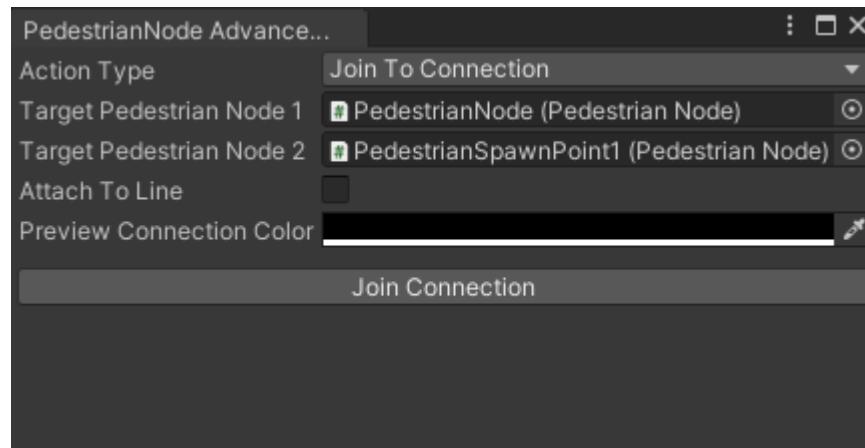


Split connection example.

Note: Split is available for already connected nodes only.

Join To Connection

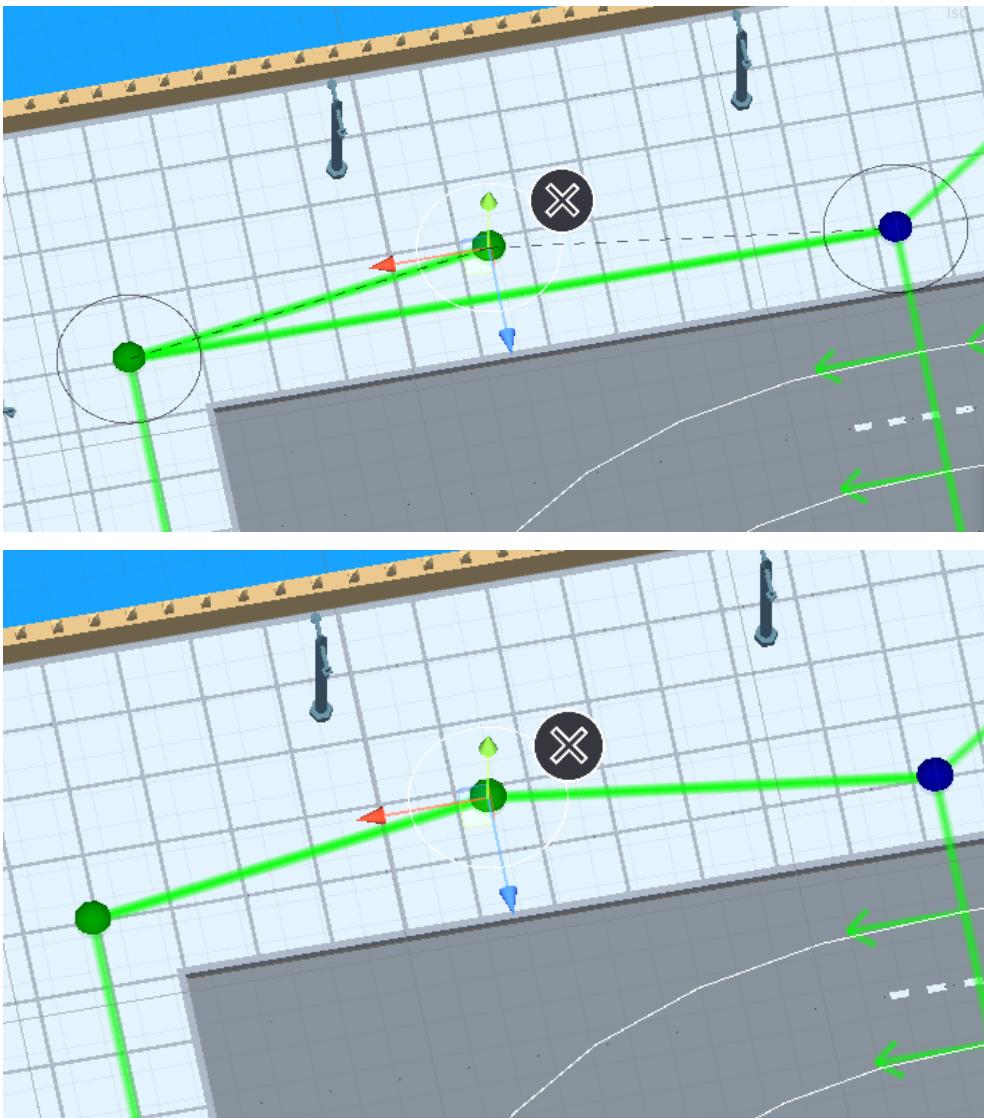
Connect the selected node to an existing connection.



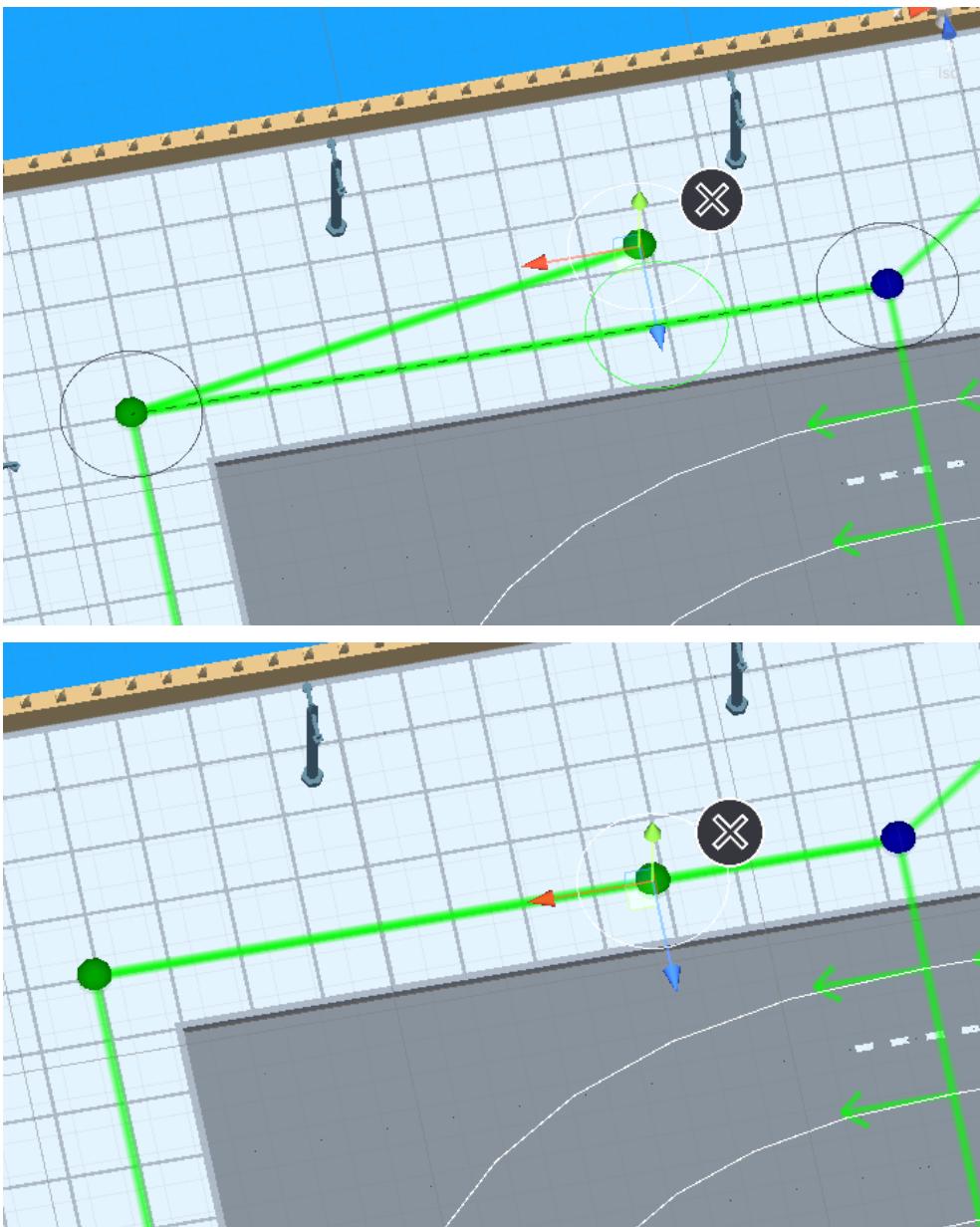
Target pedestrian node 1 : target node 1 of selected connection.

Target pedestrian node 2 : target node 2 of selected connection.

Attach to line : source node will be moved to the line connecting target nodes.



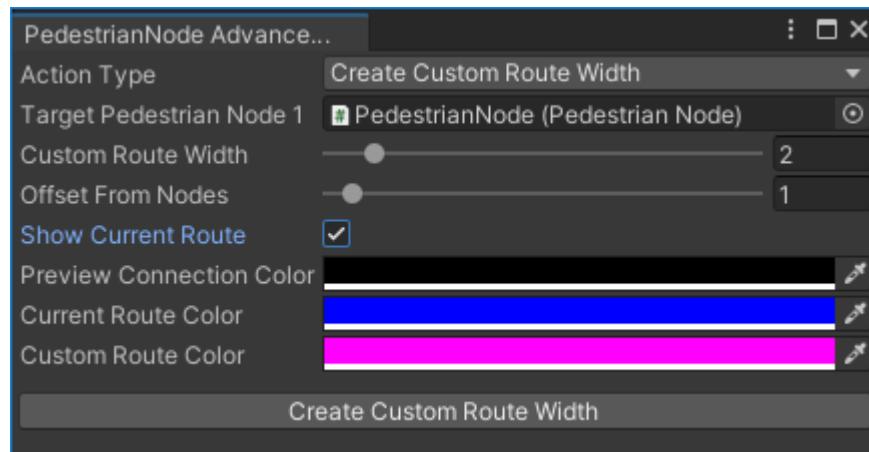
Join to connection example 1.



Join to connection example 2 (attach to line enabled).

Create Custom Route Width

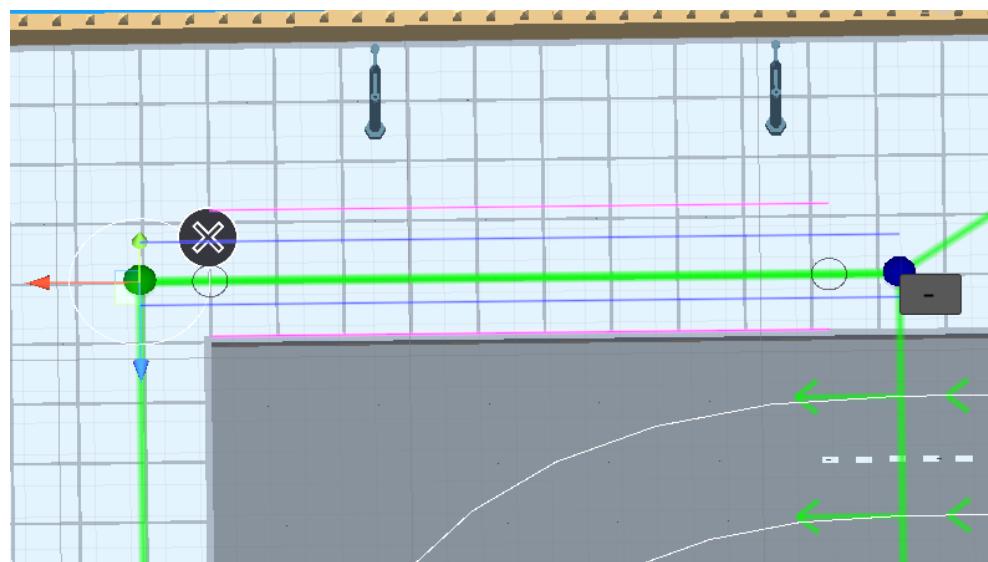
Create a custom route with custom width between two nodes.

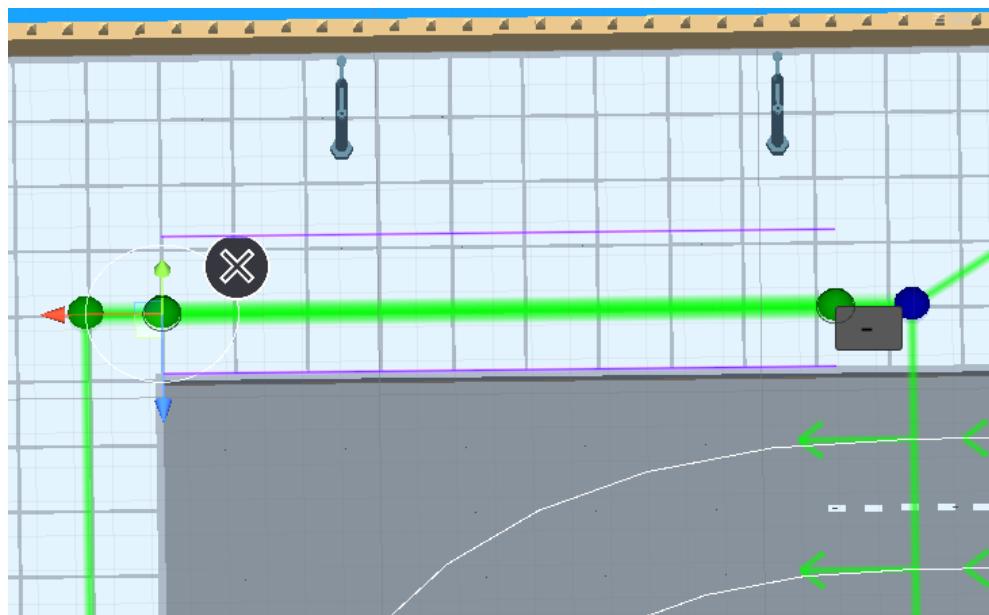


Target pedestrian node 1 : target connected node of selected connection.

Custom route width : new width of custom route.

Offset from nodes : new created nodes offset from existing nodes.

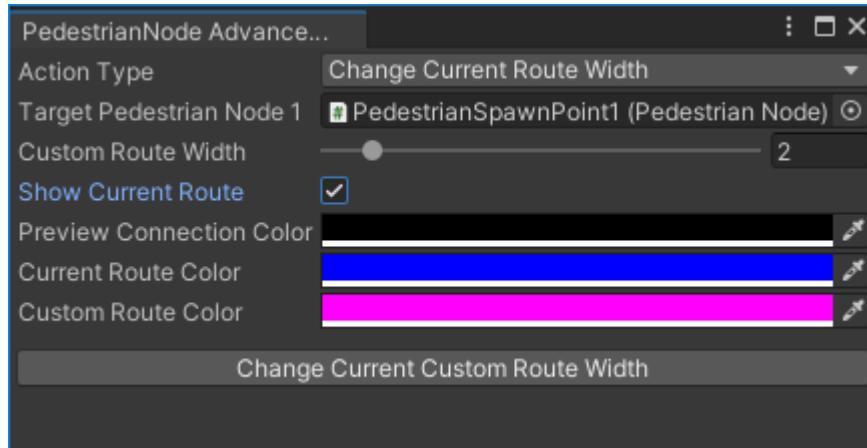




Create custom route width example.

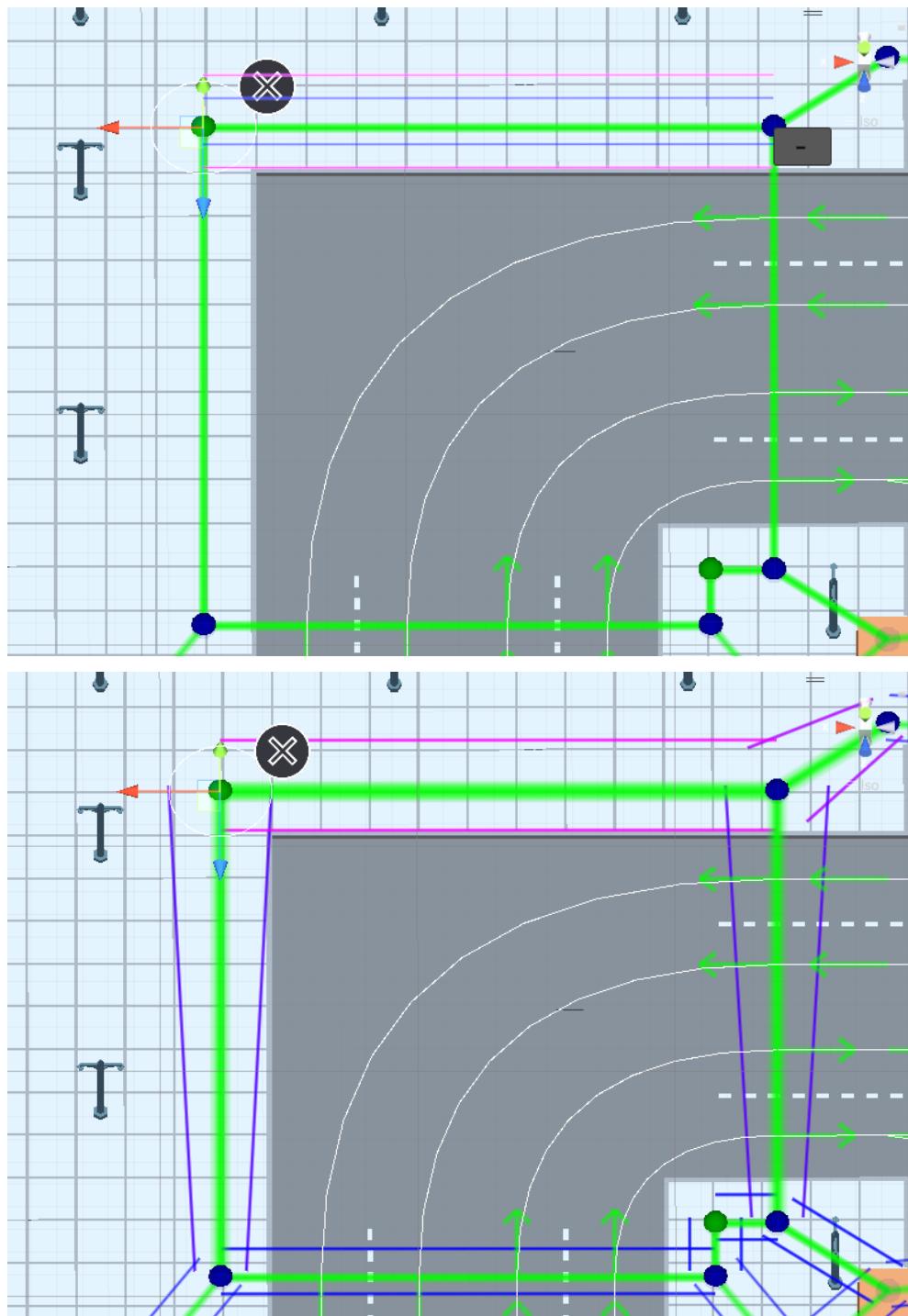
Change Current Route Width

Set the custom width to the two selected nodes.



Target pedestrian node 1 : target connected node of selected connection.

Custom route width : new width of custom route.



Change current route width example.

2.11 Traffic Area

Traffic Area is used for restricted traffic areas with an order of entry and exit.

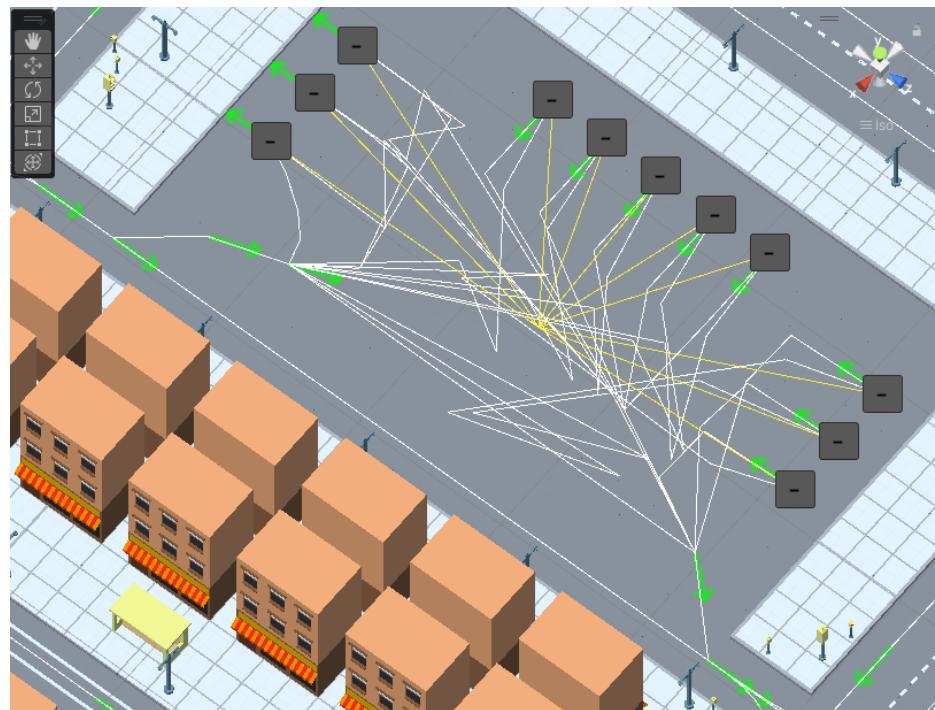
Use-case examples:

- To enter a petrol station.
- Used for *large parking areas*.

Youtube tutorial.

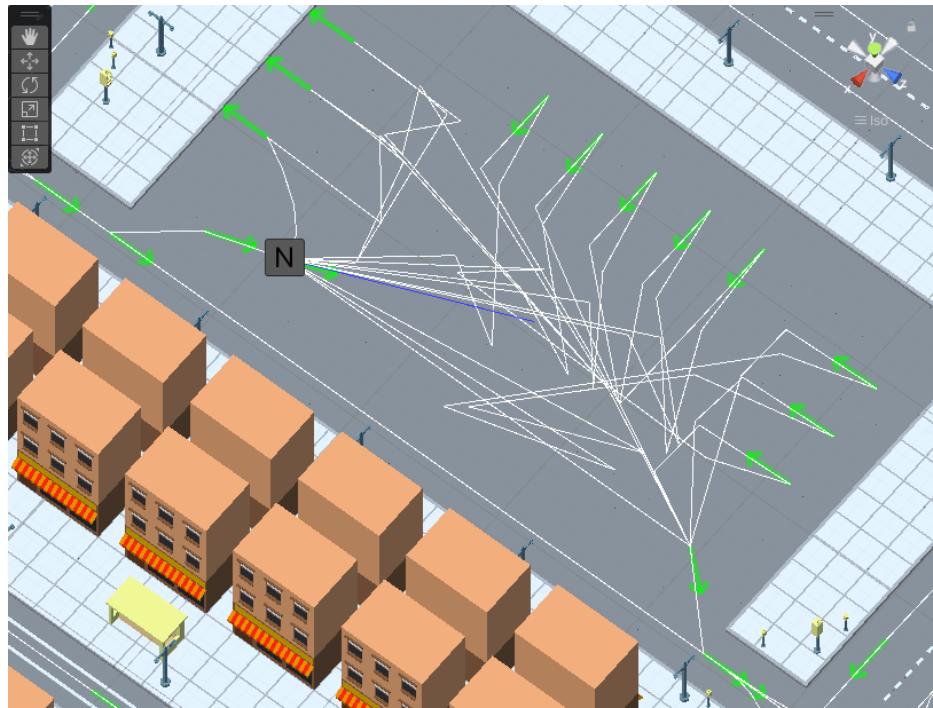
2.11.1 How To Create

1. Create an empty *GameObject* and add *TrafficAreaAuthoring* component
2. Set the *Button select type* to *Select node* value.
3. Enable the *Show scene nodes* option to see the *traffic nodes* in the scene.
4. Set *New node type* to *Default* value.
5. Select *Default* nodes in the scene.



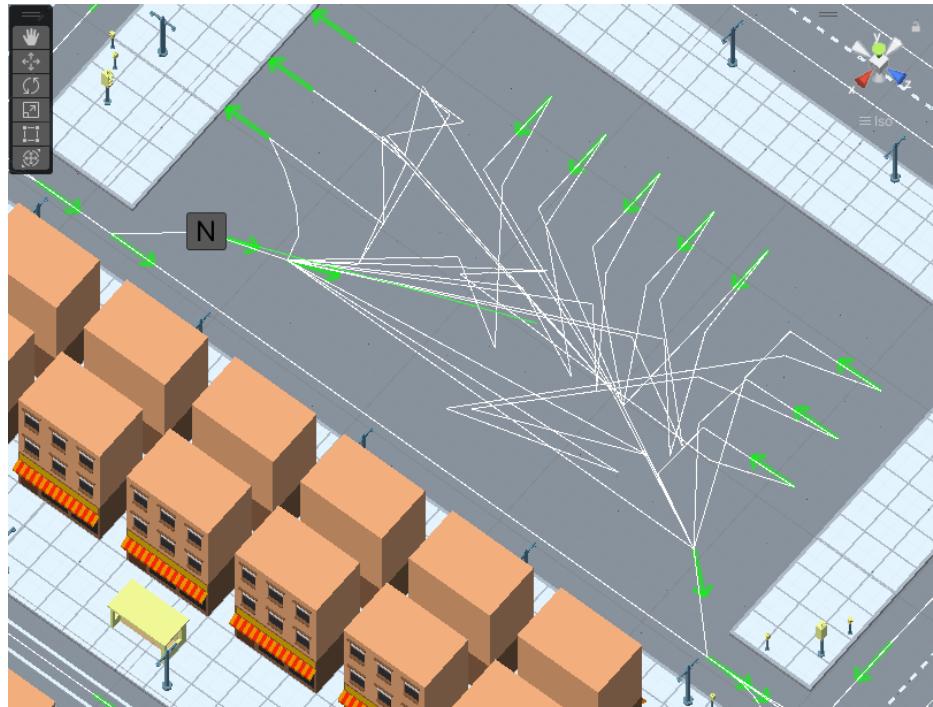
Selected Default nodes example.

6. In the same way, select the *Queue* node in the scene.



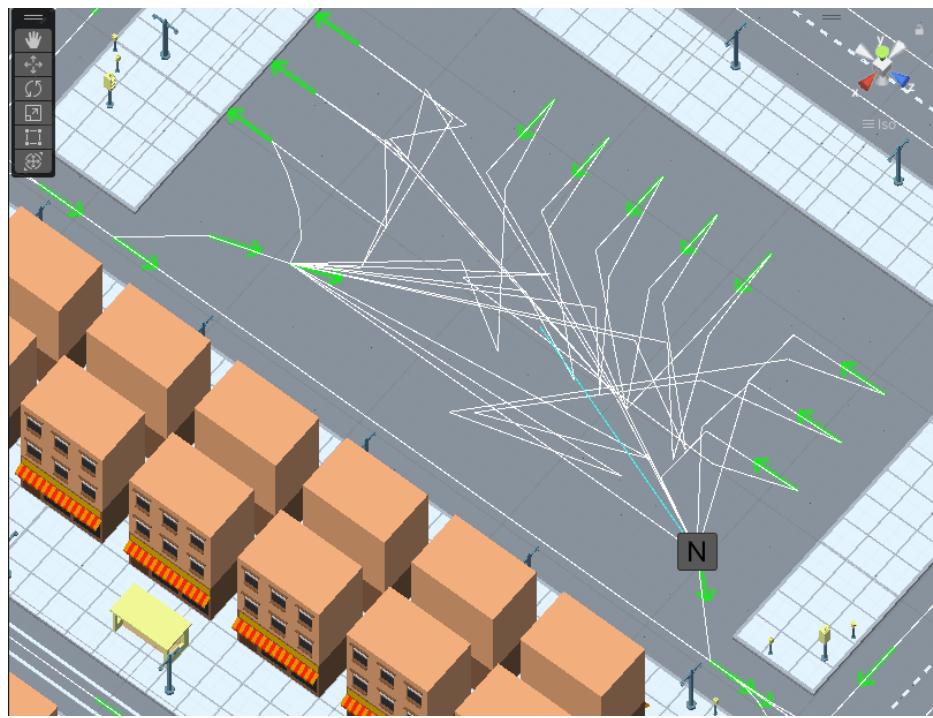
Selected Queue node example.

7. Select *Enter* node in the scene.



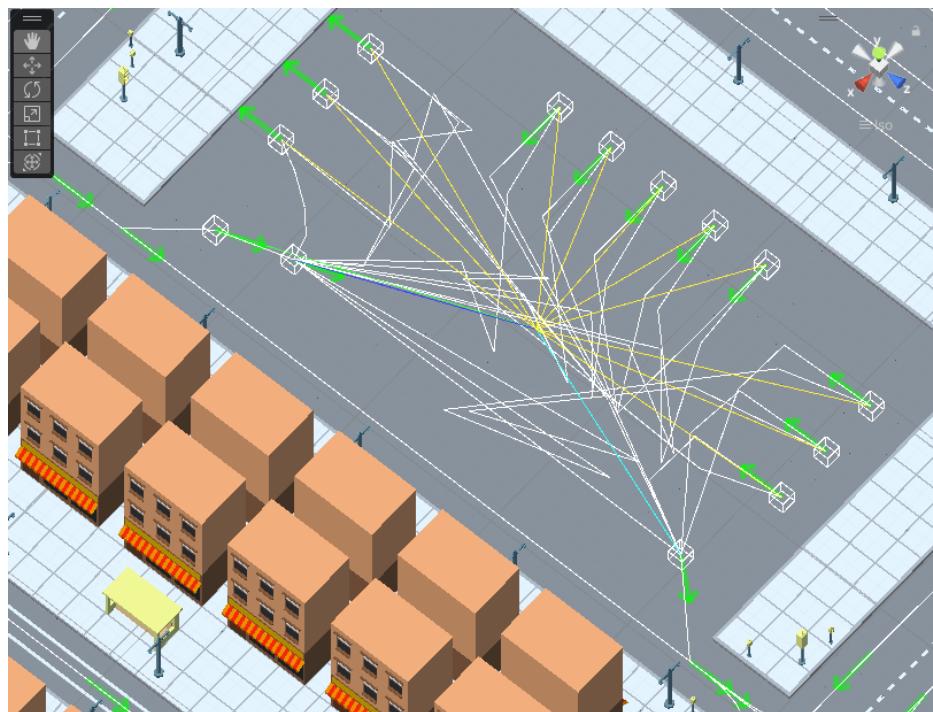
Selected Enter node example.

8. Select *Exit* node in the scene.



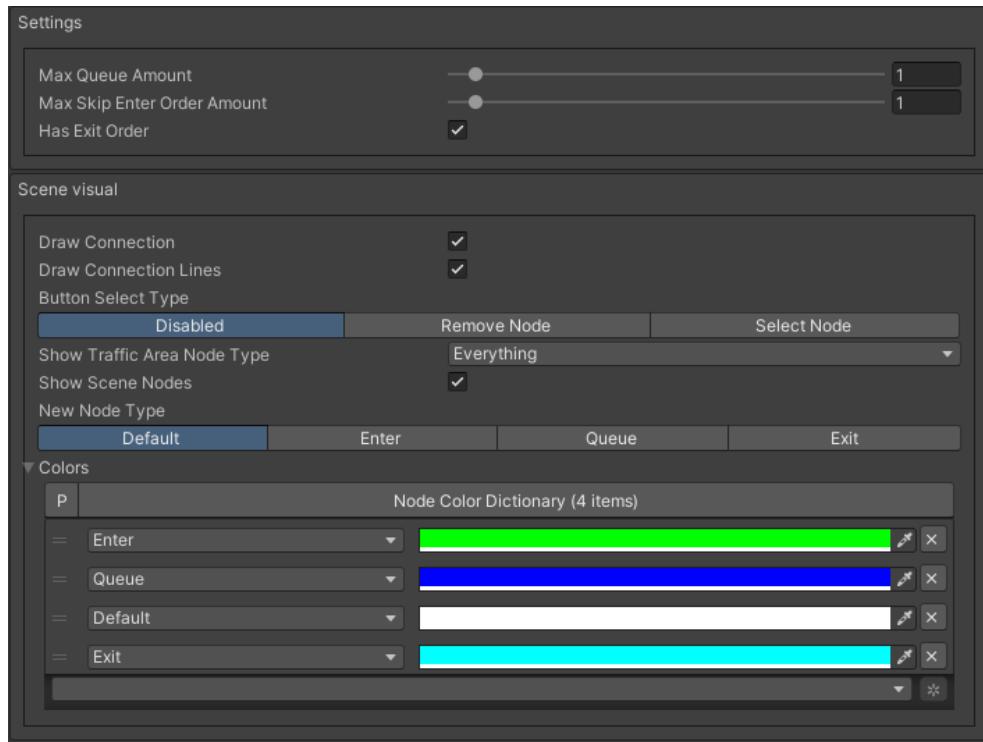
Selected Exit node example.

9. Adjust the *Max queue count* depending on how many cars can fit in the queue without blocking other cars.



TrafficArea result example (parking created by ParkingBuilder).

2.11.2 Settings



Settings

Max queue count : maximum number of cars in a queue (if the maximum number is exceeded the entrance node will be closed).

Max skip enter order count : number of vehicles that can be let in at the entrance (1 value example: 1 enters vehicle - 1 exits - 1 enters - 1 exits).

Has exit order : cars leave the *TrafficArea* on a queue basis.

Scene visual

Draw connection : on/off visual connections.

Draw connection lines : on/off connection lines to the *traffic nodes*.

Button select type:

- **Disabled**
- **Remove node** : selected node will be removed from *TrafficArea*.
- **Select node** : selected node will be added to *TrafficArea* with the select *New node type*.

Show traffic area node type : *nodes* with the selected *node type* will be displayed in the scene.

Show scene nodes : on/off display add buttons paths to *TrafficArea*.

New node type : *TrafficNode* with the selected *node type* will be added to the *TrafficArea*.

Node type

- **Default** : a node which is included in the *TrafficArea* but does not belong to one of the types listed below.
- **Enter** : entrance node to the *TrafficArea* (if the maximum number of vehicles in the queue is exceeded, the node will be closed).
- **Queue** : node in front of which a line of cars is waiting.
- **Exit** : when it passes this node, the car leaves the *TrafficArea*.

2.12 Traffic Public Route

Defined public route for *public transport*.

2.12.1 How To Create

1. Create necessary *road segments*.
2. Connect the created segments by *paths*.
3. Create an empty *GameObject* and add the *TrafficPublicRoute* component.
4. Enable the *Show path selection buttons* option.
5. Sequentially select each *path* of the route (also you make a *lane change*).



Selection route example.

6. Customize *Route settings*.

7. Make sure you have created at least one compatible (matching *TrafficPublicType* and *Car model*) *TrafficPublic* vehicle.

2.12.2 Transition Info

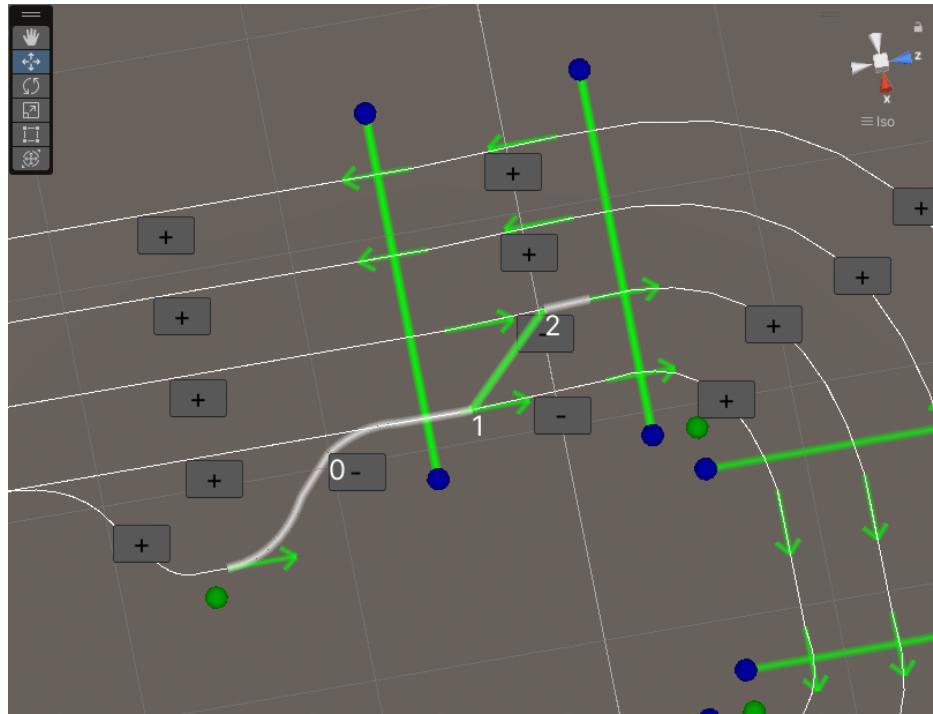
Transition paths are used for transition between lanes of public transport.

How To Create

1. Select source path.



2. Select a neighbouring path.

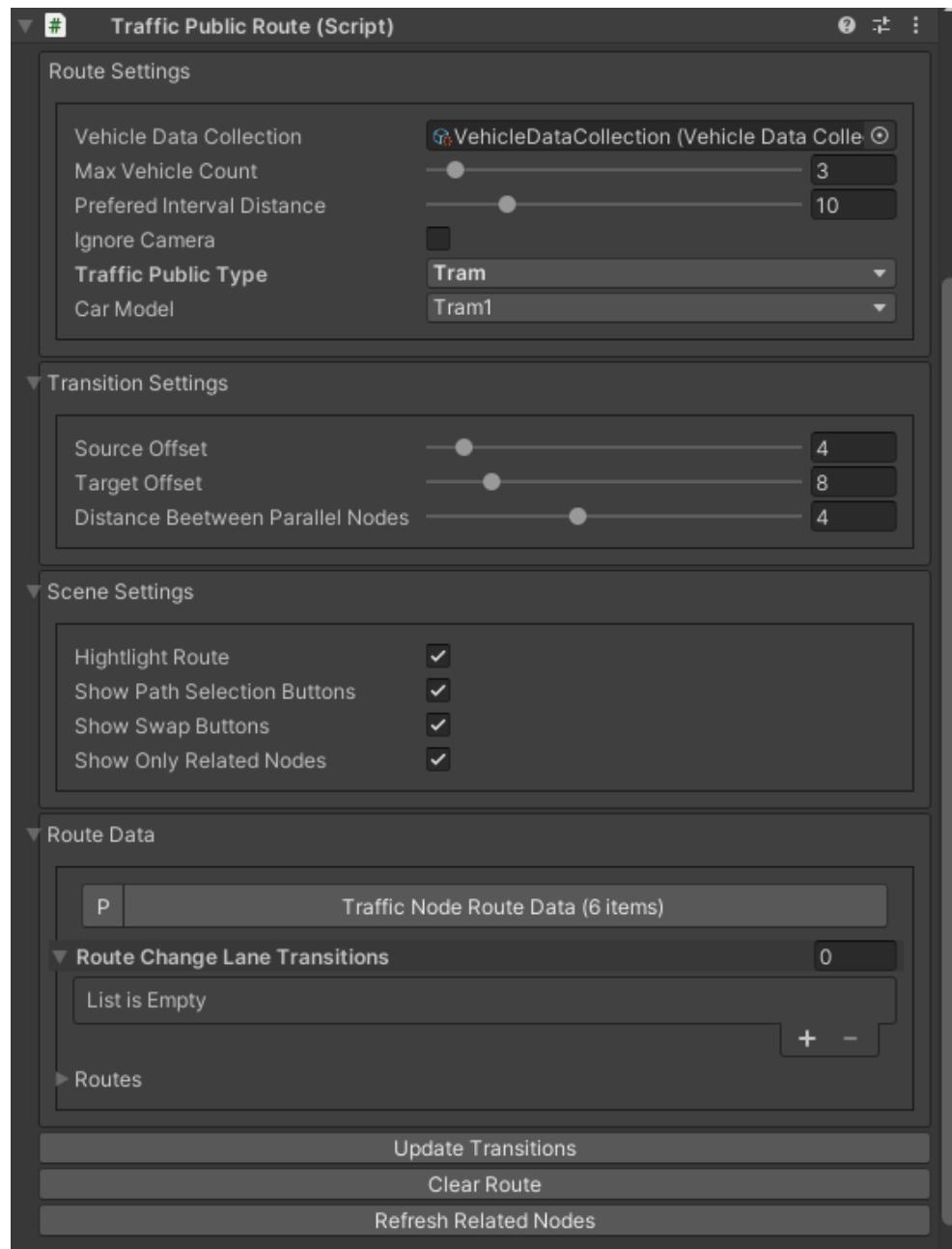


3. Customize *Transition settings*.



Transition result example.

2.12.3 Component



Route settings

Vehicle data collection : reference to the *vehicle collection*.

Max vehicle count : maximum number of vehicles on the route.

Preferred interval distance : preferred distance between public transport vehicles.

Ignore camera : if the camera is ignored, public transport can be spawned in view of the camera.

Traffic public type :

- **Bus** : for the default path.
- **Tram** : for the rail path (*rail movement* enabled by default).

Car model : *car model* of the public transport vehicle that will be spawned on the route.

Transition settings

Source offset : offset start point of transition in source path.

Target offset : offset end point of transition in target path.

Distance between parallel nodes : max distance between *traffic nodes* to find a transition path.

Scene settings

Highlight route : highlight added paths of route.

Show path selection buttons : on/off display add buttons paths to route.

Show swap buttons : show swap buttons for *transitions*.

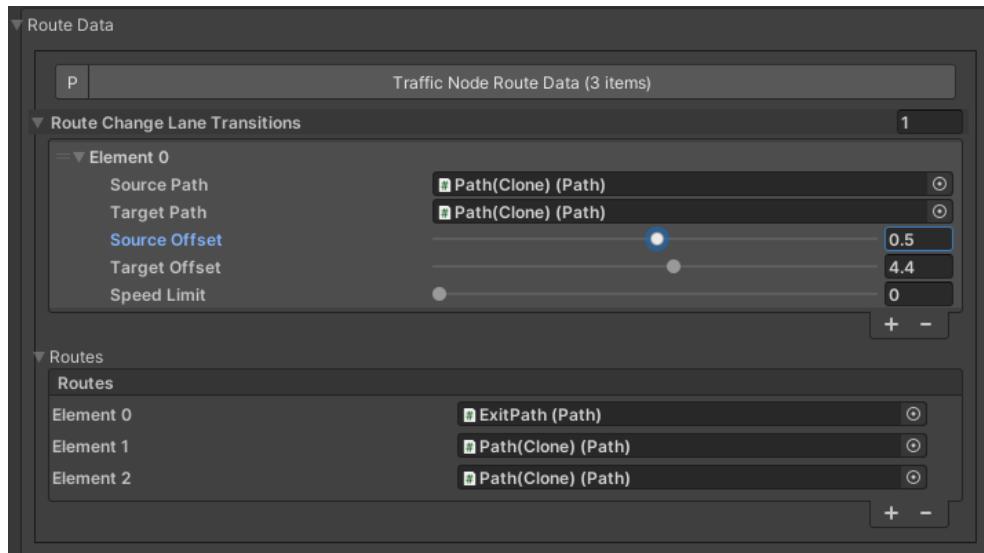
Show only related nodes : only nodes that are neighbours of nodes that have already been added will be displayed.

Route data

Traffic node route data : internal related traffic nodes route data.

Route change lane transitions : *transition* data.

Routes : sequence of paths on the route.



Transition data example.

Buttons

Update transitions

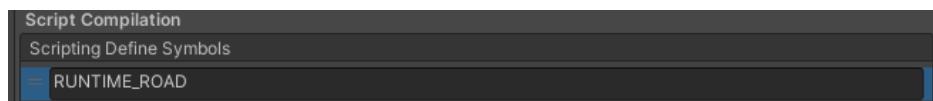
Clear route

Refresh related nodes

2.13 Runtime Road

2.13.1 Installation

- Add the *RUNTIME_ROAD* scripting define to the *Player Settings* of the project.

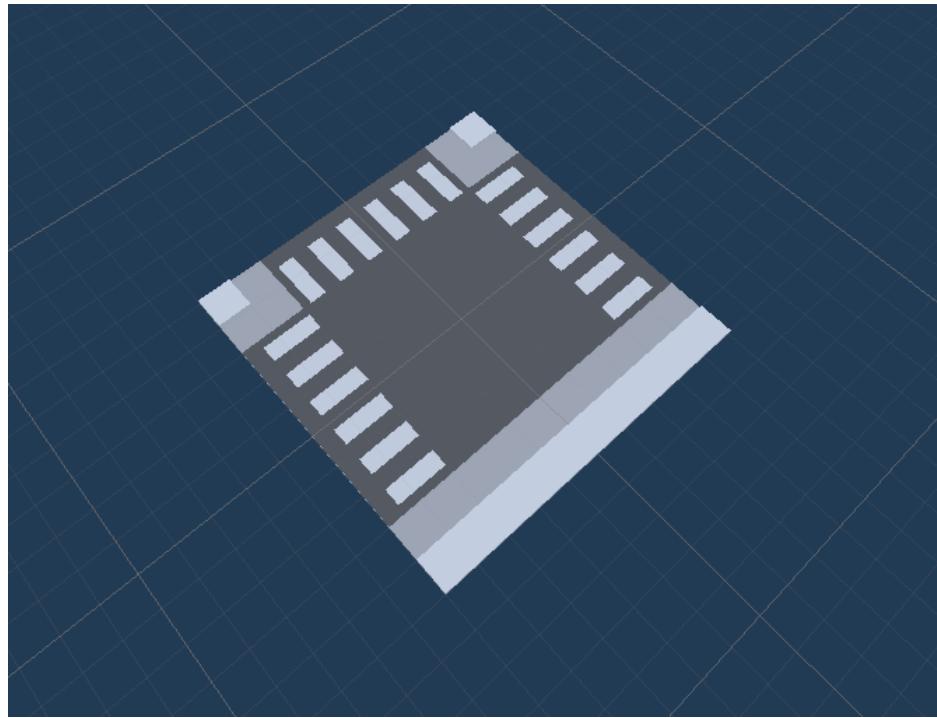


- Open the *RuntimeTileRoad Demo* to get started.

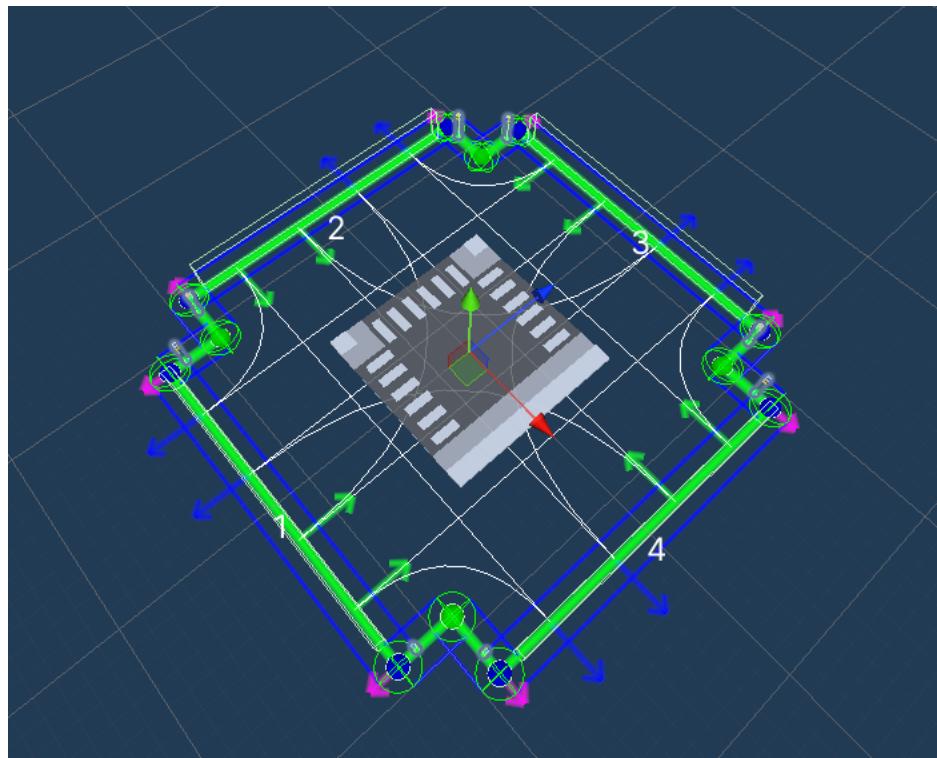
2.13.2 Tile Prefab

How To Create

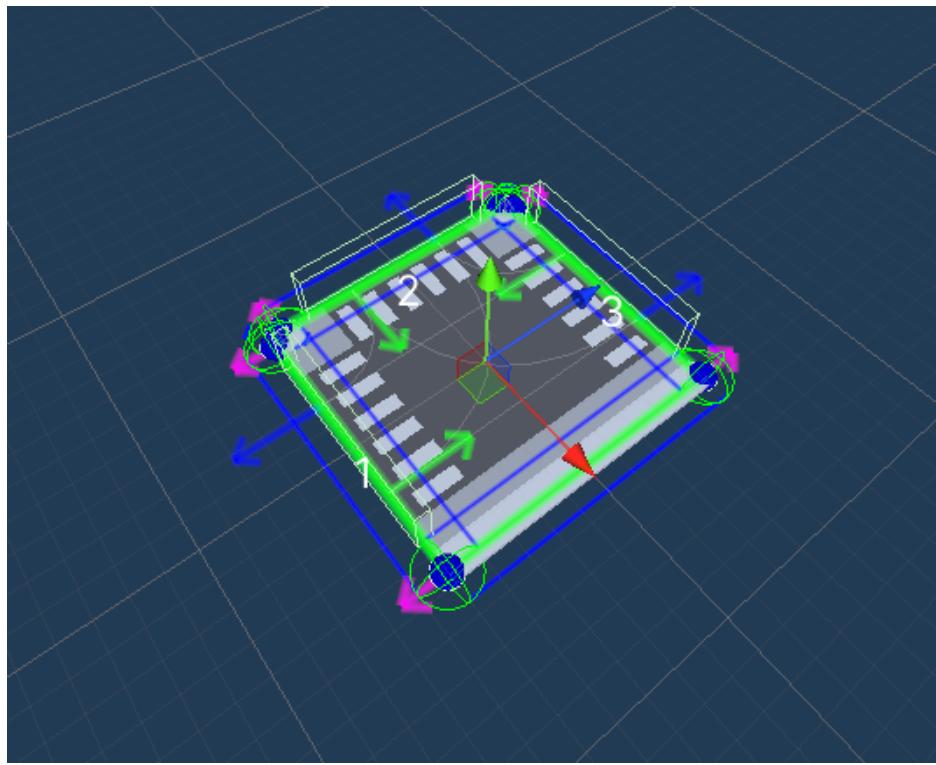
1. Open your prefab with your tile.



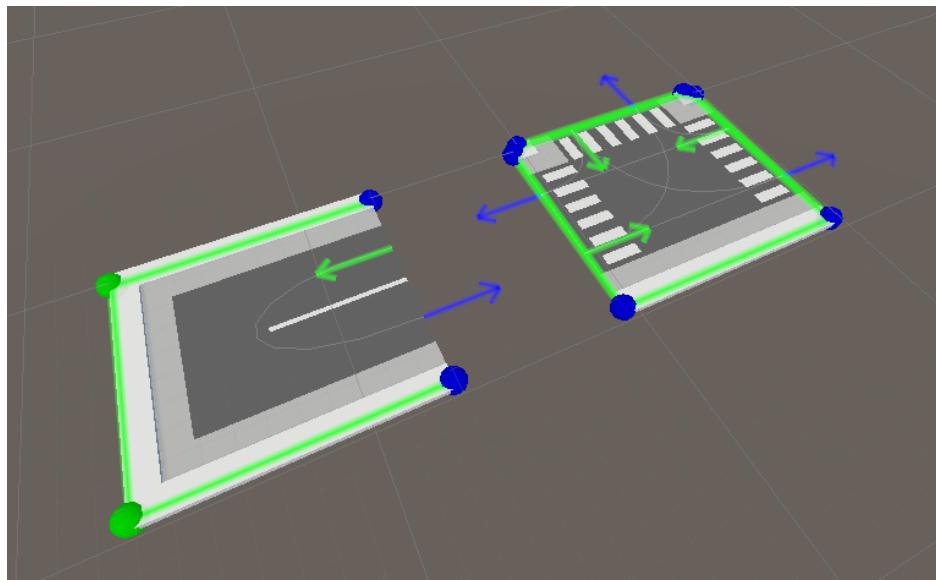
2. Create initial *Road segment* in the *Prefab stage*.



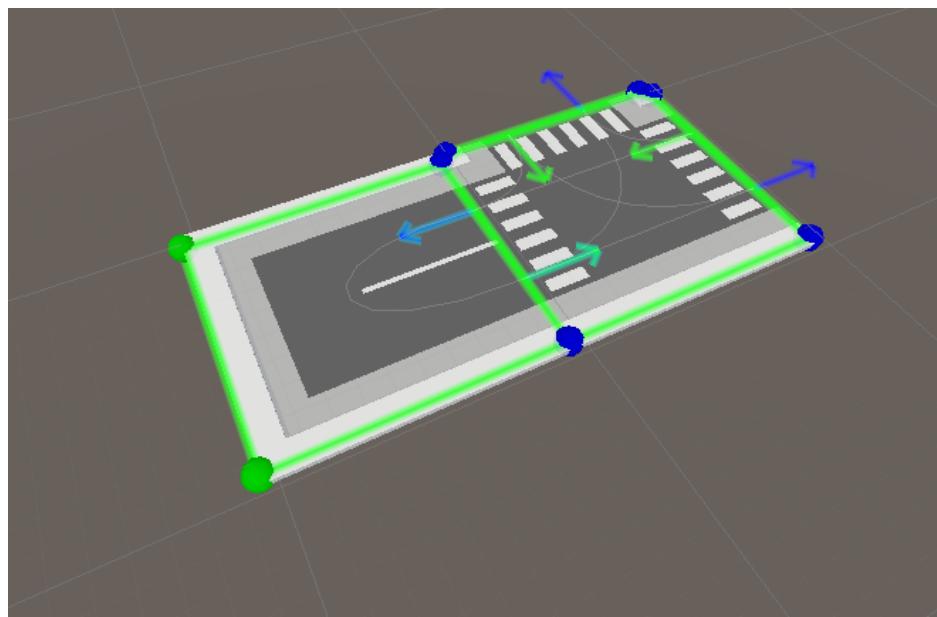
3. Adjust *segment type*, *number of lanes*, *lane width* and rotation to suit your tile.



4. *Traffic nodes & Pedestrian nodes* entrances/exits should overlap at the same position for all segments so that they can connect to other tiles at those positions.

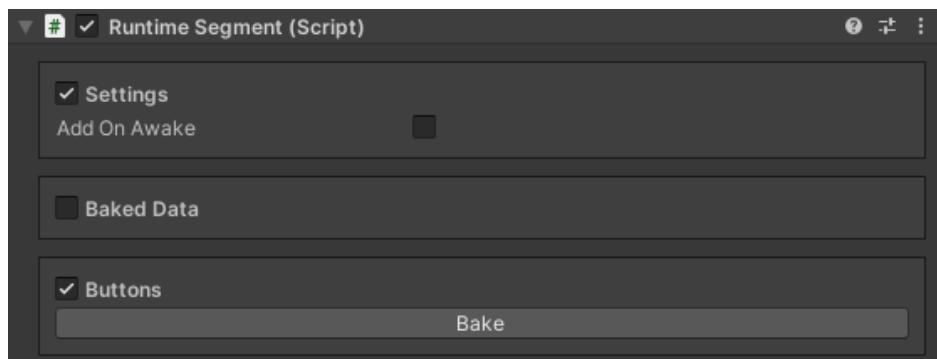


Source tiles example.



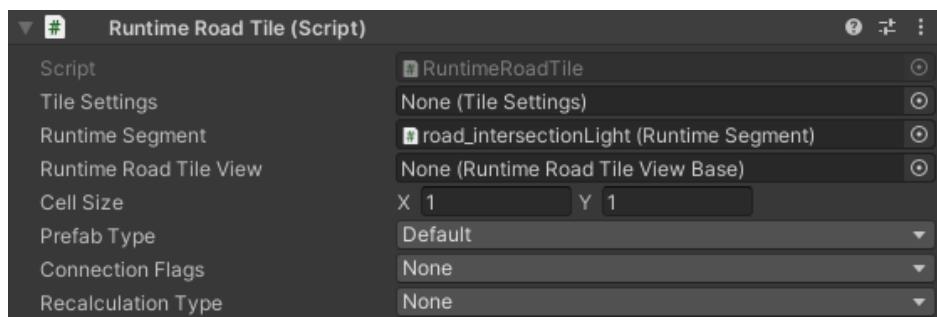
Example of overlap between traffic nodes and pedestrian nodes (Traffic node arrows & blue pedestrian node should overlap perfectly)

5. Add a *RuntimeSegment* component to the parent of the tile.

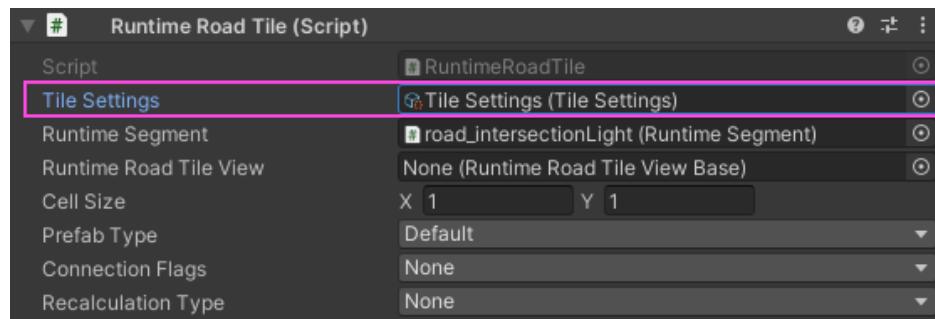


6. If you have made further changes to the road segment, press *Bake* in the *RuntimeSegment* to bake the data.

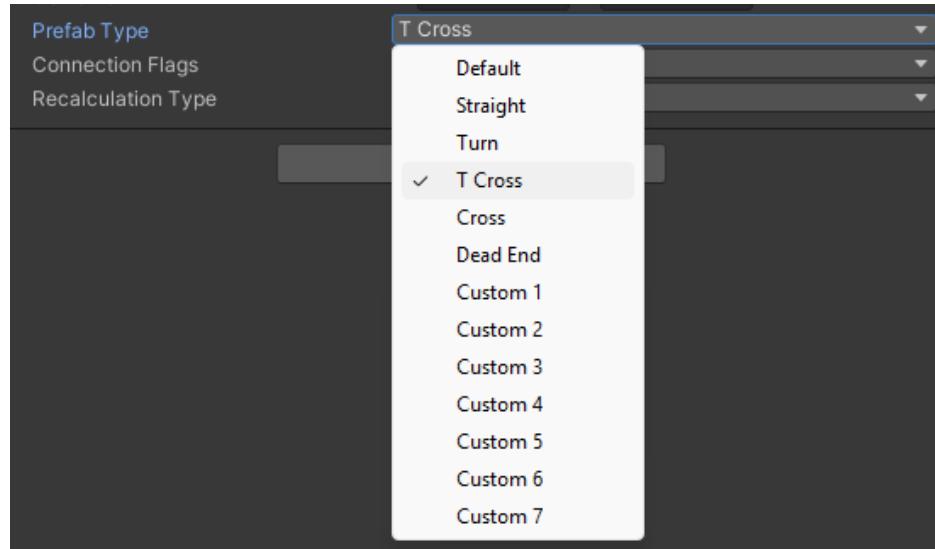
7. Add a *RuntimeRoadTile* component to the parent of the tile.



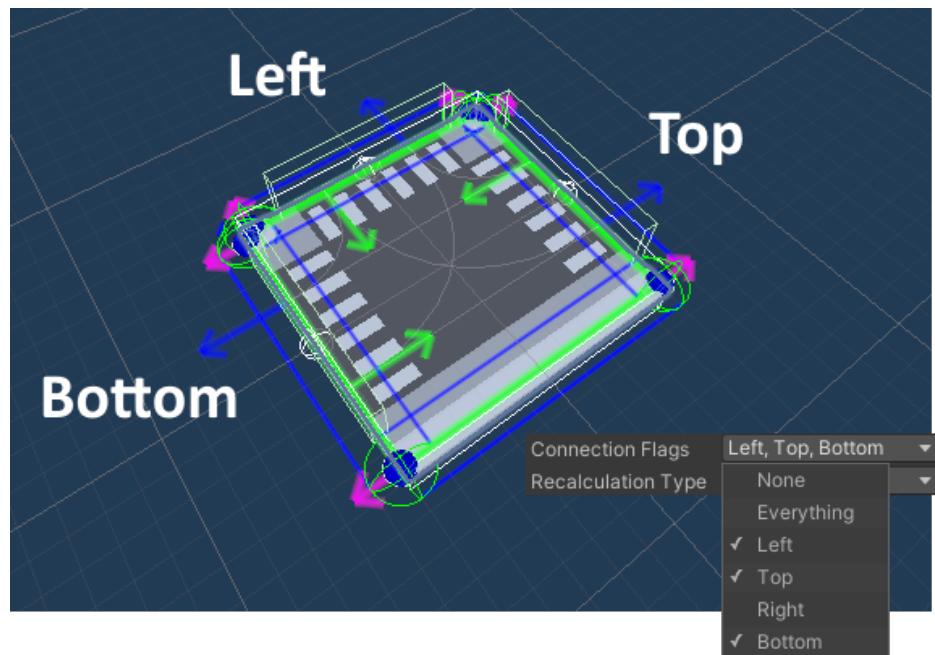
8. Assign a *Tile Settings* to the *RuntimeRoadTile*.



9. Select *Prefab type* according to the shape of the tile (custom types for custom shapes).

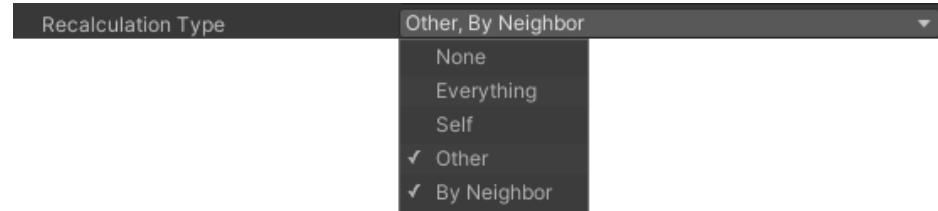


10. Select the direction of the connections if you want to use automatic tile replacement (currently only works for 1x1 tiles).

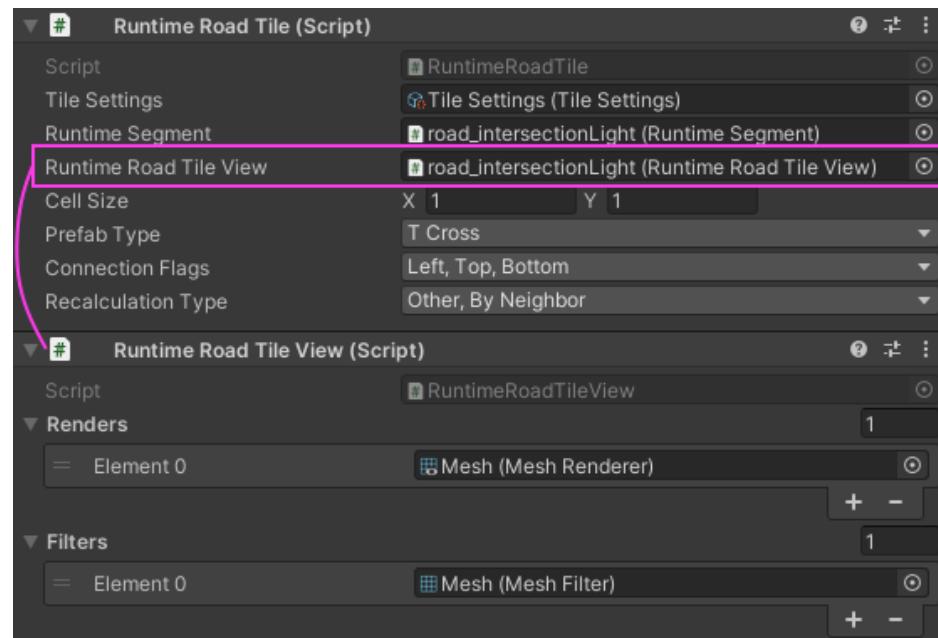


Connection example (circles on scene show direction)

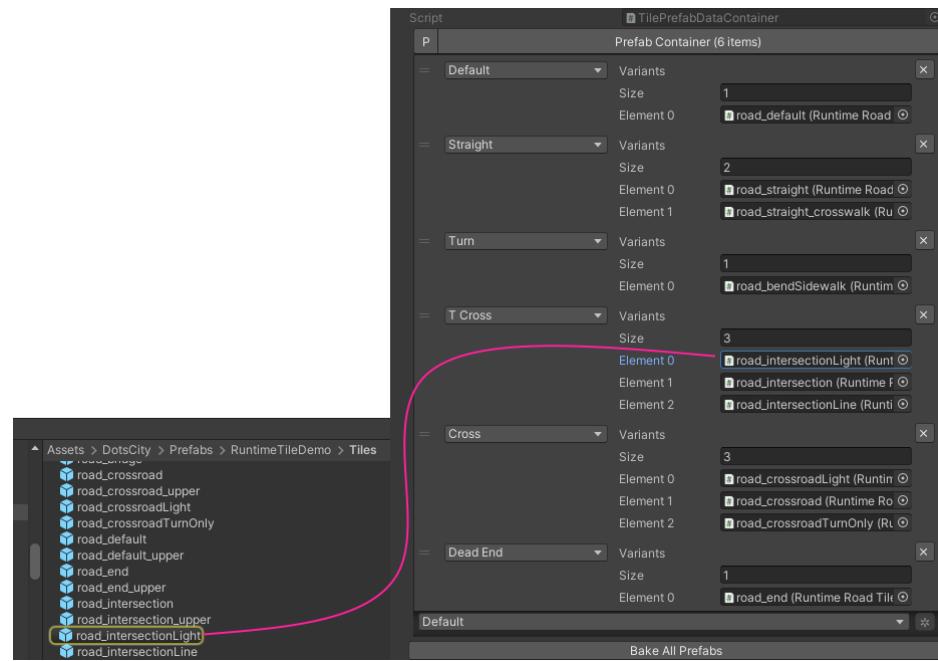
11. Select *Recalculation type*, if you want to use automatic tile replacement.



12. Add a *RuntimeRoadTileView* component & assign it to the *RuntimeRoadTile*.



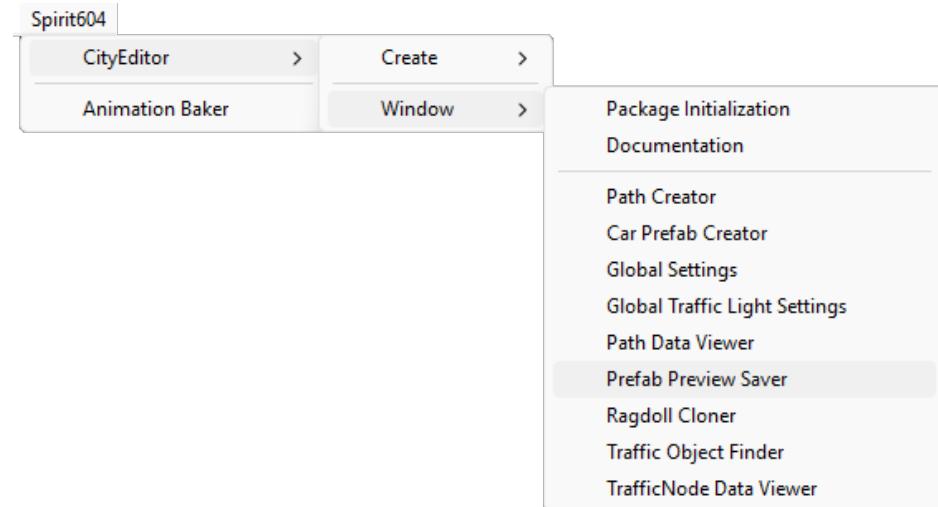
13. The next step is to assign it to a *Tile preset* according to its *Prefab type*.



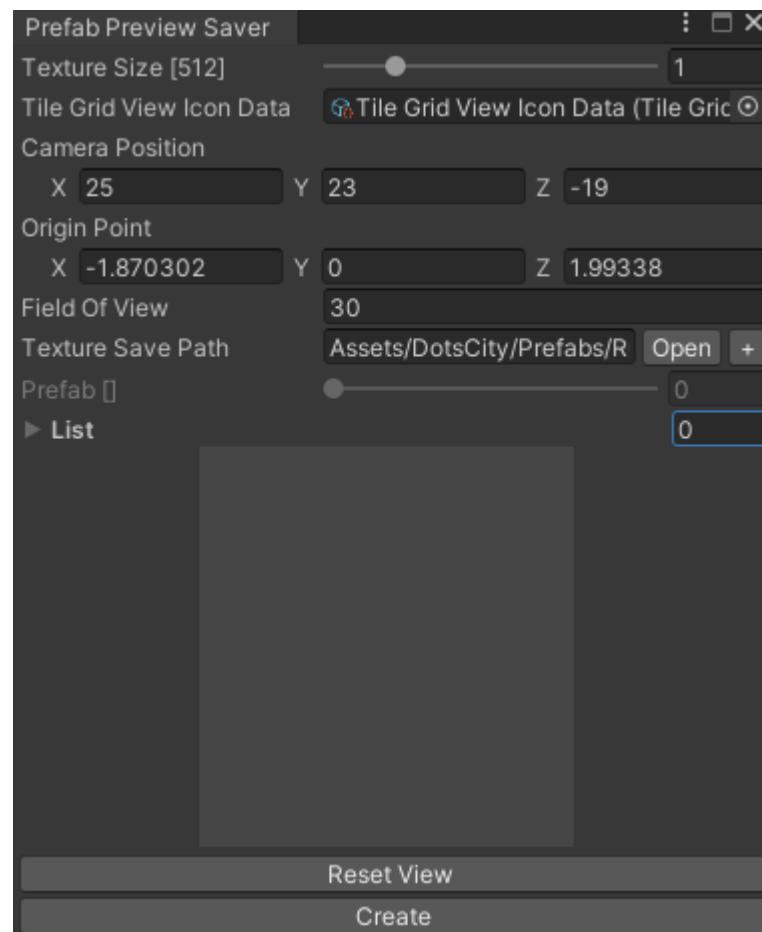
Preview Icon

To create a *Preview icon* for the *Prefab tile*, follow these steps:

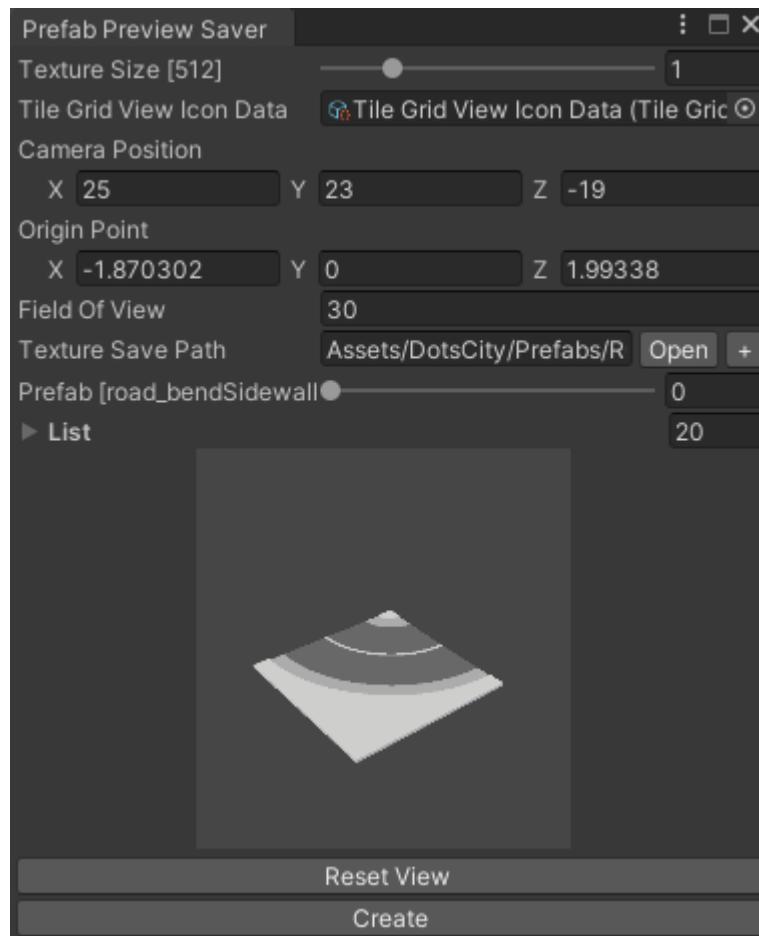
- Select from **Spirit604/CityEditor/Window/Prefab Preview Saver** the toolbar context menu.



- Drag & drop create tile prefabs into the list field.



- Adjust the camera position to adjust the *Preview icon*.



- Press *Create* Button.

How To Place

- If you are using your own placement logic, use this sample code:

```
[SerializeField] private RuntimeSegment runtimeSegmentPrefab;

private void Place(Vector3 position, Quaternion rotation)
{
    var runtimeSegment = Instantiate(runtimeSegmentPrefab, position, rotation);
    runtimeSegment.PlaceSegment();
}
```

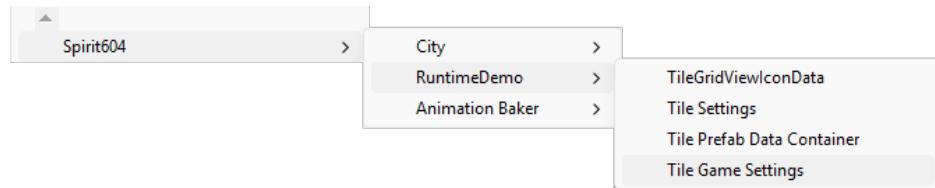
- Make sure *RuntimeRoadManager* is on the scene.

2.13.3 Tile Settings

Contain settings for the size of a single cell.

How To Create

- Select from Spirit604/RuntimeDemo/Tile Settings the project context.



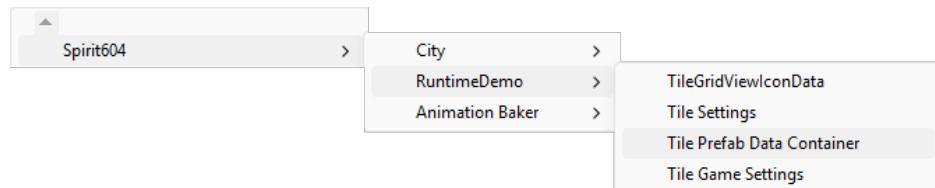
Where Is Used

- *RuntimeRoadTile*.
- *GridBoundsBase*.
- *TileGrid*.

2.13.4 Tile Preset

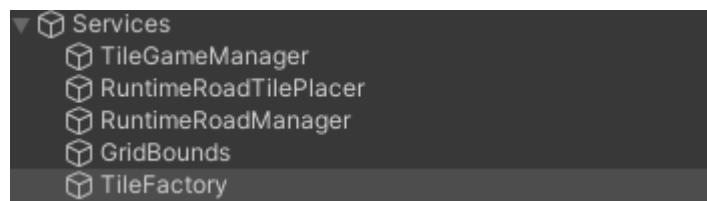
How To Create

- Select from Spirit604/RuntimeDemo/Tile Prefab Data Container the project context.

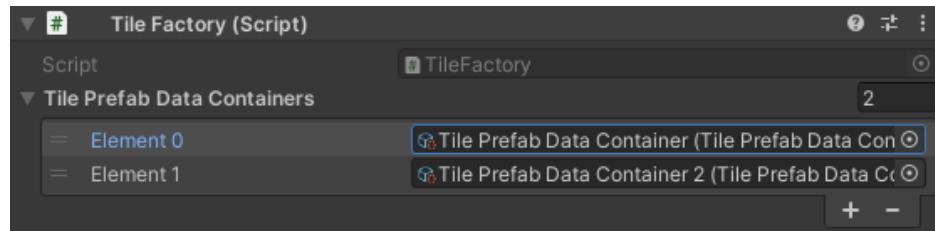


How To Assign

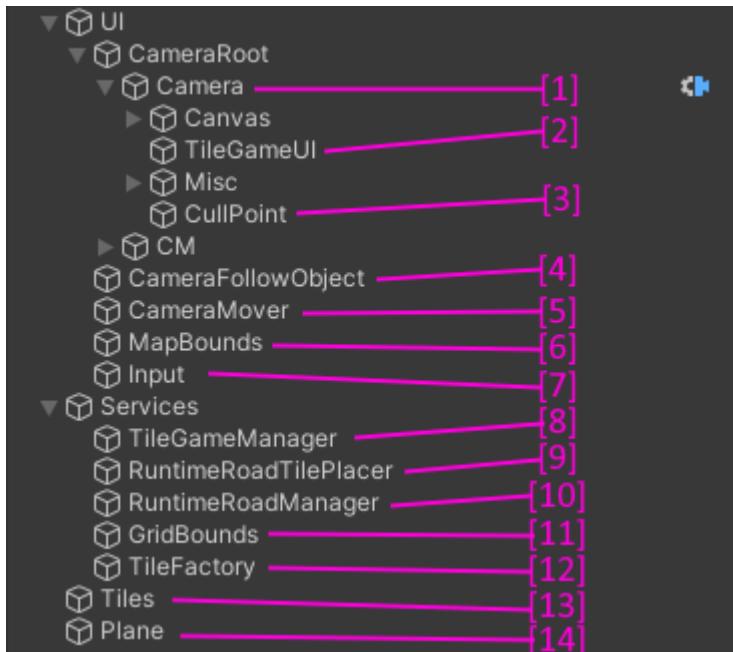
- Select *TileFactory* in the scene.



- Assign preset to the *TileFactory*.



2.13.5 RuntimeTileRoad Demo Structure



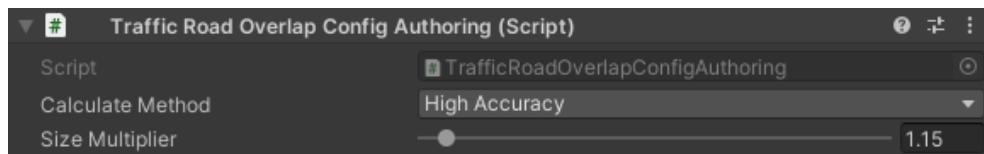
1. *Camera* : sample camera of Tile demo [optional].
2. *TileGameUI* : sample UI manager of the scene, used by *TileGameManager*. [optional, can be replaced by an implementation of `TileGameUIBase`].
3. *CullPoint* : *Cull point* of the scene [required].
4. *CameraFollowObject* : follow & aim point of the *Camera*, controlled by *CameraMover* [optional].
5. *CameraMover* : controls the movement of the aim point of the *Camera* [optional].
6. *MapBounds* : limits of the movement of the *CameraFollowObject*, used by *CameraMover* [optional].
7. *Input* : user input of the tile game, used by *TileGameManager* [optional, can be replaced by an implementation of `TileGameInputBase`].
8. *TileGameManager* : main sample tile manager, which controls all sample placement logic [required, if you are using RuntimeTileRoad demo scripts].
9. *RuntimeRoadTilePlacer*
 - [manager which responsible for tile layout on the scene [required, only if you are using RuntimeTileRoad demo scripts].]
 - *TileGrid* : map of tiles added to the scene. [optional, can be replaced by an implementation of `TileGridBase`].

- *GridSceneView* : grid display on the scene [**optional, can be replaced by an implementation of `GridSceneViewBase`**].
 - *PreviewService* : display a preview of an object on the scene before it is created [**optional, can be replaced by an implementation of `PreviewServiceBase`**].
10. *RuntimeRoadManager* : manager that converts *RuntimeSegment* road into an entity road graph. [**required**].
 11. *GridBounds* : limit of the map of cells available for the tile prefabs, used by *RuntimeRoadTilePlacer* [**optional, can be replaced by an implementation of `GridBoundsBase`**].
 12. *TileFactory* : factory containing all the *Tile* *presets*, used by *TileGameManager* [**optional, can be replaced by an implementation of `TileFactoryBase`**].
 13. *Tiles* : tile parent of the tiles, used by *RuntimeRoadTilePlacer* [**optional**].
 14. *Plane* : plane of the scene [**optional**].

2.14 Road Configs

2.14.1 Overlap Config

Config of overlapping of *traffic cars* with *traffic nodes*.



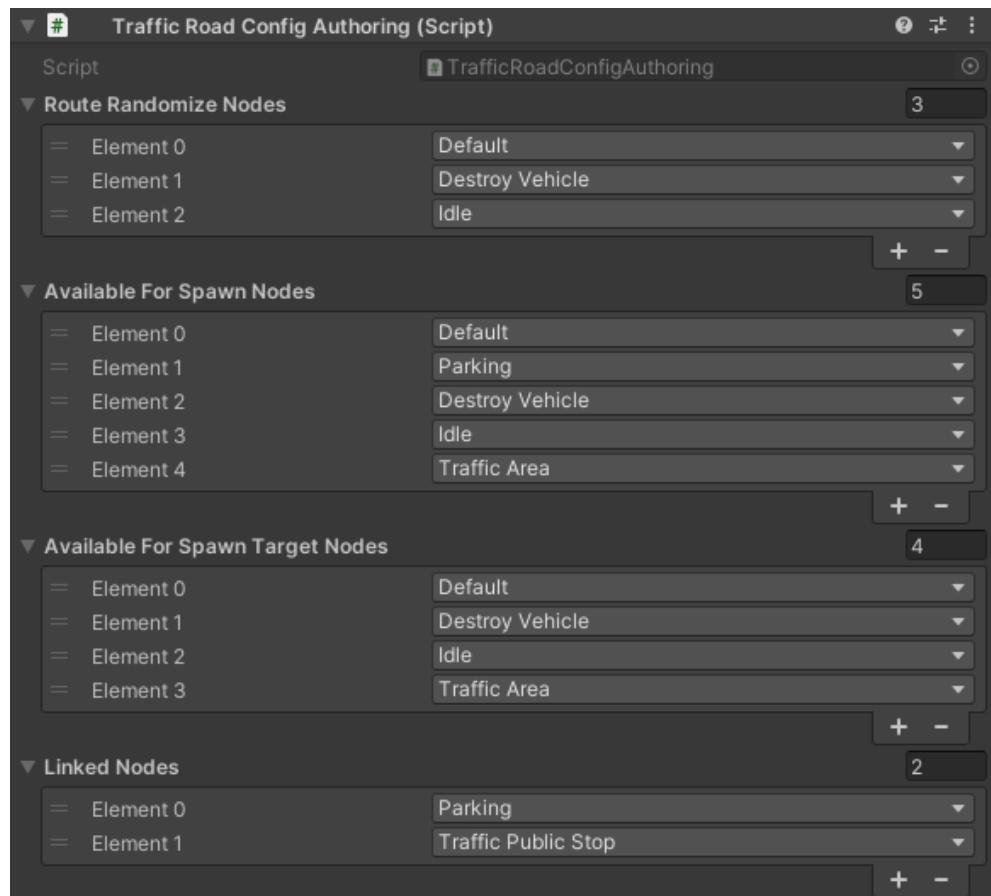
Calculate method:

- **Low accuracy** : low accuracy of calculation based on distance, but higher performance.
- **High accuracy** : higher accuracy of calculation based on bounds, but little bit lower performance.

Size multiplier : size multiplier of the *traffic car*.

2.14.2 Traffic Road Config

The config contains which categories each *traffic nodes* belongs to.



Route randomize nodes : list of allowed nodes to randomly spawn on the path between those nodes.

Available for spawn nodes : list of allowed *traffic nodes* where *traffic cars* can spawn.

Available for spawn target nodes : list of allowed *traffic nodes* that can be selected for a target at spawn.

Linked nodes : nodes that are linked to a specific *traffic car* which approaching a linked node (*parking example*).

TRAFFIC

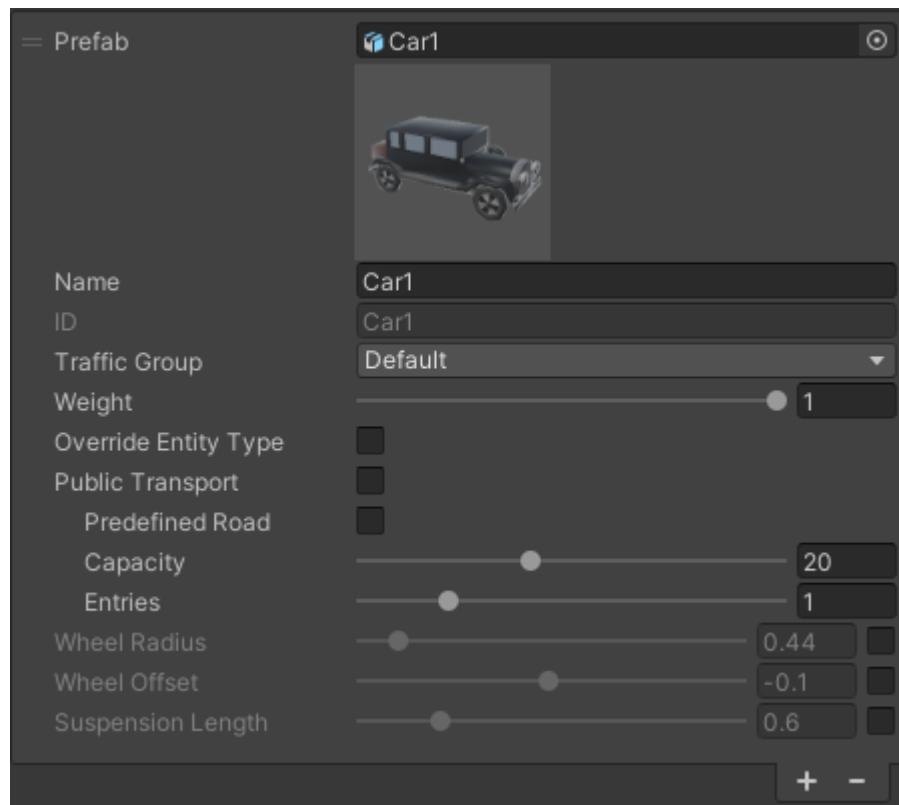
3.1 Traffic Car

- *How To Create*
- *Vehicle Physics Types*
 - *Custom Physics*
 - * *Authoring components*
 - *Vehicle Authoring*
 - *Wheel*
 - *Suspension*
 - *Friction*
 - *Transient Forces*
 - *Brakes*
 - *Engine*
 - *Scene Settings*
 - *Template Settings*
 - *Wheel Refs*
 - *PhysicsBody*
 - *PhysicsShape*
 - *Simple Physics*
 - * *Authoring components*
 - *CarWheelAuthoring*
 - *PhysicsBody*
 - *PhysicsShape*
 - *No Physics*
 - *Hybrid Mono*
 - * *Arcade (built-in sample)*

- * *Custom user controller*
- * *Input info*
- * *VehicleInput example code*
 - *Components*
- *Common Authoring Components*
 - *TrafficCarEntityAuthoring*
 - *Shared Settings*
 - *NavMeshObstacleAuthoring*
 - *CarSoundAuthoring*
 - *HealthAuthoring*
 - *CarDamageEngineAuthoring*
 - *PlayerTargetAuthoring*
- *CullState Info*
 - States
- *Parking*
 - *Entering parking states*
 - *Exiting parking states*
- *Avoidance*
- *Rail Movement*
- *Obstacle Detection*
 - *Raycast*

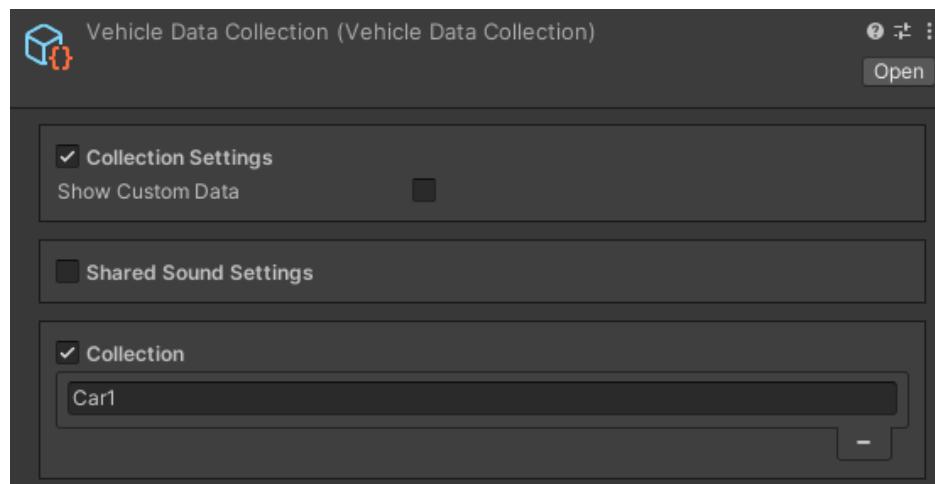
3.1.1 How To Create

1. Create a vehicle using the *Car Prefab Creator* tool (for example *Simple physics entity* type, if you need traffic with controller based on *MonoBehaviour*, read this [article](#)).

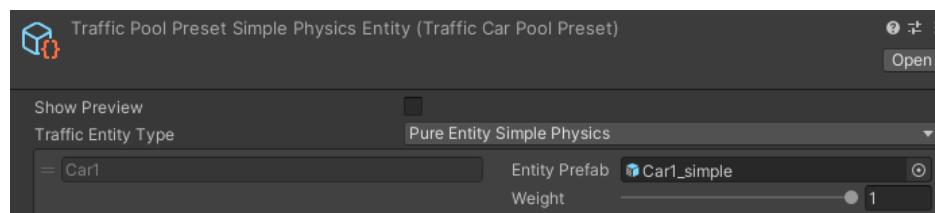


Car Prefab Creator final step example.

2. Make sure, that the *vehicle collection* contains vehicle you have created.

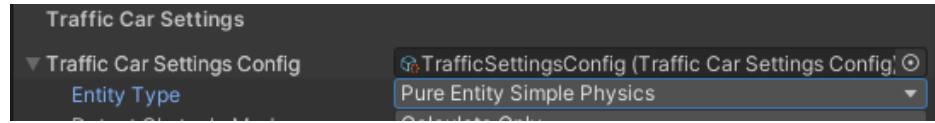


3. Make sure, that the *preset* contains the vehicle.

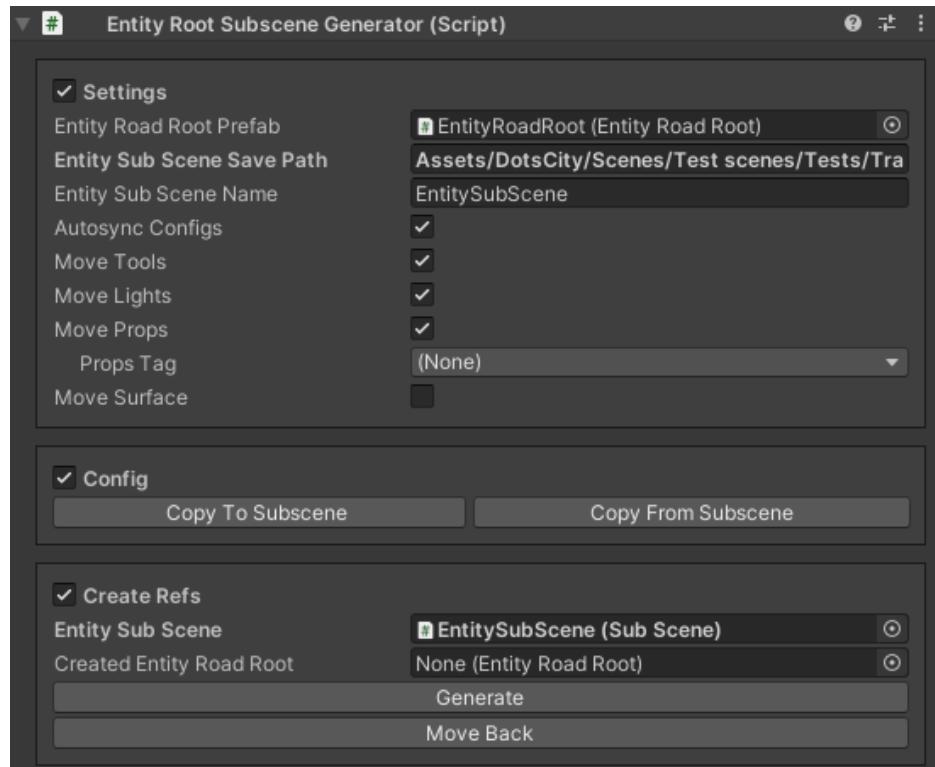


4. Open the *Traffic settings* and select the *Simple physics entity* type (if you have created a *Simple*

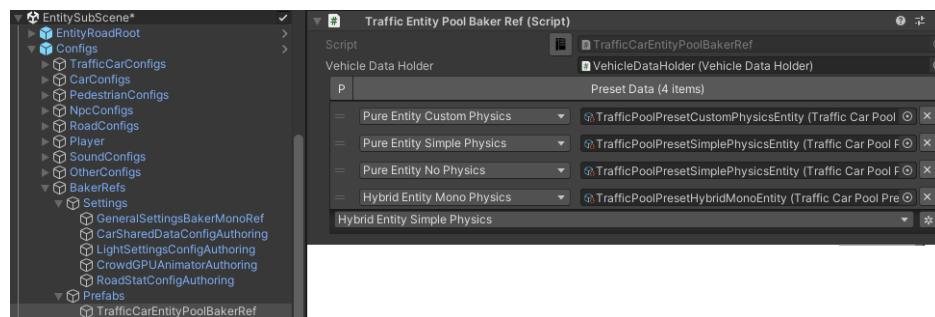
physics entity).



5. Open the *Hub* object in the scene and press the *Copy To Subscene* button.



6. Open the *EntitySubScene*, find the *TrafficCarEntityPoolBakerRef* gameobject on the main & sub-scene, make sure the correct *preset* is assigned to both scenes.



7. Adjust the traffic settings of the created vehicles.

- For *Simple physics*.
- For *Custom physics*.

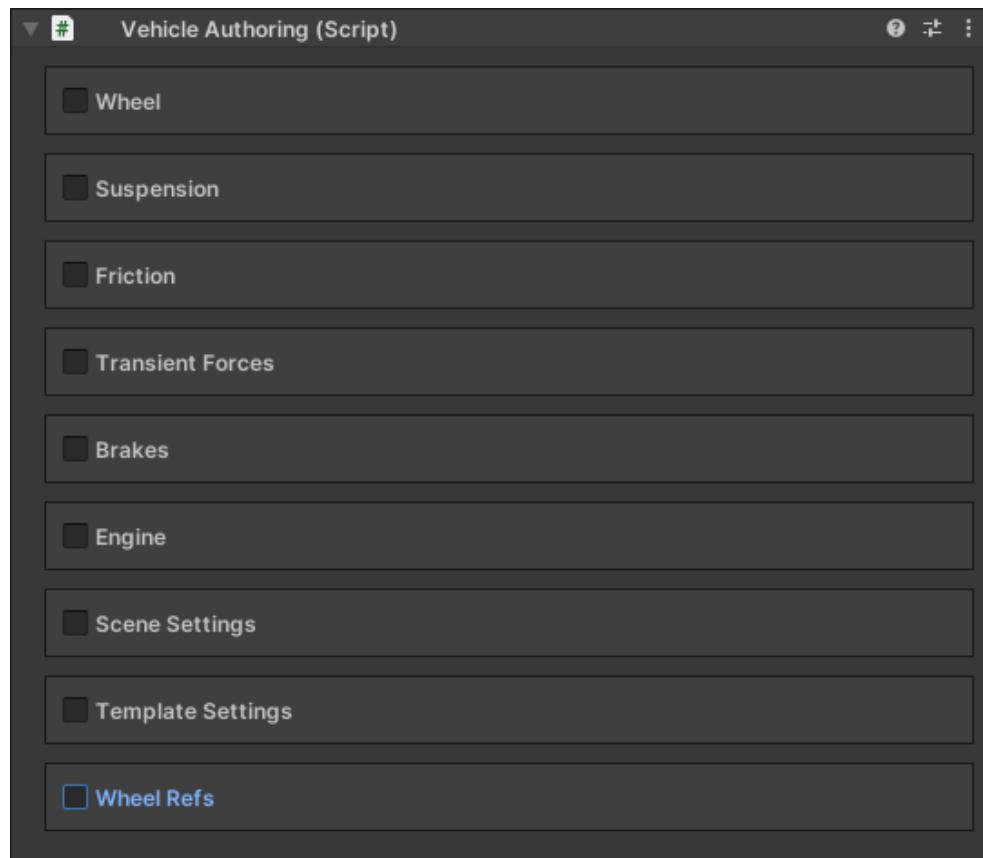
3.1.2 Vehicle Physics Types

Custom Physics

- Entities that are moved by the custom physical system.
- *Hybrid Entity Custom Physics & Pure Entity Custom Physics* types refer to this.
- YouTube tutorial.

Authoring components

Vehicle Authoring



Wheel

<input checked="" type="checkbox"/> Wheel	
Wheel Mass	20
Max Steering Angle	35
Power Steering	1.3
Custom Steering Limit	<input type="checkbox"/>
Radius	0.4
Width	0.2
Apply Impulse Offset	1
Cast Type	Ray ▾
Cast Layer	0: (Undefined Physics Category) ▾

Wheel mass : wheel mass.

Max steering angle : max steering angle of the steering wheel in degrees.

Power steering : rate of steering improvement.

Custom steering limit : limiting steering angle based on vehicle speed (Y-axis rate value (1 - max steering angle), X-axis speed in metres per second).

Radius : wheel radius.

Width : wheel width.

Apply impulse offset : applying force offset relative to lower point of wheel (without offsetting the force applied to the lower point of the wheel).

Cast type:

- **Ray** : raycast by ray.
- **Collider** : raycast by collider (collider size based on wheel radius and width).

Cast layer : physical layer that collides with the wheel.

Suspension

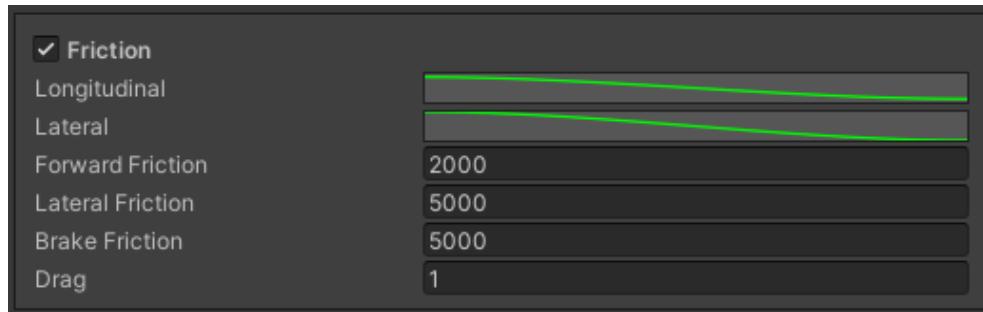
<input checked="" type="checkbox"/> Suspension	
Suspension Length	0.6
Stiffness	20
Damping	4000

Suspension length : length of suspension.

Stiffness : spring stiffness of suspension.

Damping : force to return spring to its original length.

Friction



Longitudinal : forward friction curve (Y-axis - forward slip value, X-axis forward speed in metres per second).

Lateral : lateral friction curve (Y-axis - lateral slip value, X-axis lateral speed in metres per second).

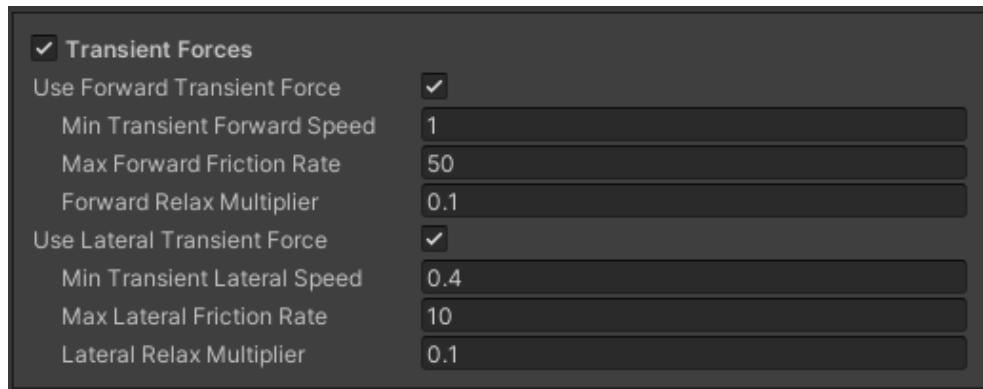
Forward friction : forward friction value.

Lateral friction : lateral friction value.

Brake friction : brake friction value.

Drag : drag value of the vehicle.

Transient Forces



Transient force is required to hold the car on an inclined ramp during manual braking.

Use forward transient force : on/off forward transient force.

Min transient forward speed : min forward speed when transient force is applied.

Max forward friction rate : max friction for transient force calculated by multiplying the entered rate by the forward friction.

Forward relax multiplier : step of forward force increase per frame.

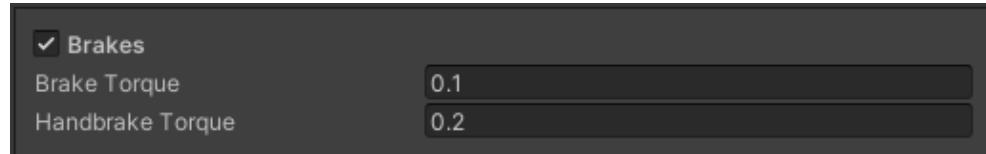
Use lateral transient force : on/off lateral transient force.

Min transient lateral speed : min lateral speed when transient force is applied.

Max lateral friction rate : max friction for transient force calculated by multiplying the entered rate by the lateral friction.

Lateral relax multiplier : step of lateral force increase per frame.

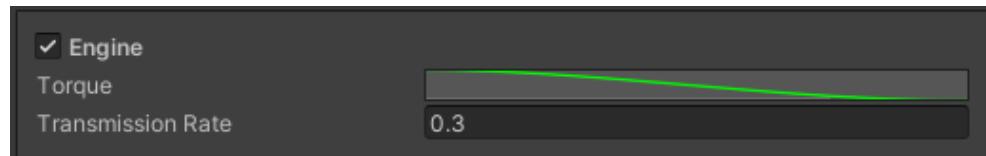
Brakes



Brake torque : torque of brake.

Handbrake torque : torque of handbrake.

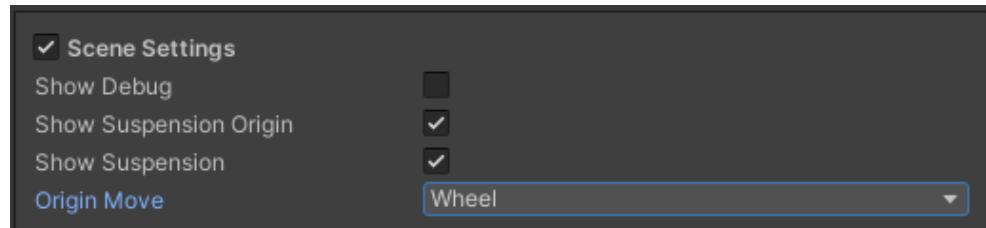
Engine



Torque : engine torque.

Transimission rate : engine torque to wheel speed ratio.

Scene Settings



Show debug : on/off debugging for wheel and suspension at runtime (*VehicleCustomDebugger*).

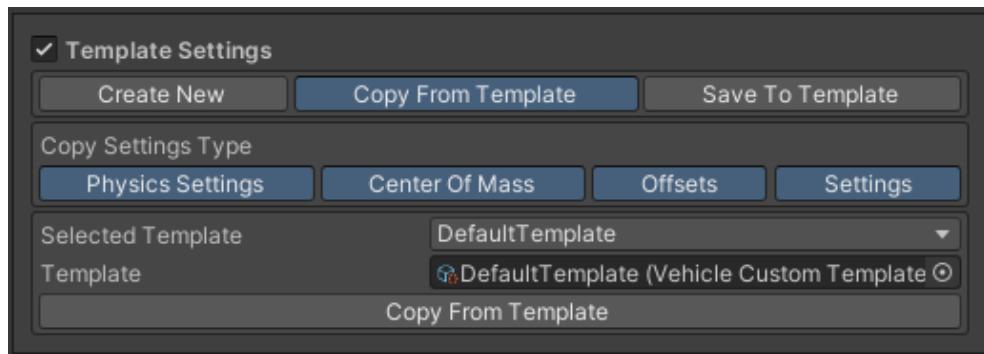
Show suspension origin : on/off display of suspension origin.

Show suspension : on/off display of suspension.

Origin move:

- **Disabled** : disabled handle.
- **Wheel** : on/off handle for wheel origin.
- **Suspension origin** : on/off handle for suspension origin.
- **Suspension** : on/off handle for suspension and wheel origin.

Template Settings



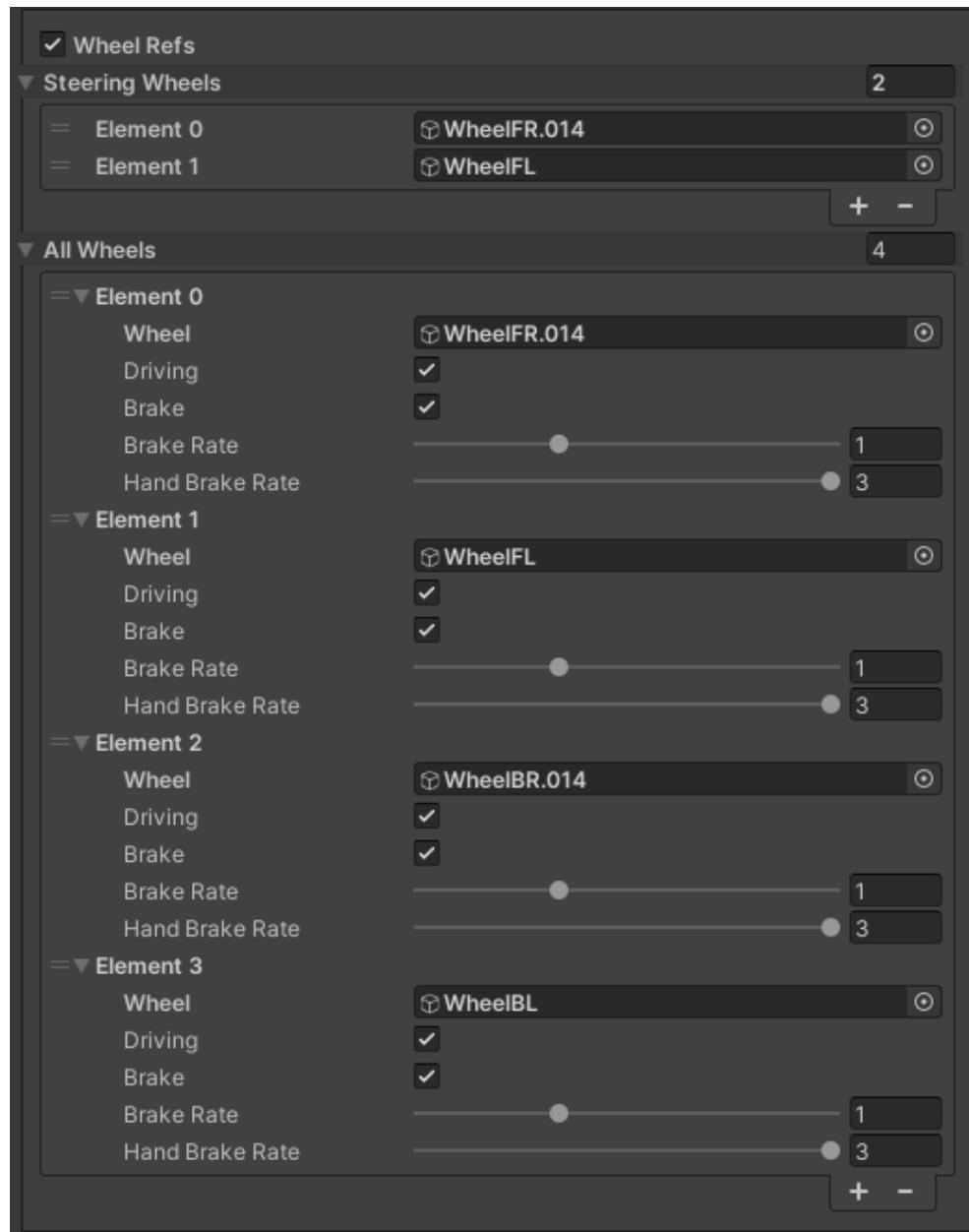
Tabs:

- **Create new** : create new template settings.
- **Copy from template** : copy settings from the selected template.
- **Save to template** : save settings to the selected template.

Copy settings type:

- **Physics settings** : copy the physics settings (mass, damping, gravity) of the *PhysicsBody* component.
- **Center of mass** : copy the center of mass local position of the *PhysicsBody* component.
- **Offsets** : copy the local offset of the wheels.
- **Settings** : copy the settings of the *VehicleAuthoring* component.

Wheel Refs



Wheel : reference to wheel.

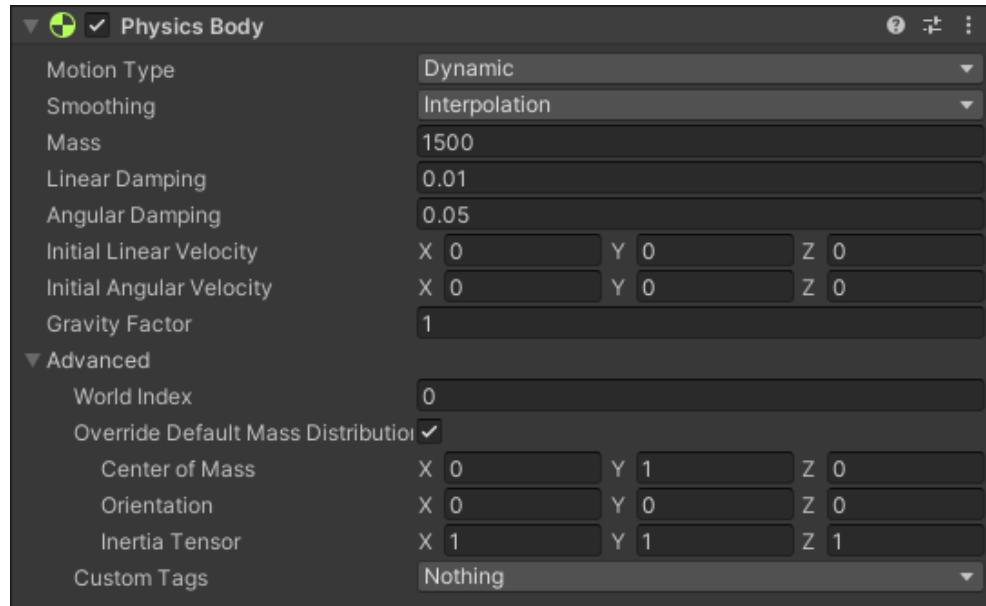
Driving : on/off driving force for the wheel.

Brake : on/off braking force for the wheel.

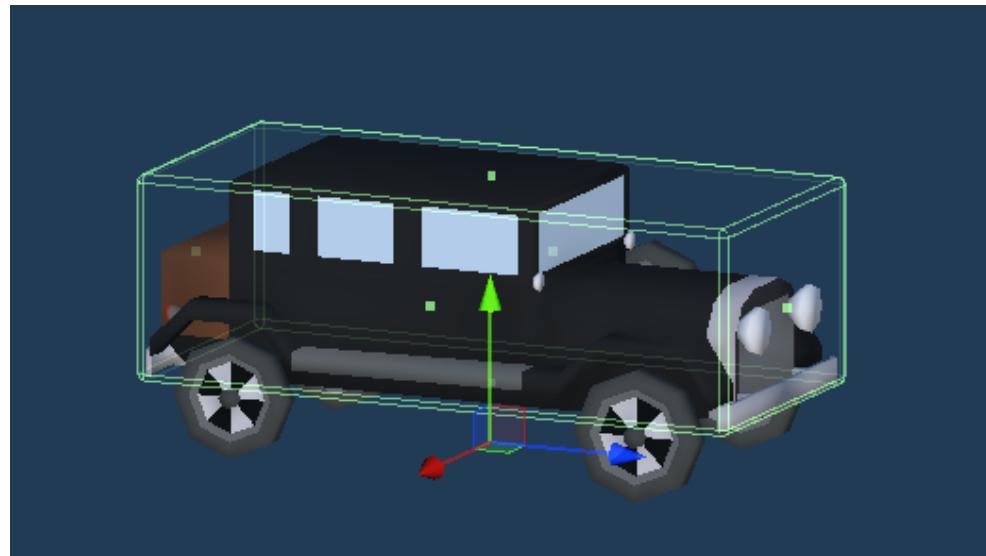
Brake rate : brake rate.

Handbrake rate : handbrake rate.

PhysicsBody



PhysicsShape



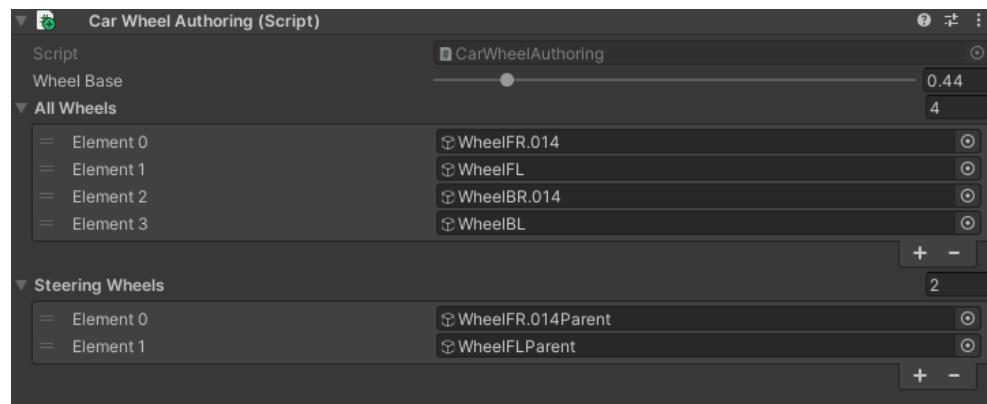
Example

Simple Physics

- Entities moved by the simple physical system (by simply adding the physics velocity to the physics body).
- *Settings*.
- *Hybrid entity simple physics & Pure entity simple physics* types refer to this.

Authoring components

CarWheelAuthoring

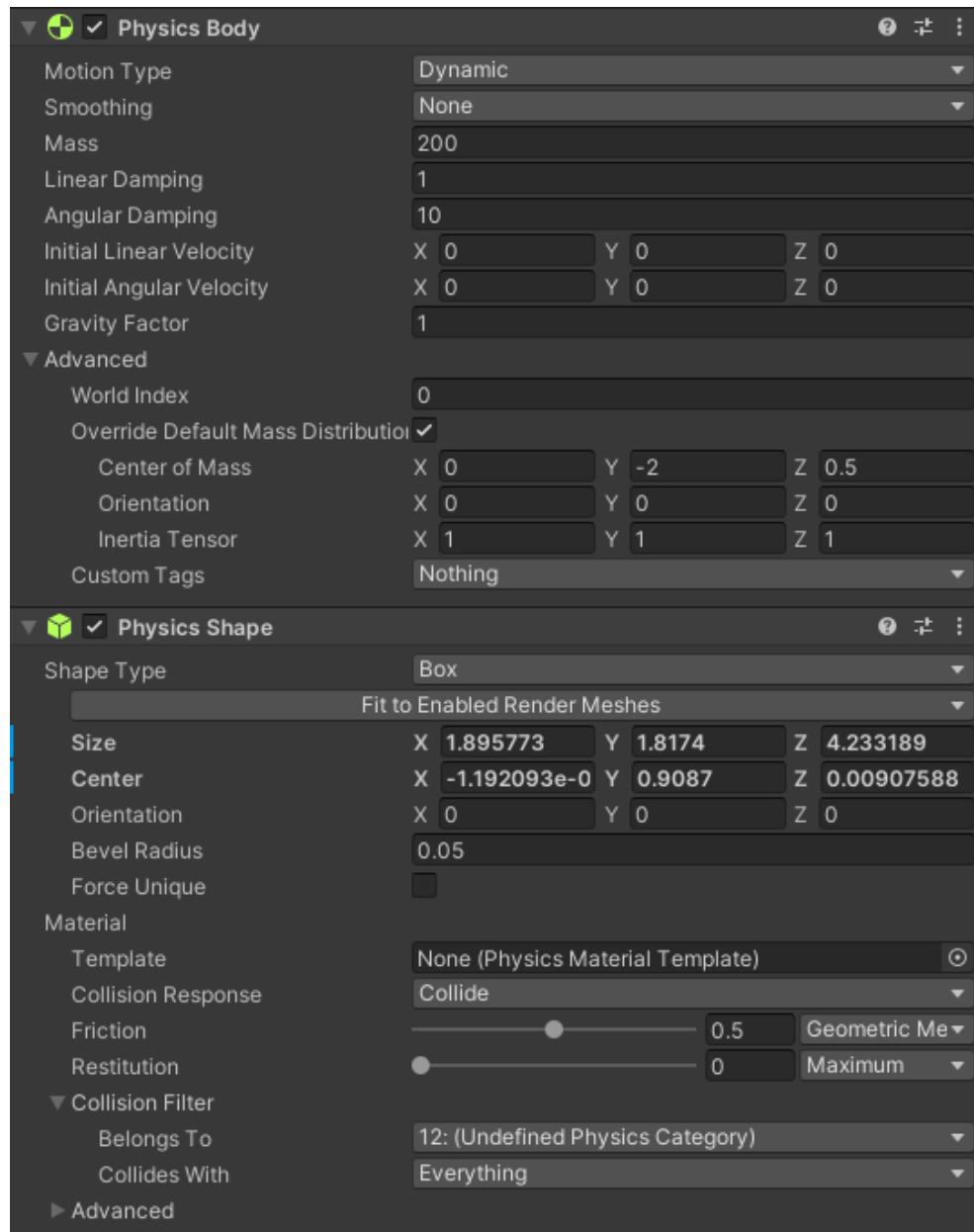


Wheel base : wheel radius.

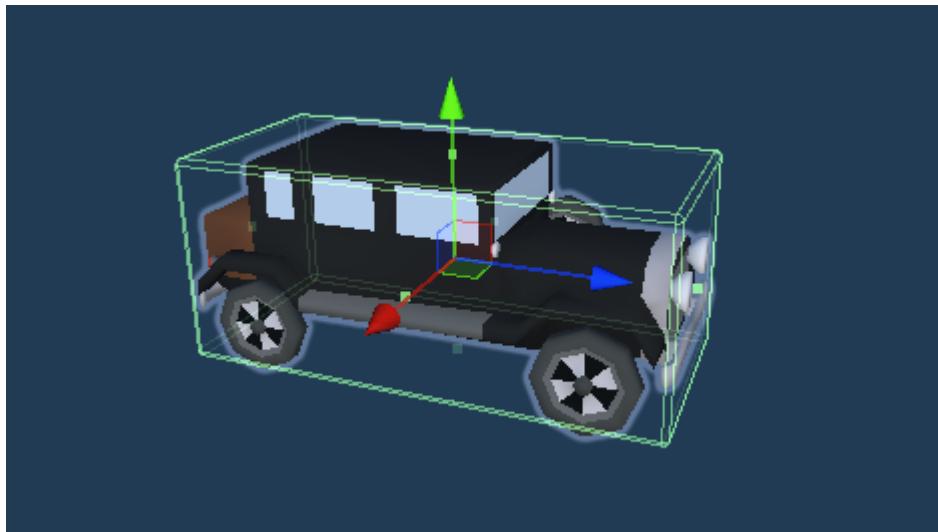
All wheels : all wheels of the vehicle.

Steering wheels : wheels that can turn.

PhysicsBody



PhysicsShape



Example

Optional components if the car moves with physics.

No Physics

- *Pure entities* that moved by transform system without physics.
- Contains the same components as *Simple Physics*.
- *Settings*.
- *Pure entity no physics* type refer to this.

Hybrid Mono

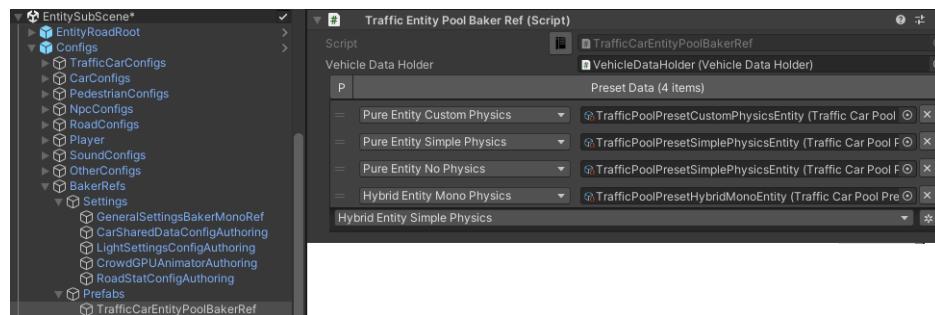
- Before using this vehicle type, make sure that you selected *World simulation type* to *Hybrid mono* in the *General settings*.
- *Hybrid entities* that moved by custom monobehaviour controller.
- *Hybrid entity mono physics* type refer to this.

Arcade (built-in sample)

If you want to use *Arcade*'s built-in sample controller, follow these steps:

1. Open the *Car Prefab Creator* tool.
2. Select *Prefab* tab.
3. Set *Car type* to *Traffic*.
4. Drag & drop your prefabs into the prefabs field (make sure your source *Prefabs* don't have any *Colliders*, *Wheel colliders* & *Rigidbodies*).
5. Press *Scan* button.

6. Select *Save* tab.
7. Set *Entity type* to *Hybrid entity mono physics*.
8. Set *Controller type* to *Arcade*.
9. Set desired preset. Also hull & save paths.
10. Select *Prefab info* tab.
11. Customize hull & wheel offsets.
12. Customize traffic related settings.
13. Press *Create* button.
14. In the *ArcadeVehicleController*, make sure the raycast layer matches your *Ground* layer.
15. Open the *EntitySubScene*, find the *TrafficCarEntityPoolBakerRef* gameobject on the main & sub-scene, make sure the correct *preset* is assigned to both scenes.

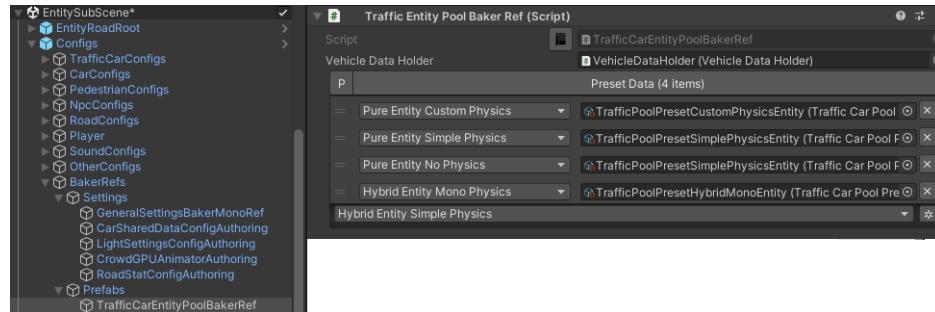


Custom user controller

If you want to use your own custom solution, follow these steps:

1. Open the *Car Prefab Creator* tool.
2. Select *Prefab* tab.
3. Set *Car type* to *Traffic*.
4. Drag & drop your prefabs into the prefabs field.
5. Press *Scan* button.
6. Select *Save* tab.
7. Set *Entity type* to *Hybrid entity mono physics*.
8. Set *Controller type* to *Custom user*.
9. Create script that implements *IVehicleInput* interface to link traffic input & your vehicle controller input (code example below)
10. Assign created script into the *Input script* field.
11. Set desired preset. Also hull & save paths.
12. Select *Prefab info* tab.
13. Set the steering angle to match the steering of your custom car controller.
14. Customize traffic related settings.

15. Press *Create* button.
16. To set your own custom Car Controller parameters to handle the most common traffic situations, use the *Traffic test scene*.
17. Open the *EntitySubScene*, find the *TrafficCarEntityPoolBakerRef* gameobject on the main & sub-scene, make sure the correct *preset* is assigned to both scenes.



Input info

- Throttle [1] : forward motion.
- Throttle [-1] : reverse motion.
- Throttle [0] : hand brake.
- Throttle [-0.9] : braking, for example, if the current speed is higher than permitted. (the value can be changed in the *Traffic settings*)
- Steering [-1, 1]

VehicleInput example code

```
public class UVC_adapter : MonoBehaviour, IVehicleInput
{
    // Replace by your custom controller script.
    // The example uses a "Universal Vehicle Controller".
    // https://assetstore.unity.com/packages/tools/physics/
    ↪universal-vehicle-controller-plus-176314
    public UVC_AIControl carControllerInput;

    public float Throttle
    {
        get => carControllerInput.Acceleration;
        set
        {
            if (value > 0)
            {
                carControllerInput.
            ↪SetAcceleration(value);
                carControllerInput.SetBrakeReverse(0);
                carControllerInput.
            ↪SetHandBrake(false);
        }
    }
}
```

(continues on next page)

(continued from previous page)

```

        }
    else
    {
        carControllerInput.SetAcceleration(0);
        carControllerInput.

        ↪SetBrakeReverse(Mathf.Abs(value));

        if (value == 0)
            carControllerInput.

        ↪SetHandBrake(true);
    }
}

public float Steering { get => carControllerInput.Horizontal; ↴
↪set => carControllerInput.SetSteer(value); }

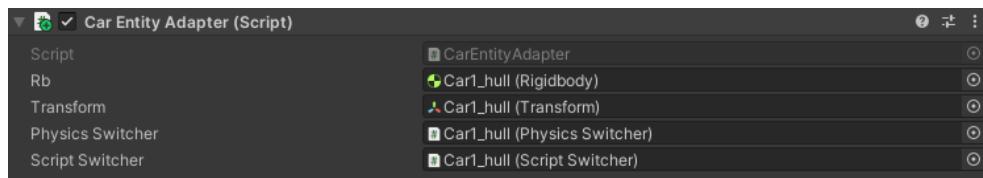
public bool Handbrake { get => carControllerInput.HandBrake; ↴
↪set => carControllerInput.SetHandBrake(value); }

public void SwitchEnabledState(bool isEnabled)
{
    carControllerInput.enabled = isEnabled;
}
}

```

Components

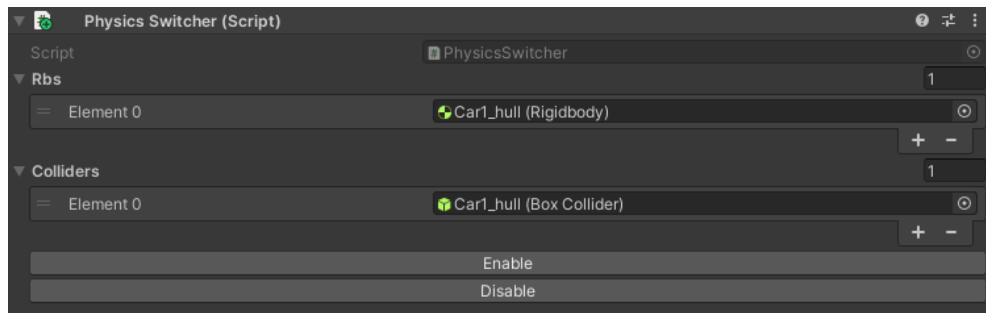
CarEntityAdapter



Component to synchronize gameobject & entity, also to switch physics & scripts of the *vehicle* when *cull state* changes with *PhysicsSwitcher* & *ScriptSwitcher* components.

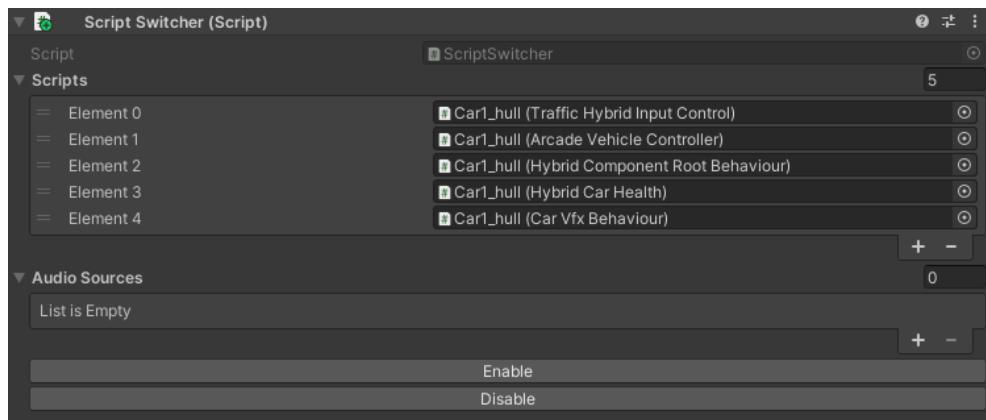
Note: You can turn off traffic physics culling in the *Traffic settings*.

PhysicsSwitcher



Component to on/off physics of the vehicle.

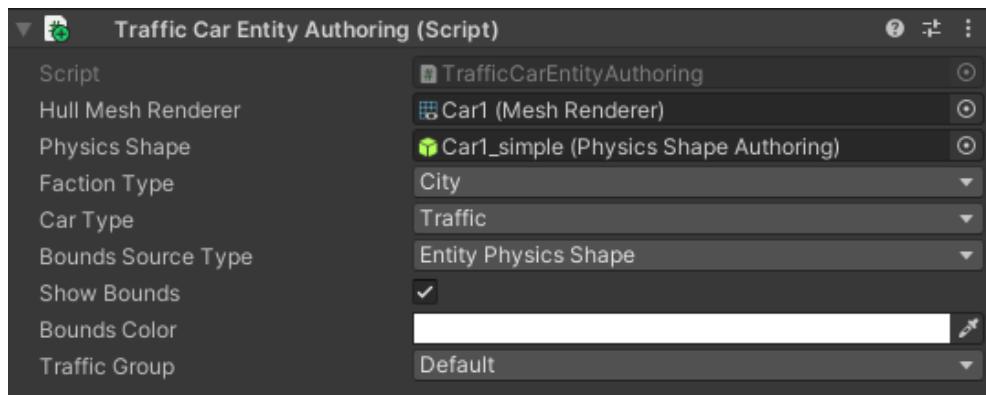
ScriptSwitcher



Component to on/off scripts of the vehicle.

3.1.3 Common Authoring Components

TrafficCarEntityAuthoring



Main component of traffic entity [required].

Hull mesh renderer : vehicle hull mesh renderer reference.

Physics shape : vehicle entity *PhysicsShape* reference.

Faction type : selected *faction type* of vehicle.

Car type : selected *car type* of vehicle.

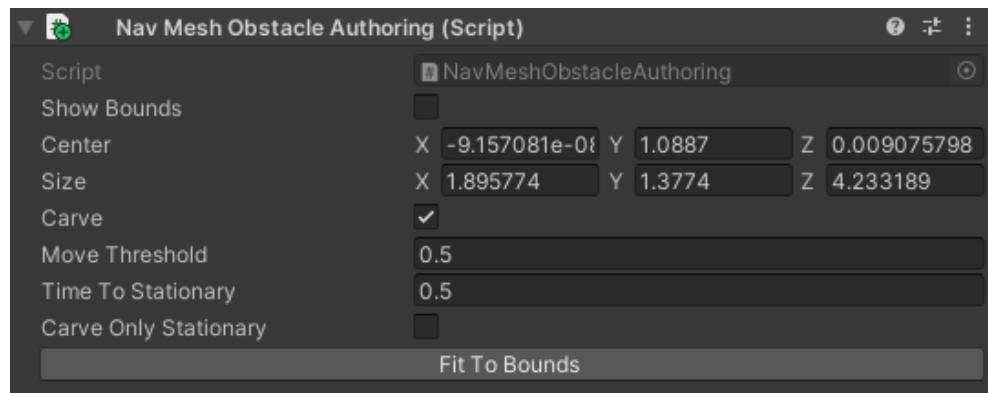
Bounds source type : selected bounds source for the entity bounds.

Traffic group : selected *traffic group*.

Shared Settings

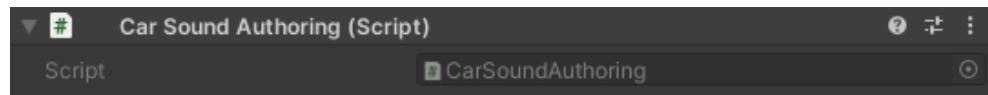
Each vehicle has a common set of settings that are described *here*

NavMeshObstacleAuthoring



NavMeshObstacleData entity component for runtime loading of *NavMeshObstacle* objects for *pedestrian navigation*. Make sure, that option is enabled in the *traffic settings* [optional].

CarSoundAuthoring



Component for vehicle *sounds* [optional].

HealthAuthoring

Vehicle health component for damage system [optional].

CarDamageEngineAuthoring

Component for visual presentation of damage in the damage systems [optional].

PlayerTargetAuthoring

Component for player targeting systems [**optional**].

3.1.4 CullState Info

States

- **Culled** : entity is destroyed.
- **CloseToCamera**:
 - Cull physics (if *enabled*) : *custom physics* & *simple physics* temporarily converted to *no physics* entity.
 - Cull wheel (if *enabled*) : disabling wheel rotating.
- **InVisionOfCamera** : entity fully enabled.

3.1.5 Parking

Car parking consists of the following states:

Entering parking states

1. The car has chosen the path containing the *parking node*.
2. The car links the *parking node* to prevent it from being selected by other cars (the list of linked nodes can be customized [here](#)).
3. When the car reaches the *parking node*, the car position correction is activated for precise parking (if disabled in the *parking config*, this step is skipped).
4. Stopping the engine state is starting (if enabled in the *stopping engine config* & vehicle in view of camera's player).
5. Pedestrian gets out of the car.

Exiting parking states

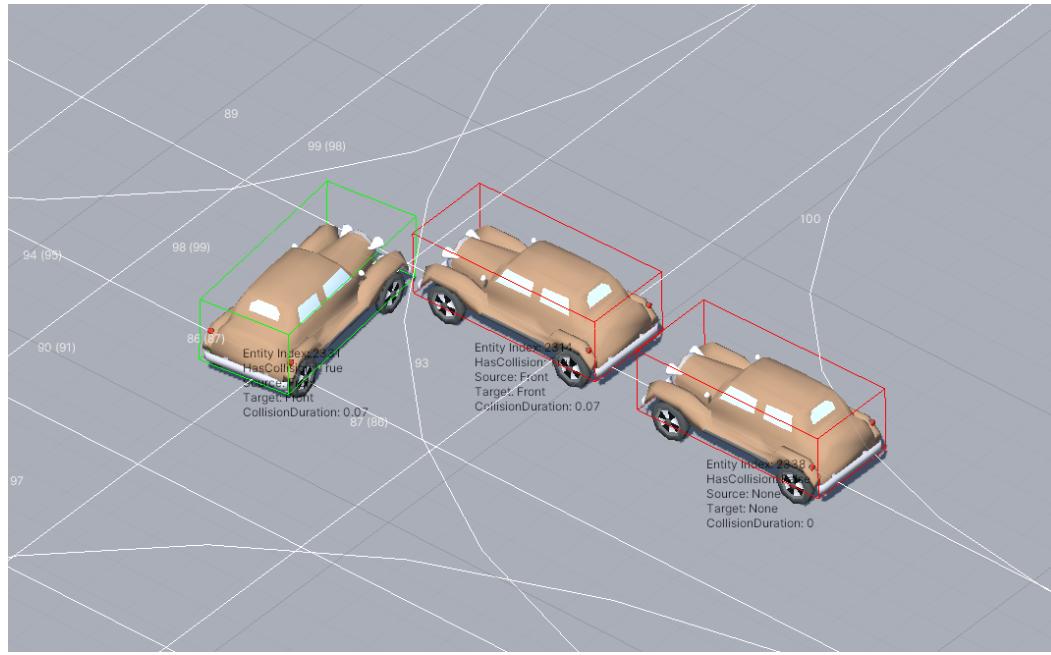
1. The Pedestrian enters the *pedestrian parking node*, if available.
2. The parking car removes link with *TrafficNode*.
3. Ignition state system starts (if enabled in *ignition config*).
4. Once the car has started the engine, the car starts moving.

3.1.6 Avoidance

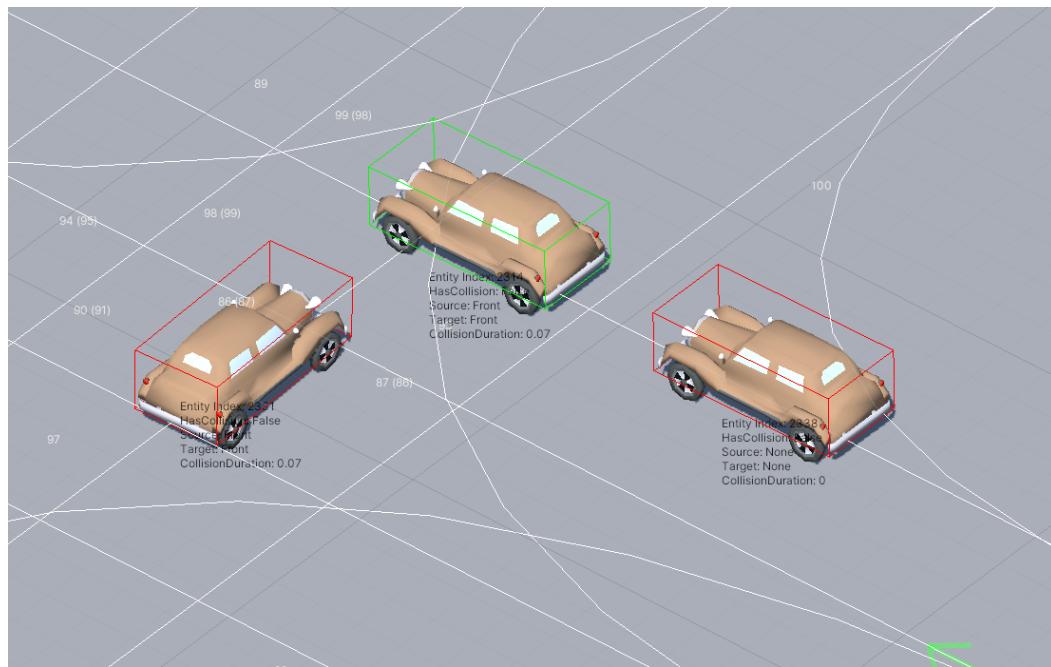
Avoidance is used in the case of stuck vehicles.

Currently enabled in the following situations:

- A cyclical obstacle where cars get stuck in each other (*avoidance config*).
- A car has collided frontally with another car (*collision config*).



Cyclical obstacle example.



Avoiding cyclical obstacle example.

Note: Test scene *example*.

3.1.7 Rail Movement

The *Rail movement* is used to drive the vehicle precisely along the *path*, which can be useful in small enclosed *parking areas*, for example. To enable rail movement, tick on the *Rail* parameter in the *path settings*. Open the *rail config* to adjust the *Rail* parameters.

Note: Enabled by default for *trams*.

3.1.8 Obstacle Detection

Raycast

Config

TrafficCarRaycastConfig.

Debugger

TrafficCarRaycastDebugger.

Modes

- *Hybrid mode* : raycast is activated only when the selected targets are close to the car.
- *Raycast only* : raycasts are sent constantly.

To define raycast targets for *Hybrid* or *Raycast only* modes, redefine the *GetTargetQuery* method in the *TrafficCarRaycastObstacleTargetQueryProvider* class, which returns the *EntityQuery* of the targets.

```
public static EntityQuery GetTargetQuery(
    SystemBase sourceSystem,
    TrafficCarDetectObstacleMode trafficCarDetectObstacleMode,
    TrafficCarDetectNpcMode trafficCarDetectNpcMode,
    out CollisionFilter tempRaycastCollisionFilter,
    out CollisionFilter raycastAlwaysCollisionFilter);
```

- *TrafficCarDetectObstacleMode*.
- *TrafficCarDetectNpcMode*.
- **Hybrid Mode Filter** : collision filter of hybrid raycast mode.
- **Raycast Only Filter** : collision filter of raycast only mode.

Note:

- You can also dynamically change the raycast target for *Hybrid mode* by adding or removing the *TrafficCustomRaycastTargetTag* component.
 - Layer constants are stored in the *ProjectConstants.cs* file.
-

3.2 Traffic Public

Traffic vehicles following public transport *routes* and picking up passengers at *stop station nodes*.

Youtube tutorial.

3.2.1 How To Create

1. Create a vehicle by following these *steps*.
2. Set *traffic group* to *Public Transport* in the *TrafficCarEntityAuthoring* component.
3. Add *TrafficPublicAuthoring* and *CarCapacity* to the created vehicle.
4. Tick on Predefined Road if public transport is to be routed via *TrafficPublicRoute*. **[Optional step]**
5. Create an empty child *GameObject*, add the *VehicleEntryAuthoring* component and assign it to the *CarCapacity* component.
6. Position the created Entry *GameObject* where the pedestrian entrances/exits will be.
7. Create the *TrafficPublicRoute* entity and set the *CarModel* according to created public transport vehicle. **[Optional step]**

3.2.2 Components

TrafficPublicAuthoring

Authoring component that contains settings for public transport.



Predefined Road : the vehicle will only be spawned on *TrafficPublicRoute* paths.

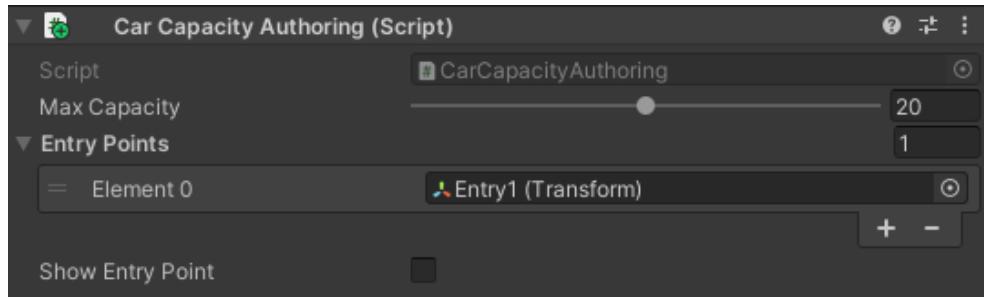
Min/Max idle time : min/max idle time at the public stop station.

Min/Max pedestrian exit count : min/max number of pedestrians that can exit the station at a time

Enter/exit delay duration : min/max delay between entrances to public transport.

Car capacity authoring

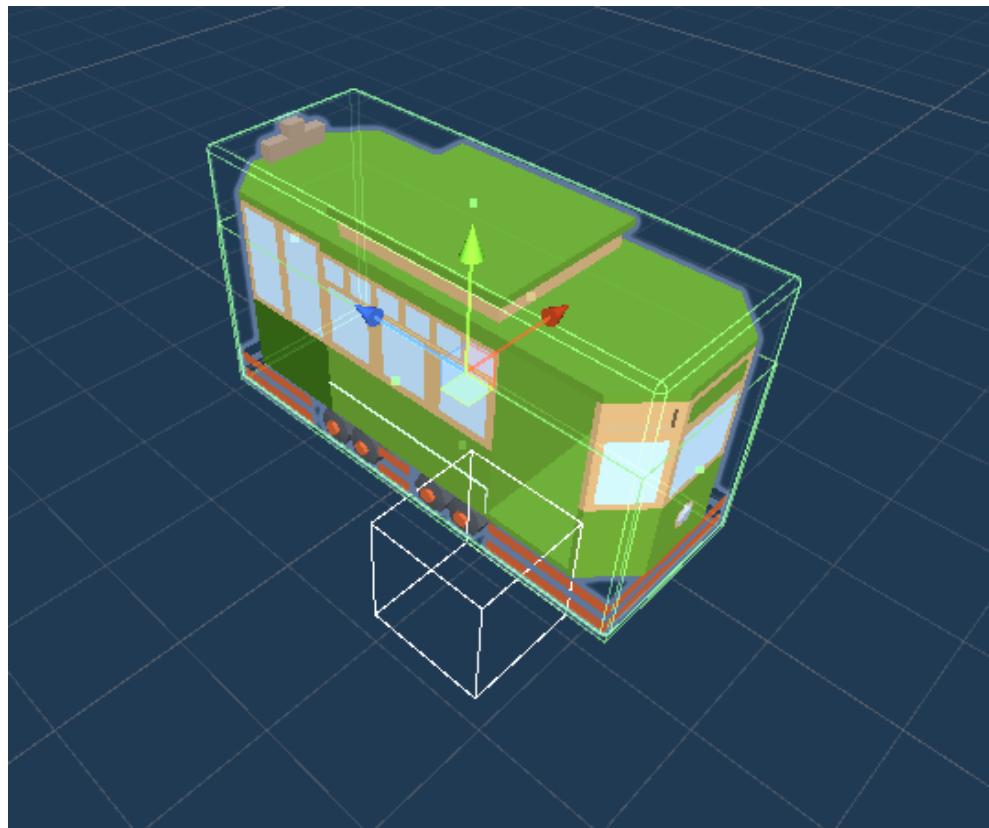
Authoring component that contains capacity settings of the vehicle.



Max capacity : max capacity of the vehicle.

Entry point : any *GameObject* that contain *VehicleEntryAuthoring* component.

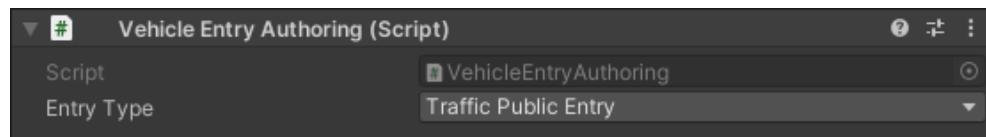
Show entry point : on/off display entry point.



Public tram example (white box - entry point).

Note: At the moment the component is only used for *TrafficPublic* vehicles.

VehicleEntryAuthoring



Entrance point for pedestrians to public transport.

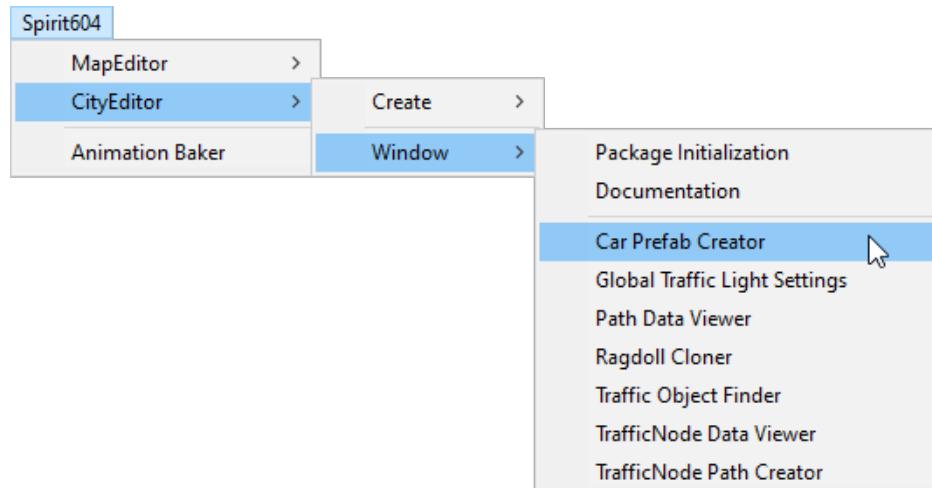
3.3 Car Prefab Creator

Youtube tutorial.

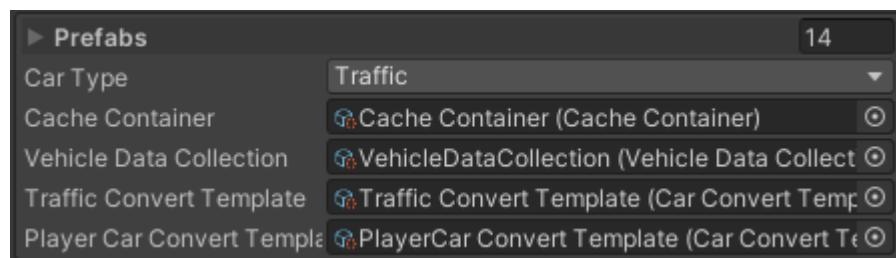
3.3.1 How To Use

- From the *Unity* toolbar, open *Car Prefab Creator*.

Spirit604/CityEditor/Car Prefab Creator



- Drag and drop car prefabs into the *Prefs* field.



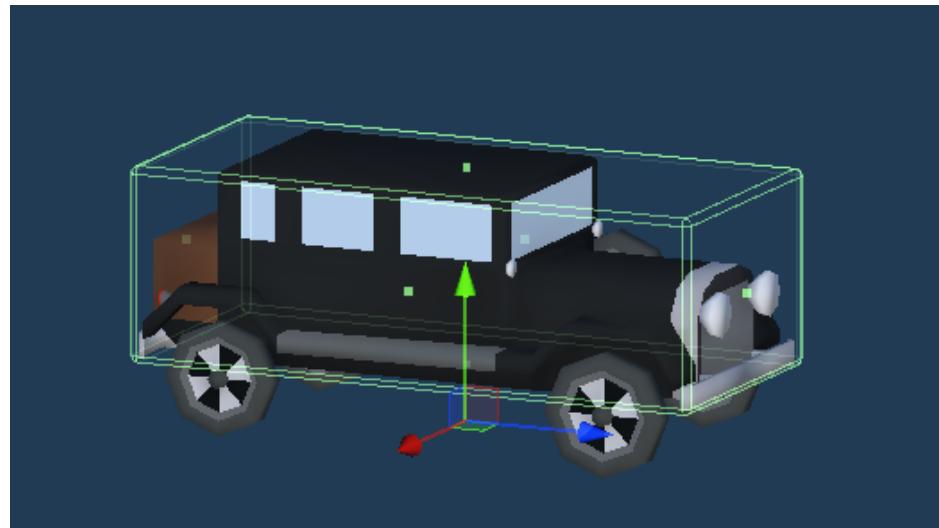
- Configure the *Common settings* for creating a car.



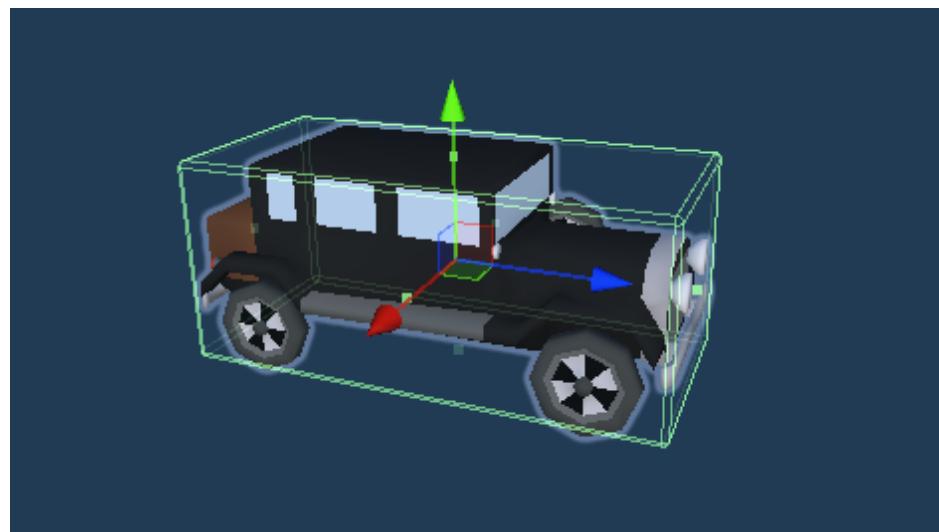
4. In the *Save* tab, select the *Entity Type* according to the desired use case.



5. If the vehicle is the *Simple physics vehicle* type, enable *Fit physics shape & Include Wheels* in the *Common settings* tab. [*Simple physics only*]

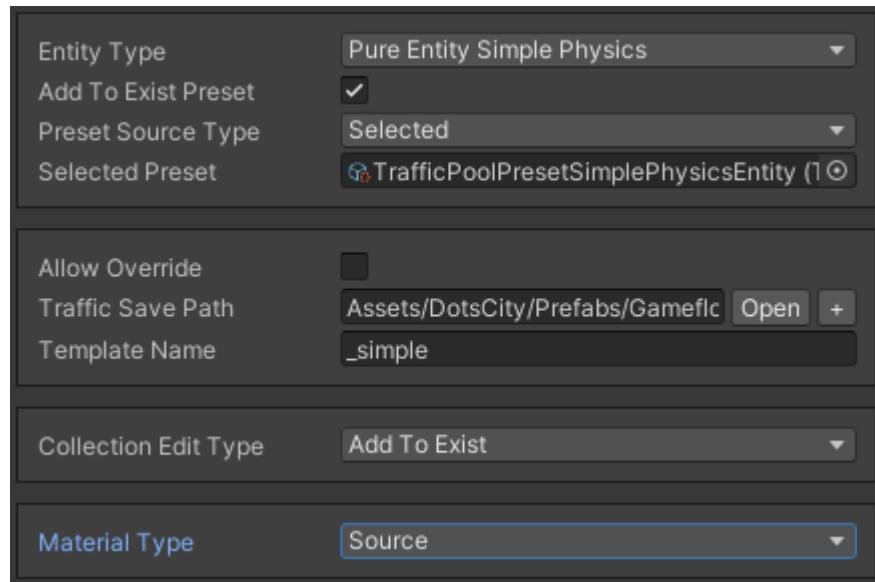


Custom physics entity shape result example

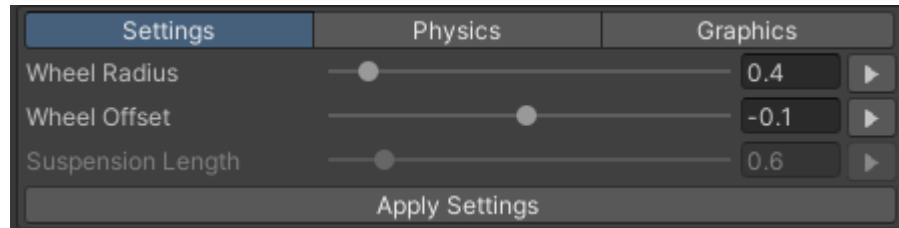


Simple physics entity shape result example

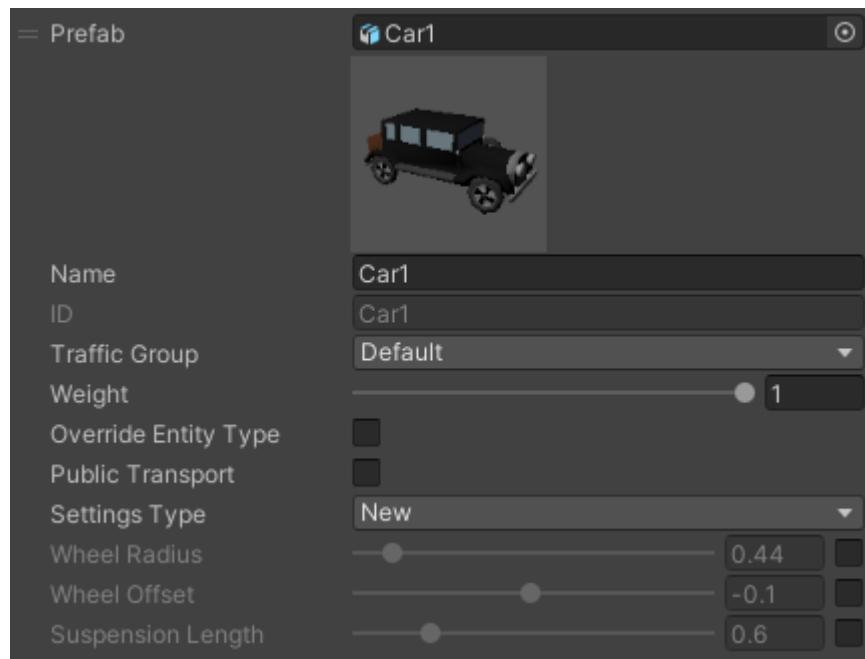
6. In the *Save* tab, configure the *Save settings*, which preset & path to use for saving the cars.



7. Click *Scan* button.
8. Open *Additional Settings* tab, adjust common settings for all vehicles.

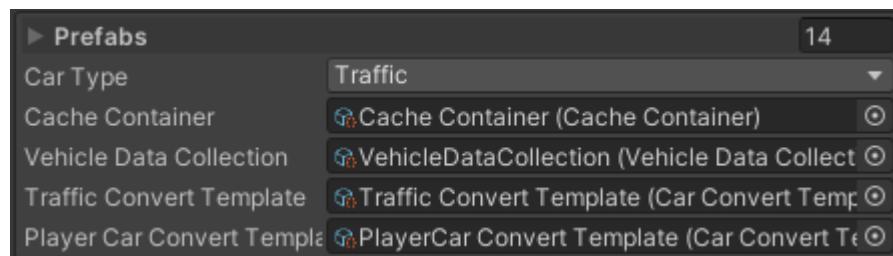


9. After, open *Prefab Info* tab, adjust unique settings for each vehicle.



10. Once all the settings have been customised, click *Create* to create the prefabs.
11. If some of the body or wheel offsets are wrong, drag and drop the created car prefab into the scene & use this tool to re-create cars with new offsets using the *Create* button again until the desired result is achieved.

3.3.2 Prefab Settings



Prefab source type:

- Scene
 - **Targets prefab parent** : prefabs will be taken from the selected root from the scene.
- Project
 - **Prefabs** : selected prefabs from the project.

Car type:

- **Traffic** : prefab car will be created for the traffic.
- **Player** : prefab car will be created for the player.

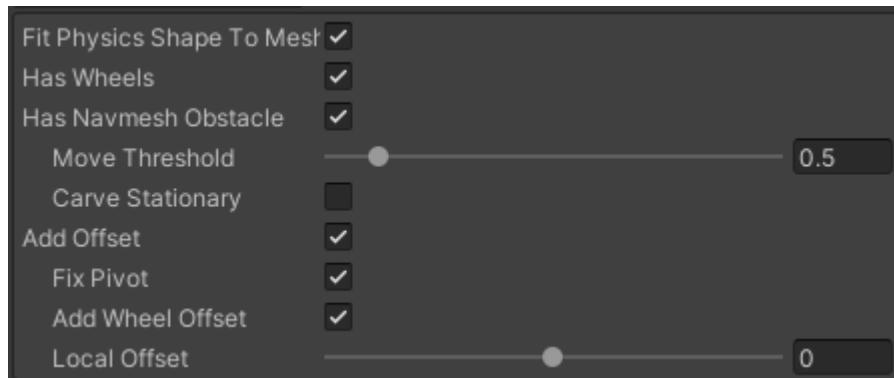
Cache container : cache data of saved vehicles.

Vehicle data collection : reference to the *collection* of all vehicles.

Traffic car convert template : template which contains traffic prefab template.

Player car convert template : template which contains player prefab template.

3.3.3 Common Settings



Assign hull mesh: should find the hull of the car.

- Parent is hull mesh : car root contains a car mesh.

Fit physics shape to mesh : physical shape will be resized to the mesh size.

Has wheels : should search for wheels on a *template*.

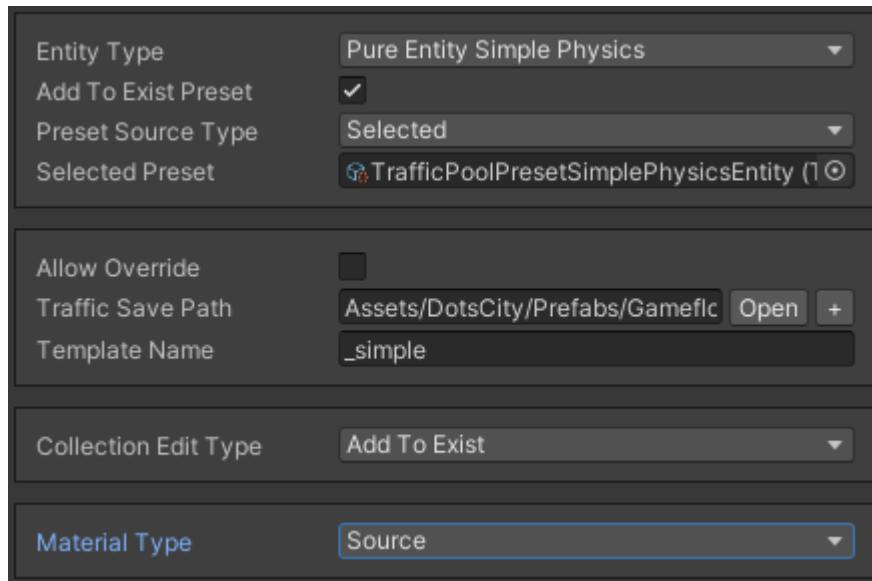
Has navmesh obstacle: does the car contain **NavMeshObstacle** component.

- Move threshold
- Carve stationary
- Carve time to stationary

Add offset: offset of the vehicle hull along the Y axis.

- Fix pivot : fixes the pivot point if the pivot point is in the centre of the mesh.
- Add wheel offset : adds wheel offset size.
- Local offset : custom offset value.

3.3.4 Save Settings



Save to exist preset:

- **Scene:** add the created prefabs to an existing *preset* in the scene.
- **Selected:** add the created prefabs to selected *preset*.

New preset settings:

- **Assign new preset to scene :** *preset* will replace an existing *preset* on scene.
- **New preset path :** project path where to create a new *preset*.
- **New preset name :** new *preset* name.

Entity type : *entity type of the vehicle*.

Prefab save type:

- **Override source :** selected prefabs will be replaced by new ones.
- **Create new if not exist :** new prefabs will be created only if there are no previously created ones by the selected path.
- **Override target :** previously created prefabs will be overwritten in case of a duplicate.

Prefab save path type:

- **Original prefab path :** prefabs will be created in the directory where the selected prefabs are located.
- **Template prefab path :** Prefabs will be created in the directory where the template is located.
- **Custom path :** user's path of creation.

New prefab template name : pattern of the name of the created prefab (for instance *Car1* (source name) + “_new” (pattern) = *Car1_new*).

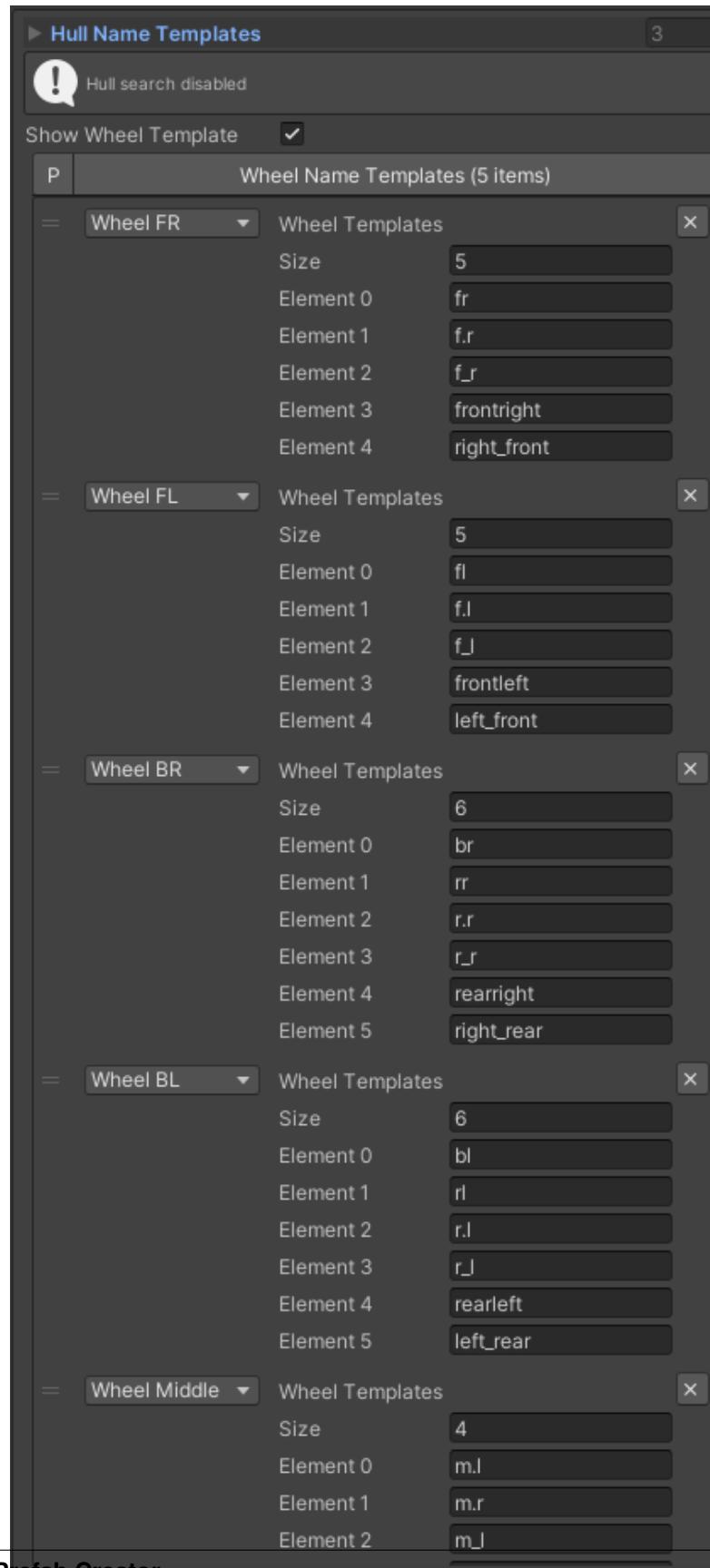
Collection edit type:

- **Add to exist** : add vehicles to exist *vehicle collection*.
- **Override** : overrides *vehicle collection* by created vehicles.

Material type:

- **Source** : material is copied from the source prefab.
- **Custom atlas material** : user's custom atlas material.
- **New unique material** : new material is generated based on the user's own material.

3.3.5 Template Settings



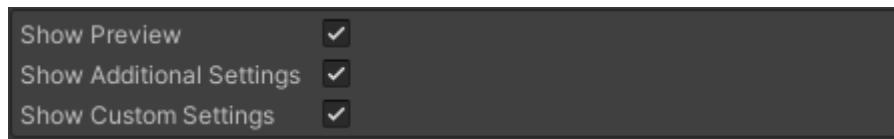
Hull name templates : keyword phrases for automatic hull searches.

Wheel name templates

[keyword phrases for automatic wheels searches.]

- **Wheel FR** : forward right wheel.
- **Wheel FL** : forward left wheel.
- **Wheel BR** : backward right wheel.
- **Wheel BL** : backward left wheel.
- **Wheel Middle** : additional wheels.

3.3.6 Preview Settings



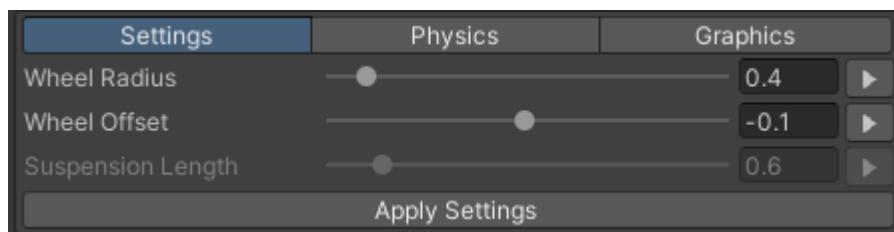
Show preview : on/off preview image of the prefab on the *Prefab Info* tab.

Show additional settings : on/off display of the additional settings of the prefab on the *Prefab Info* tab.

Show custom settings : on/off display of the custom settings of the prefab on the *Prefab Info* tab.

3.3.7 Additional Settings

Common Settings



Wheel radius : wheel radius.

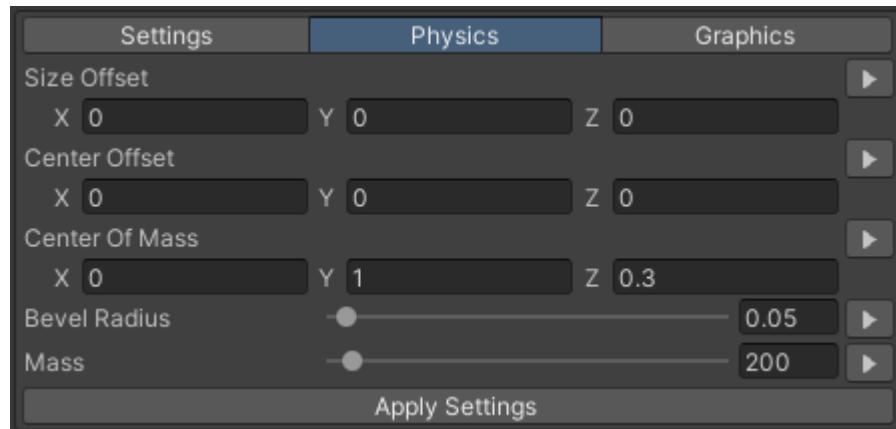
Wheel offset : wheel offset by Y-axis of the vehicle.

Suspension length : suspension length of the vehicle. [Custom physics vehicles only]

Note:

- Editing additional parameters will affect all cars in the *Prefab Info* tab, to make unique parameters check the toolbox opposite on the parameter in the *Prefab Info*.
 - Arrow-button applies the setting for the selected parameter.
-

Physics



Size offset : size offset of physics shape.

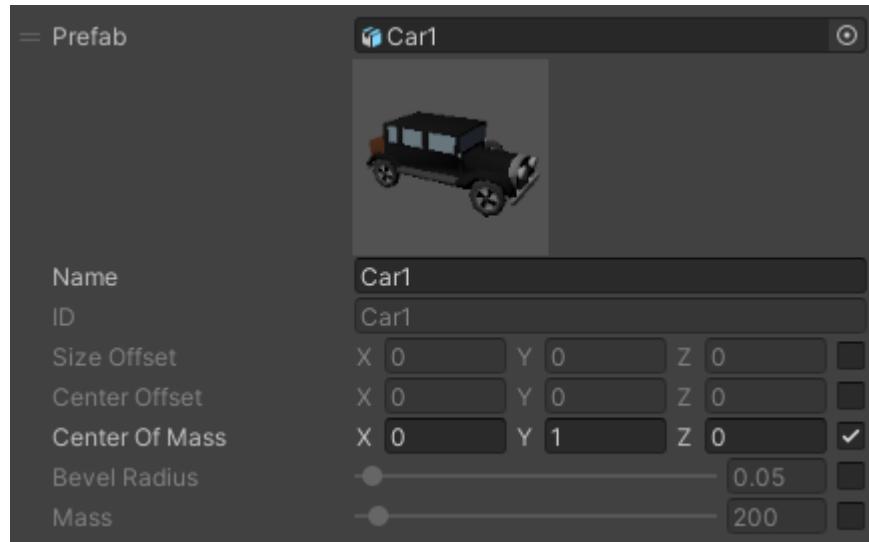
Center offset : center offset of physics shape.

Center of mass : center of mass of the vehicle.

Bevel radius : bevel radius of physics shape.

Mass : mass of the vehicle.

Info Tab



Graphics

Settings	Physics	Graphics
Wheel Source Type	Shared All	
Wheel Rotation Type	Flip Right Row	
Shared Wheel Mesh	WheelFL	
Shared Wheel Mesh LOD1	None (Mesh)	
Shared Wheel Mesh LOD2	None (Mesh)	
Has Lods	<input checked="" type="checkbox"/>	
Lod 0 Screen Size	1.2	
Lod 1 Screen Size	0.3	
Lod 2 Screen Size	0	

Wheel source type:

- **Model unique** : the wheels remain as in the original model.
- **Shared from model** : the wheel model selected by the user from the original model is used for all wheels.
- **Shared all** : the wheel model selected by the user shared between all wheels.

Wheel rotation type [shared wheel only]:

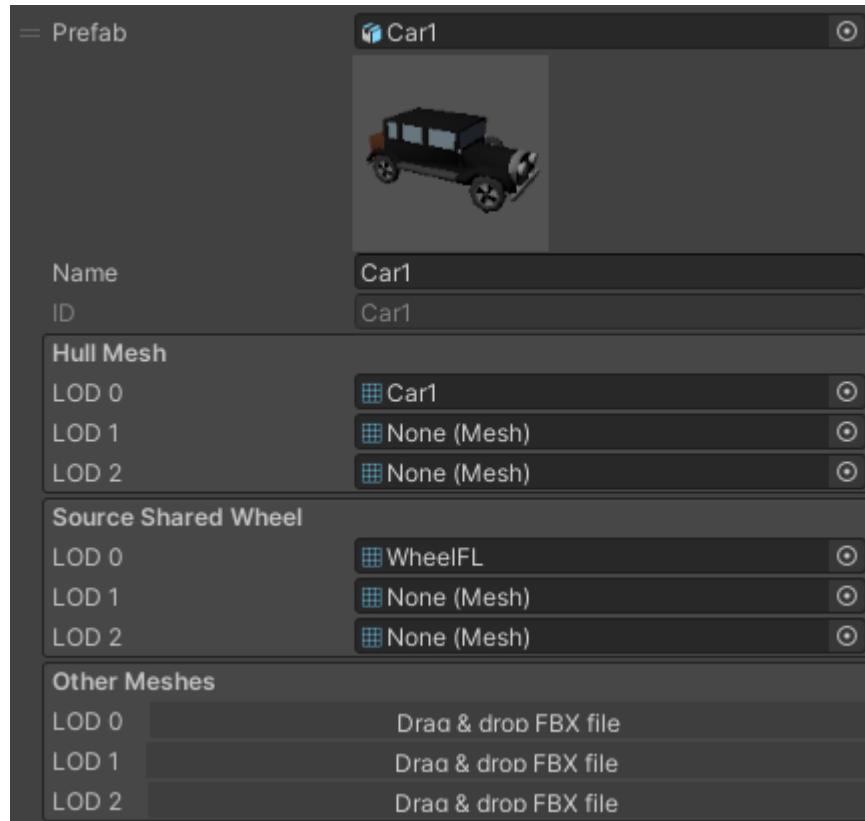
- **Source** : the wheel rotation remains unchanged.
- **Flip left row** : rotate the wheel in the left-hand row by 180° if you are using the wheel model from the right-hand row.
- **Flip right row** : rotate the wheel in the right-hand row by 180° if you are using the wheel model from the left-hand row.

Has lods: on/off LODs for vehicle.

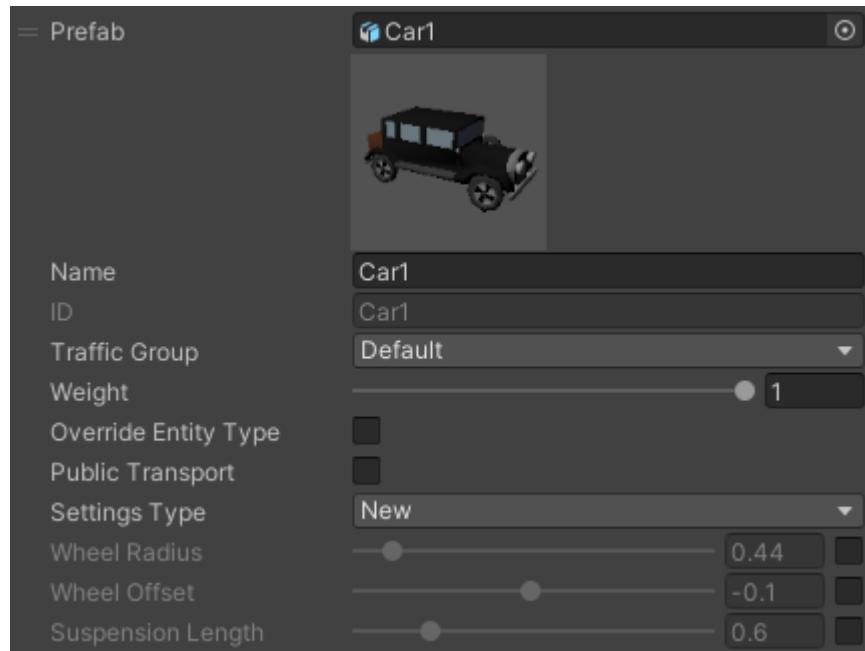
- **Lod 0, 1, 2 screen size** : screen size of LOD.

Note: Wheel sharing is useful for using the same wheel model for all wheels to reduce drawcalls.

Info Tab



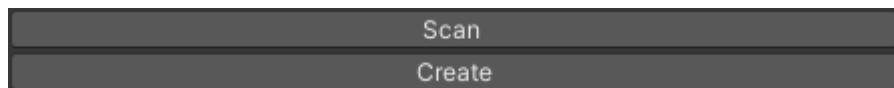
3.3.8 Prefab Info



Car Info

- **Prefab** : reference to source prefab.
- **Name** : user's *name* of the vehicle.
- **ID** : new *ID* entry for *vehicle collection*.
- **Traffic group** : *traffic group* of the vehicle.
- **Override entity type**
[new *entity type* for selected vehicle (might be useful for specific vehicles such as *tram*).]
 - **Entity type**
- **Public transport**
[on/off *public transport* feature. (Settings)]
 - **Predefined road**
 - **Capacity**
 - **Entries**
- **Settings type:**
 - **New** : user-defined settings.
 - **Template** : vehicle settings are copied from the selected template [**custom physics vehicle only**].
 - **Clone model** : vehicle settings are copied from the selected *CarModel* in the list.
- **Wheel radius** : wheel radius. (**can be unique value**)
- **Wheel offset** : wheel offset by Y-axis of the vehicle. (**can be unique value**)
- **Suspension length** : suspension length of the vehicle. (**can be unique value**) [**Custom physics vehicles only**]

3.3.9 Buttons

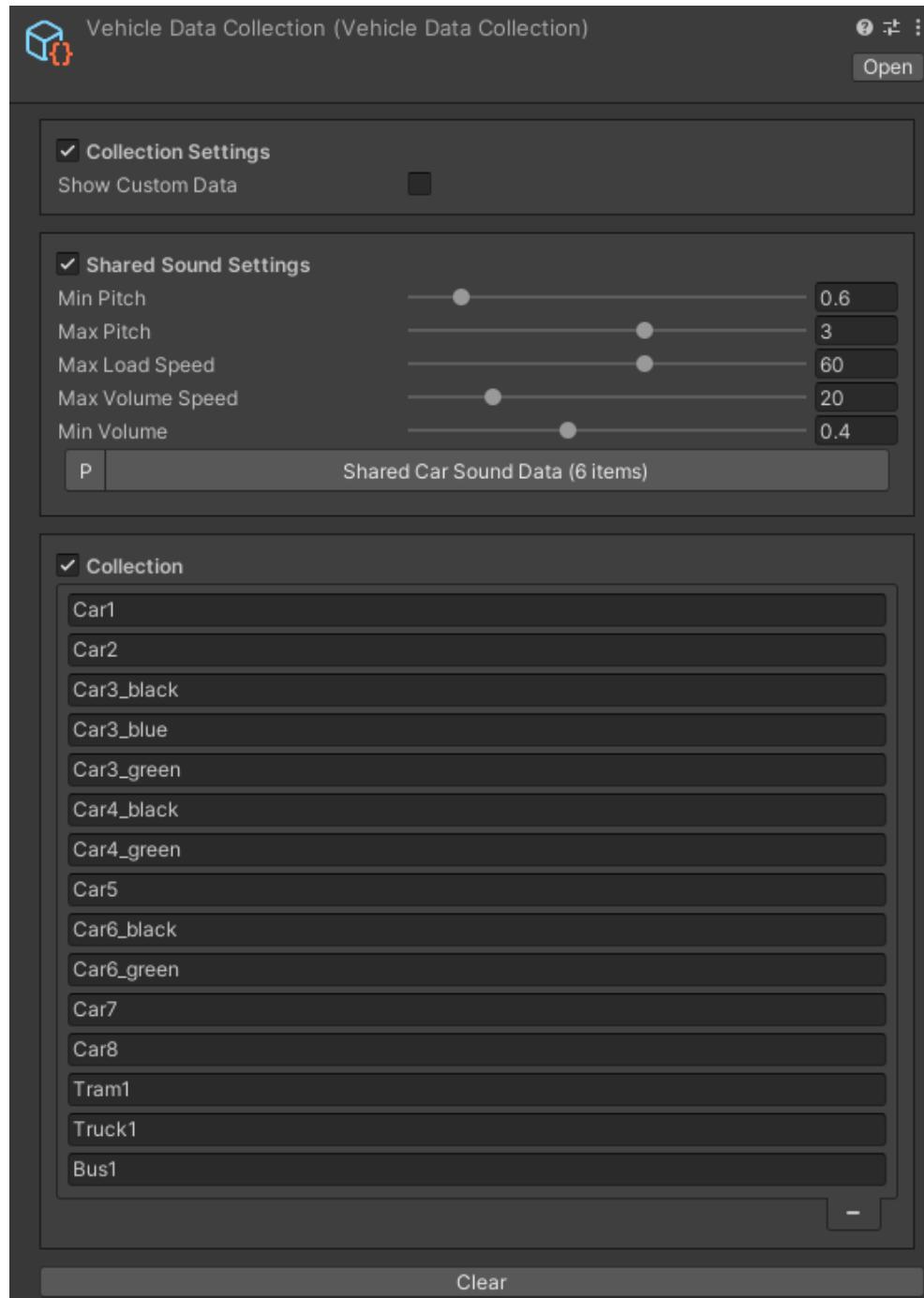


Scan : scan the added prefabs and add information about new ones to the *Prefab Info* tab.

Create : create new entity prefabs based on the added prefabs.

3.4 Vehicle Collection

The *Vehicle collection* contains data on all vehicles in the city and their *shared settings*.



3.4.1 How To

Add To Collection

Vehicles can only be added to the collection using the *Car Prefab Creator* tool.

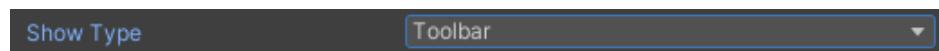
Override Settings

Steps

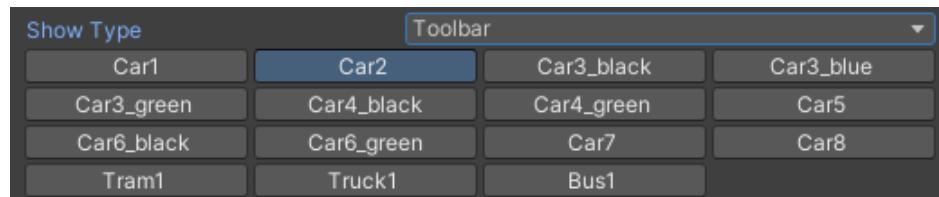
1. Tick on *Show Custom Data*.



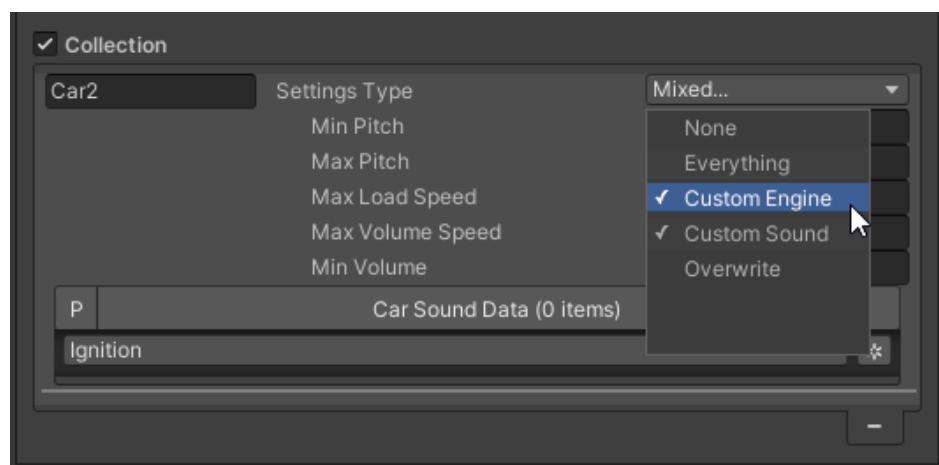
2. Change *Show Type* to *Toolbar*.



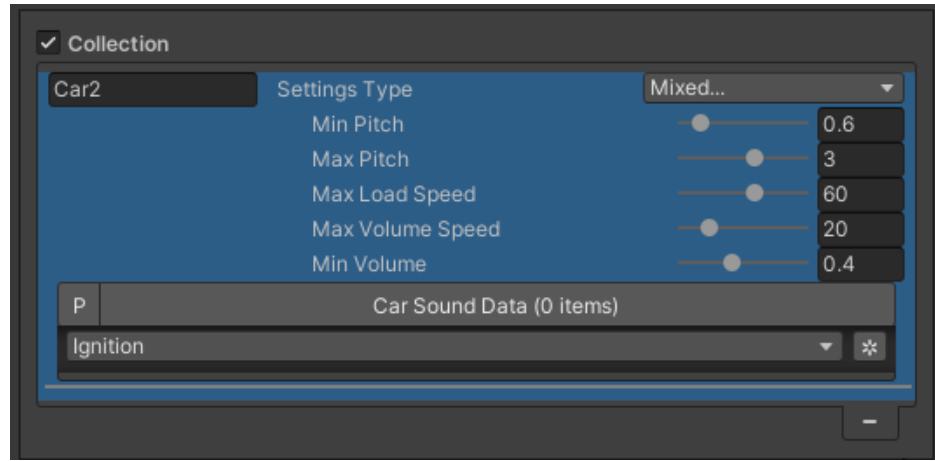
3. Select desired vehicle.



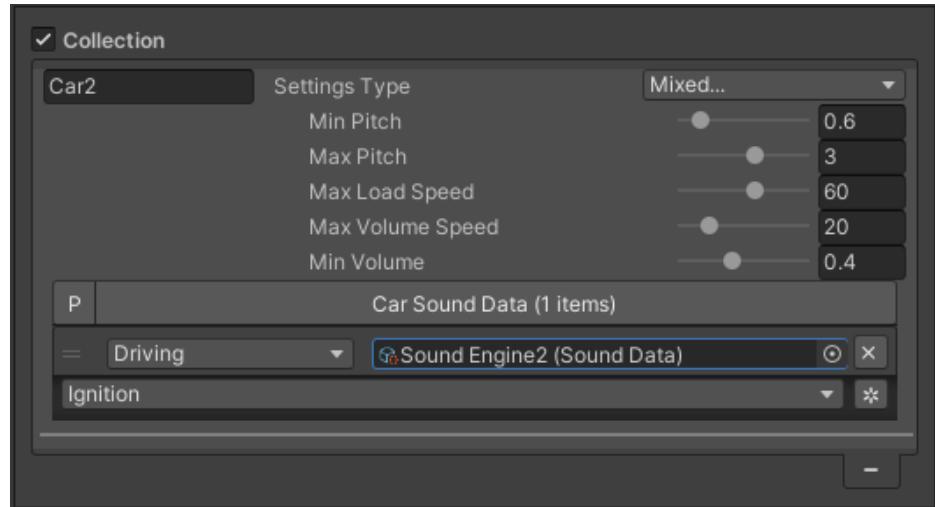
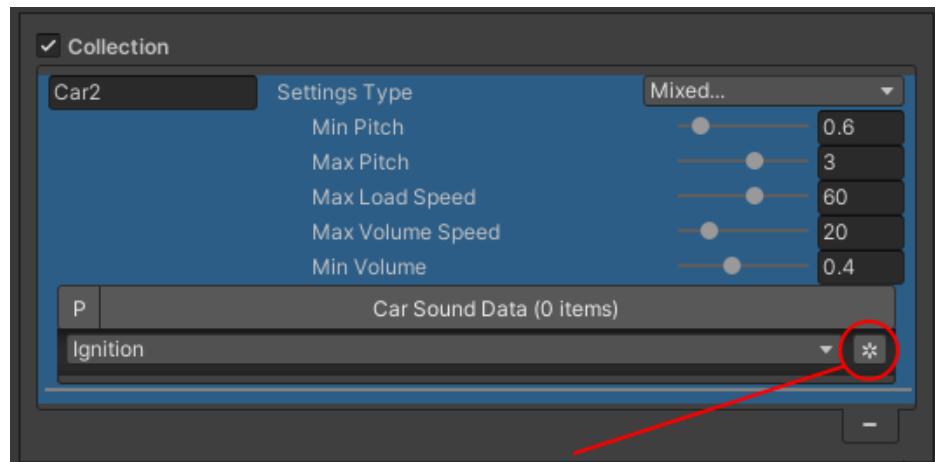
4. Select the *Settings Type* to *Custom Engine* and *Custom Sound* (if you want to override both).



5. Customize *custom sound settings*.

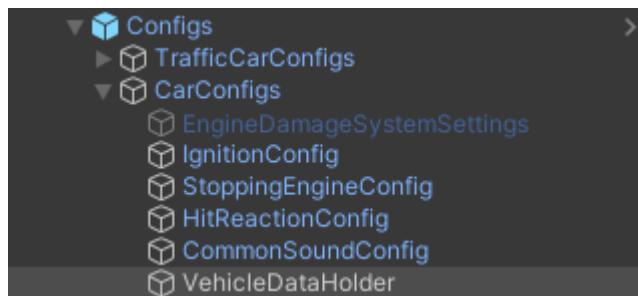


6. Add *custom sounds*.



Entity Conversion

Vehicle collection assigned to the *VehicleDataHolder* and converted in the *EntitySubScene* subscene.



3.4.2 Unique ID

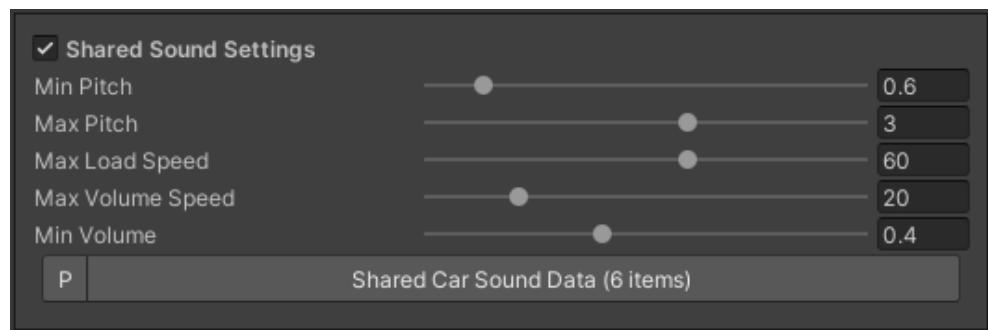
ID - is the unique, immutable ID based on the main mesh name of the vehicle.

3.4.3 Car Model

Car model - the name of the vehicle that is assigned to the *Vehicle data* and associated with an immutable *ID*.

Note: You can change the *CarModel* name at any time in the *Collection* tab.

3.4.4 Sound Settings



Min pitch : minimum pitch of the car engine.

Max pitch : maximum pitch of the car engine.

Max load speed : speed at which the engine has the maximum pitch.

Max volume speed : speed at which the engine has the maximum volume.

Min volume : minimum engine volume.

3.4.5 Sounds

Shared Car Sound Data (6 items)			
= Ignition	▼	Car Ignition (Sound Data)	⊕ X
= Idle	▼	Sound idle (Sound Data)	⊕ X
= Driving	▼	Sound Engine2 (Sound Data)	⊕ X
= Horn	▼	Sound horn (Sound Data)	⊕ X
= Enter Car	▼	Open Car Sound (Sound Data)	⊕ X
= Exit Car	▼	Exit Car Sound (Sound Data)	⊕ X

- **Ignition** : playback during engine start of the vehicle.
- **Idle** : idle sound of the vehicle.
- **Driving** : plays when pedestrian or player driving the vehicle.
- **Horn** : plays when the horn of the vehicle is active.
- **Enter car** : plays when pedestrian or player enters vehicle.
- **Exit car** : plays when pedestrian or player exits vehicle.

3.5 Presets

- Preset contain prefabs that are converted to entities.
- There are several *Entity type*.
- The preset for conversion is selected from the *Entity type* in the *Traffic Settings*.

Warning: If the *Vehicle Collection* does not contain the appropriate *Prefab ID*, the *Prefab* will be ignored in the preset.

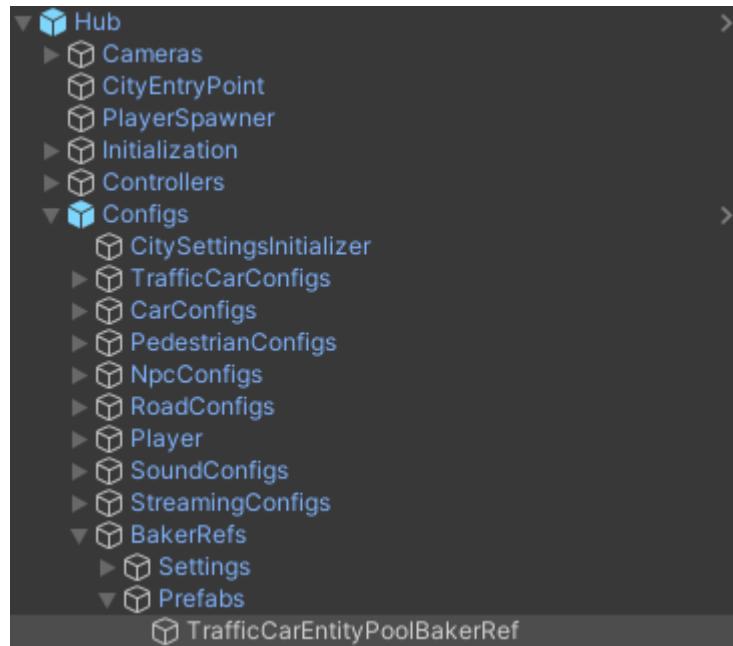
3.5.1 How To Create

Presets can only be created by using the *Car Prefab Creator* tool (step №6).

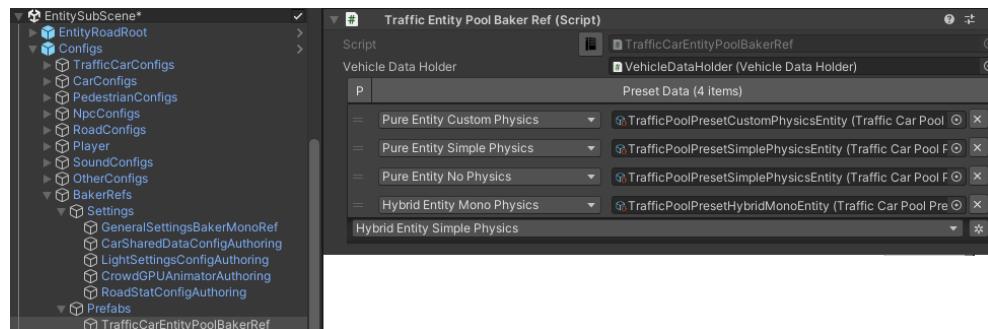
3.5.2 Where To Find

Make sure, that *TrafficCarEntityPoolBakerRef* in the *main scene* & *subscene* has the correct preset. To sync both scenes, read more about *config editing*

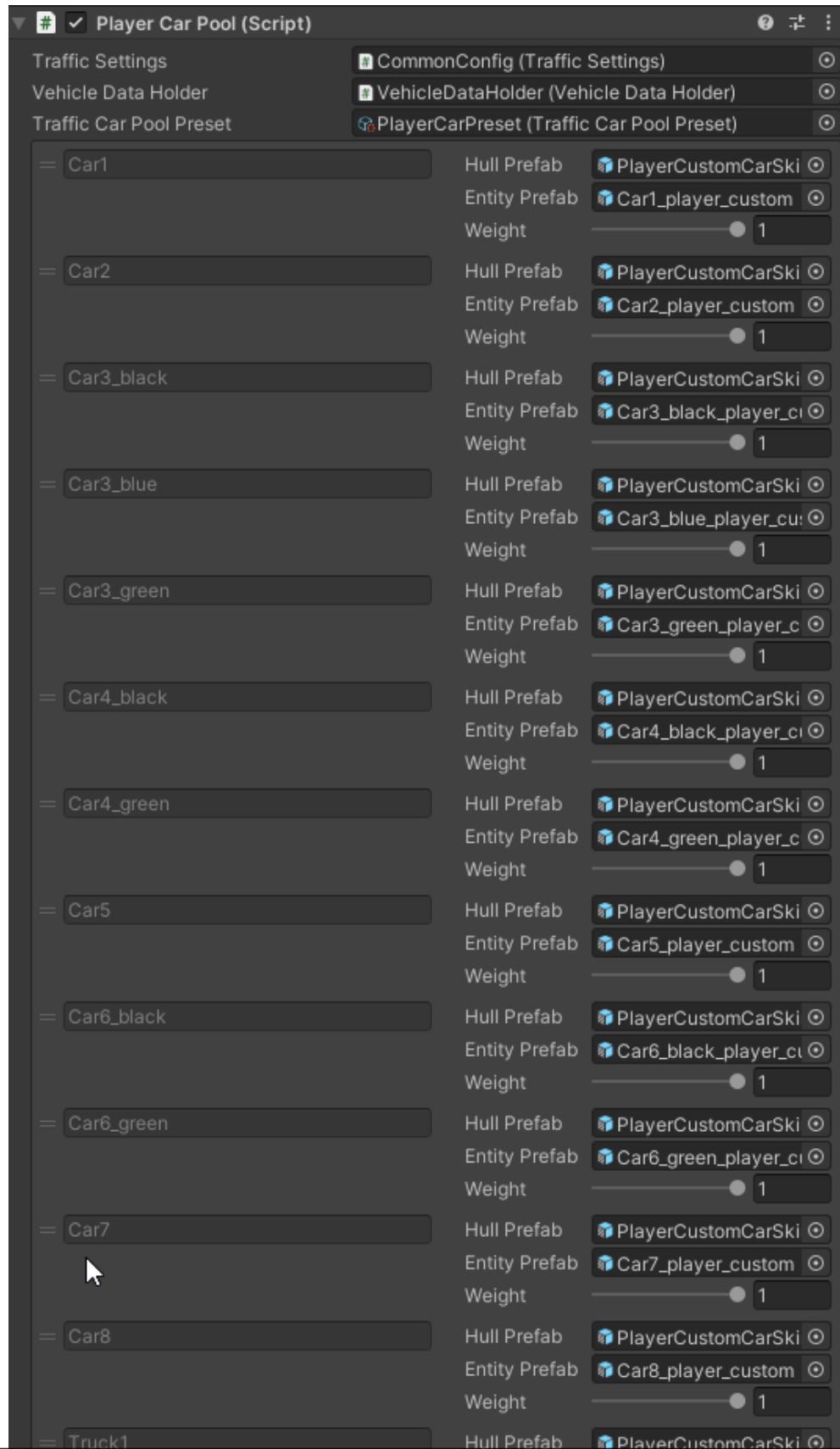
Main scene

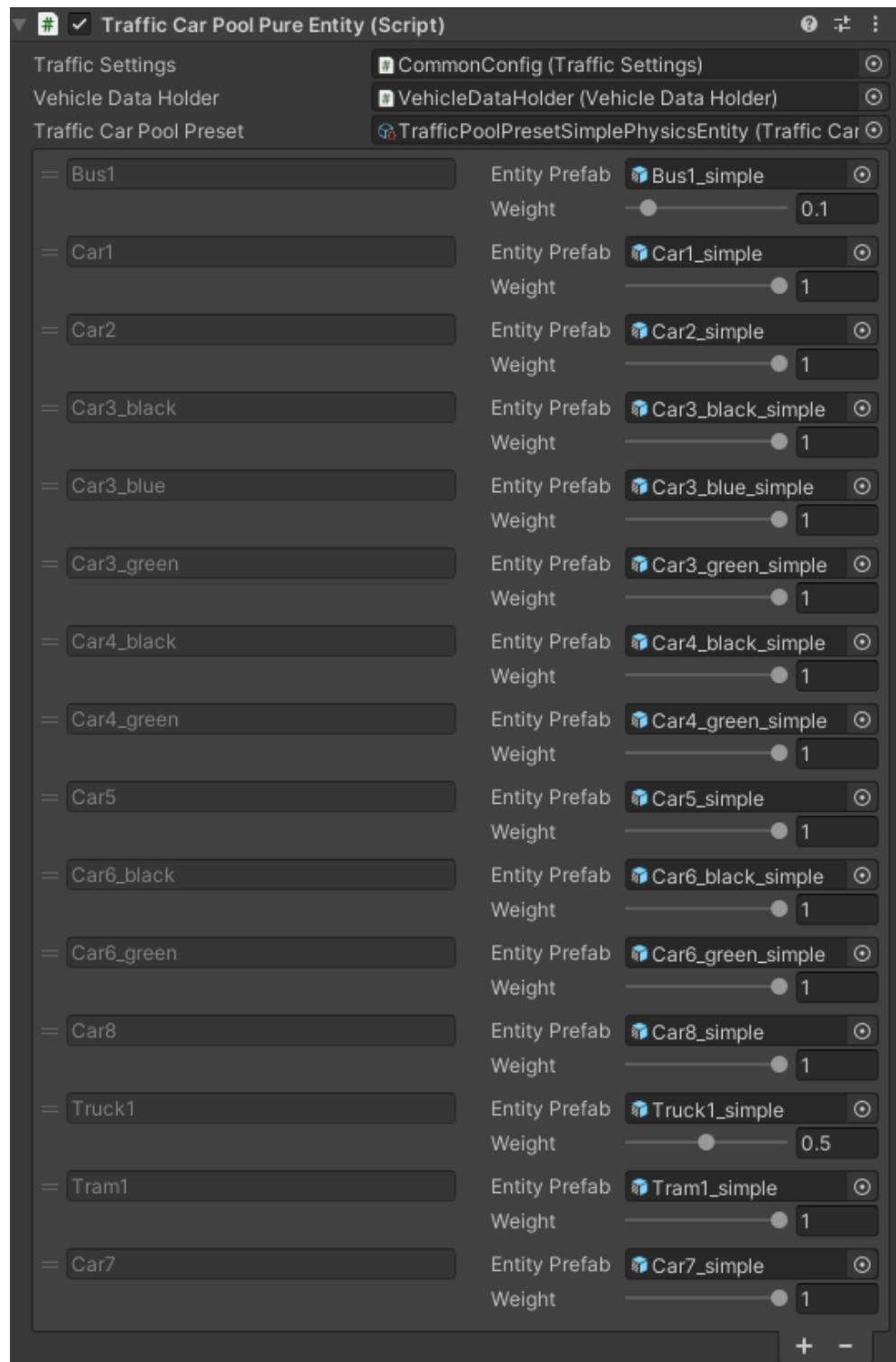


Sub scene



3.5.3 Examples



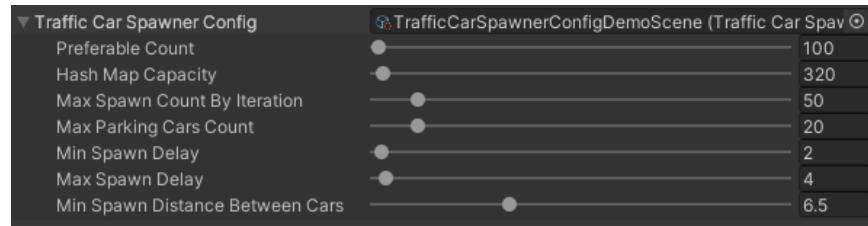
Hybrid custom entity player preset example*Simple physics entity traffic preset example*

Note: Pure entity simple physics type and Pure entity no physics type has the same preset.

3.6 Traffic Configs

3.6.1 Traffic Configs

Traffic Car Spawner Config



Preferable count : maximum number of cars in the city.

HashMap capacity : initial capacity of the hashmap that contains the data of the traffic cars.

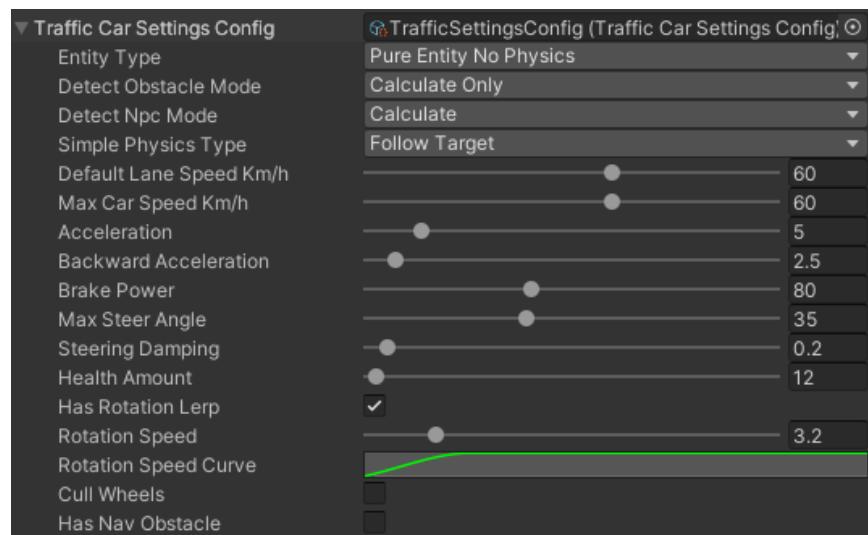
Max spawn count by iteration : maximum number of cars that will be spawned in one iteration.

Max parking cars : maximum number of parked in the city.

Min/Max spawn delay : minimum/maximum duration between spawns.

Min spawn distance : minimum distance for spawning between cars.

Traffic Car Settings



Entity Type

- **Hybrid entity simple physics** : *hybrid entities* moved by the simple DOTS physical system ([description](#)).
- **Hybrid entity custom physics** : *hybrid entities* moved by the custom DOTS physical system ([description](#))
- **Hybrid entity mono physics** : *hybrid entities* moved by the custom *Monobehaviour* controller ([description](#)) (new)
- **Pure entity custom physics** : *pure entities* moved by the custom DOTS physical system ([description](#))
- **Pure entity simple physics** : *pure entities* moved by the simple DOTS physical system ([description](#)).
- **Pure entity no physics** : *pure entities* that moved by transform system without physics ([description](#)).

Note: Depending on which *entity type* such a *preset* is converted.

Detect Obstacle Mode

- **Hybrid** : combine types *Calculate* and *Raycast*.
- **Calculate only** : mathematically calculates the obstacle.
- **Raycast only** : detect obstacle by raycast ([more info](#)).

Note: In *Hybrid mode*, raycast is activated only when the selected targets are close to the car ([more info](#)).

Detect Npc Mode

- **Disabled**
- **Calculate** : mathematically calculates the npc.
- **Raycast** : detect obstacle by raycast (npc should have *PhysicsShape* component).

Simple Physics Movement Type

- **Car input** : simple emulation of real movement based on traffic input.
- **Follow target** : the vehicle rotation is set based on the destination direction.

Common Settings

Default lane speed km/h : default lane speed (if the lane speed limit is set to 0 the default speed will be selected).

Health count : number of hit points of the car (health systems should be enabled).

Simple Vehicle Settings

Max car speed km/h : maximum speed of the car.

Acceleration magnitude : vehicle acceleration speed.

Backward acceleration magnitude : backward vehicle acceleration speed.

Brake power : brake power.

Max steer angle : max steer angle of the wheels.

Steering damping : wheel turn speed.

Has rotation lerp :

- **Rotation speed** : vehicle rotation speed.
- **Rotation speed curve** : curve on the dependence of the speed of the car on its speed.

Custom Vehicle Settings

Braking input rate : braking rate if current vehicle speed limit is greater than speed limit.

Other Settings

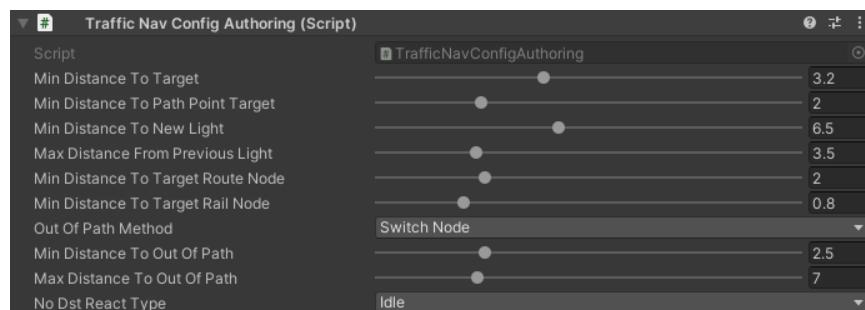
Cull physics : off physics for the vehicle if they are outside the camera.

Cull wheels : on/off wheel handling if they are outside the camera.

Has nav obstacle : on/off navmesh obstacles for traffic *config*.

Traffic Nav Config

Config distance to target nodes and traffic light handlers.



Min distance to target : min distance to target *TrafficNode*.

Min distance to path point target : min distance to connected *path point*.

Min distance to new light : minimum distance to the *TrafficNode* entity that contains the *traffic light handler* entity to assign it to the car entity (if the traffic node entity does not contain a traffic light entity, the index is -1).

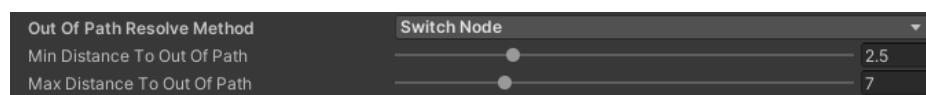
Min distance from previous light : minimum distance from the *TrafficNode* entity that contains the *traffic light handler* entity to unassign it from the car entity (if the traffic node entity does not contain a traffic light entity, the index is -1).

Min distance to target route node : minimum distance to switch to the next waypoint of the *path*.

Min distance to target rail route node : minimum distance to switch to the next waypoint of the *path* (rail movement only (tram etc...)).

Out of path resolve method: resolving method in case the car is out of the path.

- **Disabled** : no actions.
- **Switch node** : switching to the next waypoint.
- **Backward** : car will try to reach the missed waypoint by reversing.
- **Cull** : car will be culled.



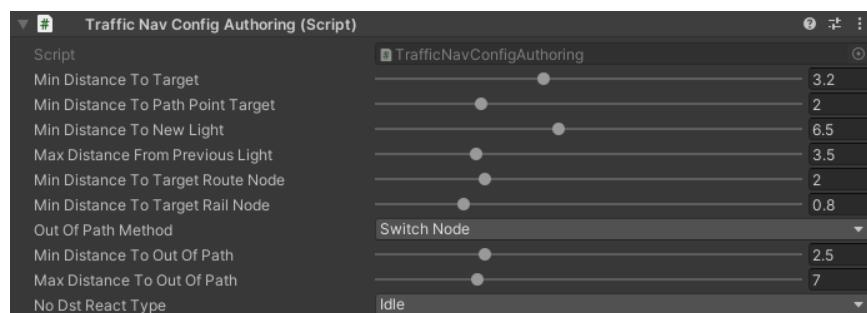
Out of path resolve method [enabled]:

- **Min distance to out of path** : minimum distance from the missed waypoint to the car.
- **Max distance to out of path** : maximum distance from the missed waypoint to the car.

No Dst React Type : the reaction if there is no further destination for the cars (for example, when the road was unloaded by *road streaming*).

Traffic Obstacle Config

Config to calculate obstacles on the path.



Max distance to obstacle : minimum distance to an obstacle (*example*) (*test scene*).

Min distance to start approach : minimum distance to the last car in the current lane to start (*approaching*) (stay at the same speed as the target car) (*example*) (*test scene*).

Min distance to start approach soft : minimum distance to the last car in the current lane to start (*approaching*) (long distance approach at a *soft* speed) (*test scene*).

Min distance to check next connected path : minimum distance to check the next path for obstacles (*example*) (*test scene*).

Short path length : if the next path is too short, start checking the next connected paths for obstacles (*example*) (*test scene*).

Calculate distance to intersect point : distance to intersected paths when they are checked for obstacles (*example*) (*test scene*).

Obstacle intersect calculation method: method of calculating the intersection of the vehicle and the intersect point.

- **Distance** : distance between car and intersect point.
- **Bounds** : calculate intersect point that inside the car bounds.

Size offset to intersect point : additional offset to the length of the car bounds to check the closeness to the intersect point (*test scene*).

Close enough distance to stop before intersect point : car is close enough to stop in front of the intersect point if necessary (*example*) (*test scene*).

Close enough distance to stop before intersect same target node : current car is close enough to stop in front if another car approaches the same target node but with a higher priority (*example*) (*test scene*).

Close distance to change lane point : car that is too close to the lane change point is always an obstacle (*example*) (*test scene*).

Max distance to obstacle change lane : maximum distance to the obstacle in the target change lane (*example*).

Same direction value : direction of the vehicle to check for obstacles in neighboring paths (paths that start from the same point) (*example*).

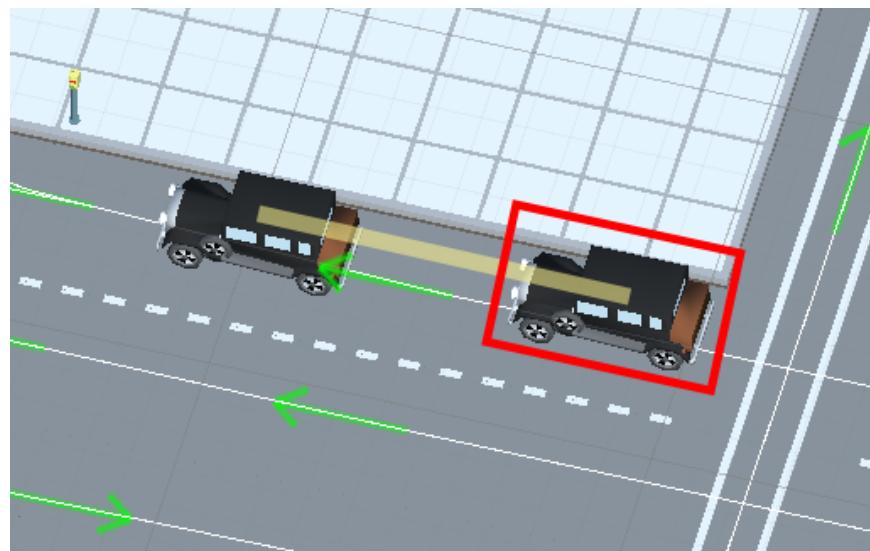
Avoid crossroad jam : car doesn't enter an crossroad if it cannot pass it without jamming (*example*) (*test scene*).

Note:

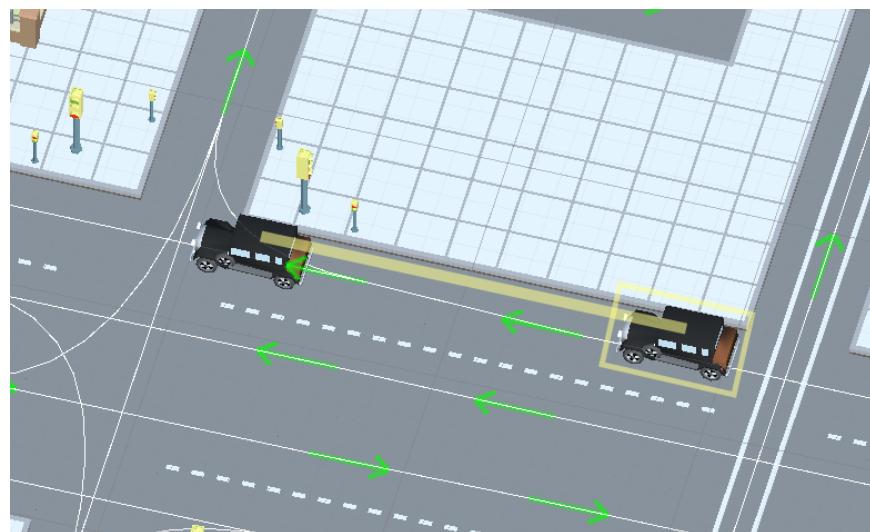
How to calculate the parameters relative to the size of the vehicle hull:

- Select the mesh renderer of the vehicle hull and insert to the *Target Car Mesh* field.
 - Press *Recalculate* button.
 - On the traffic test scene, calibrate the parameters depending on your needs.
-

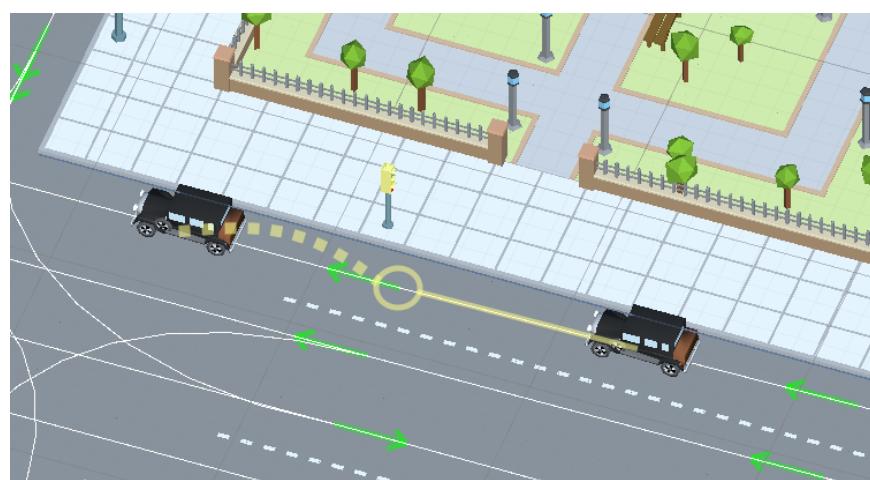
Parameter visualization:



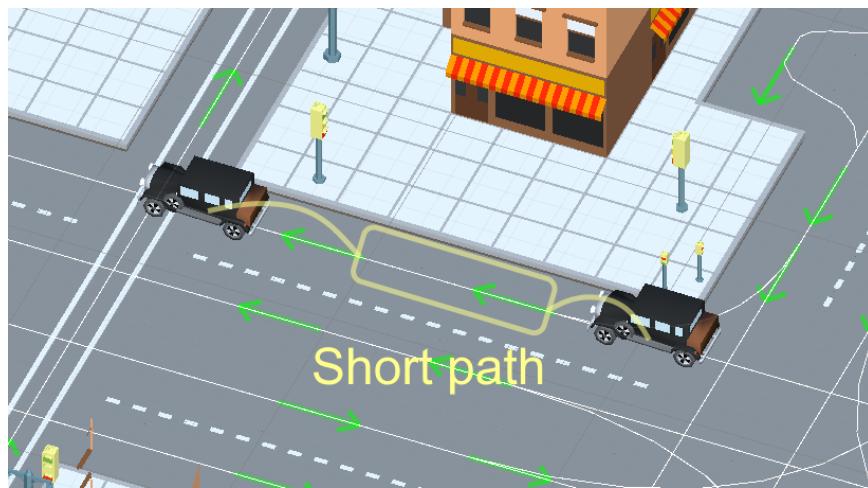
Obstacle distance example.



Approach distance example.



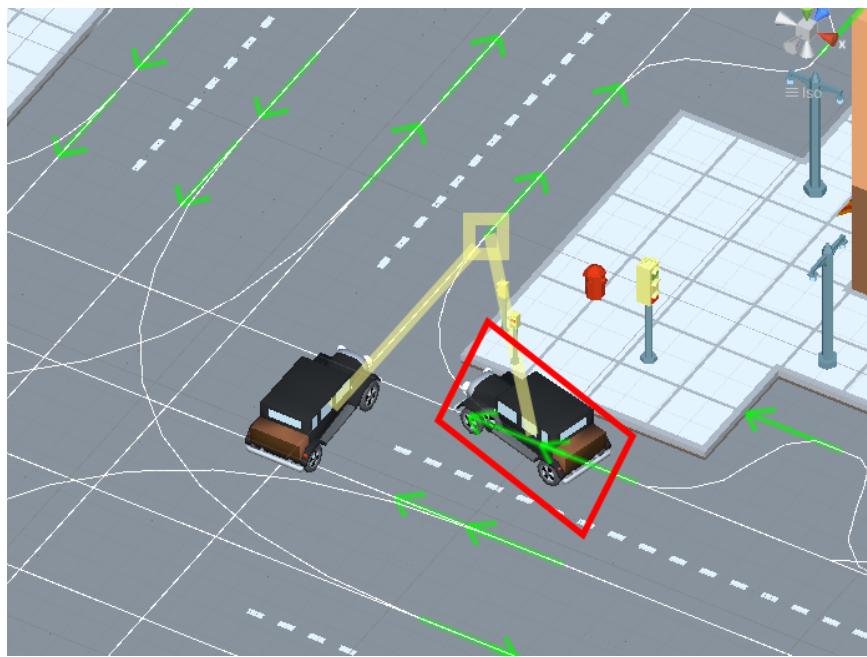
Min distance to check next ConnectedPath example.



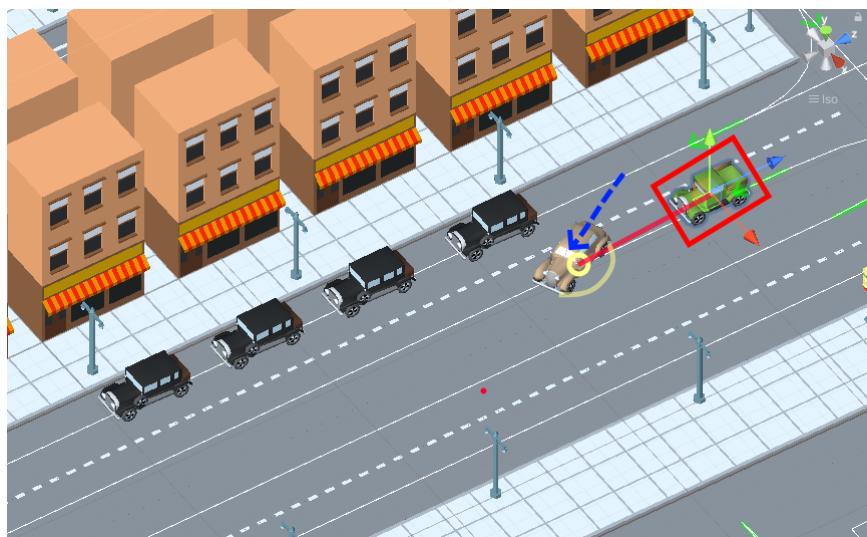
Short path example.



Calculate distance to intersect example.



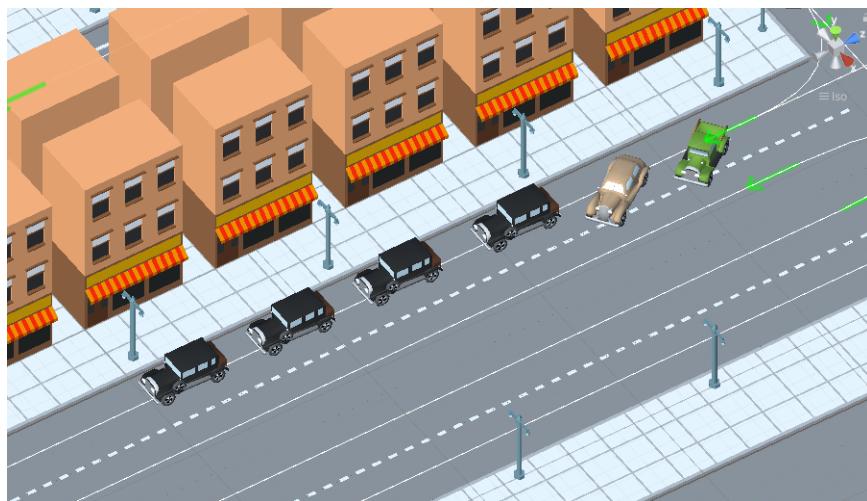
Calculate distance to intersect same target example.



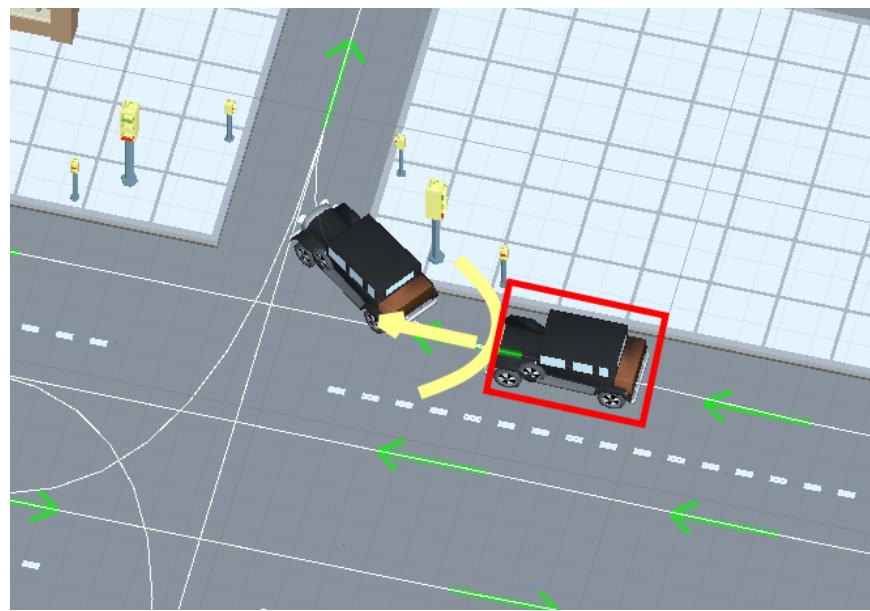
Change lane close distance to point example.



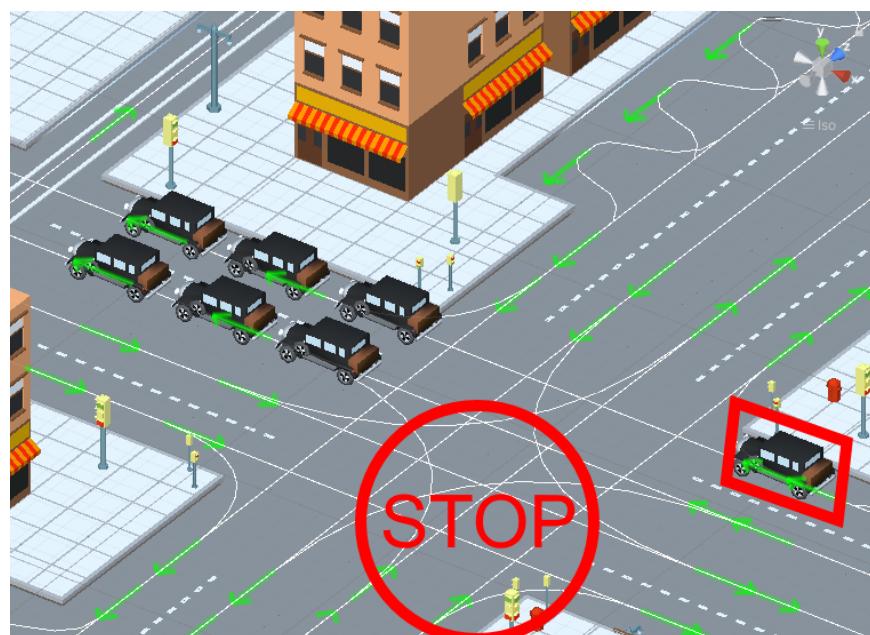
Maximum distance to the obstacle in the target change lane example.



Short path example.



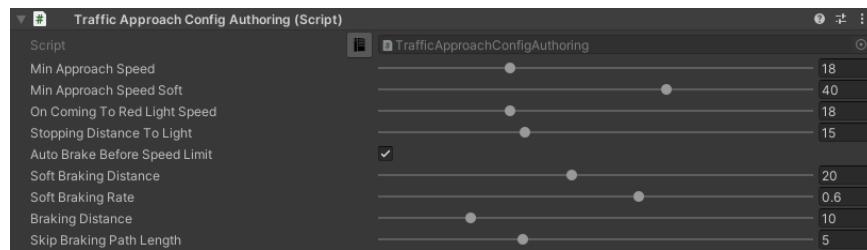
Same direction example.



Avoid crossroad jam example.

Traffic Approach Config

Config of approaching obstacles and lights (*test scene*).



Min approach speed : min approach speed.

Min approach speed soft : min approach speed soft at the long distance to obstacle.

On coming to the red light speed : slowing down the speed of the car when approaching a red light (if the segment speed limit is lower or the speed of the obstacles is lower, the lowest speed of all the conditions will be selected).

Stopping distance to light : distance at which the car slows down.

Auto brake before speed limit

The car automatically brakes to the new speed limit at the selected distance.

Soft braking distance : soft braking distance to new speed limit.

Soft braking rate : soft braking rate when the car is at a soft distance (if it is closer soft distance than the default *braking rate*).

Braking distance : braking distance to new speed limit.

Skip braking path length distance : if the next path is too short, the next one is selected.

Note: Approach distance set in *Traffic obstacle config*.

Traffic Raycast Config

Traffic raycast Config (*TrafficDetectObstacleMode* raycast or hybrid should be enabled) ([debug example](#)) ([more info](#)).



Side offset : width of raycast box.

Min/Max ray length : lenght of raycast box.

Boxcast height : height raycast box.

Ray Y axis offset : y-offset position box.

Dot direction : if the raycast is set to *Hybrid mode* than only those targets that are in front of the car with the set dot parameter will be raycasted.

Bounds multiplier : value by which the bounds is multiplied.

Traffic Change Lane Config

Config for automatic calculation of lane change by traffic (works for *paths* with the *Straight road road type* only) (*test scene*).



Can change lane : on/off ability to change lanes.

Min max change lane offset : min/max offset in the target lane depending on the speed of the car. (*example*)

Max distance to end of path : maximum distance before the end of a current path at which car can change lanes.

Min distance to last car in current lane : minimum distance to the last car in the current lane. (*example*)

Min Max distance to other cars in other lane : distance to the car in the target lane, the distance is chosen based on the current speed of the calculated car (lerp between 0 speed and max speed of the car (60 km/h by default)) (*example*)

Max distance to intersected path : distance to the crossing, if the car is close to the crossing, the ability to change lanes is disabled. (*example*)

Check frequency : frequency of lane change calculation.

Block duration after change lane : blocking the ability to change lanes after a lane change has been performed.

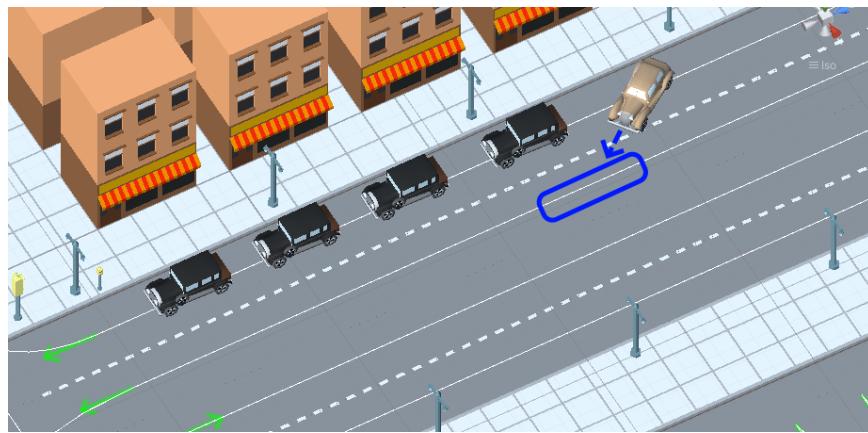
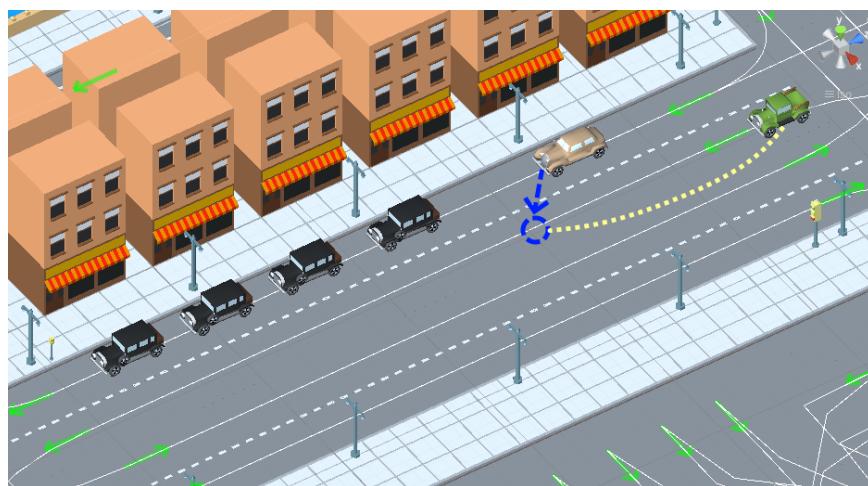
Achieve distance : distance to achieve the target lane point.

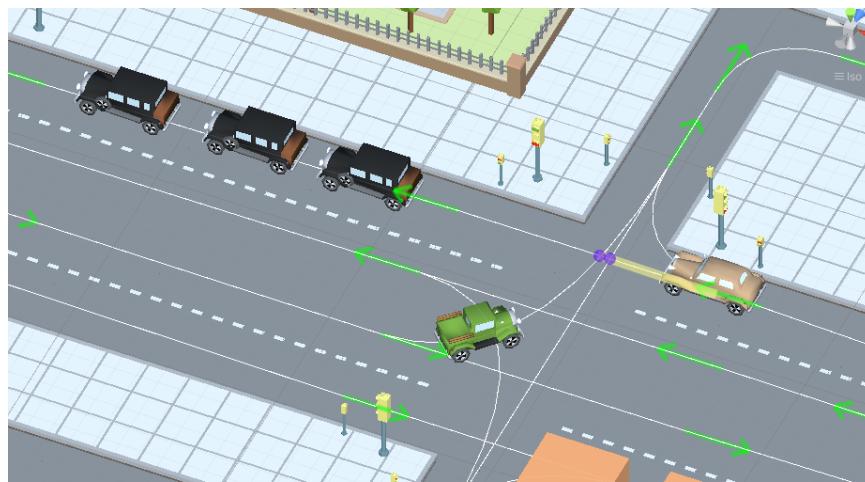
Min car count in current lane to change lane : minimum number of cars in the current lane to change lanes.

Min car lane difference count to start change lane : minimum car difference in the nearest lane to change lanes.

Change lane car speed : lane change speed.

Change lane HashMap capacity : initial capacity hashmap containing data about cars that change lanes.

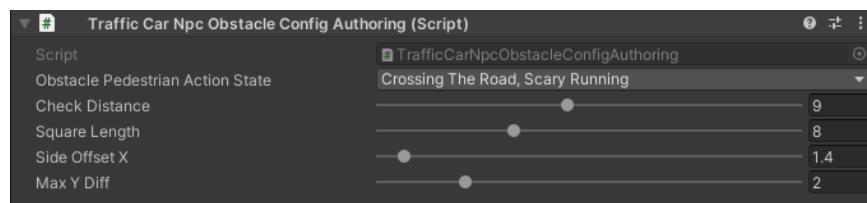
Parameter visualization:*Min/max change lane offset example.**Min distance to last car in current lane example.**Min distance to other cars in other lane example.*



Min distance to intersected path example.

Traffic Npc Obstacle Config

Config to calculate npc obstacles (*example*).



Obstacle pedestrian action state : the car only reacts to pedestrians with the selected *Action State*.

Check distance : obstacle calculation length.

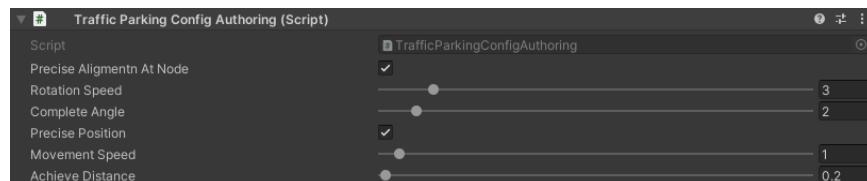
Square length : length of the obstacle calculation square.

Side offset X : width of the obstacle calculation square.

Max Y diff : maximum difference in Y-axis position between the car and the npc.

Traffic Parking Config

Config for parking cars (*test scene*).



Precise Alignments At Node : on/off precise positioning of the car's parking space.

Rotation speed : rotation speed.

Complete angle : angle at which the rotation is complete.

Precise position : on/off minor driving correction speed to parking point.

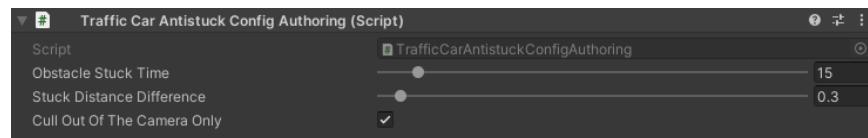
Movement speed : on/off minor driving correction speed to parking point.

Achieve distance : on/off minor driving correction speed to parking point.

Note: Read more about *parking states*.

Traffic Antistuck Config

Config to culling car in case of stuckness.



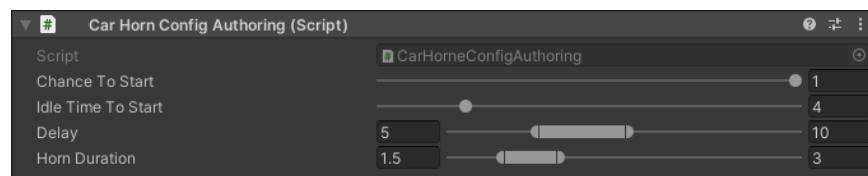
Obstacle stuck time : duration of sighting of the obstacle after which the car will be culled.

Stuck distance difference : if the car moved more than the parameter distance the *Obstacle stuck time* is reset.

Cull of out the camera only : car will be culled only if it is out of the camera's range of vision.

Traffic Horn Config

Config to sound random horns when an obstacle is detected. It can be disabled ([here](#)).



Chance to start : chance to start the horn.

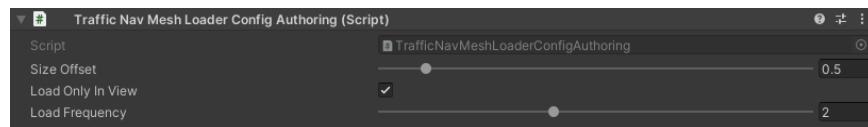
Idle time to start : idle time to start the horn.

Delay : delay between horns.

Horn duration : horn duration.

Traffic NavMesh Loader Config

Config to load Navmesh obstacles for traffic.



Size offset : size offset of loaded NavMeshObstacle.

Load only view : load NavMeshObstacle in view of camera only.

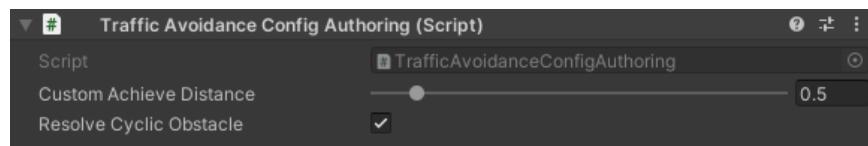
Load frequency : load frequency of NavMeshObstacle for the car.

Note:

- NavMeshObstacle loading is enabling in the *Traffic Settings* config.
 - Make sure, that the pedestrians have *NavMesh navigation*.
 - Make sure, that the *NavMeshSurface* is generated.
-

Traffic Avoidance Config

Config of traffic *avoidance*.

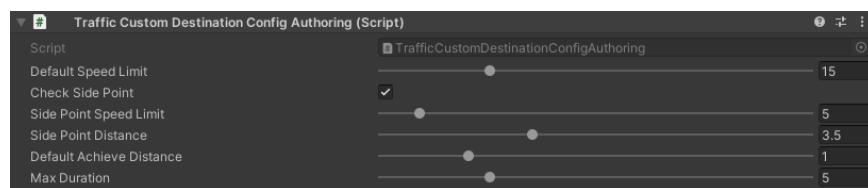


Custom achieve distance : custom achieve distance of avoidance point.

Resolve cyclic obstacle : overcome the cyclical obstacle of cars getting stuck in each other.

Traffic Custom Destination Config

Config for custom destination of the vehicles. Also used by *traffic avoidance*.



Default speed limit : default speed limit for custom destination (if user doesn't set custom).

Check side point : check that the destination is on the side of the car.

Side point speed limit : custom speed limit when destination is at side point.

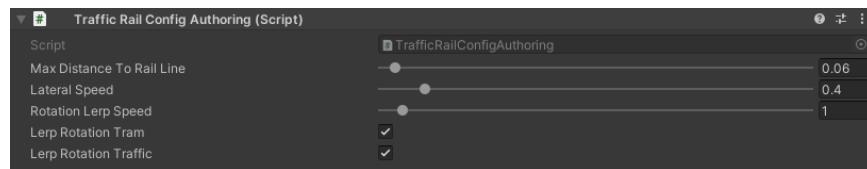
Side point distance : distance to side point.

Default achieve distance : achieve distance of destination point.

Max duration : max duration of custom destination state enabled.

Traffic Rail Config

Config for *rail movement* of the vehicles.



Max distance to rail line : maximum distance between the rail and the vehicle.

Lateral speed : lateral speed of the vehicle to align with the rail.

Rotating lerp speed : rotation lerp speed.

Lerp rotation tram : on/off rotating lerp for tram.

Lerp rotation traffic : on/off rotating lerp for default traffic.

Traffic Collision Config

The config is used in a two-car collision.



Idle duration : idling time after collision with other vehicle.

Avoid stucked collision : attempt to *avoid* a stuck collision vehicle.

Collision duration : collision time with vehicle to start *avoidance*.

Ignore collision duration : duration of collision ignoring since last event.

Calculation collision frequency : calculation frequency of the *avoidance*.

Repeat avoidance frequency : frequency of collision *avoidance* attempts.

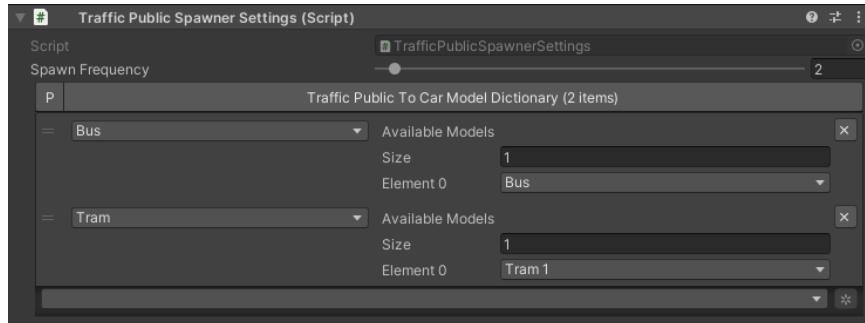
Forward direction value : dot value of the direction of the colliding vehicles along the Z axis.

Side direction value : dot value of the direction of the colliding vehicles along the X axis.

3.6.2 Public Traffic Configs

Traffic Public Spawner Settings

Spawning config for *Traffic Public* vehicles.



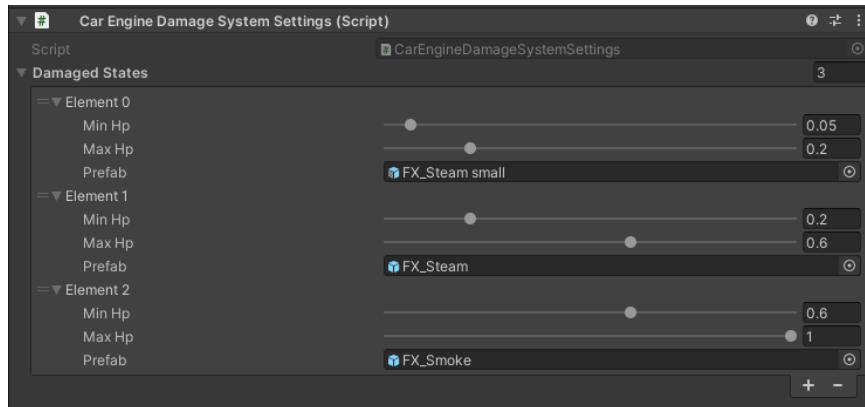
Spawn frequency : spawning frequency.

Traffic public to car model dictionary : contains data on which *CarModel* belong to *TrafficPublicType*.

3.6.3 Common Car Configs

- *Car Engine Damage System Settings*
- *Car Ignition Config*
- *Car Stopping Engine Config*
- *Car Common Sound Config*

Car Engine Damage System Settings



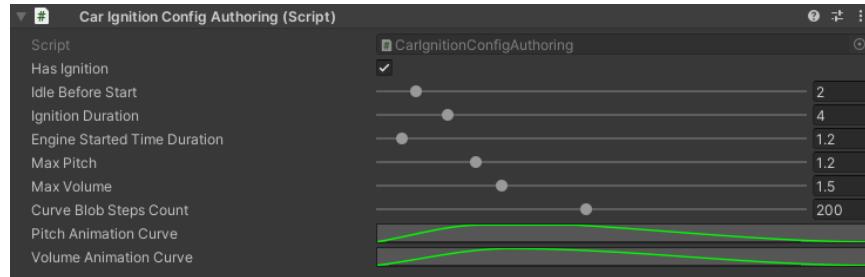
Damage states:

- **Min/max hp** : min/max vehicle health (as % of maximum health) at which engine damage starts to appear.

- **Prefab** : VFX prefab.

Car Ignition Config

Config used by *parking states*.



Has ignition : on/off ignition state of the car when the NPCs enters the car.

Idle before start : idle before starting ignition.

Ignition duration : ignition duration.

Engine started time duration : time to end of ignition state after which the engine start sound is emitted (if value = 0, engine start sound is not emitted).

Max pitch : max pitch of the ignition sound.

Max volume : max volume of ignition sound.

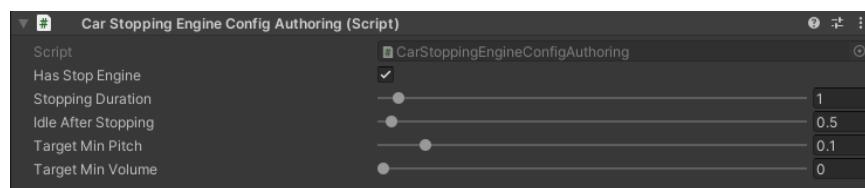
Curve blob steps count : count of steps in the blob curve.

Pitch animation curve : pitch ignition curve. Y - pitch value. X - normalized duration time.

Volume animation curve : volume ignition curve. Y - volume value. X - normalized duration time.

Car Stopping Engine Config

Config used by *parking states*.



Has stop engine :

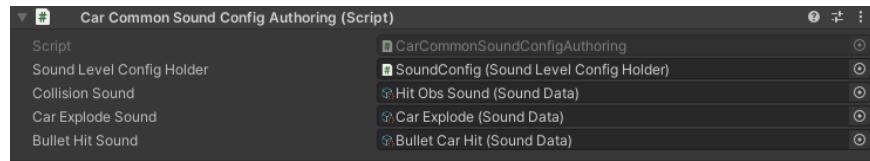
Stopping duration :

Idle after stopping :

Target min pitch :

Target min volume :

Car Common Sound Config



Collision sound :

Car explode sound :

Bullet hit sound :

3.7 Traffic Test Scene

An isolated *Traffic Test Scene* is used to test the created vehicles.

PEDESTRIAN

4.1 Pedestrian

- *How To Create*
 - *Hybrid Legacy Skin*
 - * *Factory*
 - * *Animations*
 - * *Animation authoring*
 - *Pure GPU Skin*
 - * *How To Create*
 - * *Animation authoring*
 - * *Animation Texture Data*
 - * *Crowd GPU Custom Animator*
 - *Hybrid and GPU*
 - * *How To Create*
 - * *Cull state*
 - *Ragdoll*
 - * *How To Create*
- *Navigation*
 - *NavMesh Navigating*
 - * *Installation*
 - * *How To Setup*
 - * *Pros And Cons*
 - *Local Avoidance*
 - * *How To Setup*
 - * *Pros And Cons*
 - *Agents Navigation*

- * *How To Setup*
- *Animation*
 - *Custom Animation*
- *States*
 - *Common Logic*
 - *How To Change*
 - *Custom State System*
 - *Movement State*
 - *Action State*
- *Common Info*
 - *Collision*
- *Authoring Components*
 - *PedestrianAuthoring*
 - *PlayerTargetAuthoring*
 - *Physics*
- *Configs*
 - *Pedestrian Spawner Config*
 - *Pedestrian Settings Config*
 - * *Skin Type*
 - * *Rig Type*
 - * *Entity Type*
 - * *Common Settings*
 - * *Obstacle Avoidance Type*
 - * *Pedestrian Navigation Type [NavMesh **navigation only**]*
 - * *Collision type*
 - *NavAgent Config*
 - *Obstacle Local Avoidance Config*
 - *Antistuck Config*
 - *Trigger Config*
 - *Scary Trigger Config*
 - * *Trigger settings*
 - * *Sound settings*
 - *Bench Config*
 - *Common Sound Config*
 - *State Authoring*

- * *State Dictionary*
- * *Movement State Binding Dictionary*

4.1.1 How To Create

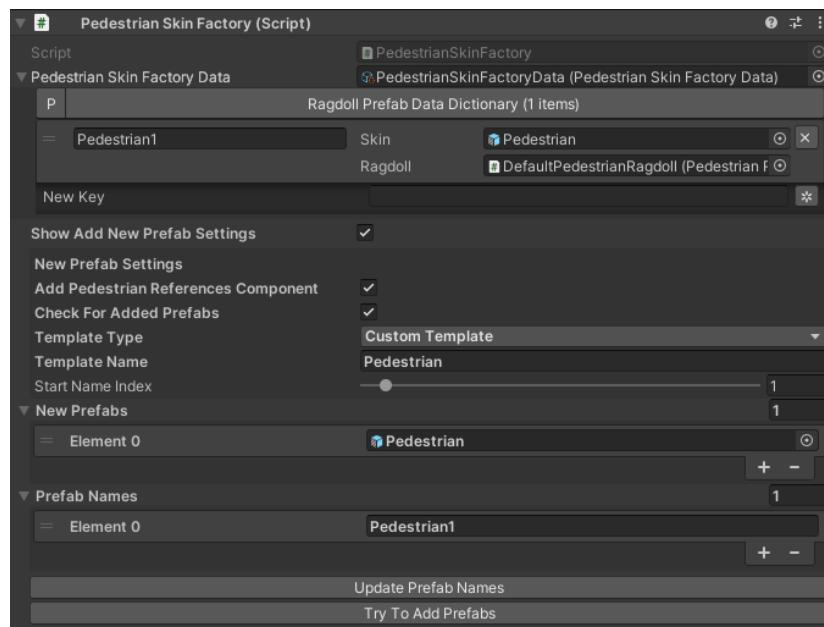
Hybrid Legacy Skin

A *Hybrid legacy skin* is a *hybrid entity* that combines the default *GameObject* (with *animator*) and the DOTS entity.

Factory

1. Open *PedestrianSkinFactory* in the scene.

Hub/Pools/Npc/Pedestrian/PedestrianSkinFactory



2. Enable the *Show Add New Prefab Settings*.
3. Drag & drop source prefabs into the *New Prefabs* field.
4. Customize the prefab names.
5. Click the *Try To Add Prefabs*.
6. If necessary, configure *Ragdoll* and assign to the *Pedestrian Skin Factory Data* (make sure *Ragdoll* is *enabled*).

Note: Each *Hybrid legacy* pedestrian prefab should have *PedestrianEntityRef* component.

Animations

By default, each pedestrian has a *PedestrianBaseController* animator.

Animation List:

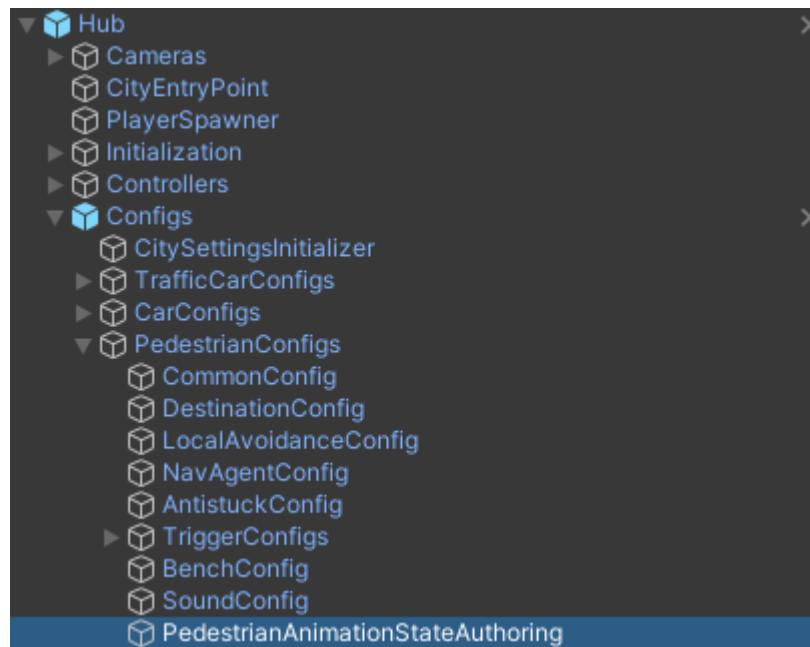
Animation name	Parameters	Value	When it starts
Walking	<ul style="list-style-type: none"> • yInput • SideMovement 	<ul style="list-style-type: none"> • 0.3 • 0 	By default
Running	<ul style="list-style-type: none"> • yInput • SideMovement 	<ul style="list-style-type: none"> • 1 • 0 	By default
Idle	<ul style="list-style-type: none"> • yInput • SideMovement 	<ul style="list-style-type: none"> • 0 • 0 	By default
Stand To Sit	<ul style="list-style-type: none"> • IsSitting 	<ul style="list-style-type: none"> • true 	By default
Sitting Idle			Starts when <i>Stand To Sit</i> is completed
Sit To Stand	<ul style="list-style-type: none"> • IsSitting 	<ul style="list-style-type: none"> • false 	Starts after <i>Sitting Idle</i>
Talking 1, 2, 3	<ul style="list-style-type: none"> • Talking 	<ul style="list-style-type: none"> • 0,1,2 	By default

Used in systems:

- LegacyAnimatorSystem
- LegacyAnimatorCustomStateSystem

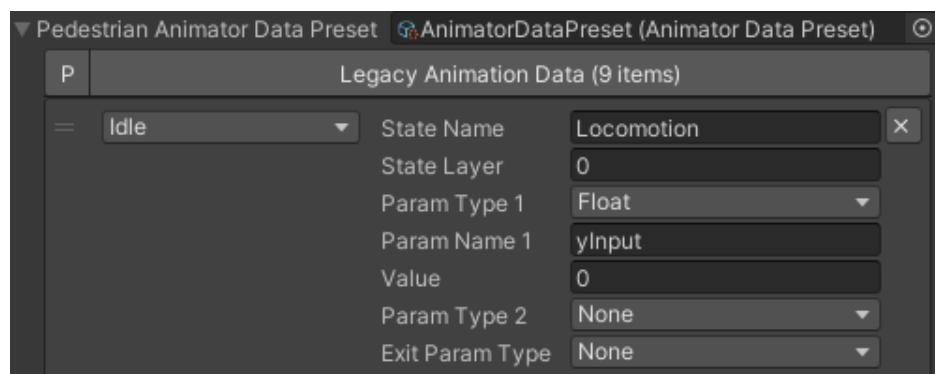
Animation authoring

- Add your animation to *AnimationState* script file.
- In the scene find:



Hub/Configs/PedestrianConfigs/PedestrianAnimationStateAuthoring.

- Add your animation to the list & enter condition to start the animation from the assigned *Animator*:



- **State name** : state name of the animation in the *Animator*.
- **State layer** : number of the layer where the animation is stored in the *Animator*.
- **Param 1** : first parameter to start animation in the *Animator*.
- **Param 2** : second parameter to start animation in the *Animator* [optional].
- **Exit param** : parameter to exit current animation in the *Animator* [optional].

- How to play animation described [here](#).

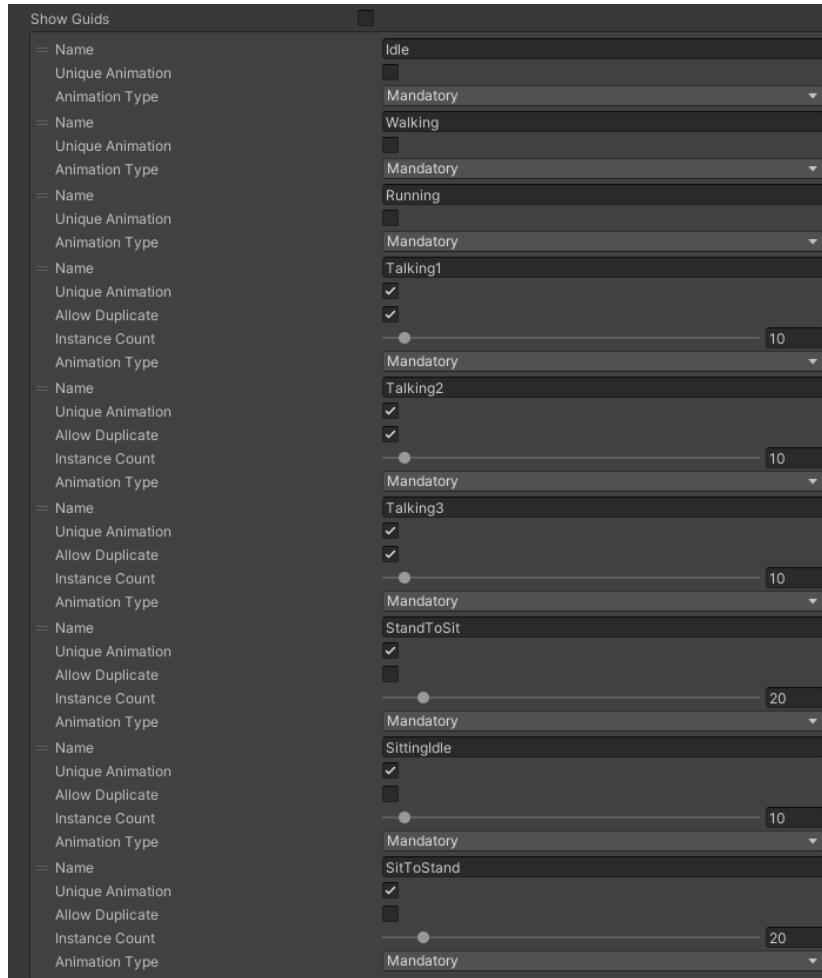
Pure GPU Skin

Pure GPU skin is a *pure entity* that combines the GPU texture animations and the DOTS entity.

How To Create

1. *Create textures and animation sheets* in the *Animation baker* tool.
2. Create *Animation Collection* from the project context .

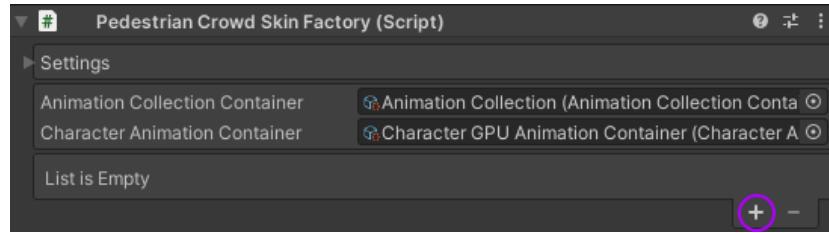
Spirit604/Animation Baker/Animation Collection



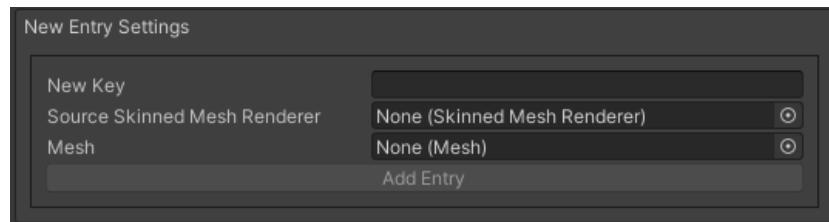
3. Add and customize desired animations data.
4. Open in the scene *PedestrianCrowdSkinFactory*.

Hub/Pools/Npc/Pedestrian/PedestrianCrowdSkinFactory

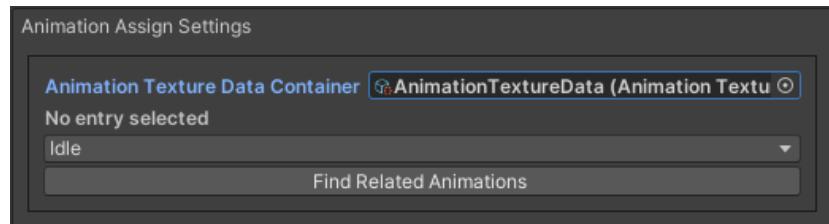
5. Assign *Animation Collection* to *PedestrianCrowdSkinFactory*.



6. Click + to show *New Entry* panel.

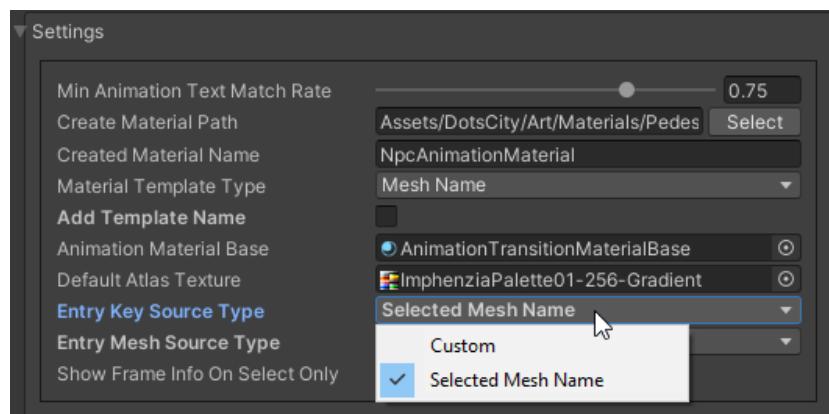


7. Select created *Baked Animation Sheet Data*.

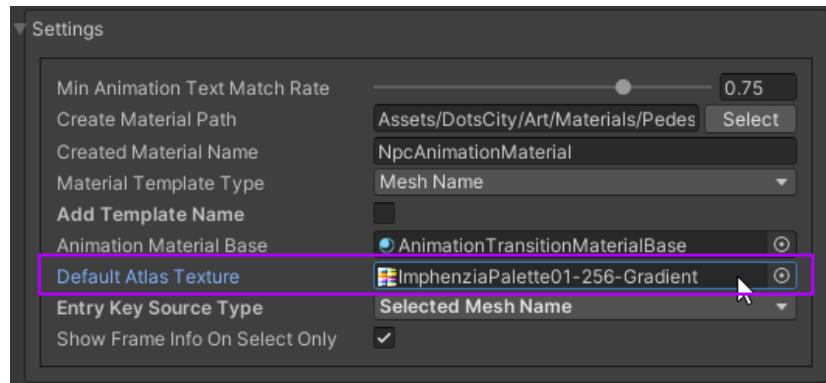


8. Open factory settings.

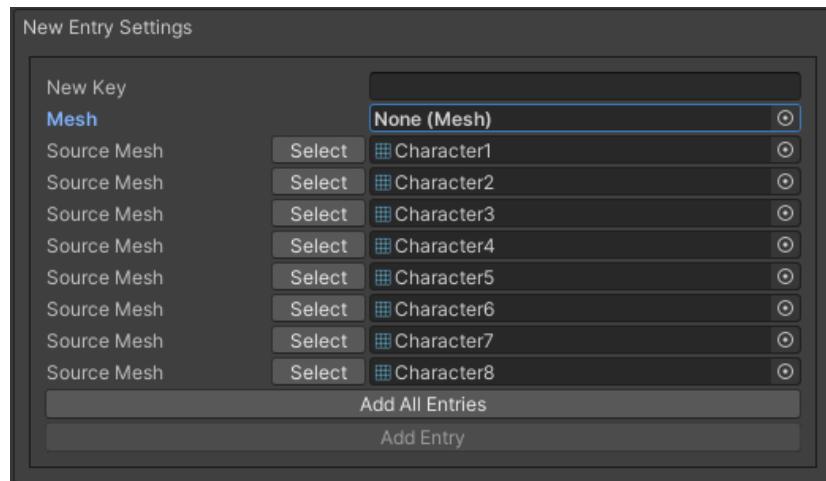
9. Select *Entry Key Source Type* to *Selected Mesh Name* (or select *Custom* if you want to enter the name manually).



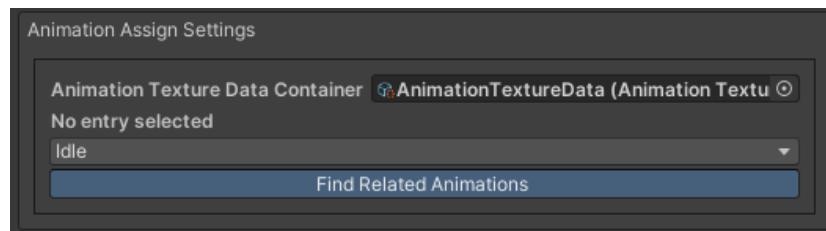
10. Assign *Default Atlas Texture* (if selected pedestrians have the same texture material).
[optional step]



11. One by one click *Select* and *Add entry* button. Or click the *Add all entries* button to add all entries in the container.

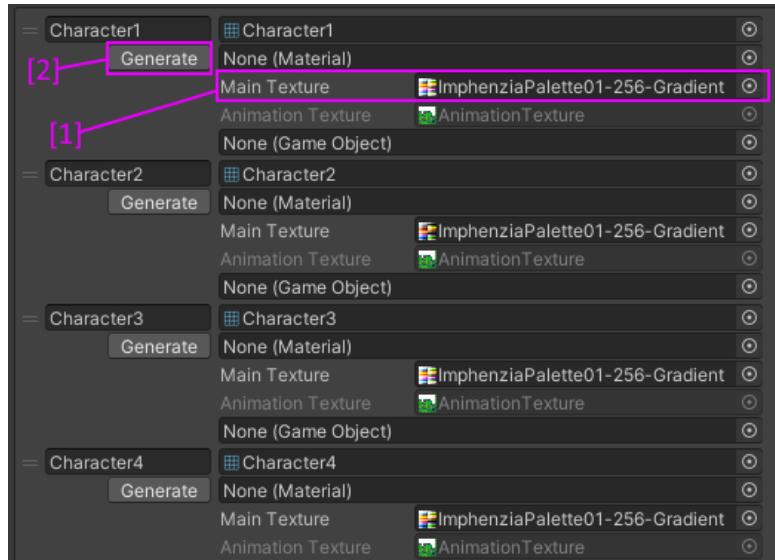


12. Turn on the *Find Related Animations* button.



13. Generate Animation Material.

1. Assign main texture of selected model [if missing].
2. Press the *Generate* button.



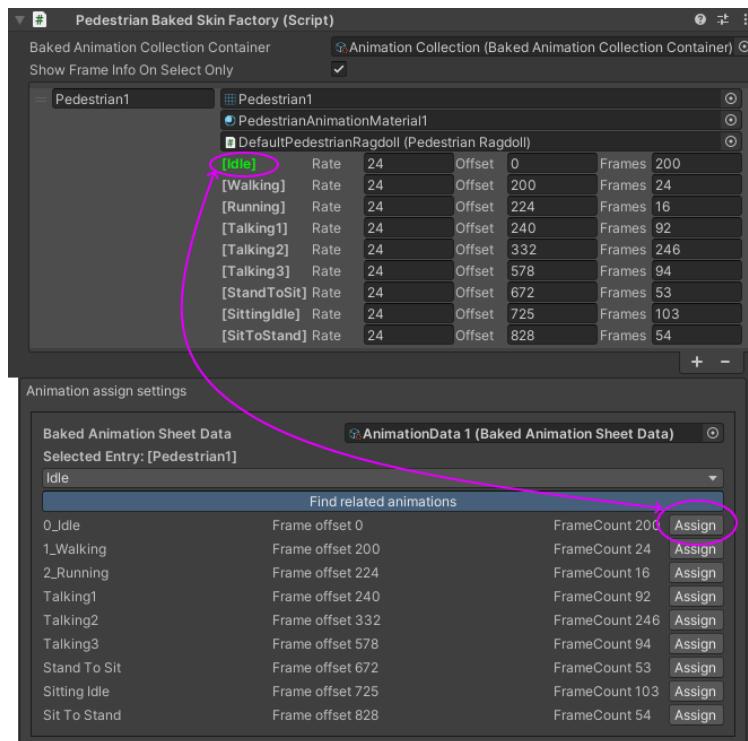
14. Select entry & assign animations:

1. Animation baker binding:

1. *Bind* the animation on the baking texture step.

2. Manual way:

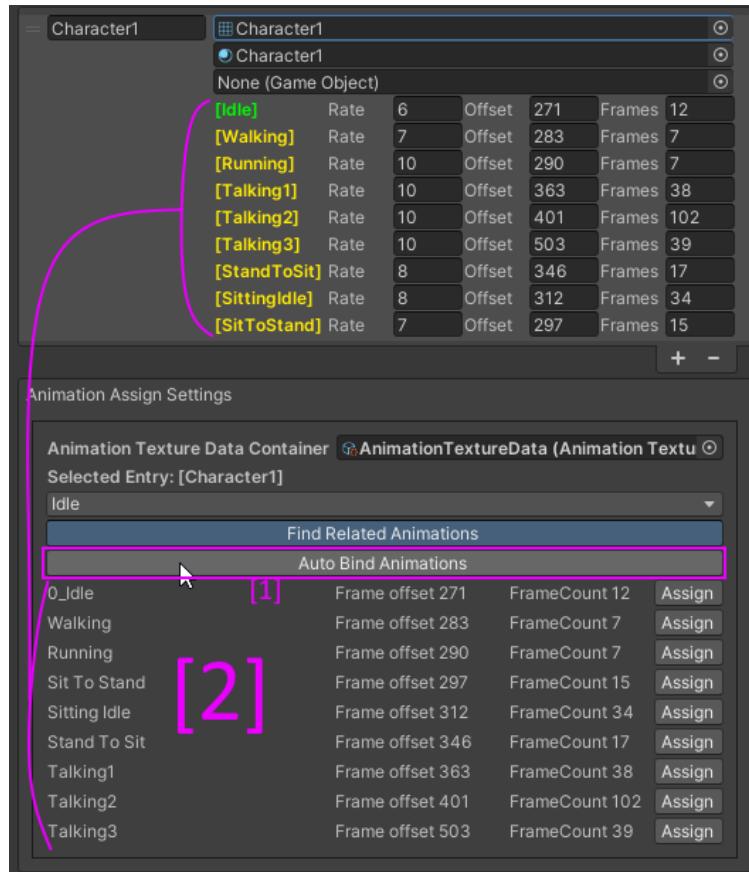
1. In the inspector, select the animation that you want to assign to the selected character.



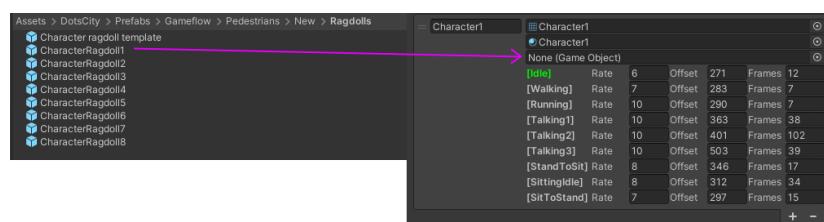
2. Press the *Assign* button according to the selected animation in *Animation Texture Data*.

3. Automated way:

1. Automatic assignment works if the animation in the list matches (or partially matches) the animation name in the selected container.
2. Press the *Auto Bind Animations* button.
3. Make sure, that all animations are assigned.



15. Assign animations to each entry in the same way.
16. **Add custom optional animations for the desired pedestrians [optional step].**
 1. In the *Animation Collection* add new *Optional* animations.
 2. Tick on *Show optional animation popup* in Pedestrian crowd skin factory settings.
 3. Add desired optional animations in the character list of the factory.
 4. Bind added animations.
17. Assign *Ragdolls* [optional step].

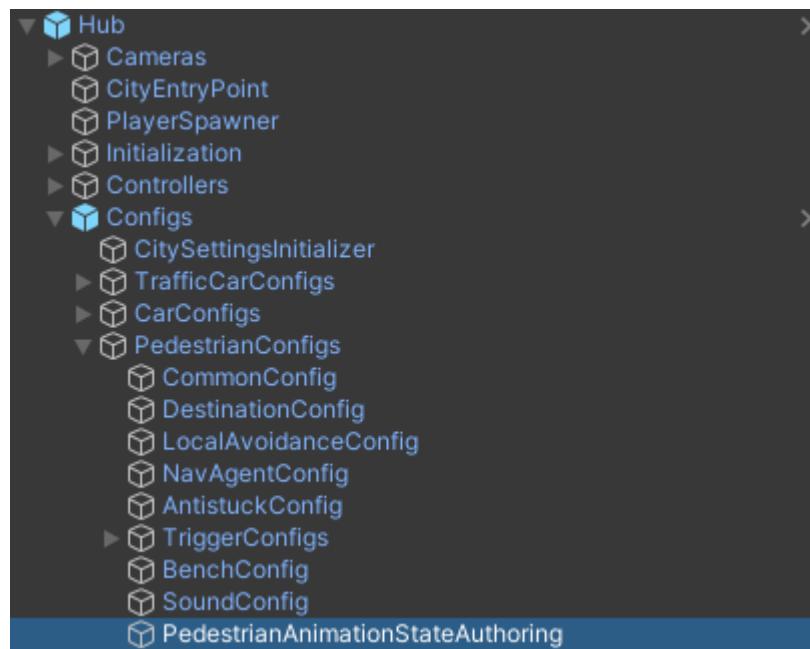


Used in systems:

- GPUAnimatorSystem
- GPUAnimatorCustomStateSystem

Animation authoring

- Add your animation to *AnimationState* script file.
- In the scene find:



Hub/Configs/PedestrianConfigs/PedestrianAnimationStateAuthoring.

- Add binding in the list (*AnimationState* is a key, *Animation* from *Animation collection* is a value)

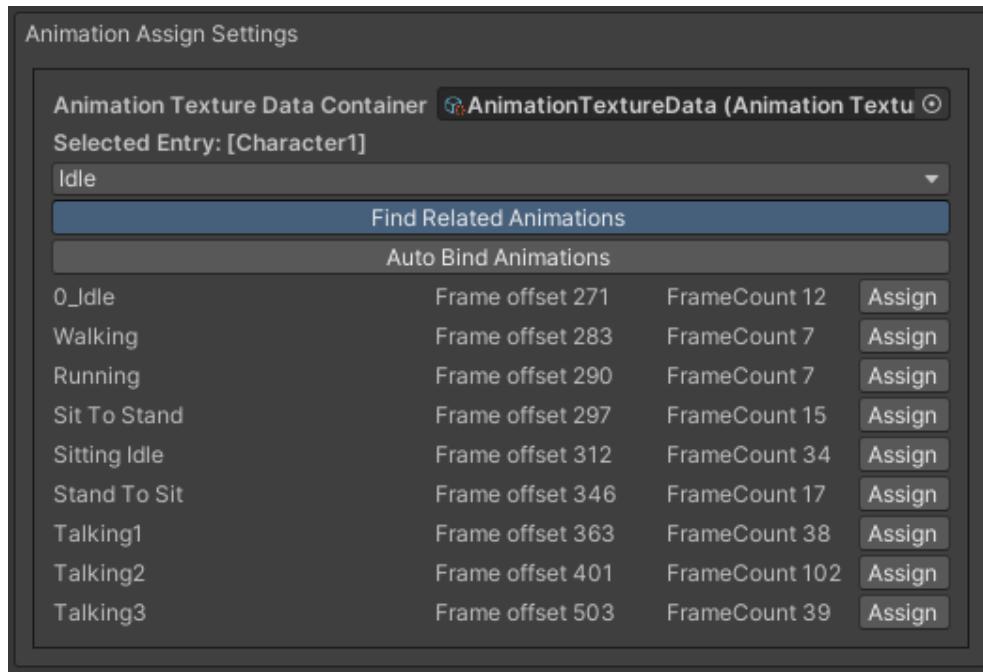
Pure Animation Data (9 items)			
= Walking	Animation	Walking	x
= Idle	Animation	Idle	x
= Running	Animation	Running	x
= Talking 1	Animation	Talking1	x
= Talking 2	Animation	Talking2	x
= Talking 3	Animation	Talking3	x
= Stand To Sit	Animation	StandToSit	x
= Sitting Idle	Animation	SittingIdle	x
= Sit To Stand	Animation	SitToStand	x
Default		*	

Example.

- How to play animation described [here](#).

Animation Texture Data

Data about baked animations in texture ([How to create](#)).



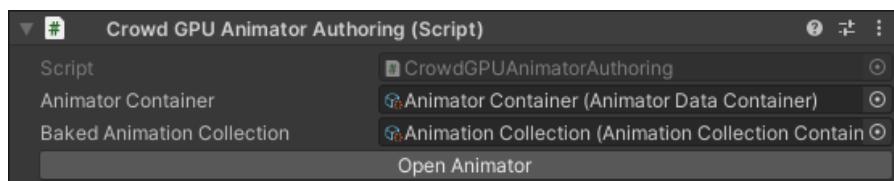
Crowd GPU Custom Animator

The Crowd GPU Custom animator is used for transitions between baked animations (implemented by *CrowdAnimatorTransitionSystem* system).

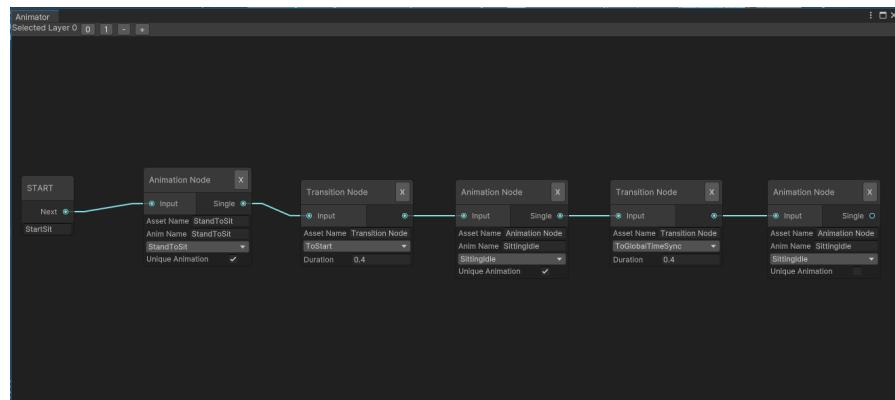
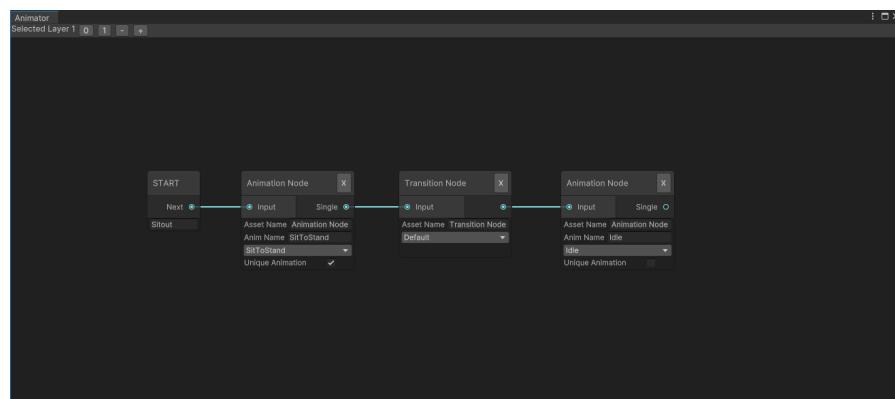
How To Create Transition:

1. Open in the scene *CrowdGPUAnimatorAuthoring*.

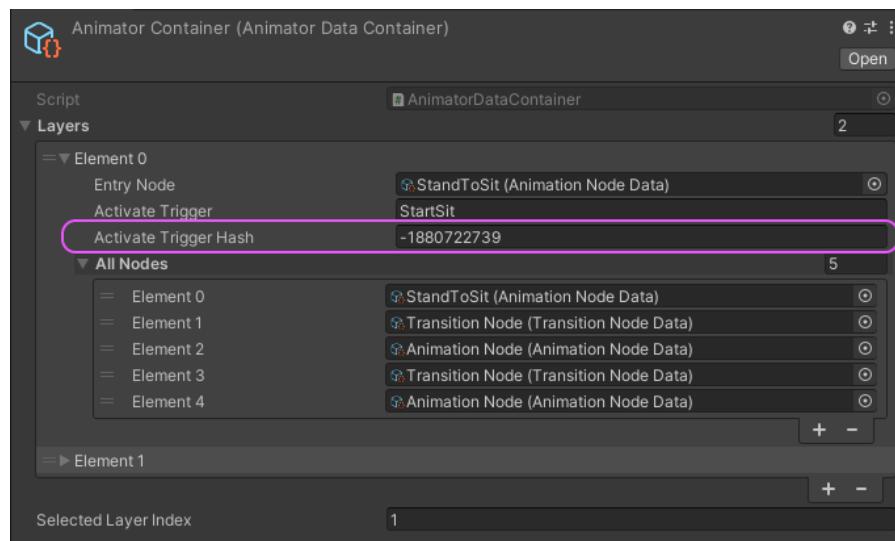
Hub/Configs/BakerRefs/Settings/CrowdGPUAnimatorAuthoring



2. Create an *Animator Data Container* from the project context and assign it to the animator (if required).
3. Assign *Animation Collection* the same as in the *PedestrianCrowdSkinFactory*.
4. Press the *Open Animator* button.
5. Create a *new transition layer* (if needed).
6. Enter the name of the trigger in the *StartNode*.
7. Create and connect *AnimationNode* and *TransitionNodes*.

*Start sit transition example.**Sitout transition example.*

8. Copy & paste the *generated hash* from the *AnimatorContainer* into the code (usage example).



Used in systems:

- GPUAnimatorCustomStateSystem

Hybrid and GPU

New hybrid GPU mode that allows you to mix hybrid animator models for near and GPU animation for far at the same time.

How To Create

- Create *Legacy* pedestrians.
- Add desired animations in the *Animation state authoring* for *Legacy* pedestrians.
- Create *GPU* pedestrians.
- Add desired animations in the *Animation state authoring* for *GPU* pedestrians.
- Make sure that the number & the order of *Legacy* & *GPU* of the models are the same in the factories (*PedestrianSkinFactory* & *PedestrianGPUSkinFactory*).
- How to play animation described [here](#).

Cull state

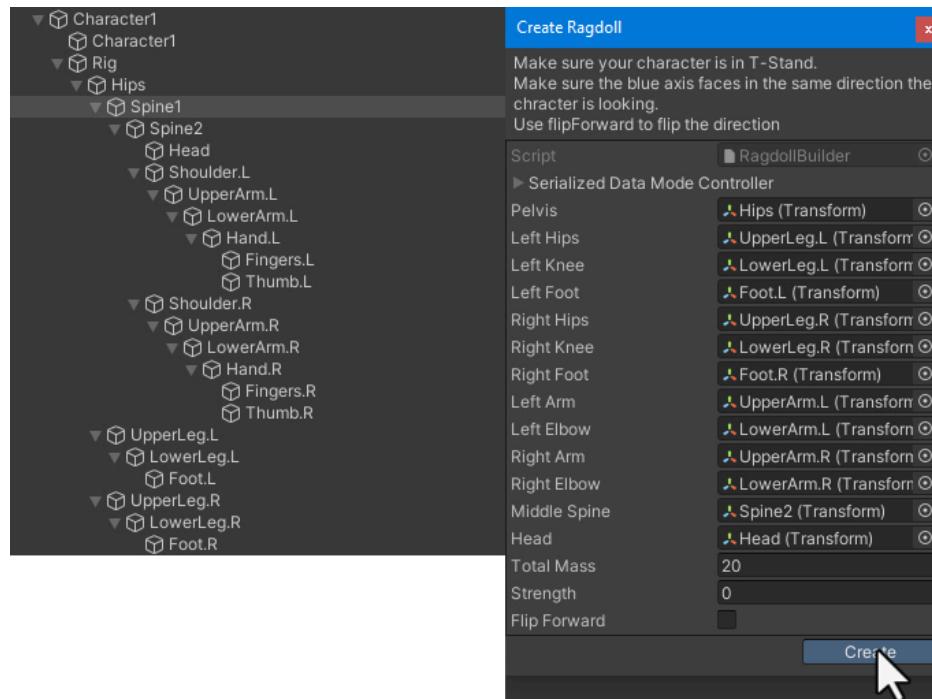
- *InViewOfCamera*: *Hybrid* legacy skin is enabled.
- *CloseToCamera*: *GPU* skin is enabled.

Ragdoll

Ragdoll is created at the scene of the pedestrian's death. Make sure ragdoll is *enabled*.

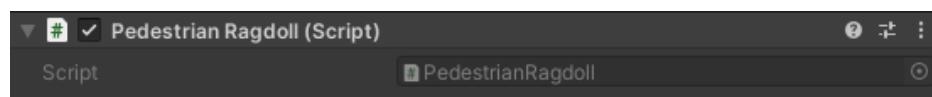
How To Create

1. Add all the colliders and rigidbodies to character according to the [RagdollWizard](#) tutorial.

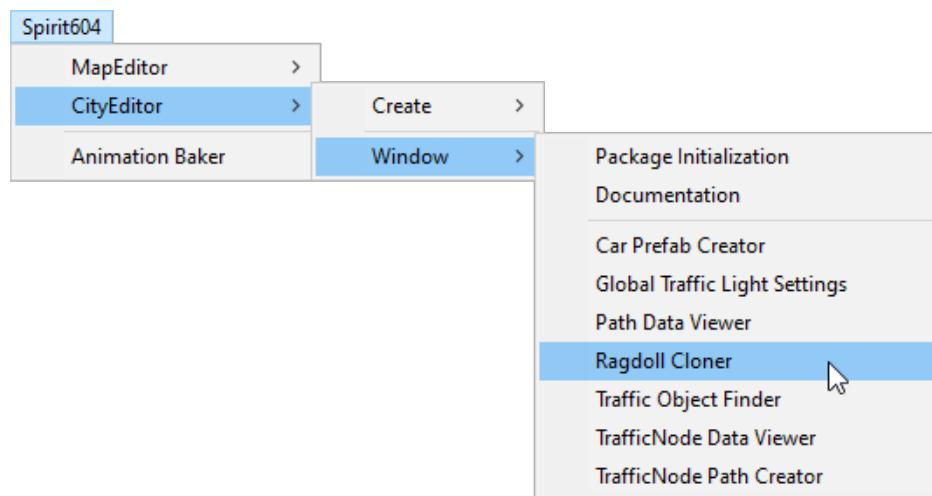


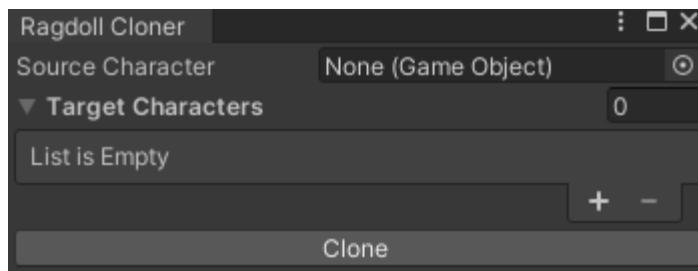
RagdollWizard example.

2. Add the *PedestrianRagdoll* component.

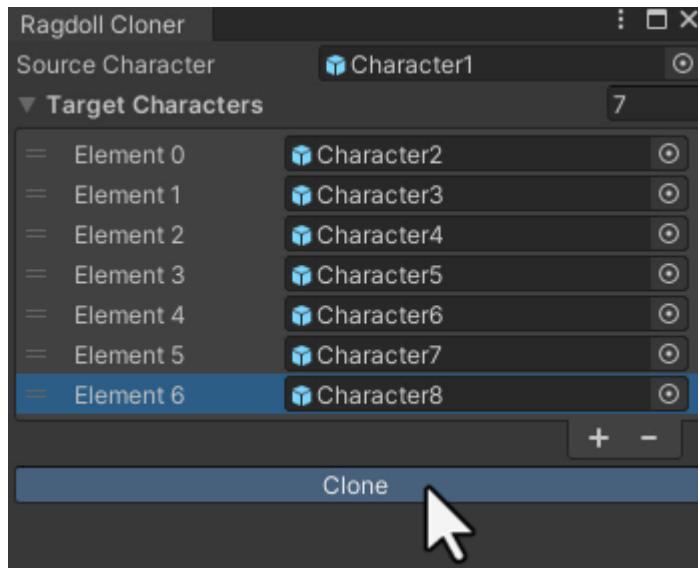


3. For the remaining characters, open the *RagdollCloner* tool.





4. Assign the source character created first and the target remaining characters.



5. Click the *Create* button.

6. Assign the result to *PedestrianHybridLegacyFactory* or *PedestrianCrowdSkinFactory* depending on the *type of rig* you have chosen.

Note:

- Implemented by *RagdollSystem*.
 - Currently only collides with default colliders
 - Make sure, that the scene contains default colliders.
 - Read more info about the *Physics Transfer Service* on how to clone legacy colliders.
-

4.1.2 Navigation

Navigation is used for pedestrian obstacle avoidance.

There are 3 types of navigation:

NavMesh Navigating

DOTS navigation on [NavMeshSurface](#).

Useful links:

- [NavAgent Config](#)
- [Test scene.](#)

Installation

- Check that the [Navigation package](#) is installed.
- Make sure that navigation is enabled in the [General Config](#).
- Ensure that [NavMeshObstacle](#) is enabled for traffic.
- Each dynamic object in the scene must have a [NavMeshObstacle](#) component.

How To Setup

- Create new gameobject & add [NavMeshSurface](#) component.
- Set [Agent type](#) to *Pedestrian* & press the *Bake* button in the created [NavMeshSurface](#).
- Set [Avoidance type](#) to *Calc Nav Path*.
- Set [Pedestrian navigation type](#) to *Temp* or *Persist* mode.

Pros And Cons

Pros:

- High precision.
- Can avoid any obstacle.

Cons:

- High CPU load.

Local Avoidance

DOTS system to avoid local obstacles (vehicles).

Useful links:

- [Local Avoidance Config](#)
- [Test scene.](#)

How To Setup

- Set the *Avoidance type* to *Local Avoidance*.
- Configure *Local Avoidance Config*.

Pros And Cons

Pros:

- Low CPU load.

Cons:

- Can avoid vehicles only.
- Works on flat surfaces only.

Agents Navigation

DOTS navigation on NavMeshSurface using [Agents Navigation](#) plugin.

How To Setup

- Make sure that you purchased & downloaded [Agents Navigation](#) plugin.
- Set the *Avoidance type* to *Agents Navigation*.
- Enable the *Auto Add Agent Components* option for quick prototyping & customize the settings in the *Agents Navigation Config Authoring* tab, or add agent authoring components to the *PedestrianEntity* prefab from the *Agents Navigation* sample for more flexible settings. ([Agents Navigation doc](#))
- Ensure that *NavMeshObstacle* is enabled for traffic.

4.1.3 Animation

Custom Animation

To handle custom animation, follow these steps:

- Add custom animations in the *Animation state authoring* for pedestrians.
 - *Hybrid skin* (if you are using Hybrid animations).
 - *GPU skin* (if you are using GPU animations).
- Add custom animator state by code:

```
// IJobEntity entity example
void Execute(
    Entity entity,
    ref AnimationStateComponent animationStateComponent)
{
    // Some condition
    bool condition = true;
```

(continues on next page)

(continued from previous page)

```

        if (condition)
    {
        // Replace 'AnimationState.StandToSit' with your animation.
        AnimatorStateExtension.AddCustomAnimatorState(ref_
        ↵CommandBuffer, entity, ref animationStateComponent, AnimationState.
        ↵StandToSit);
    }
}

```

- Change to new state if required, code:

```

// IJobEntity entity example
void Execute(
    Entity entity,
    ref AnimationStateComponent animationStateComponent)
{
    // Some condition
    bool condition = true;

    if (condition)
    {
        // Replace 'AnimationState.SitToStand' with your animation.
        AnimatorStateExtension.ChangeAnimatorState(ref_
        ↵CommandBuffer, entity, ref animationStateComponent, AnimationState.
        ↵SitToStand);
    }
}

```

- After all the custom animations have been played, turn off the custom animation state.

```

// IJobEntity entity example
void Execute(
    Entity entity,
    ref AnimationStateComponent animationStateComponent)
{
    // Some condition
    bool condition = true;

    if (condition)
    {
        AnimatorStateExtension.RemoveCustomAnimator(ref_
        ↵CommandBuffer, entity);
    }
}

.. note::
    For an example of a system, please read the script below:
    * BenchStateSystem.cs.

```

4.1.4 States

Common Logic

1. Custom system set the next *Action state* in the *NextStateComponent* by utils method.
 - **bool NextStateComponent.TryToSetNextState(ActionState.WaitForGreenLight, ref destinationComponent)**
Example method, if state can't be set, then target swap back.
 - **bool NextStateComponent.TryToSetNextState(ActionState.WaitForGreenLight)**
Example method without retargeting.
2. *PedestrianStateSystem* is checking *NextStateComponent* for non-default next *Action state* and checks if the list of available states contains that state.
Available state list for the current state can be defined [here](#).
3. If state is available, set *StateComponent* to the new state and set *Movement state* according to *Movement binding data*.
4. After the *Movement state* is set to a new state, the *MovementStateChangedEventTag* tag is enabled & new animation movement animation is running in the appropriate animation system.
 - For Legacy skin *LegacyAnimatorSystem*.
 - For GPU skin *GPUAnimatorSystem*.
5. If you want to set the *Custom animation* for pedestrian read [this](#).

How To Change

```
// Switch state example

[WithDisabled(typeof(WaitForGreenLightTag))]
[BurstCompile]
public partial struct CheckTrafficLightJob : IJobEntity
{
    void Execute(
        ref DestinationComponent destinationComponent,
        ref NextStateComponent nextStateComponent,
        EnabledRefRW<WaitForGreenLightTag> waitForGreenLightTagRW,
        EnabledRefRW<CheckTrafficLightStateTag> checkTrafficLightStateTagRW)
    {
        // Tag is triggering system
        checkTrafficLightStateTagRW.ValueRW = false;

        //Example red traffic light flag logic
        bool redLight = true;

        if (redLight)
        {
            // If the next state is available, start waiting for a
            ↵green light.
    }
}
```

(continues on next page)

(continued from previous page)

```

        if (nextStateComponent.TryToSetNextState(ActionState.
    ↵WaitForGreenLight, ref destinationComponent))
    {
        // Some logic

        waitForGreenLightTagRW.ValueRW = true;

        // If the entity has a custom animation for this
    ↵state, use the 'AnimatorStateExtension.AddCustomAnimatorState' method
    }
    else
    {
        // Otherwise return to previous destination, for
    ↵example
    }
}
else
{
    // Not red traffic light then set cross the road state
    nextStateComponent.TryToSetNextState(ActionState.
    ↵CrossingTheRoad);
}
}
}

```

Custom State System

```

// Custom state system example

[BurstCompile]
public partial struct CustomStateJob : IJobEntity
{

    void Execute(
        ref StateComponent stateComponent,
        ref NextStateComponent nextStateComponent,
        EnabledRefRW<WaitForGreenLightTag> waitForGreenLightTagRW)
    {
        // Some logic for waiting traffic light
        bool greenLight = true;

        if (!greenLight)
        {
            // Some logic while waiting for the green light
        }

        // If the traffic light is green or another system has changed
    ↵state, leave current system
        var leaveState = greenLight || !stateComponent.HasActionState(in
    ↵nextStateComponent, ActionState.WaitForGreenLight);
    }
}

```

(continues on next page)

(continued from previous page)

```

        if (leaveState)
        {
           waitForGreenLightTagRW.ValueRW = false;

            if (greenLight)
            {
                nextStateComponent.TryToSetNextState(ActionState.
←CrossingTheRoad);
            }
            else
            {
                // Otherwise logic if the state is interrupted
←with another system
            }
        }
    }
}

```

Movement State

- Default
- Idle
- Walking
- Running

Action State

- **Default** : no state.
- **Idle** : when a pedestrian is waiting.
- **MovingToNextTargetPoint** : when going from *PedestrianNode* to *PedestrianNode* (excluding crosswalk).
- **WaitForGreenLight** : when a pedestrian is waiting for a green traffic light.
- **CrossingTheRoad** : when a pedestrian goes crossing a crosswalk.
- **ScaryRunning** : activated when a pedestrian runs away in a panic (for example, the sound of a gunshot or the death of a pedestrian nearby).
- **Sitting** : when a pedestrian is sitting.
- **Talking** : when a pedestrian is talking.

Note: You can edit state logic [here](#).

4.1.5 Common Info

Collision

In some cases pedestrians can get stuck in obstacles (vehicles), to solve this problem, adjust the [Antistuck config](#).

4.1.6 Authoring Components

Authoring components that make up the pedestrian entity.

PedestrianAuthoring

Contains the main components of pedestrian entity **[required]**.

PlayerTargetAuthoring

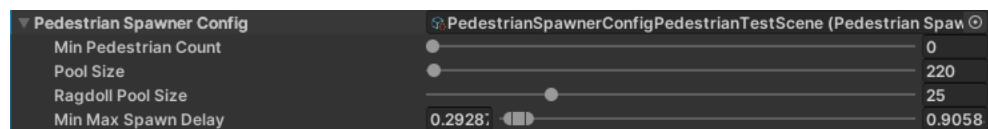
Component for player targeting systems **[optional]**.

Physics

PhysicsBody and *PhysicsShape* components for physics related systems **[optional]**.

4.1.7 Configs

Pedestrian Spawner Config



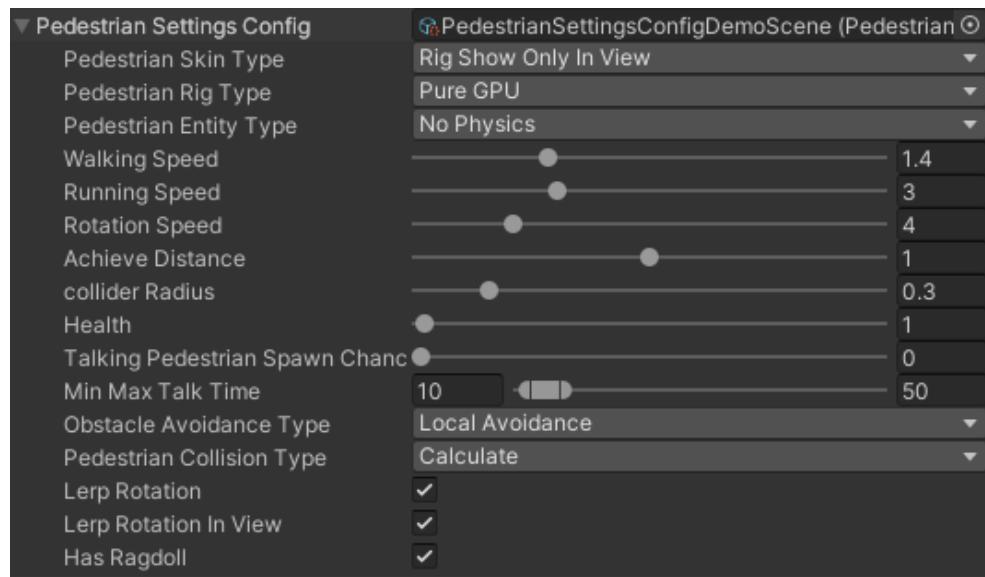
Min pedestrian count : number of pedestrians in the city.

Pool size : pool size of [HybridLegacy](#) skins.

Ragdoll pool size : [pedestrian ragdoll pool size](#).

Min/Max spawn delay : minimum and maximum delay between spawn iterations.

Pedestrian Settings Config



Skin Type

- **Rig show only in view** : rig skin will be loaded in the camera's view area.
- **Rig and dummy** : rig will be in the camera's view, and the dummy skin will be out of the camera's view.
- **Dummy show only in view** : dummy skin will be loaded in the camera's view area.
- **Rig show always** : rig skin will be loaded when the entity is created and will exist until it is destroyed.
- **Dummy show always** : dummy skin will be loaded when the entity is created and will exist until it is destroyed..
- **No skin** : entities without a skin will be created.

Rig Type

- **Hybrid legacy** : *hybrid entity with animator component.*
- **Pure GPU** : *pure entity with gpu animations.*
- **Hybrid and GPU** : *New hybrid GPU* mode that allows you to mix hybrid animator models for near and GPU animation for far at the same time.

Entity Type

- **No physics** : pedestrian not contains *PhysicsShape* component.
- **Physics** : pedestrian contains *PhysicsShape* component.

Common Settings

Pedestrian collider radius : pedestrian collider radius for *No physics* type.

Walking speed : walking speed.

Running speed : running speed.

Rotation speed : rotation speed.

Health : number of hit points for pedestrians.

Talking pedestrian spawn chance : chance of spawning talking pedestrians

Min/Max talk time : min/max talk time.

Obstacle Avoidance Type

Calc nav path : navigating based on *NavMesh* (*config*).

Local avoidance : simple *obstacle avoidance* navigation (*config*).

Agents navigation : navigating with *Agents Navigation* plugin (*how to setup*).

Pedestrian Navigation Type [NavMesh navigation only]

- **Temp** : navigation will be enabled if there is an obstacle in front of pedestrian.
- **Persist** : navigation is always on.
- **Disabled**

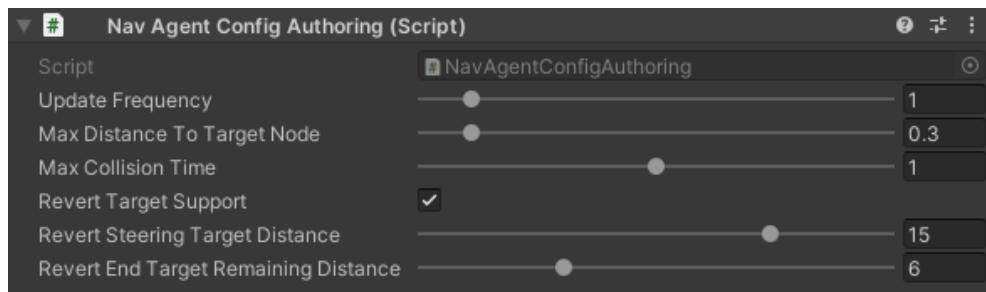
Collision type

- **Calculate** : collision is calculated manually (*for NoPhysics type*).
- **Physics** : collision is calculated with *Unity.Physics* (*for Physics type*).
- **Disabled**

Has ragdoll : on/off *ragdoll* for pedestrian.

NavAgent Config

Config for *NavMesh* navigating.



Update frequency : how often the nav target can be updated.

Max distance to target node : distance to nav path node.

Max collision time : if the pedestrian is stuck for more than the collision time, the anti-stuck will be activated.

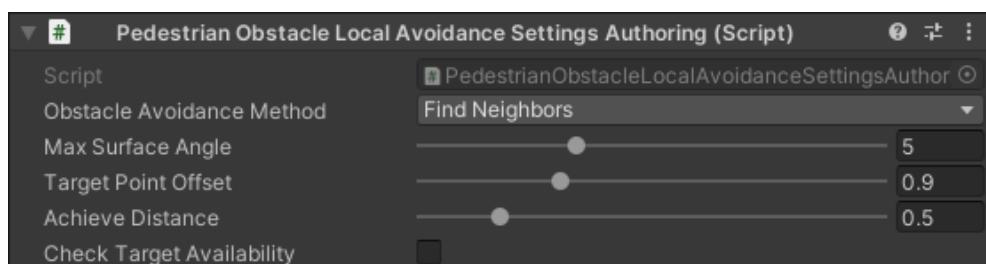
Revert target support

[if steering target is much further than final target with a given value the target will be reverted.]

- **Revert steering target distance** : distance to steering target logic for target return.
- **Revert end target remaining distance** : distance to final target logic for target return.

Obstacle Local Avoidance Config

Config for *Local Avoidance* navigating.



Obstacle avoidance method:

- **Simple** : is able to avoid only 1 object.
- **Find neighbors** : multiple objects close to each other are grouped as one (more costly in performance).

Max surface angle : maximum surface tilt angle at which the avoidance is calculated.

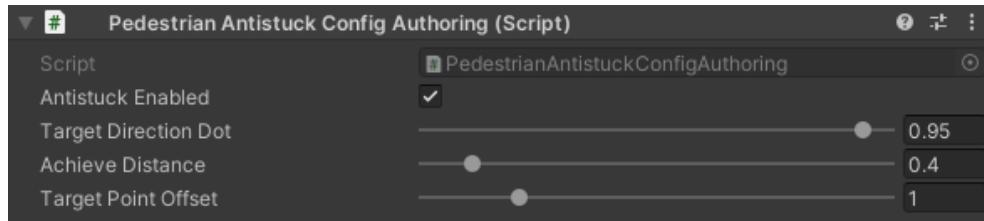
Target point offset : offset between an obstacle and avoidance waypoints.

Achieve distance : distance to achieve the avoidance waypoint.

Check target availability : check if destination can be reached, if not and can't be found new, destination returns.

Antistuck Config

Anti-stuck config for pedestrians stuck in a collision.



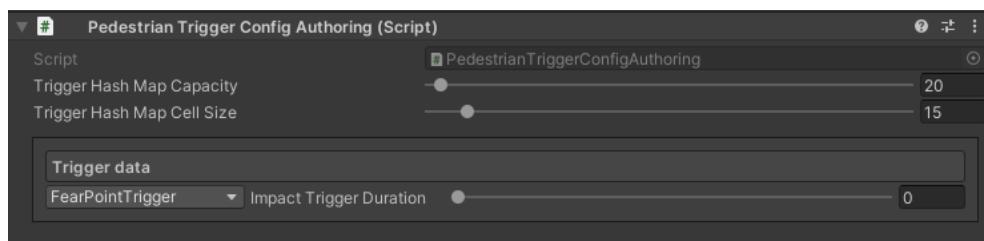
Antistuck enabled : on/off anti-stuck feature (if disabled previous target will be selected).

Target direction dot : direction between the pedestrian's forward and the anti-stuck point.

Achieve distance : achieve distance to the antistuck target point.

Target point offset : distance between collision and anti-stuck point.

Trigger Config



Trigger HashMap capacity : initial hashmap capacity that contains data of triggers.

Trigger HashMap cell size : hashmap cell size.

Trigger data:

- **Fear Point Trigger :**

– **Impact trigger duration** : duration of the *trigger* on the pedestrian.

Scary Trigger Config



Trigger settings

Death trigger squared distance : death trigger squared distance (squared distance == distance * distance).

Death trigger duration : death trigger duration.

Sound settings

Has scream sound : on/off scream sound.

Scream entity limit : maximum number of screaming pedestrians at the same time.

Chance to scream : chance of a pedestrian screaming.

Scream delay : delay between screams.

Scream sound data : scream *sound data* source.

Bench Config



Min/Max idle time : min/max idle duration on the bench.

Custom achieve enter point distance : distance to achieve the entry point on the bench.

Idle after achieved exit duration : idle after achieved exit point duration.

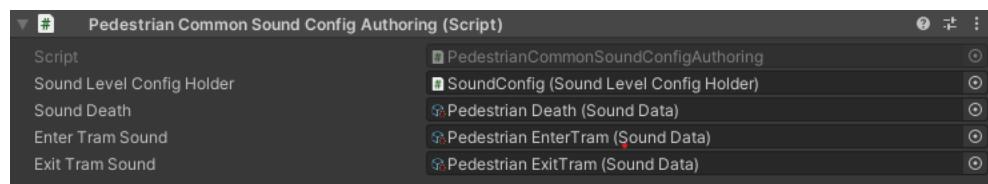
Sitting movement speed : pedestrian movement speed when sitting on the bench.

Sitting rotation speed : pedestrian turn speed when sitting on the bench.

Custom achieve sit point distance : distance to achieve the sit point on the bench.

Common Sound Config

Common pedestrian sound settings



Sound death : *sound* when a pedestrian died.

Enter tram sound : *sound* when entering a tram.

Exit tram sound : *sound* when exiting a tram.

State Authoring

State Dictionary

P	State Dictionary (3 items)		
=	Achieved Target	Next States	Mixed... ▾
		State Type	External System ▾
=	Scary Running	Next States	Mixed... ▾
		State Type	Additive ▾
=	Reset	Next States	Default ▾
		State Type	Default ▾
Default			▼ *

Next states : which *states* can override the current *state*.

State type:

- **Default** : the state processed by *PedestrianStateSystem* system (code processing for state should be there PedestrianStateSystem.cs:144).
- **External system** : the state processed by external system (code processing for state should be in the separate system).
- **Additive** : additive state flag adds to the current state and is processed by the *External system*.
- **Additive any** : additive state flag adds to the current state and is processed by the *External system* & ignores available next state flags.

Movement State Binding Dictionary

P	Movement State Binding Dictionary (7 items)		
=	Moving To Next Target P ▾	Walking	▼ ✖
=	Crossing The Road	Walking	▼ ✖
=	Scary Running	Running	▼ ✖
=	Wait For Green Light	Idle	▼ ✖
=	Talking	Idle	▼ ✖
=	Sitting	Idle	▼ ✖
=	Idle	Idle	▼ ✖
Default			▼ *

Contains data - which *Movement state* is assigned after the *Action state* is assigned.

Note:

- Read more the *state info* & *available states*.
-

4.2 Animation Baker

- *How To Bake*
- *How To Bake Multi-Mesh*
- *Baker Window*
 - *Settings*
 - *Source Data*
 - * *Toolbar*
 - * *Clip Data*
 - *How To Preview*
 - *How To Bind*
 - * *Texture Data*
 - * *Transition Data*
 - *How To Use*
 - * *Buttons*
 - *Animation Texture Data*
 - *Crowd GPU Animator*
 - *How To*
 - * *Open*
 - * *Initial Set Up*
 - * *Create Node*
 - * *Create Transition*
 - * *Create Transition Layer*
 - * *Test*
 - *Graph Nodes*
 - * *Start Node*
 - * *Animation Node*
 - * *Transition Node*
 - * *Transition example*
 - *Animation Collection*
 - *How To Create*

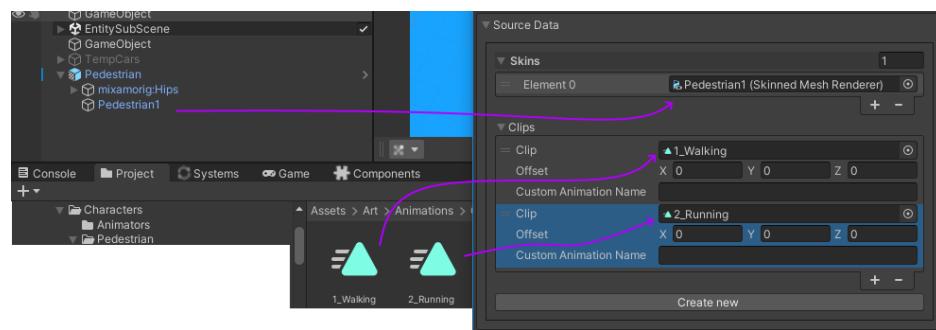
- *Settings*
- *Animator Data Container*

4.2.1 How To Bake

1. Open *Baker* from the *Unity* toolbar.

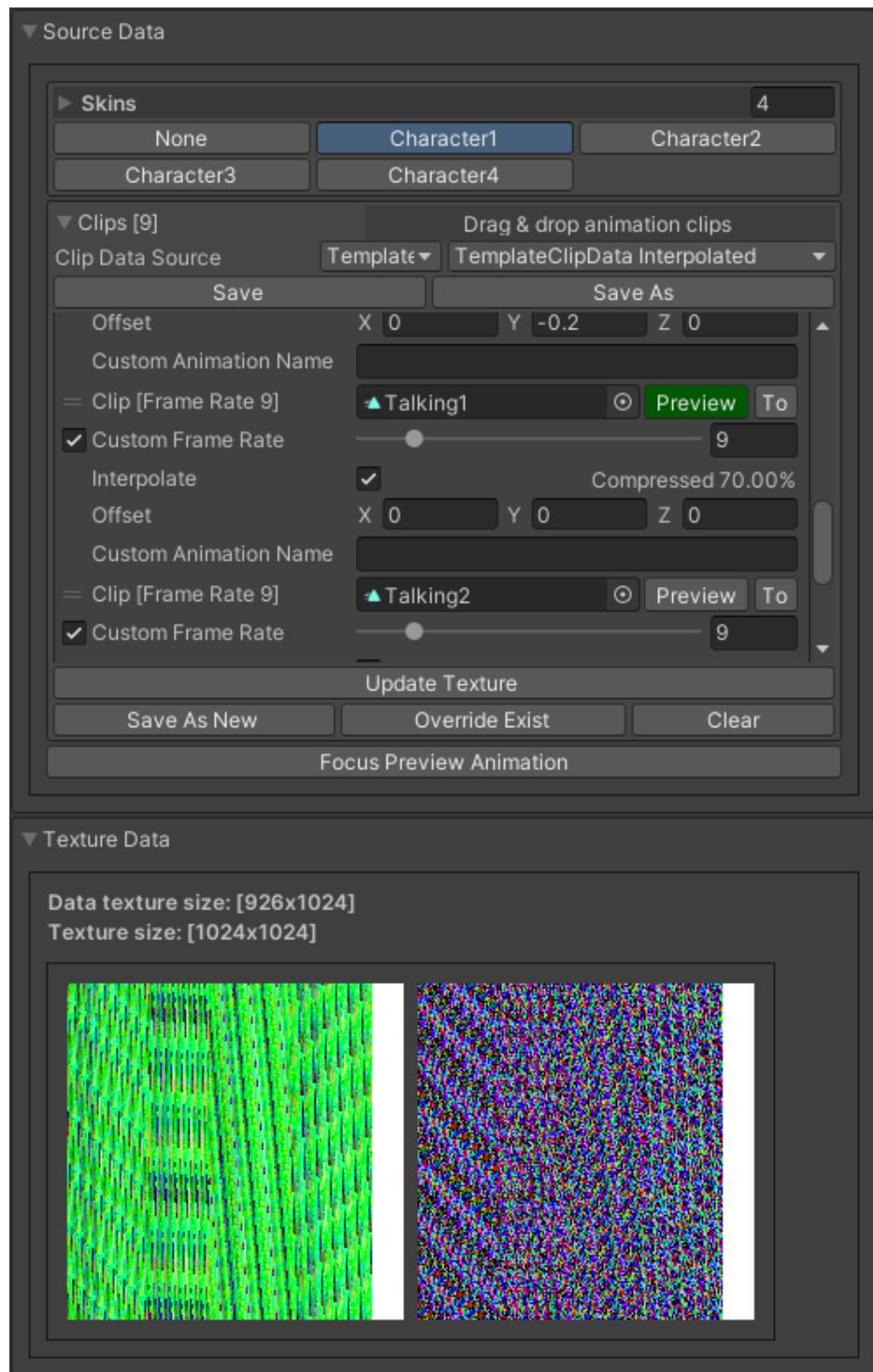
Spirit604/Animation Baker

2. Customize texture *settings*.
3. Drag & drop selected *SkinnedMeshRenderer* of pedestrians into *Skins field*.
4. Drag & drop selected animations of pedestrians into the *Clips field*.



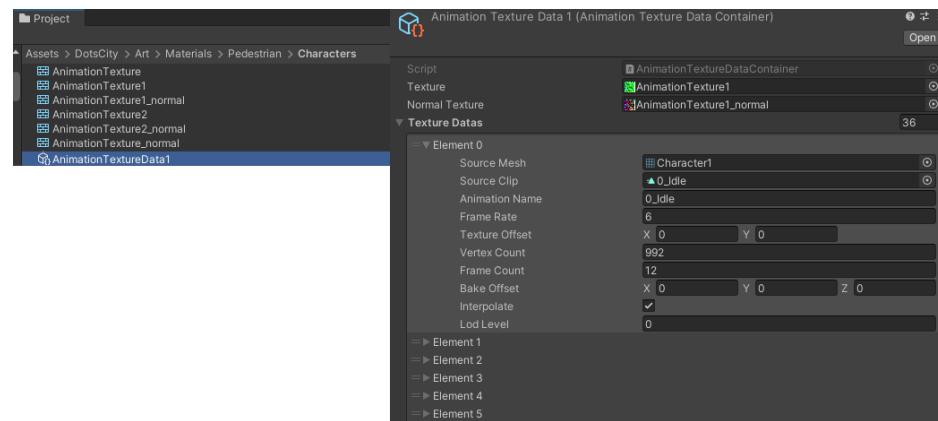
Drag & drop example.

5. Press the *Create new* button.



Multiple animation result example.

6. In the project view, check the *Animation Texture Data* result.



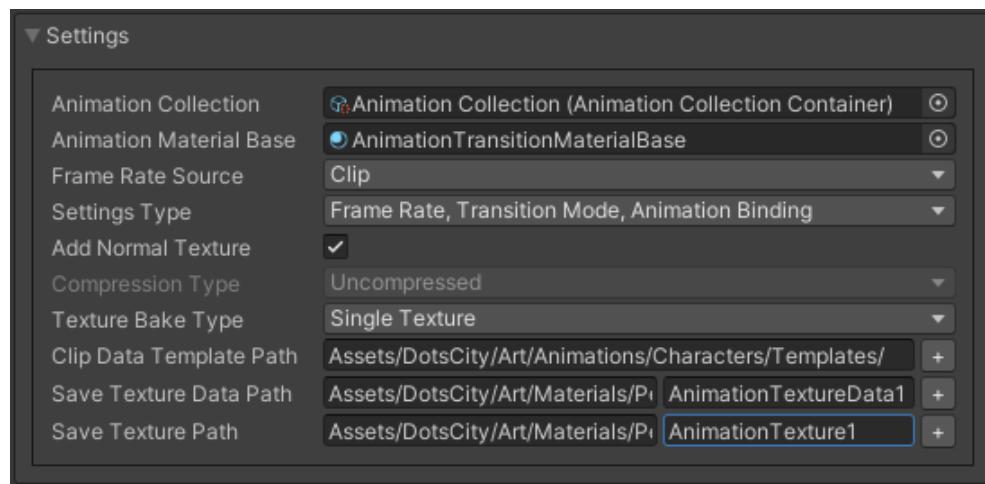
AnimationData result example.

4.2.2 How To Bake Multi-Mesh

1. Enable *Multimesh* type in the *Settings type*.
2. Customize modular character with desired skinned mesh renderers.
3. Add gameobjects with the desired attachments to the child of the desired bone [**optional**].
4. Drag & drop character parent to source skin list.
5. Make the same *steps* for baking.

4.2.3 Baker Window

Settings



Animation material base : base animation material.

Frame rate : frame rate of baked animation.

Settings type:

- **Frame rate** : shows frame rate settings.
- **Common** : shows common settings (custom offset & custom animation name).
- **Transition mode** : allows you to preview the transition from one animation to another in the editor.
- **Animation binding** : shows binding settings of the *Animation collection*.
- **Multimesh** : allows you to bake animations with multi-skin renderers or with custom attachments [v1.0.5+].

Add normal texture : add normal texture.

Compression type:

- **Uncompressed** : uncompressed format of baked texture.
- **Compressed** : compressed format of baked texture. [currently not available]

Texture bake type:

- **Single texture** : bake all characters to single texture.
- **Multiple textures** : bake each character to unique texture.

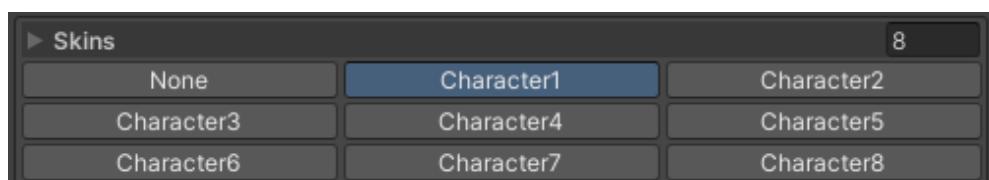
Clip data template path : path to clip templates.

Save texture data path : creating path of *texture animation data*.

Save texture path : creating path of baked textures.

Source Data

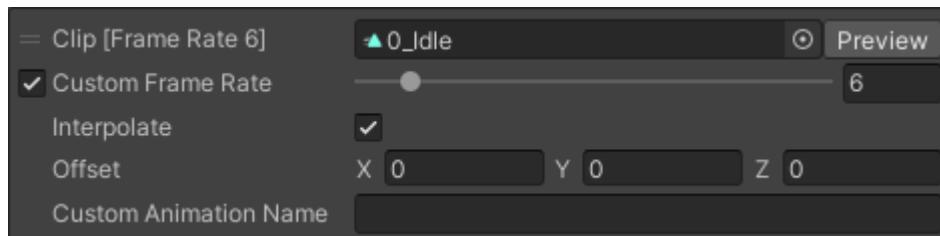
Toolbar



Skins : source *SkinnedMeshRenderer* of characters.

Skin toolbar : character selection toolbar for selecting preview animation.

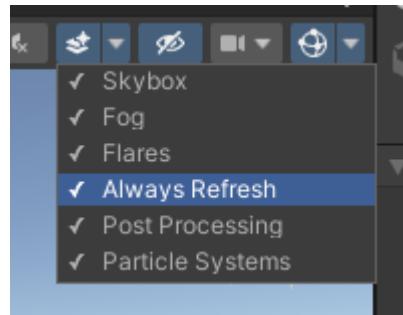
Clip Data



- **Clip** : reference to the clip.
- **Custom frame rate** : custom frame rate of the clip.
- **Interpolate** : on/off interpolation feature for the clip.
- **Offset** : local offset of vertices in baked animation.
- **Custom animation name** : custom animation name (if the field is empty the name from the clip will be taken).
- **Preview** : on/off preview playback of baked animation (make sure the texture is created and any character is selected in the *toolbar*).

How To Preview

1. *Bake* the texture.
2. Select any character (for example *Character1*).
3. Press the *Preview* button.
4. Make sure that *Refresh Always* option is enabled in the editor.

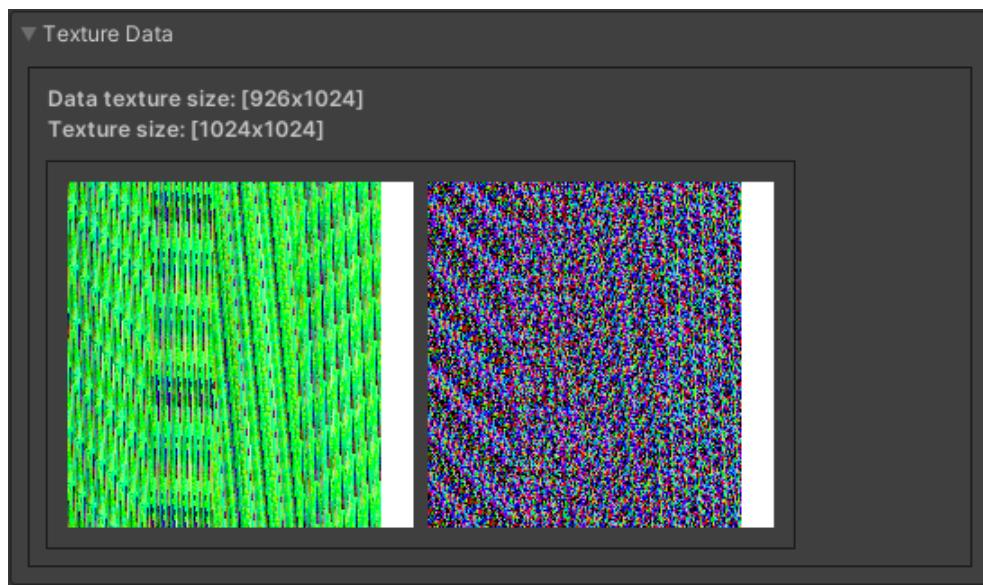


How To Bind

1. Select *Animation collection* in the *settings*.
2. Enable *Animation binding* type in the *Settings type*.
3. In the *Clip data* select the animation from the *Animation collection* according to required animation.

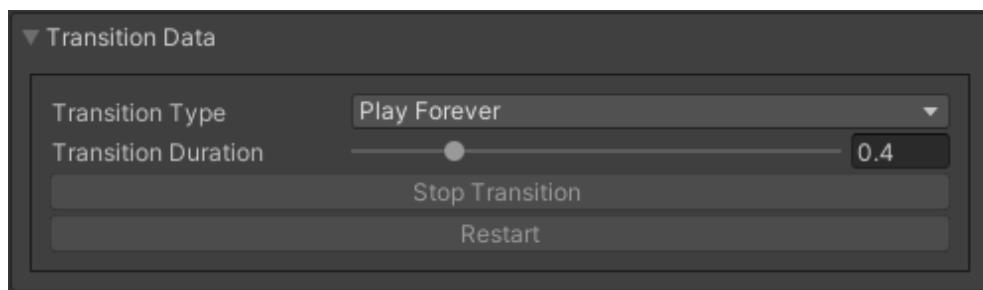
Texture Data

Shows a preview of the created texture.



Transition Data

Shows a preview of the transition animation between two selected animations.



- **Transition type:**
 - **Play once** : animation plays only once.
 - **Play forever** : animation loops forever.
- **Transition duration** : time to end of current animation when interpolation transition between animations is enabled.

How To Use

1. Enable *Transition mode* in the settings.
2. *Bake* the texture.
3. Select any character (for example *Character1*).
4. Press the *Preview* button for the source transition animation.
5. Then, press the *To* button next to the animation you want to target.
6. Adjust the *Transition duration* if required.

Buttons

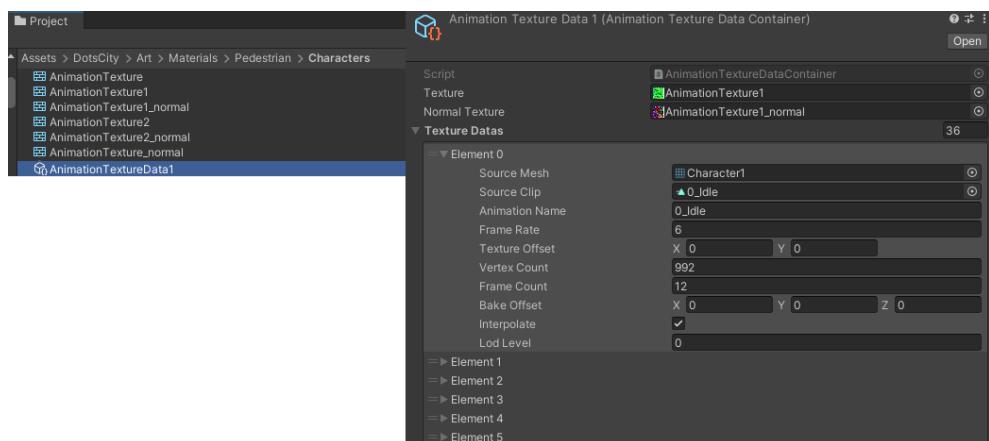
Create new : create a new texture.

Save as new : save the texture as a new asset.

Override exist : overrides existing texture & data.

Clear : clean up the texture.

Animation Texture Data



Example.

4.2.4 Crowd GPU Animator

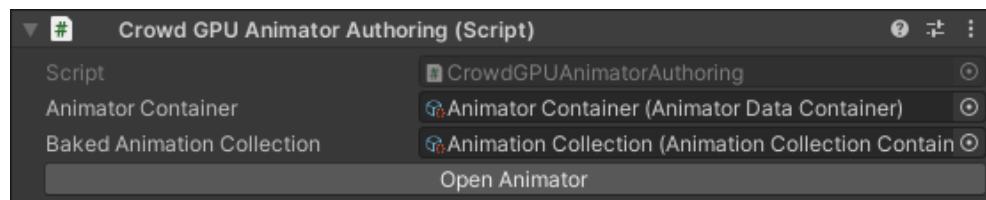
The *Crowd GPU Animator* is used for transitions between GPU animations.

How To

Open

Open in the scene *CrowdGPUAnimatorAuthoring*.

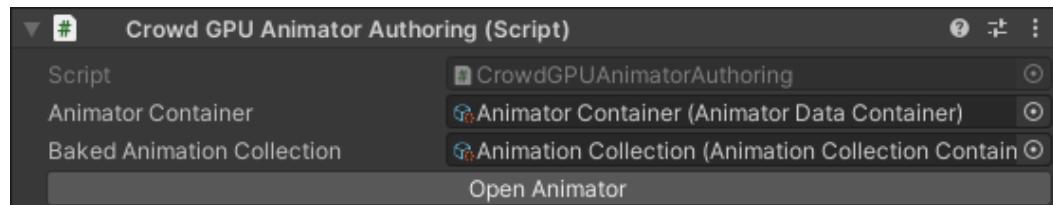
`Hub/Configs/BakerRefs/Settings/CrowdGPUAnimatorAuthoring`



Initial Set Up

Steps:

1. Create an *Animator Data Container* from the project context and assign it to the custom animator (if necessary).
2. Create (if necessary) and assign *Animation Collection* the same as in the *PedestrianCrowdSkinFactory*.



Create Node

Right-click in the window and select the *desired node* from the context menu.

Create Transition

Transition is a sequential set of nodes StartNode→AnimNode→TransitionNode→AnimNode→TransitionNode→AnimNode→... (*example*).

Steps:

1. Create a *new transition layer* (if required).
2. Enter the name of the trigger in the *StartNode*.
3. Create and connect *AnimationNodes* and *TransitionNodes*.

Create Transition Layer

Press the + button on the main toolbar at custom animator to create a new layer, or press - to delete the currently selected layer.

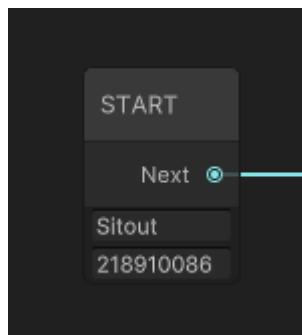
Test

You can test the transition between two animations & adjust the transition duration in the editor in *AnimationBakerWindow*.

Graph Nodes

Start Node

Node where the transition begins by trigger.



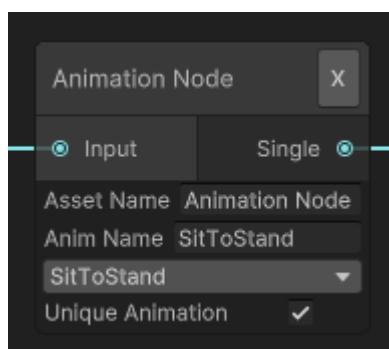
Trigger name : name of the trigger on which the transition starts.

Hash : hash of the trigger on which the transition starts.

Note: Hash from trigger name generated by Unity method `Animator.StringToHash`

Animation Node

Animation playback node.



Asset name : asset data name.

Anim name : animation name (by default is taken from *Anim enum*).

Anim enum : list of available animations in the *Animation Collection*

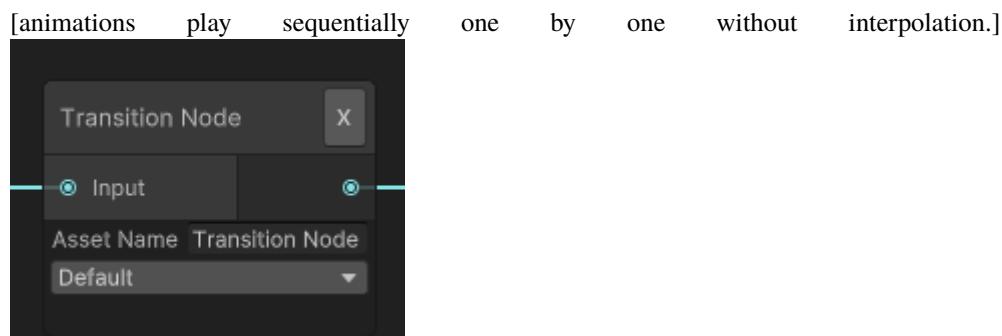
Unique animation : unique animation mesh instance will be created for this animation.

Transition Node

Node with settings for switching between animations.

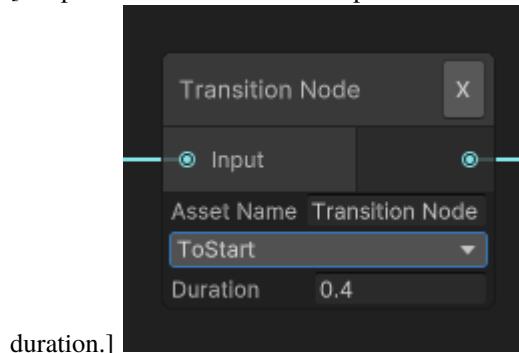
Node Type:

- **Default**



- **To Start**

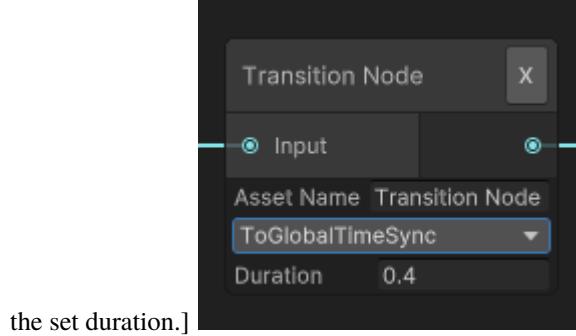
[the previous animation is interpolated to the beginning of the next animation with the set



duration.]

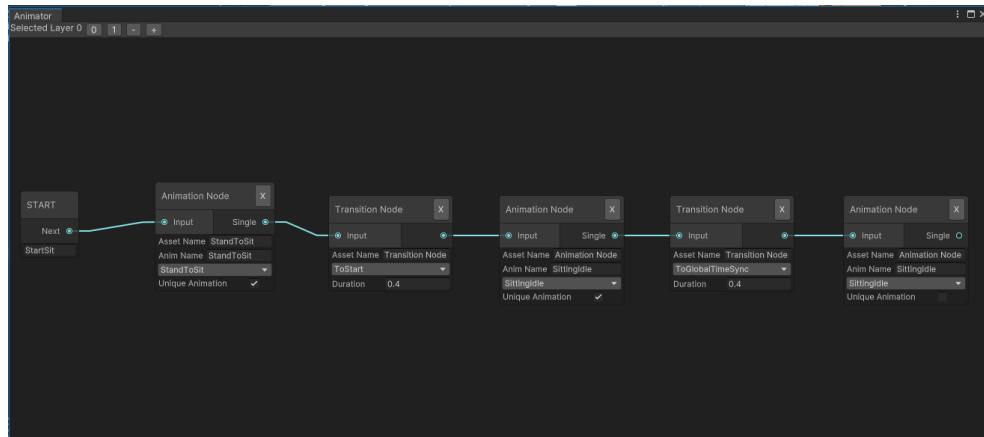
- **To Global Sync**

[the previous animation is interpolated to the global playback time of the next animation with

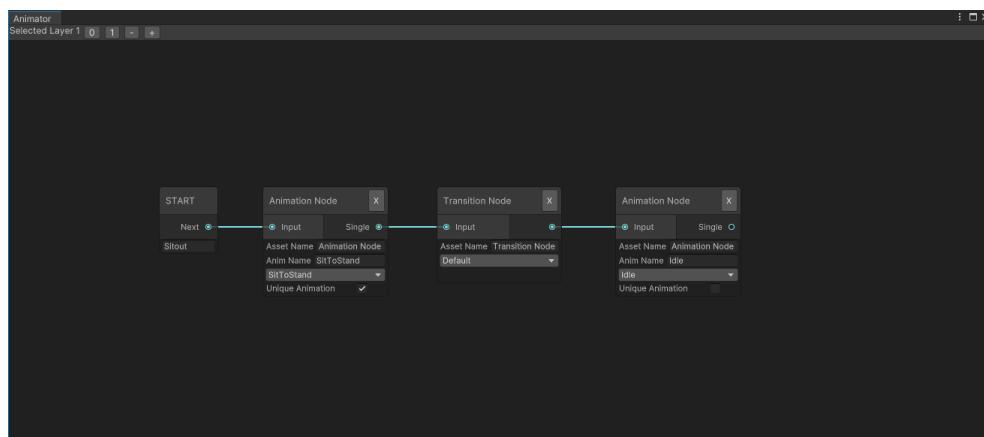


the set duration.]

Transition example



Start sit transition example.



Sitout transition example.

4.2.5 Animation Collection

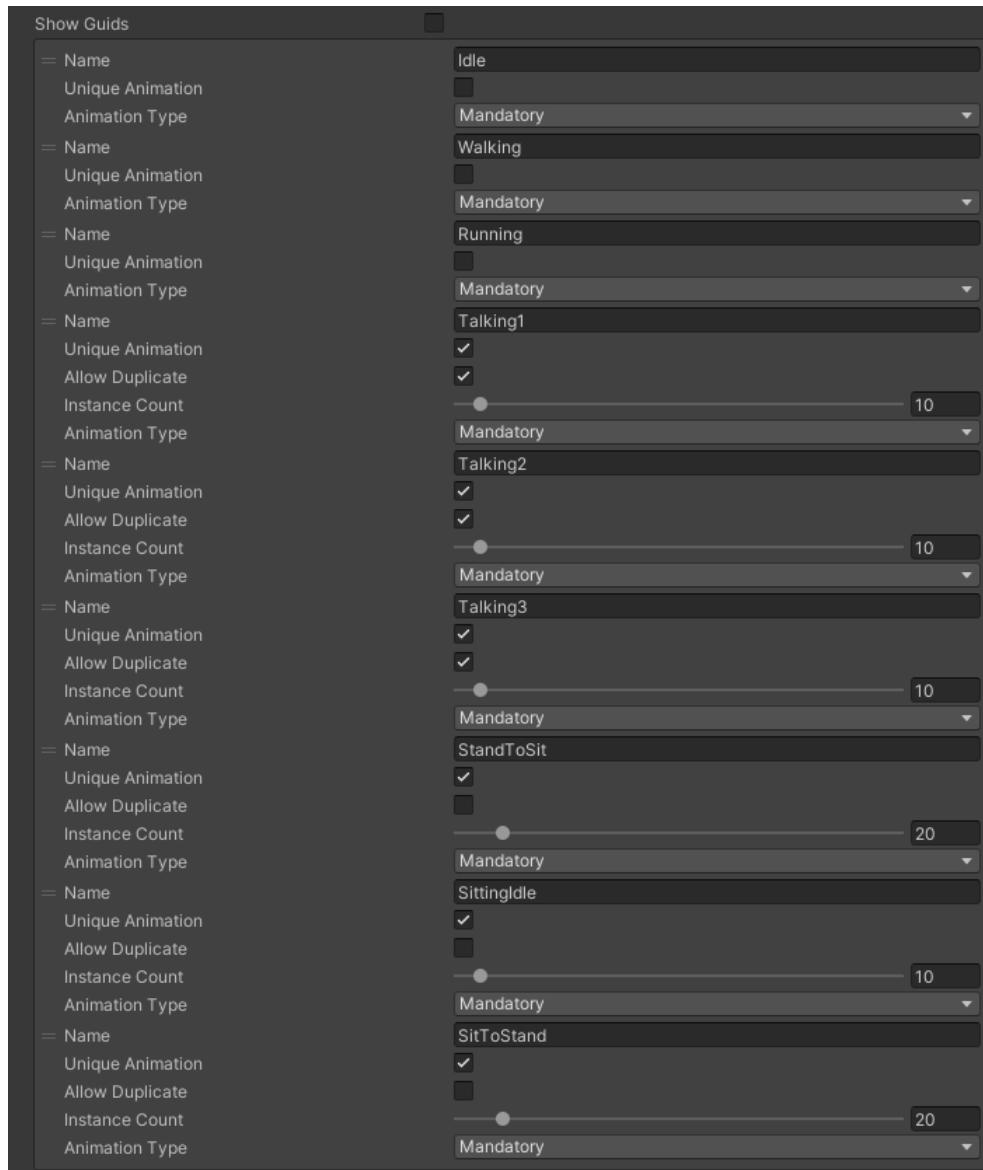
Contains meta-data of existing animations for the pedestrians.

How To Create

From the project context :

`Spirit604/Animation Baker/Animation Collection`

Settings



Name : animation name.

Unique animation

[a unique animation mesh instance pool will be created for this animation.]

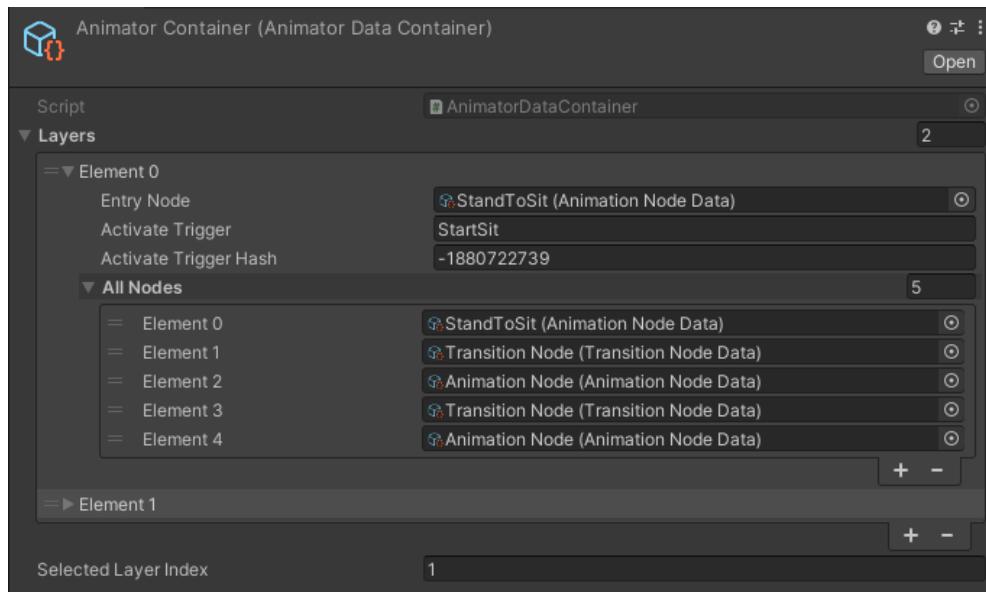
- **Allow duplicate** : is it allowed to take an animation from the pool if it is already being used by another character.
- **Instance count** : animation pool size.

Animation type:

- **Mandatory** : each entity of the crowd must have this animation.
- **Optional** : each entity of the crowd doesn't necessarily have to have this animation [*step 16*].

4.2.6 Animator Data Container

Contains data about animation transitions.



Layer Data

[contains data about transition.]

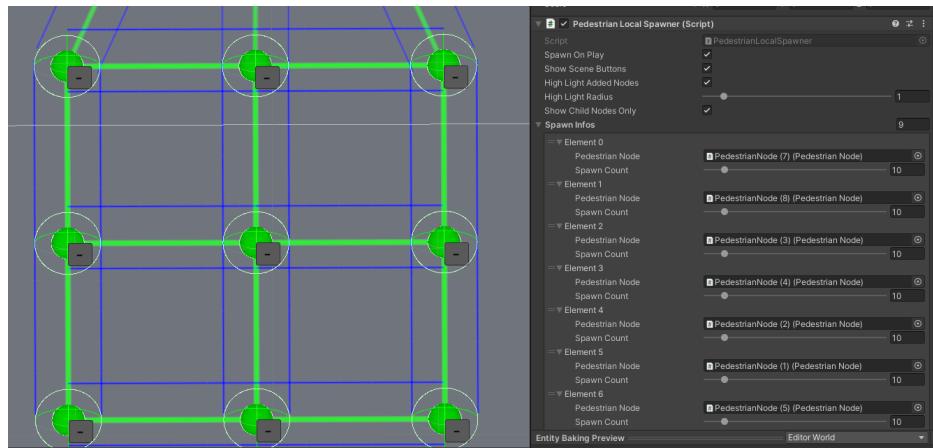
- **Entry node** : asset node where the transition begins.
- **Activate trigger** : name of the transition activation trigger.
- **Activate trigger hash** : hash of the transition activation trigger.
- **All nodes** : all transition nodes.

4.3 Pedestrian Test Scene

4.3.1 How To Use

Youtube tutorial.

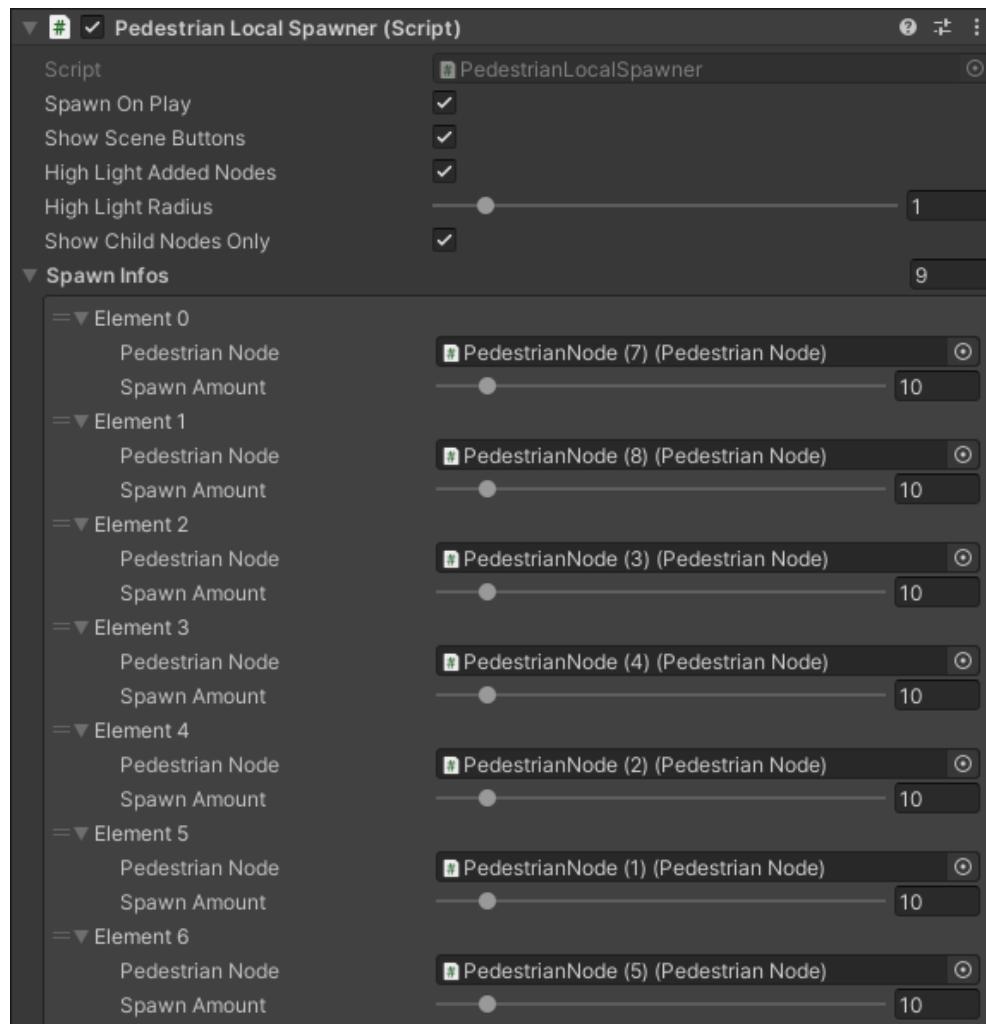
1. Create *pedestrian nodes* & connect them.
2. Create a parent *GameObject* and add the *Pedestrian Local Spawner* component.
3. In the created component, press the *Show scene buttons*.
4. Select the *pedestrian nodes* in the scene where you want the pedestrians to spawn.



Selection example.

5. Adjust the *Spawn count*.
6. Start the scene.
7. Click on the *Spawn* button in the component.
8. Learn more about the *Pedestrian Local Spawner* settings.

Pedestrian Local Spawner



Common settings

Spawn on play : spawn the pedestrian after the start of the scene.

Show scene buttons : show add *pedestrian node* button to the component in the scene

Highlight added nodes : on/off highlight added *pedestrian nodes*.

Highlight radius : highlight radius.

Show child nodes only : only the child *pedestrian nodes* will be shown.

Spawn info

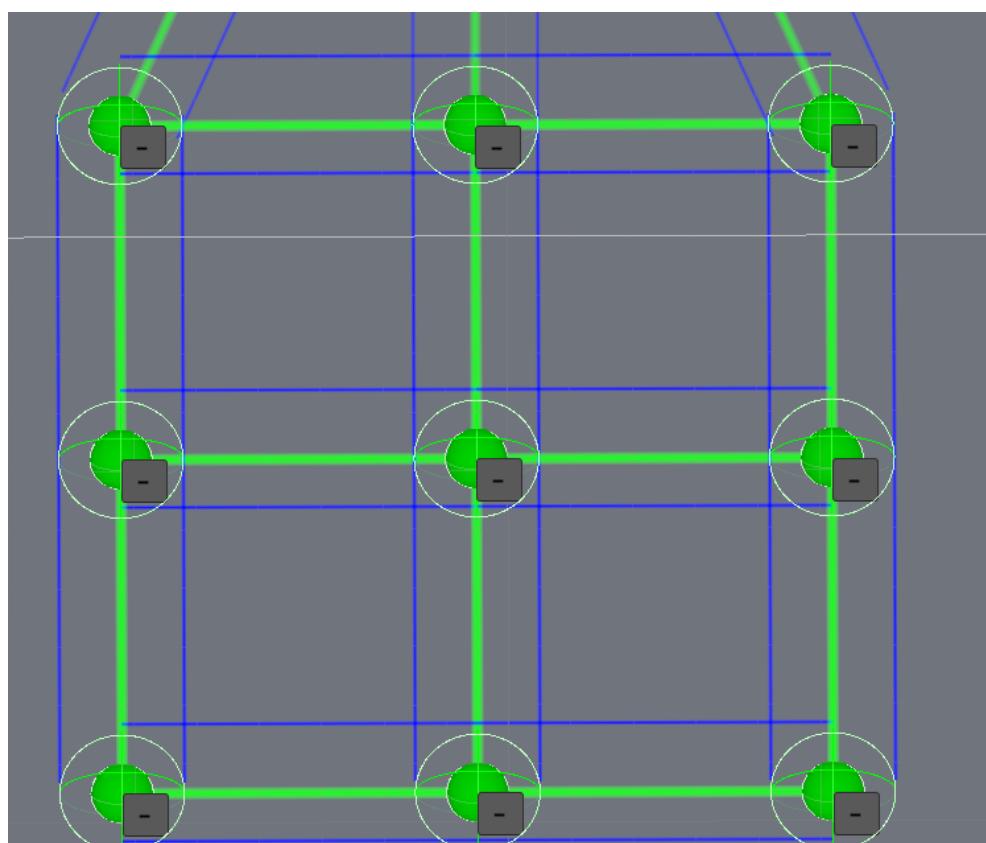
Pedestrian node : linked *pedestrian node*.

Spawn amount : number of pedestrians that will be spawned in the *pedestrian node*.

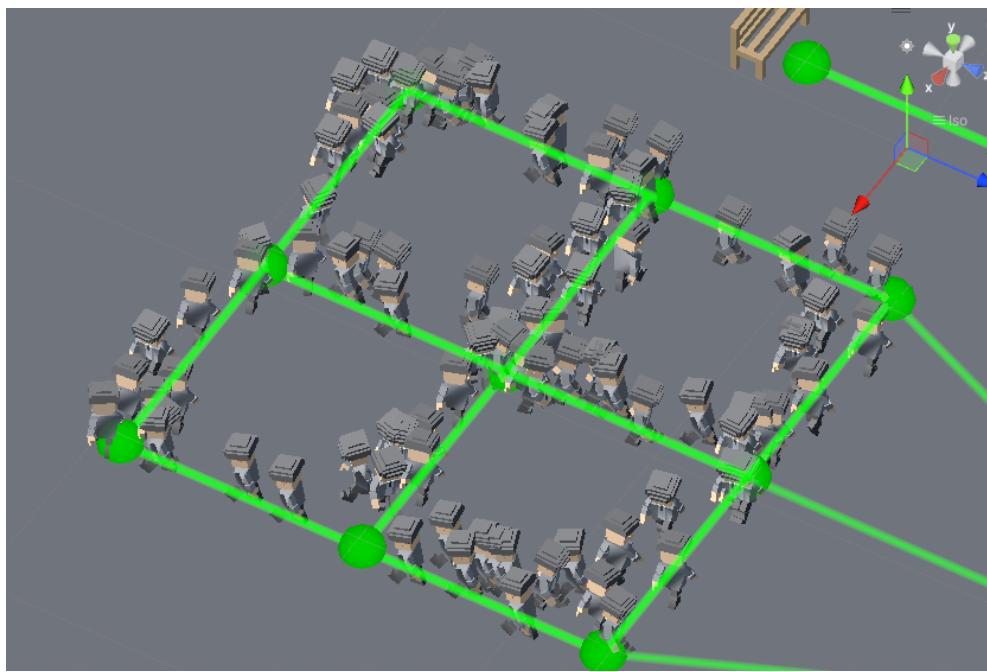
4.3.2 Test Cases

Walking Test

Test case to test the *walking parameters*.

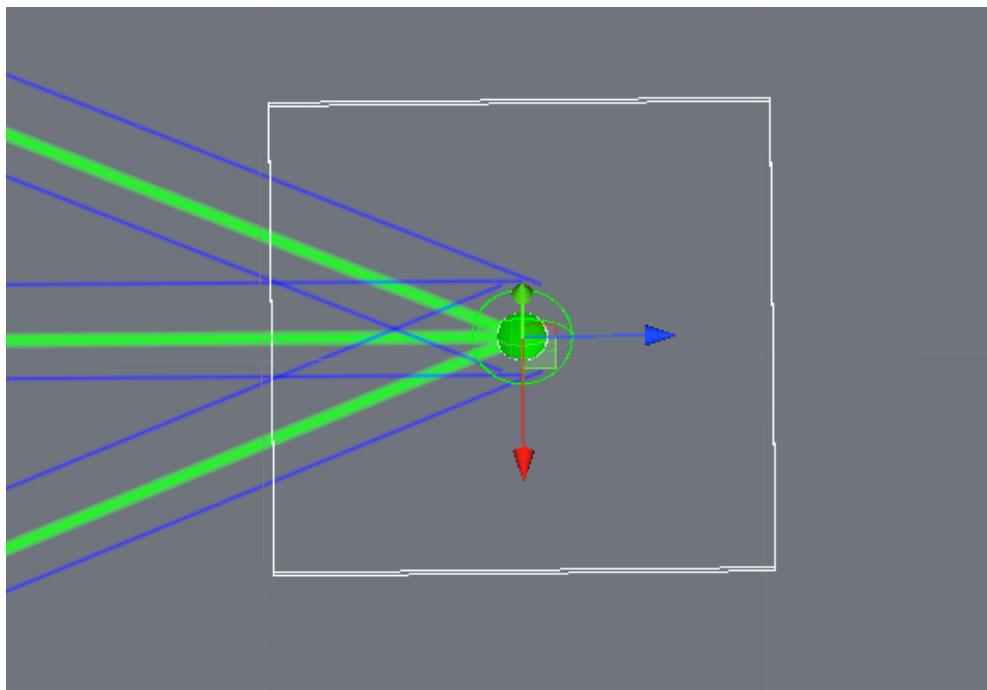


Source nodes.

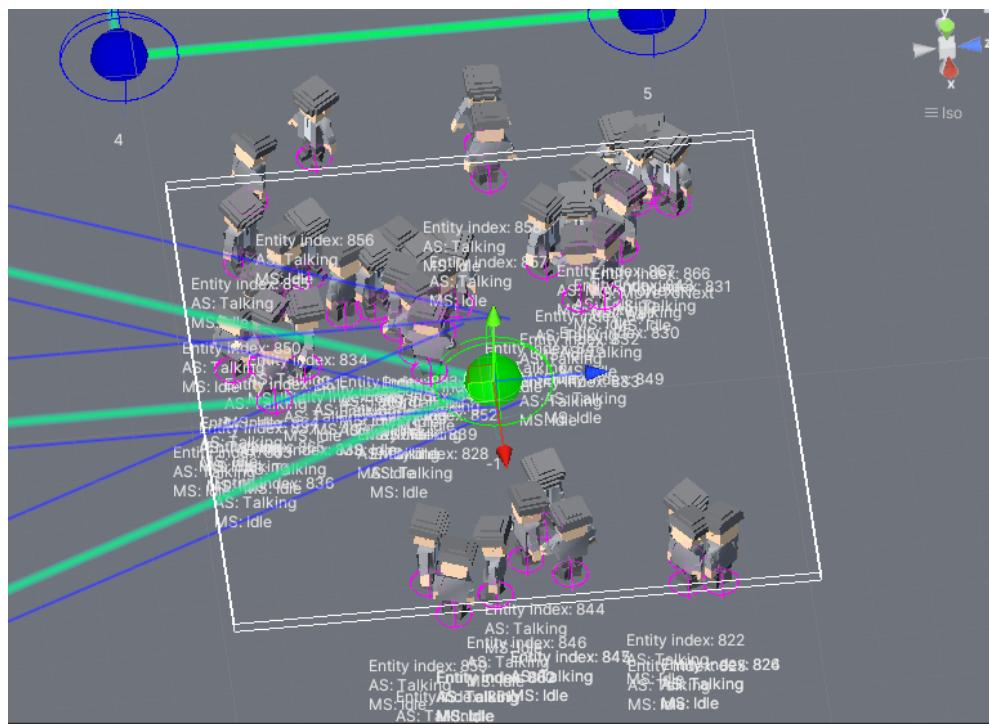


Result.

TalkArea Test



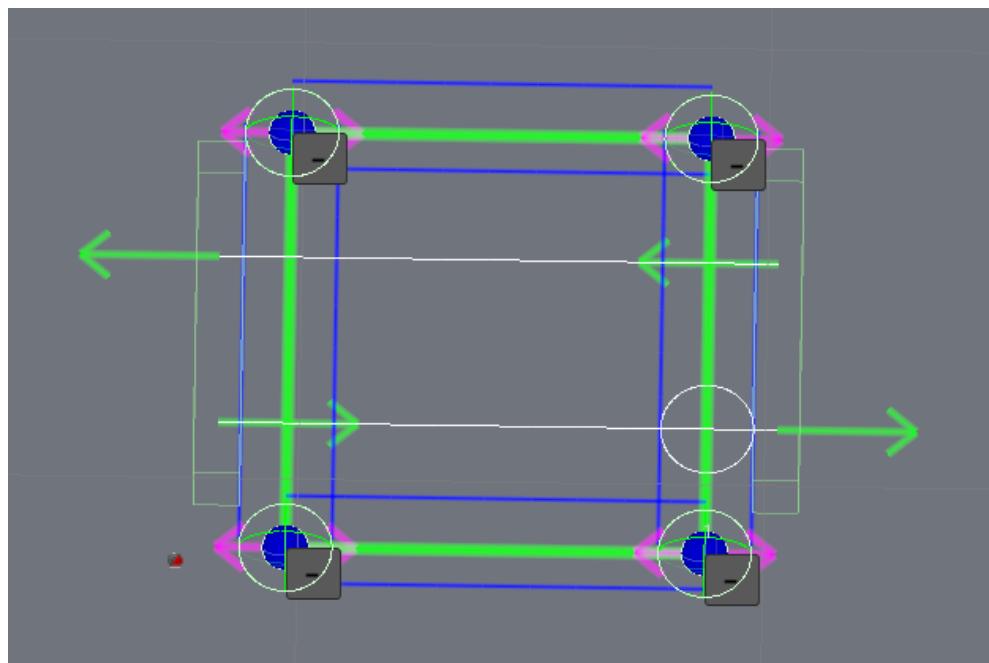
Source node.



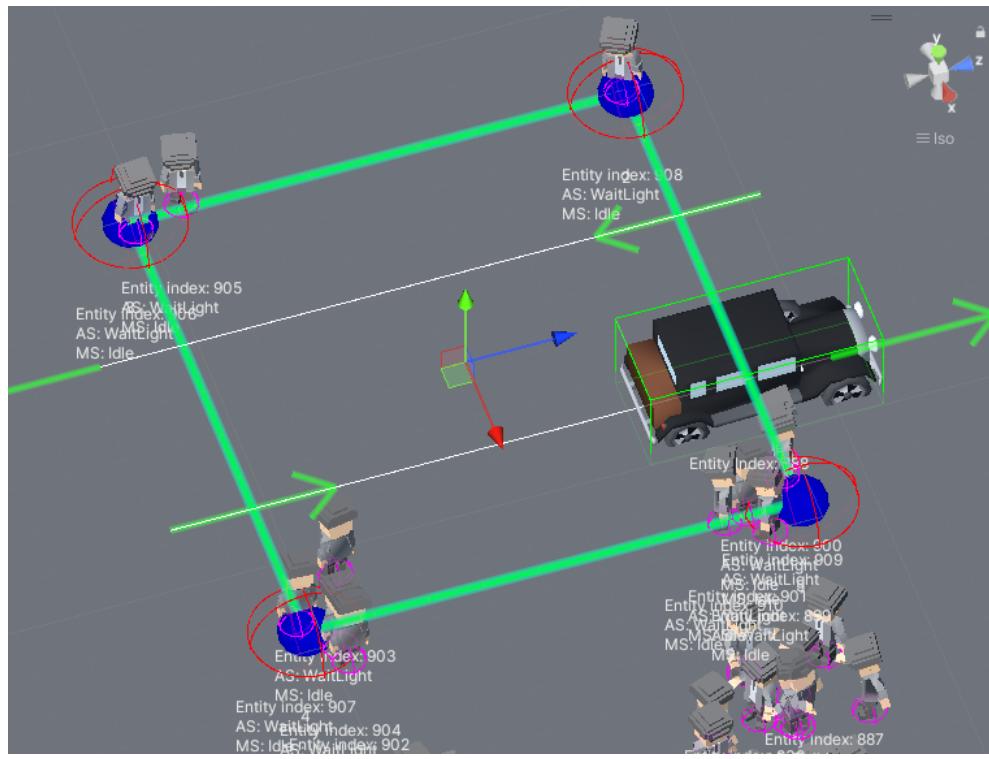
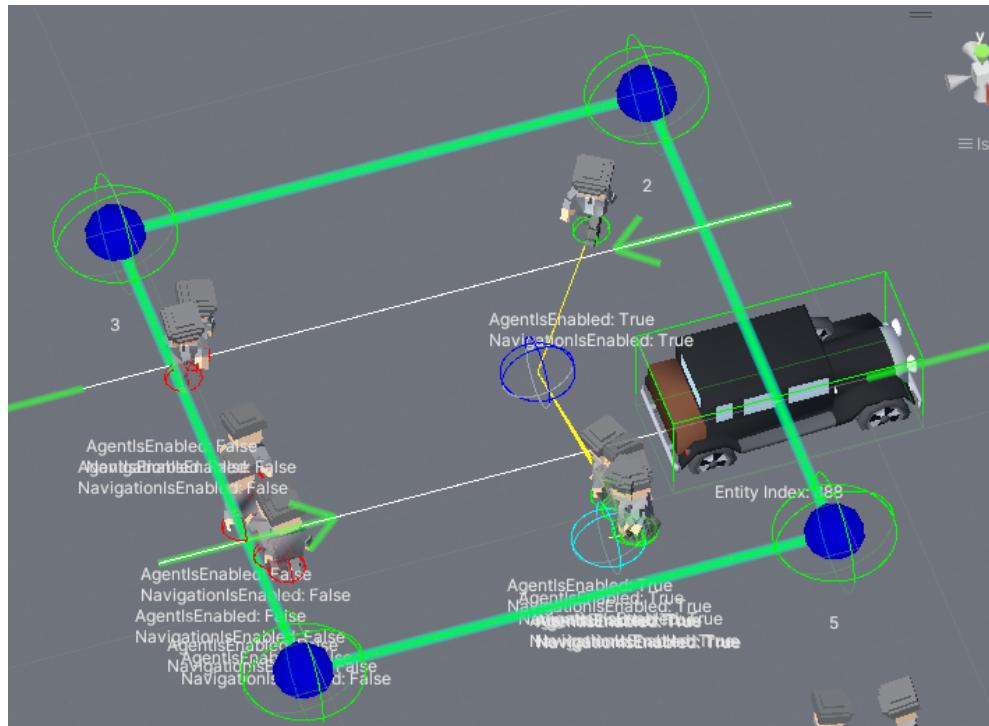
Result.

Crossroad Test

Test case of pedestrians waiting at traffic lights and crossing the crossroad.

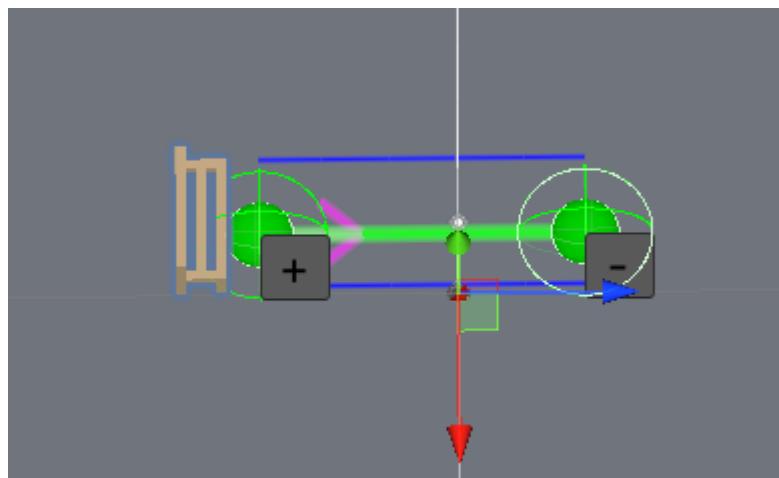


Source nodes.

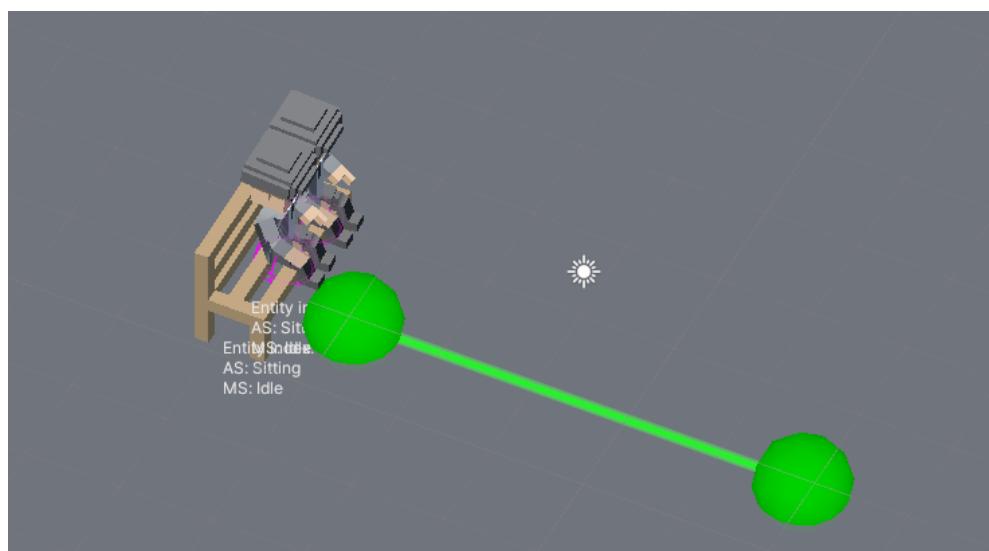
*Traffic waiting.**Crossing the road.*

Bench Test

Test case to test bench *seating*.



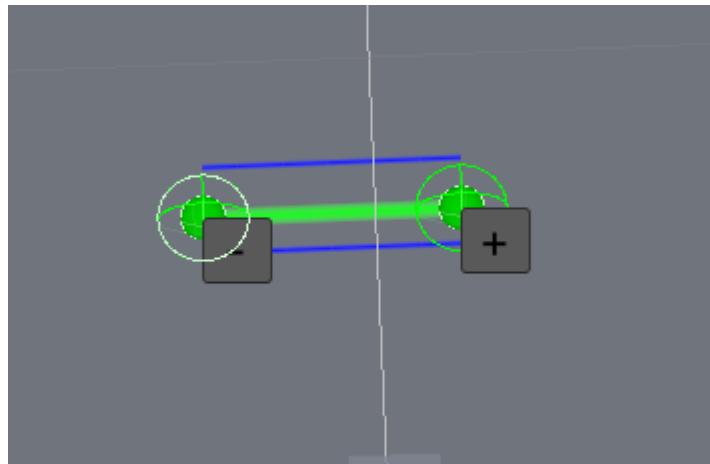
Source nodes.



Result.

House & Idle Test

Test case for *idling* and entering the *house*.



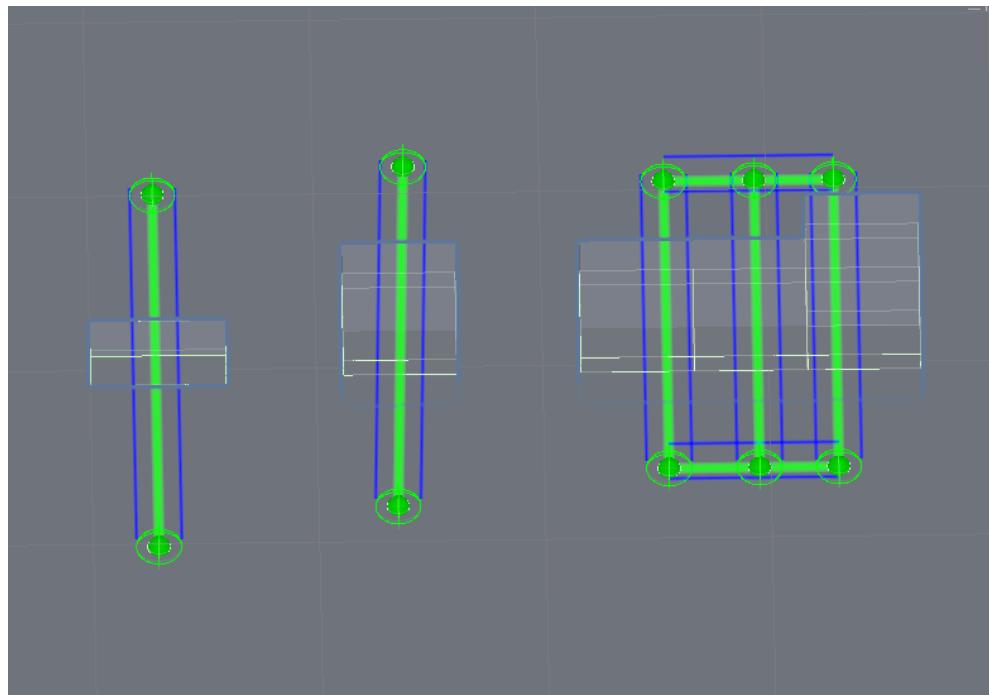
Source nodes.



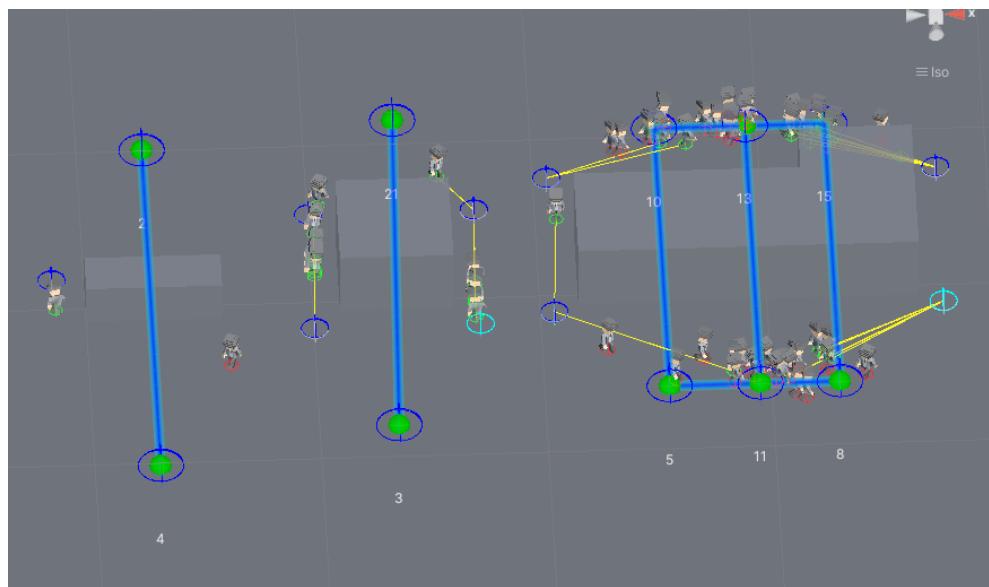
Result.

Navigation Test

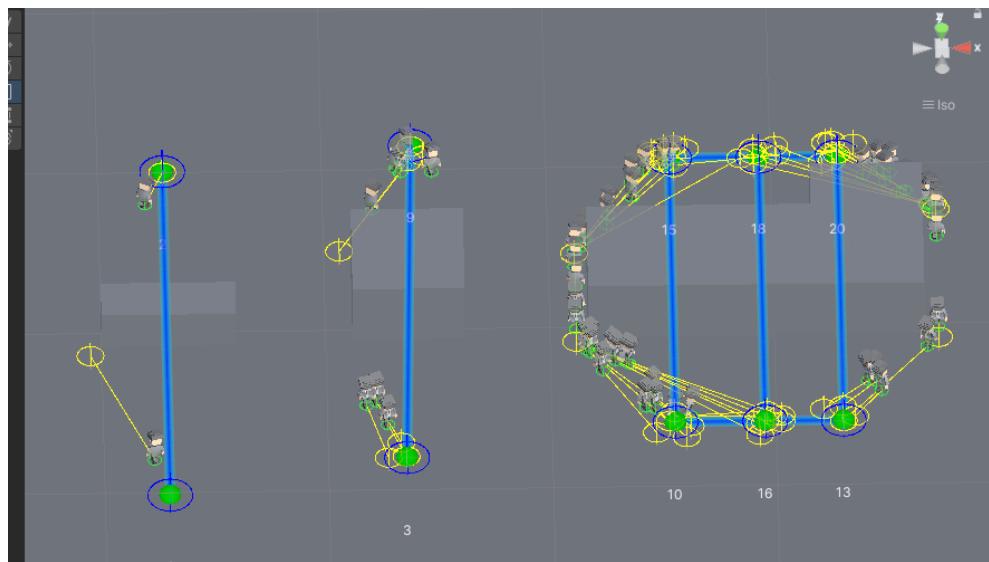
Test case for *navigation*. Red circle navigation is disabled. Green circle navigation is enabled.



Source nodes.



Local avoidance example.



NavMesh navigating example.

PLAYER

5.1 Built-in Solution

5.1.1 Player Spawner

- *Info*
- *Where To Find*
- *How To Use*
- *How To Replace*
 - *Steps*

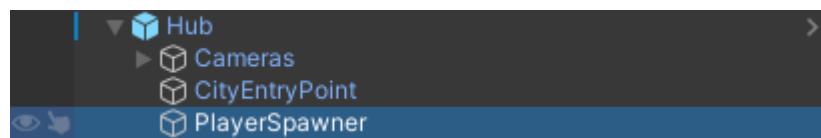
Info

Player Spawner Service is used to spawn player in the city.

Available spawn types:

- NPC: spawns the *Player NPC*.
- Car: spawns the *Player Car*.
- Free fly camera : spawns free fly camera (should be set in the *General Settings*)

Where To Find



How To Use

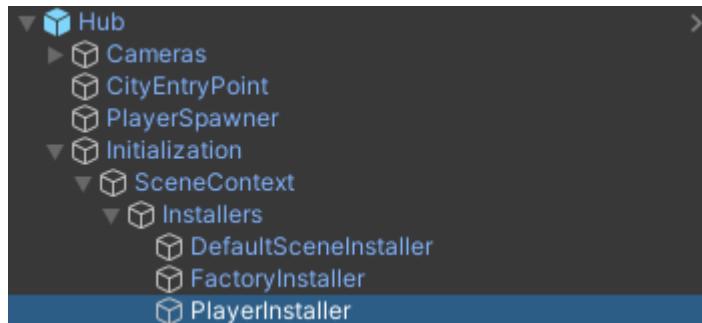
- Create *Player NPC & Player Car*.
- Select spawn type.

How To Replace

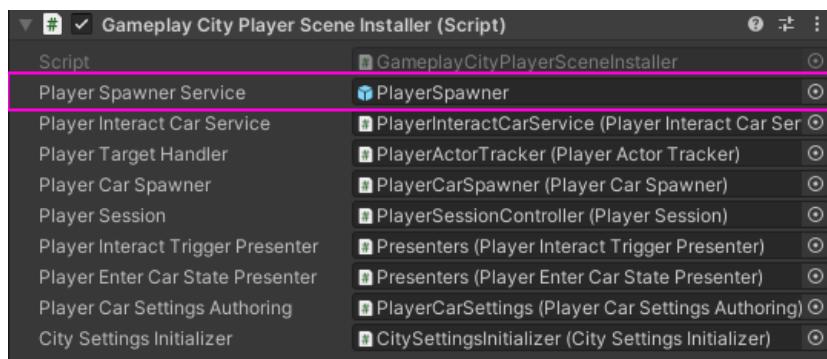
You can replace spawner by your own solution. By default, *Player Spawner Service* implements *IPlayerSpawnerService* interface & called by *PlayerSpawnCommand.cs* script, create your own *Monobehaviour* class that implements *IPlayerSpawnerService* & assign here:

Steps

- Find on the scene *PlayerInstaller*:



- Assign *Player Spawner Service*:



5.1.2 Player Npc

- *Hybrid DOTS*
 - *How To Create*
 - *Components*
 - *Factory*
- *Hybrid Mono*

- *How To Create*
- *Components*
- *Factory*

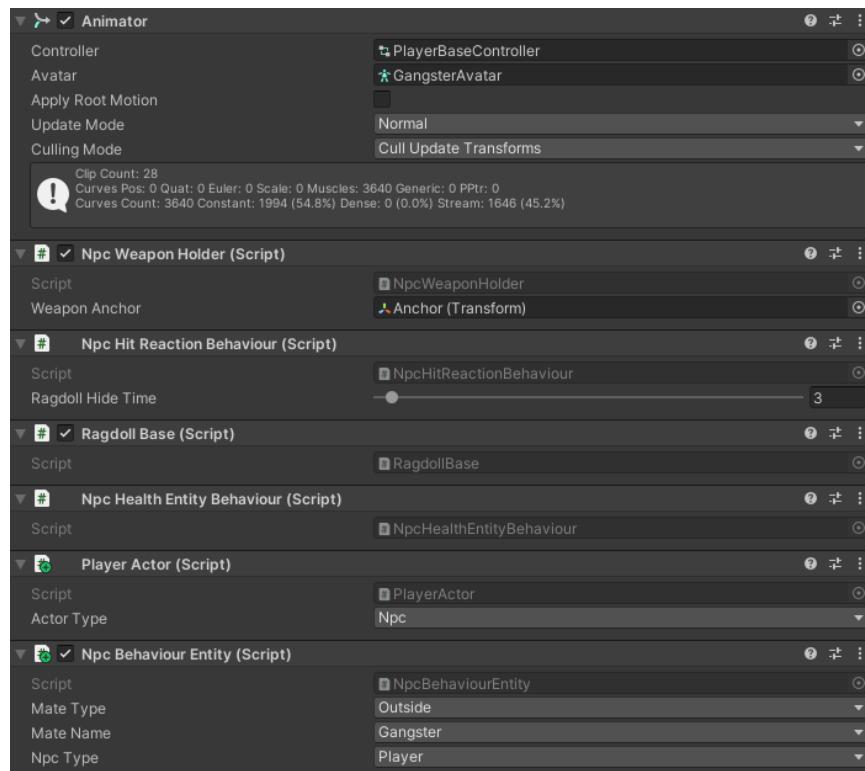
Hybrid DOTS

The player NPC behaviour is handled by the *DOTS* systems.

How To Create

1. Add an *Animator* component to your character if it's missing & assign a *PlayerBaseController* if you want to use project animations.
2. Add all required components to your character from the list below.
3. Create a new *ID* in the enum *NpcId.cs* file.
4. Add the result to the *PlayerNpcFactory*.
5. Make sure that *NPC* & *Hybrid DOTS* are selected in the *PlayerSpawner*.
6. Select your character from the list in the *PlayerSpawner*.

Components



- **Player actor** : camera tracking component [required].

- **Npc behaviour entity** : holds reference to bind entity & read input from *DOTS entity world*, also controls animator & npc weapon behaviour [required].
- **Npc weapon holder** : reference to the anchor bone that holds the weapon & reference to the weapon.
- **Npc hit reaction behaviour** : behaviour that handles the reaction to a bullet hit.
- **Ragdoll base** : activates *Ragdoll* when the NPC dies.
- **Npc health entity behaviour** : wrapper for entity health in the *Monobehaviour* world.

Note:

- Spawns by *PlayerSpawner*.



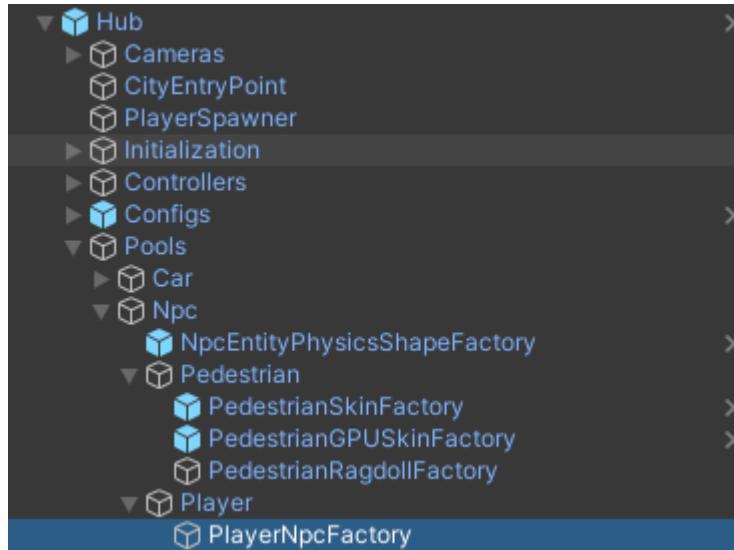
- Entity movement controlled by systems:

- * *NpcControllerSystem*.
- * *NpcGroundStateSystem*.
- * *NpcRaycastGroundSystem*.
- * *NpcFreezeVerticalRotationSystem*.

- *Gangster outside Player* is example prefab.
-

Factory

Factory that contains player *Hybrid DOTS* NPCs.



Hybrid Mono

The player NPC behaviour is handled by the mono controller.

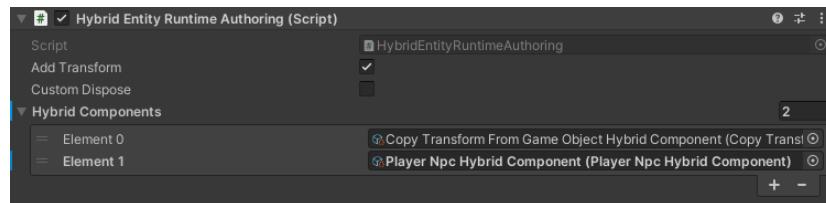
How To Create

1. Set the *World simulation type* to *Hybrid mono* in the *General settings* config.
2. Add animator to your model & assign *PlayerBaseController* into the controller field.
3. Add all required components to your character from the list below.
4. Add the result to the *PlayerHybridMonoFactory*.
5. Make sure that *NPC & Hybrid Mono* are selected in the *PlayerSpawner*.
6. Select your character from the list in the *PlayerSpawner*.

Note: *Demo Mono* scene & *Gangster Mono outside Player* prefab are examples.

Components

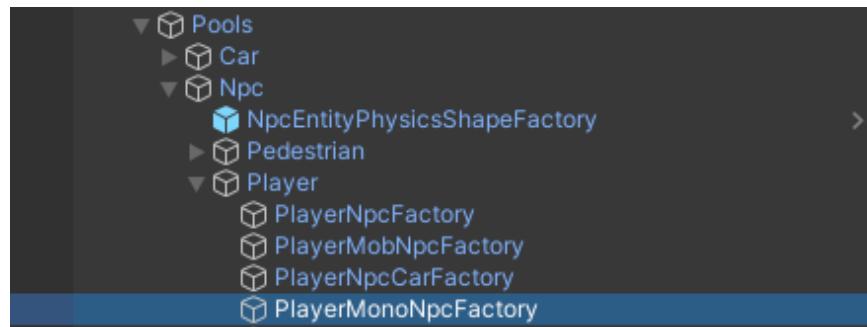
- **Player actor** : camera tracking component **[required]**.
- **Character controller** : default unity component **[required]**.
- **Npc motion behaviour** : component that handles NPC behaviour **[required]**.
- **Player npc input behaviour** : contains input from the player **[required]**.
- **Npc weapon holder** : reference to the anchor bone that holds the weapon & reference to the weapon.
- **Npc hit reaction behaviour** : behaviour that handles the reaction to a bullet hit.
- **Ragdoll base** : activates *Ragdoll* when the NPC dies.
- **Hybrid entity runtime authoring** : automatically load the entity on enable at runtime for this gameobject. **[required]**.



- **Npc health behaviour** : health component.

Factory

Factory that contains player *Hybrid Mono* NPCs.



5.1.3 Player Car

- *Common Info*
- *Hybrid DOTS*
 - *How To Create*
- *Hybrid Mono*
 - *How To Create*
- *Player Car Pool*
 - *Where To Find*
 - *How To Use*
 - *Example*

Common Info

- Each player car only works in *Hybrid custom physics* type & consists of 2 parts: *DOTS* entity is handled by *DOTS* systems & *Monobehaviour* hybrid skin (required for camera tracking & npc car layout) that follows the binded *DOTS entity*.
- Player hybrid skin car should have following components:
 - **Player Actor** : camera tracking component.
 - **PlayerNpcCarBehaviour** : logic of the NPC shooting from the car.
 - **CarSlots** : stores the NPC data that is in the car (Npcs sitting in the car are taken from the *PlayerNpcCarFactory*).

Note: Look at *PlayerCarSkinBase* prefab example.

Hybrid DOTS

- Player vehicle physics handled by *DOTS* systems.
- The hybrid skin follows the *DOTS* entity.

How To Create

1. Open *Car Prefab Creator* & set *Car type* to *Player* in the *Prefab* tab.
2. Create a vehicle using the *Car Prefab Creator* tool.
3. Create prefab variant from the *PlayerCarSkinBase* prefab & assign to *Player Car Pool* according to the vehicle.
4. Customize the car slots in the created prefab if you plan to make the NPC's shooting option from the car [**optional step**].
5. Enable spawn type to *Car* & select desired *Car model* from the list in the *PlayerSpawner*, if you want the player to spawn in a car initially [**optional step**].
6. Once all the steps have been completed, reopen *subscene*.

Hybrid Mono

- Player vehicle physics handled by custom user's *Monobehaviour physics plugin*.
- Unlike the *Hybrid DOTS*, the entity following & presents collider for *DOTS* world without mesh representation.
- **List of vehicle controllers from the Asset Store that can be used for (e.g.)**
 - Edy's Vehicle Physics
 - Realistic Car Controller Pro
 - NWH Vehicle Physics 2
 - Universal Vehicle Controller Plus
 - MS Vehicle System
 - Sim-Cade Vehicle Physics

How To Create

1. Set the *World simulation type* to *Hybrid mono* in the *General settings* config (make sure that config on the *subscene* has the same value).
2. Open the *Car Prefab Creator* & set *Car type* to *Player* in the *Prefab* tab.
3. Set *Entity type* to *Hybrid entity mono physics* in the *Save* tab.
4. Drag & drop your desired prefabs into the *Prefs* field.
5. Click the *Scan* button.
6. Customize *Save settings* in the *Save* tab.
7. In the *Prefab Info* tab, enter the vehicle *ids* (*ids* should match the traffic cars *ids* if you want to make option enter & exit for the player npc).

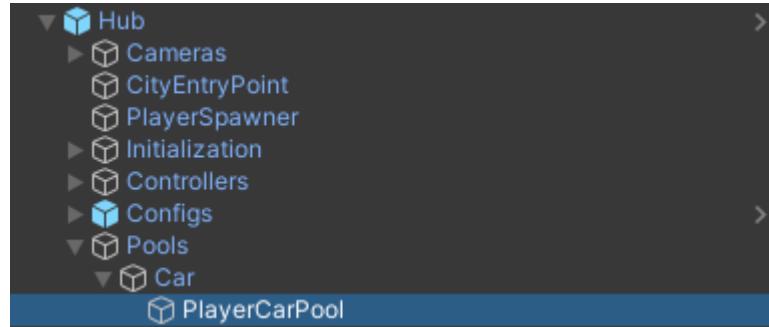
8. Click the *Create* button.
9. Ensure that the bounds of the entities created match the prefabs you have selected.
10. Input for the player vehicle is implemented according to your vehicle controller plugin.
11. The input enable & disable for the car when the player's npc exits & enters the car should be implemented in the *PlayerInteractCarService.cs* in the *EnterCar* & *ExitCar* methods.
12. Enable spawn type to *Car* & select desired *Car model* from the list in the *PlayerSpawner*, if you want the player to spawn in a car initially [optional step].
13. Once all the steps have been completed, reopen *subscene*.

Player Car Pool

Where To Find

In the scene:

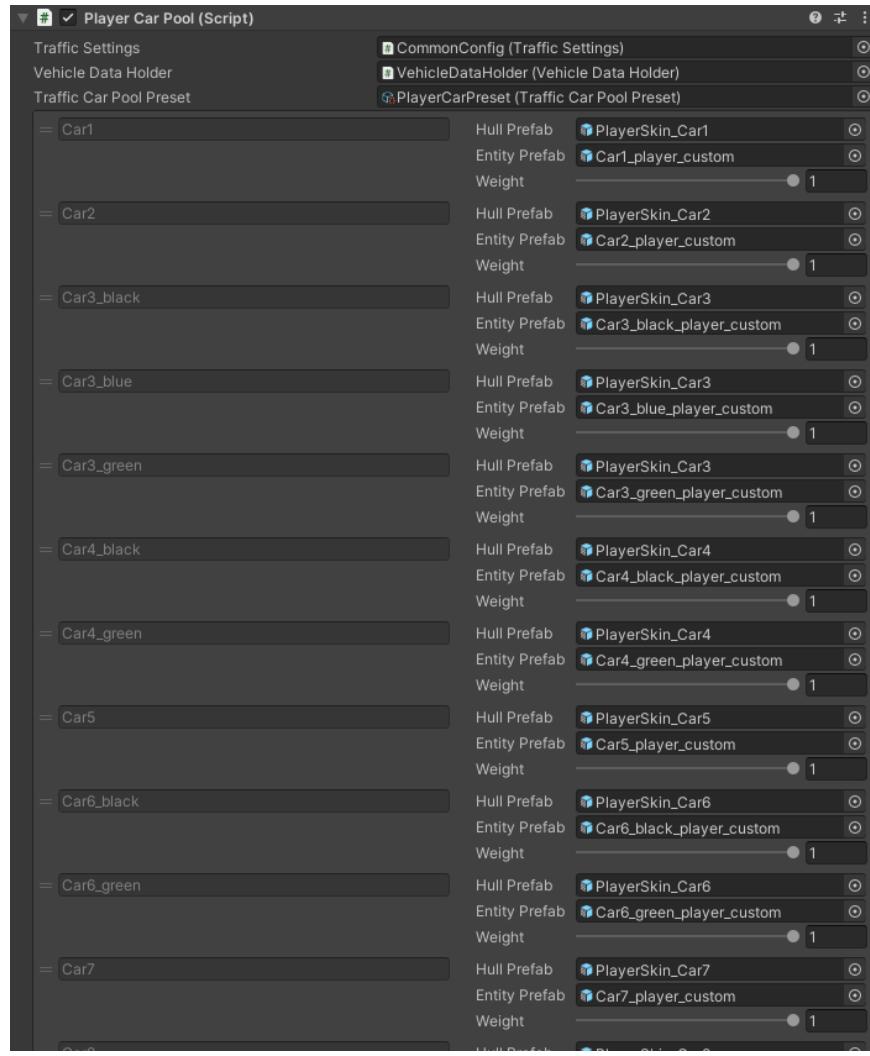
Hub/Pools/Car/PlayerCarPool



How To Use

Player cars spawned by *PlayerCarSpawner*.

Example



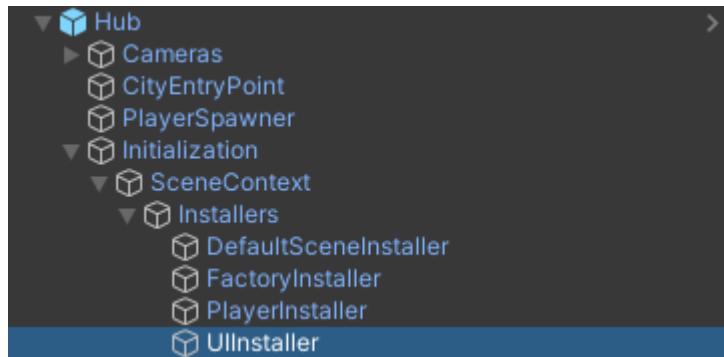
5.1.4 Custom Camera

Info

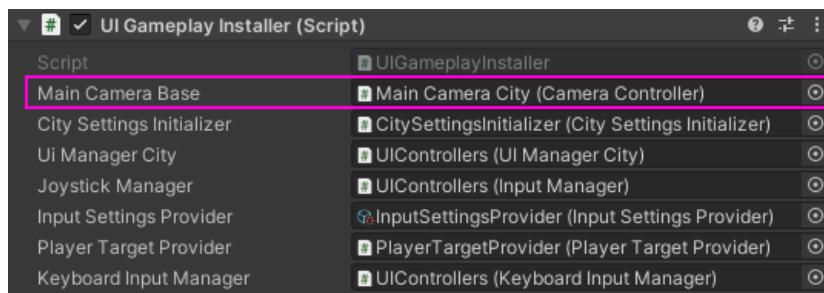
If you want to add your own camera solution, follow these steps:

Steps

1. Create a new *MonoBehaviour* script & implement the class that derived from *CameraBase* & add it to the root of your custom camera (the code example in the *Code Example* section below).
2. Find the *UIInstaller* object on the scene.



3. Assign the new camera to the *Main Camera Base* field.



4. Create a *Cull point* child for the new camera by adding a *CullPointRuntimeAuthoring* to the new gameobject.
5. Make sure you have deleted or disabled the old camera.

Code example

```
// The new MonoBehaviour script that derived from 'CameraBase'
public class ExampleCamera : CameraBase
{
    // Your own camera solution example
    public vThirdPersonCamera cam;

    // Injecting PlayerActorTracker via Zenject (mandatory)
    [Inject]
    public override void Construct(PlayerActorTracker
        playerActorTracker)
    {
        base.Construct(playerActorTracker);
    }
}
```

(continues on next page)

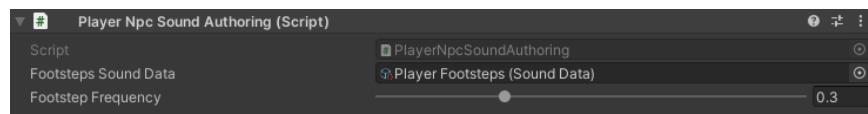
(continued from previous page)

```
// Override this event method to set target for your custom camera, pseudo code example:
protected override void PlayerActorTracker_OnSwitchActor(Transform newPlayerActor)
{
    // Set the player transform target
    cam.SetTarget(newPlayerActor);
}
```

5.1.5 Player Configs

- *Player Npc Sound Config*

Player Npc Sound Config



Footstep frequency : *sound* frequency of the player's footsteps.

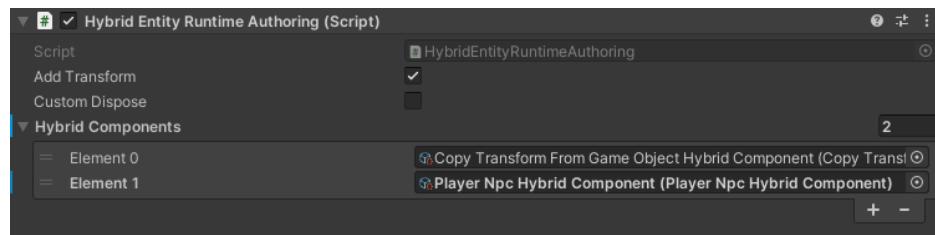
5.2 User Custom Solution

This topic about how to replace the player with a custom user solution.

5.2.1 Player Npc

If you want your own script to spawn player npc, follow these steps:

1. Set the *World simulation type* to *Hybrid mono* in the *General settings* config.
 2. Set the *Player controller type* to *Custom* in the *General settings* config.
 3. Disable built-in camera on the scene.
- 
4. Make sure your camera has a *cull point* object as a child (set local position to zero).
 5. Add the *HybridEntityRuntimeAuthoring* component to your prefab.
 6. Add *Copy Transform From Game Object*, *Player Npc* components in the *Hybrid components* list.



7. If you haven't created your own scene yet, you can try the *Demo Mono* scene for tests.
8. Custom player only works with *Hybrid Mono* vehicles.
9. Your player prefab npc is ready.

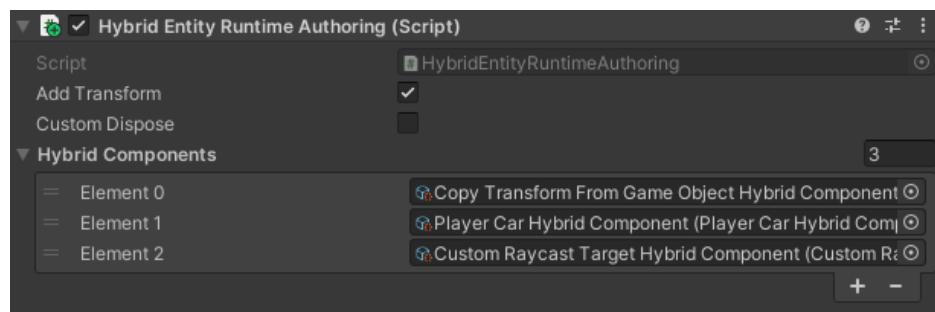
5.2.2 Player Car

If you want your own script to spawn player car, follow these steps:

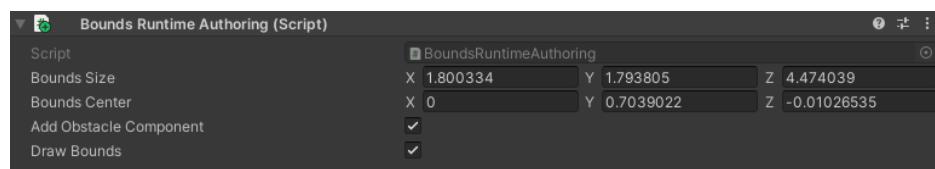
1. Set the *World simulation type* to *Hybrid mono* in the *General settings* config.
2. Set the *Player controller type* to *Custom* in the *General settings* config.
3. Disable built-in camera on the scene.



4. Make sure your camera has a *cull point* object as a child (set local position to zero).
5. Make sure that you created & selected *Hybrid mono* traffic.
6. Add the *HybridEntityRuntimeAuthoring* component to your prefab.
7. Add *Copy Transform From Game Object*, *Custom Raycast Target*, *PlayerCar* components in the *Hybrid components* list.



8. Add the *Bounds runtime authoring* & *Velocity runtime authoring* components to your prefab if you want to have *calculated collisions* with pedestrians. (*optional step*)



Bounds component example.



Velocity component example.

9. Make sure that the traffic *Raycast config* includes a player car layer.
10. If you haven't created your own scene yet, you can try the *Demo Mono* scene for tests.
11. Custom player car only works with *Hybrid Mono* vehicles.
12. Your player prefab car is ready.

5.3 Pure Traffic Simulation

If you want to remove all the sample code, including the player feature & extras, and just keep the traffic simulation, follow these steps:

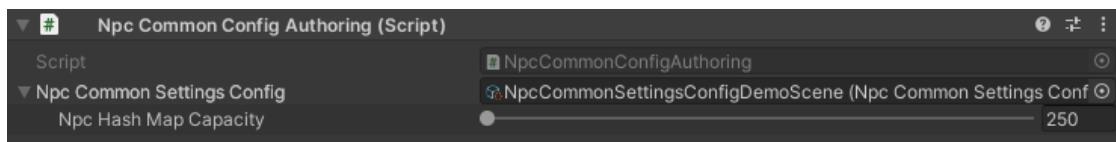
1. Add the *DOTS_SIMULATION* scripting define to your project's *Player settings*.
2. Only works on newly created scenes.
3. Check out the *PureCityStressTest* scene for an example.

NPC

- *Configs*
 - *Npc Common Config*
 - *Npc Ground Config*

6.1 Configs

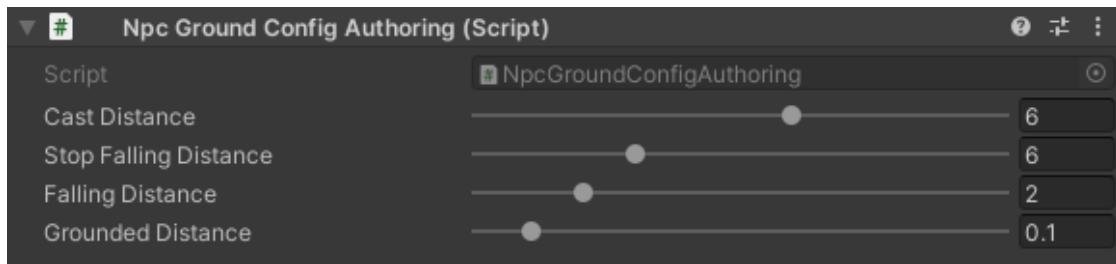
6.1.1 Npc Common Config



Npc HashMap capacity : initial capacity of the hashmap containing data about the NPC (position, state, etc...).

Note: HashMap contains pedestrian npcs, common nps and player npcs.

6.1.2 Npc Ground Config



Cast distance : raycast distance.

Stop falling distance : distance from the surface where the landing animation starts.

Falling distance : min distance from the surface where the falling state starts.

Grounded distance : distance from the surface for ground state.

Note: Currently only used for player NPCs.

STREAMING

7.1 Cullpoint Info

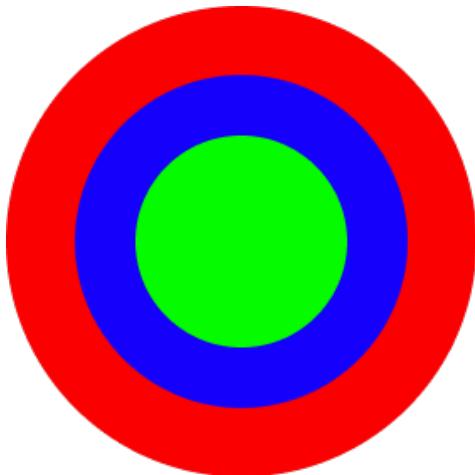
The cull point is the origin for the surrounding entities (by default, it's a child of the camera). The *cull state* of the surrounding entities varies depending on the distance to the culling point (*example*). You can change the distances in the cull config & *debug*.

7.1.1 States

- **Culled** : entity is far away (by default, the entity is destroyed or disabled).
- **CloseToCamera** : entity is enabled but with limited or modified functionality for better performance.
- **PreInitInCamera** : *state* between to *CloseToCamera* and *InViewOfCamera*, currently used to activate static physics objects [*optional*].
- **InViewOfCamera** : entity is fully enabled.

Default State List

The default list is used for most objects, contains *Culled*, *Close to camera*, *In view of camera* states.



- 1) In view of camera
- 2) Close to camera
- 3) Culled

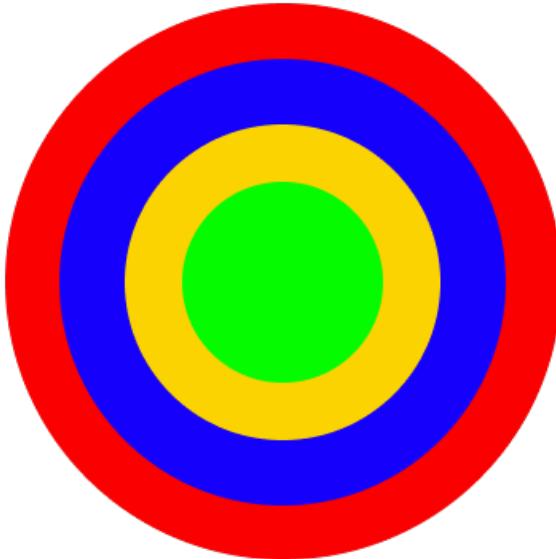
Default state list example.

Note:

-
- States add to the prefab entity by `CullComponentsExtension.CullComponentSet` extension method.
-

Extended State List

The extended state list is used for objects that require a pre-init state before viewing in camera state, but earlier than the *close to camera* state, contains *Culled*, *Close to camera*, *Pre-init in camera*, *In view of camera* states.



- 1) In view of camera**
- 2) Pre-init camera**
- 3) Close to camera**
- 4) Culled**

Extended state list example.

Note:

- States add to the prefab entity by `CullComponentsExtension.PreinitCullComponentSet` extension method.
 - It is used in the project for static physics objects to load them earlier than the dynamic physics objects to avoid dips through static surfaces.
-

7.2 Scene Streaming

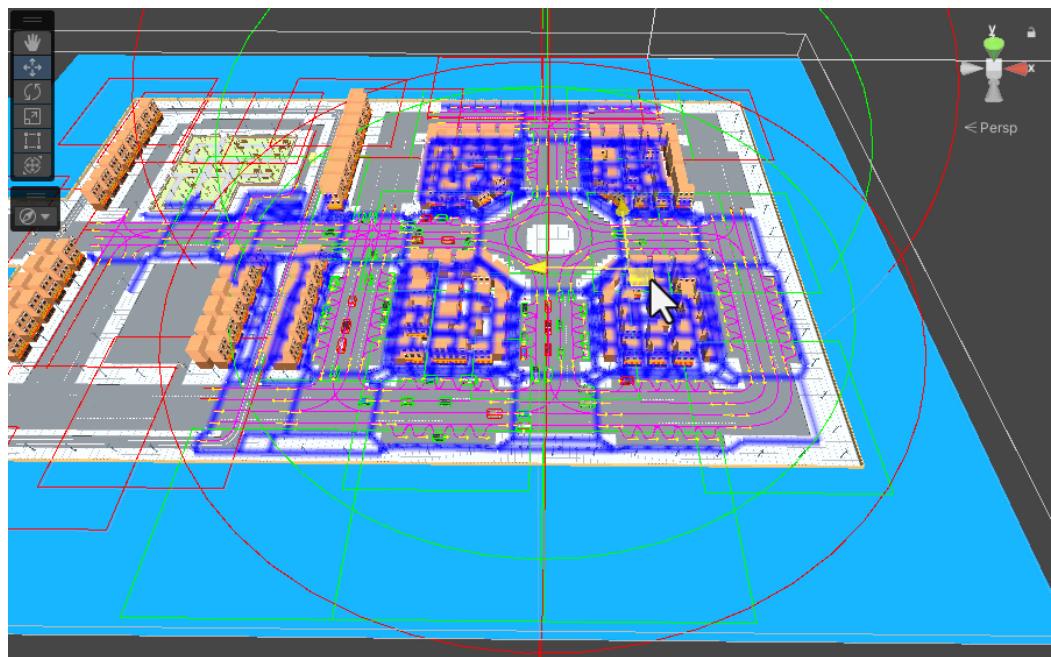
7.2.1 Road Streaming

Road Streaming is needed to split the map into chunks to create huge maps, so that road entities: *TrafficNodes*, *PedestrianNodes*, etc. are only loaded where the player is.

Youtube tutorial.

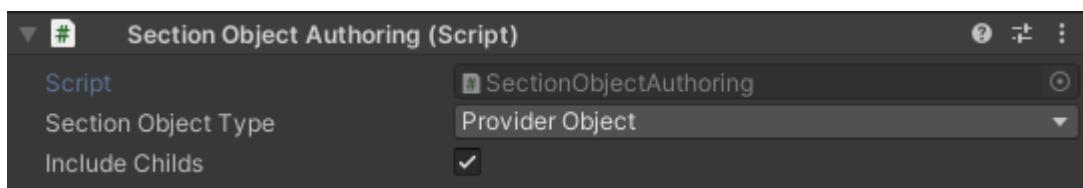
How To Adjust

1. Enable streaming in the Road Streaming Config to load road sections at the runtime.
2. Adjust load/unload distance and section cell size.
3. *TrafficNode*, *PedestrianNode*, *TrafficLightHandler* automatically attaches to related *RoadSegment*.
4. *Traffic lights* has a *SectionObjectAuthoring* component.
5. If you want to add your own section object, add the *SectionObjectAuthoring* component and select the appropriate *Section object type*.
6. *Debug streaming* distance and section size.



Road streaming example.

Section Object Authoring



Section object type:

- **Attach to closest** : attach to nearest road section.
- **Create new if necessary** : create a new road section if doesn't exist with the currently computed section hash.
- **Provider object** : object has a component that implements the *IProviderObject* interface, that provides a reference to the associated object section.
- **Custom object** : user's own associated object section.

Include childs : all child objects are included in the section of the parent object.

7.2.2 Scene Streaming

You can split the scene content into chunks for partial loading at runtime.

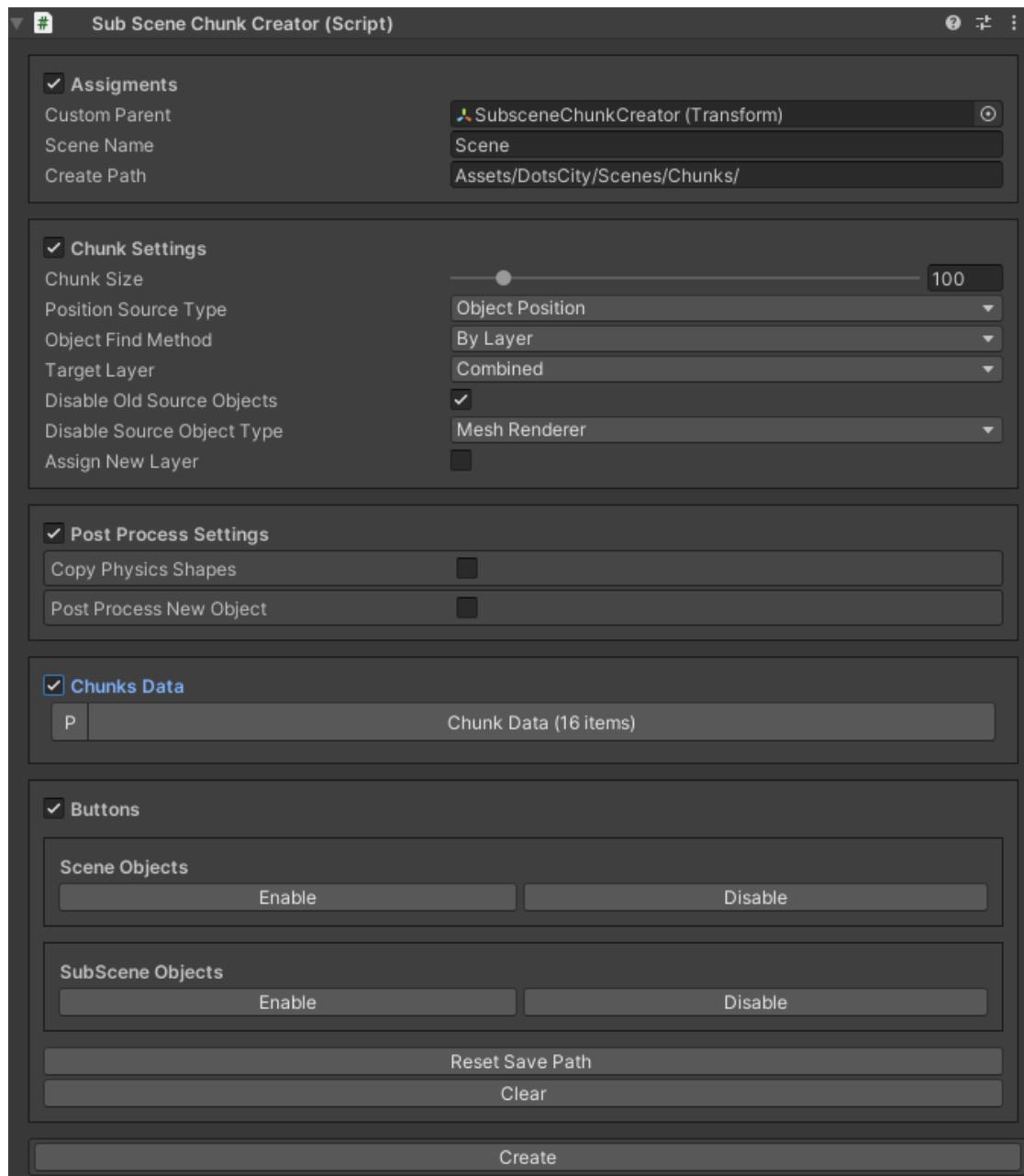
[Youtube tutorial](#).

How To Create

1. Create a new empty *GameObject* and add the *SubSceneCreator* component.
2. Adjust the *chunk settings*.
3. If necessary, enable *post process settings* [**optional step**].
4. Press the *Create* button.
5. Adjust the Streaming Level Config to load/unload subscenes at the runtime.

7.2.3 SubScene Chunk Creator

Content chunking tool to split the scene into chunks. Old objects remain disabled in the old scene and are used to create duplicates in the chunk sub-scenes.



Assignments

Custom parent : custom parent of subscene.

Scene name : subscene template name.

Create path : create subscenes path.

Chunk Settings

Chunk size : chunk size.

Position source type

[source position of the object to be assigned to the chunk.]

- **Object position**
- **Mesh center**

Destroy previous created : destroy previously created chunks.

Object find method

[method for finding an object to add to a chunk.]

- **By tag** : by *Unity* tag.
- **By layer** : by *Unity* layer.

Target tag : search tag.

Disable old source objects : turn off the source objects.

Disable source object type

- **Mesh renderer** : disable meshRenderers of source objects.
- **Parent** : disable parents of source objects.
- **Parent if no mesh** : disable the meshRenderer, but if not, disable the parent.

Assign new layer : assign new layer to objects created from new chunks.

Post Process Settings

Copy physics shape : on/off *PhysicsShape Transfer* tool.

Post process new object : on/off post processing of the object.

Component type name : target component name.

Post process type:

- **Delete component** : the component found will be deleted.
- **Delete object** : the object with the found component will be de.

Chunk Data

Buttons

Create : create subscene chunks.

Enable/disable scene objects : enable/disable source scene objects.

Enable/disable sub scene objects : enable/disable created subscene objects.

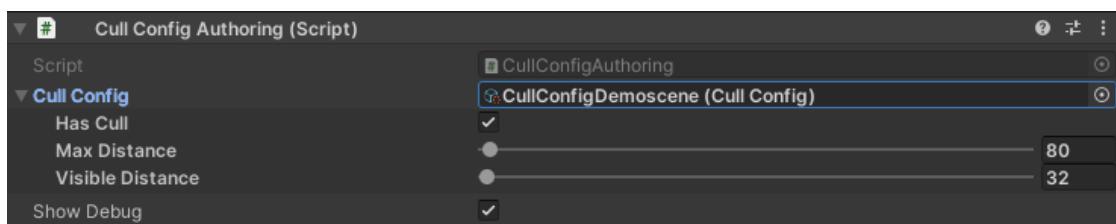
Reset save path : reset save path of the subscenes.

Clear : clear created subscene chunks.

7.3 Configs

7.3.1 Cull Config

Config of the *cull point*.



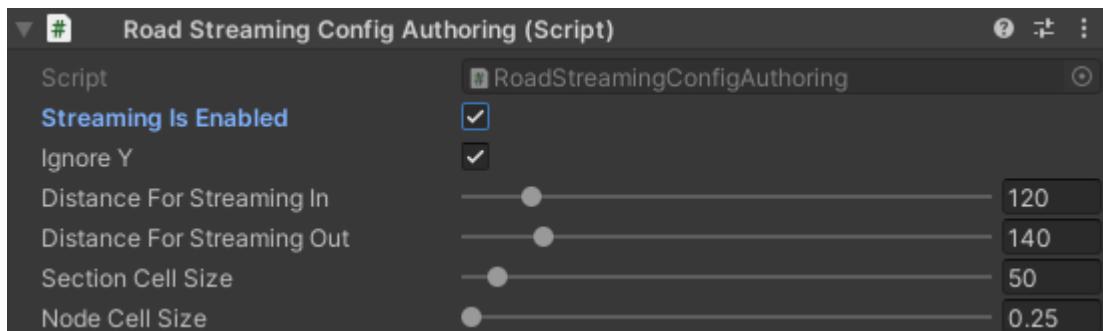
Has cull:

- **Max distance** : maximum distance to activate entities.
- **Visible distance** : distance to activate visual features of entities.

Show debug : on/off visual culling circle in the scene.

7.3.2 Road Streaming Config

Config for *load/unload* road sections from the entity *subscene*.



Streaming is enabled : on/off streaming.

Ignore Y : ignore calculation of distance to road section for Y axis.

Distance for streaming in : distance at what the road section is loaded.

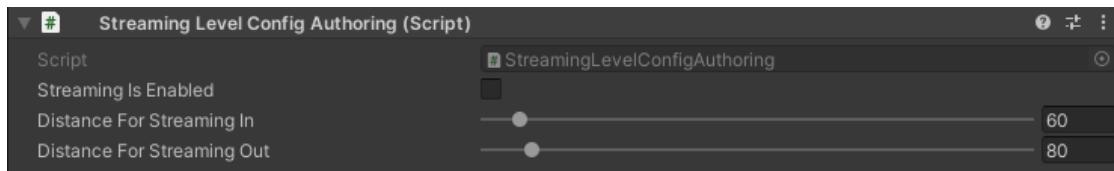
Distance for streaming out : distance at what the road section is unloaded.

Section cell size : cell size of the road section.

Node cell size : node size for *TrafficNode* and *PedestrianNode* in order to compute a unique position hash for them.

7.3.3 Streaming Level Config

Config for *load/unload* content subscenes.



Streaming is enabled:

- **Distance for streaming in** : distance at what the subscene is loaded.
- **Distance for streaming out** : distance at what the subscene is unloaded.

COMMON INFO

8.1 Common Info

- *Hybrid Entities*
 - *How To Create*
- *Props*
 - *How To Use*
 - *Props Authoring*

8.1.1 Hybrid Entities

Entities that combine *DOTS* entities and default *GameObjects* (game objects are tied by position to an entity).

How To Create

1. Create a prefab entity through the [baking](#).
2. Add the `CopyTransformToGameObject` component and add your custom init component to the [baking](#) process for initialization, pseudocode example:

```
public struct InitComponentExample : IComponentData, ↴  
    IEnableableComponent  
{  
}
```

3. Spawn a prefab entity at runtime.
4. Create your own init system to initialize your hybrid entity, pseudocode example:

```
[UpdateInGroup(typeof(InitializationSystemGroup))]  
public partial class InitSystemExample : SystemBase  
{  
    private ExampleFactory exampleFactory;  
    private EndInitializationEntityCommandBufferSystem ↴  
    entityCommandBufferSystem;
```

(continues on next page)

(continued from previous page)

```

protected override void OnCreate()
{
    base.OnCreate();

    entityCommandBufferSystem = World.
    ↪GetOrCreateSystemManaged
    ↪<EndInitializationEntityCommandBufferSystem>();
}

protected override void OnUpdate()
{
    var commandBuffer = entityCommandBufferSystem.
    ↪CreateCommandBuffer();

    Entities
        .WithoutBurst()
        .WithStructuralChanges()
        .WithAll<InitComponentExample>()
        .ForEach<(
            Entity entity) =>
    {
        var exampleObject = exampleFactory.Get();

        //Bind transform to entity
        EntityManager.AddComponentObject(entity, ↪
    ↪exampleObject.transform);

        //Add required component if missing
        commandBuffer.AddComponent
    ↪<CopyTransformToGameObject>(entity);

        //Disable init component
        commandBuffer.SetComponentEnabled
    ↪<InitComponentExample>(entity, false);

    }) .Run();

    entityCommandBufferSystem.
    ↪AddJobHandleForProducer(Dependency);
}

//Some factory that is assigned from outside

public void Initialize (ExampleFactory exampleFactory)
{
    this.exampleFactory = exampleFactory;
}
}

```

Warning: All code provided in the example is not part of *DOTS City* and is not intended for production.

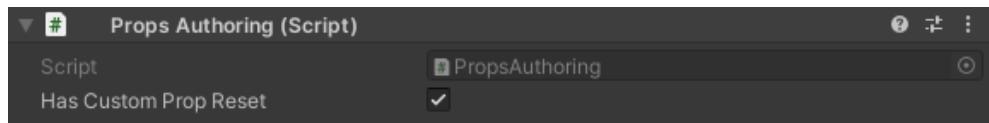
8.1.2 Props

Props are active entities that react to damage.

How To Use

1. Create props prefab.
2. Add *Props Authoring* component.
3. Tick if necessary *Has Custom Prop Reset*.
4. Make sure that *Props damage system support* option is enabled.
5. Use a *test scene* to check that the props work.

Props Authoring



Has custom prop reset : if checked, a custom reset system must be implemented for this object that contains *PropsCustomResetTag* component.

Custom reset of hydrant, example code:

```
Entities
.WithoutBurst()
.WithAll<PropsCustomResetTag>()
.WithAll<HydrantTag>()
.ForEach<(
    Entity entity,
    ref PropsVFXData propsVFXData) =>
{
    if (propsVFXData.RelatedEntity != Entity.Null)
    {
        var particleSystem = EntityManager.GetComponentObject<ParticleSystem>(propsVFXData.RelatedEntity);
        particleSystem.Stop();
        particleSystem.gameObject.ReturnToPool();

        commandBuffer1.DestroyEntity(propsVFXData.RelatedEntity);
        propsVFXData.RelatedEntity = Entity.Null;
    }
}
```

(continues on next page)

(continued from previous page)

```
commandBuffer.SetComponentEnabled<PropsCustomResetTag>(entity, false);
commandBuffer.SetComponentEnabled<PropsDamagedTag>(entity, false);

}).Run();
```

8.2 Common Configs

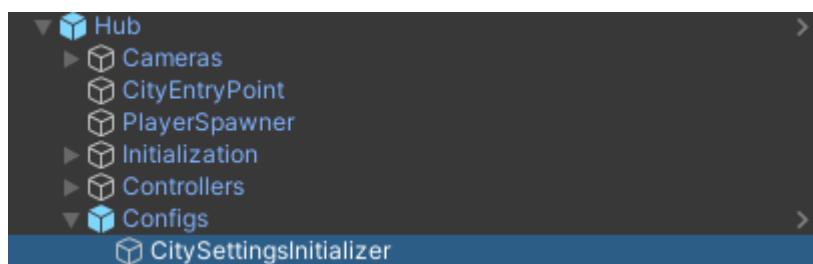
- *Common Configs*
 - *General Settings Config*
 - *Where To Find*
 - *Config Example*
 - * *Player Settings*
 - * *Player Target Settings*
 - * *Common Car Settings*
 - * *Traffic Car Settings*
 - * *Pedestrian Settings*
 - * *Other Settings*
- *Sound Configs*
 - *Common Sound Config*
 - *Crowd Sound Config*

8.2.1 Common Configs

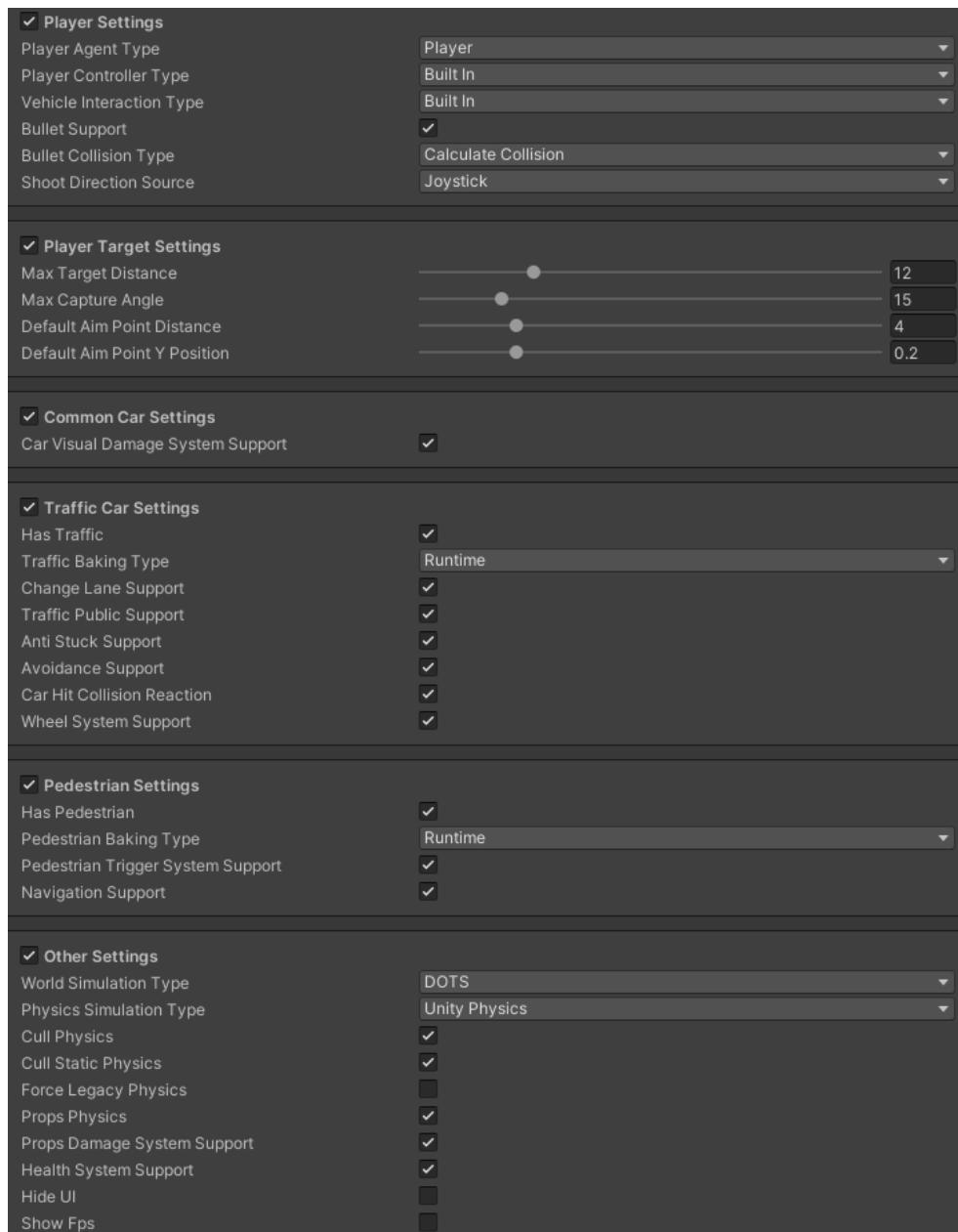
General Settings Config

Config to quickly on/off optional features.

Where To Find



Config Example



Player Settings

Player agent type:

- **Player** : player will be spawned.
- **Free fly camera** : flying camera will be spawned.

Player controller type:

- **Built In** : the player spawned by the built-in solution & has an example built-in controller.

- **Built In Custom** : the player spawned by the built-in solution, but the player NPC has a custom character controller & taken from *PlayerCustomHybridMonoNpcFactory*.
- **Custom** : the player is spawned & handled entirely by the user's custom solution.

Vehicle interaction type:

- **Built In** : the player interacting with cars by the built-in solution.
- **Custom** : the player interacting with the cars by the custom user solution.

Bullet collision type: method of calculating collisions for a bullet.

- **Calculate collision** : manual calculating.
- **Raycast** : by raycast.

Shoot direction source:

- **Joystick** : target of the firing in the direction of the joystick.
- **Crosshair** : target of the shooting in the direction of the crosshair position.

Player Target Settings

Max target distance : maximum distance for crosshair target capture (*crosshair mode only*).

Max capture angle : maximum angle for crosshair target capture (*crosshair mode only*).

Default aim point distance : distance between the player and the crosshair if there is no target.

Default aim point Y position : default Y-axis crosshair position.

Common Car Settings

Car visual damage system support : on/off visual hit feature for traffic vehicles by bullets.

Traffic Car Settings

Has traffic : on/off traffic vehicle in the city.

Traffic baking type:

- **Editor subscene** : all traffic entities are converted at editor subscene time.
- **Runtime** : all traffic entities are converted at runtime.

Change lane support : on/off feature to change lanes for traffic.

Traffic public support : on/off public traffic vehicle in the city.

Antistuck support : on/off *antistuck* feature for traffic vehicles.

Avoidance support : on/off *avoidance* of the vehicles.

Car hit collision reaction : on/off traffic collision reaction to other traffic cars.

Wheel system support : on/off simple wheel system for traffic vehicles.

Pedestrian Settings

Has pedestrian : on/off pedestrians in the city.

Pedestrian baking type:

- **Editor subscene** : all pedestrian entities are converted at editor subscene time.
- **Runtime** : all pedestrian entities are converted at runtime.

Pedestrian trigger system support : on/off trigger feature for pedestrians (fear running due bullets etc...).

Other Settings

World simulation type:

- **DOTS** : simulation of traffic & pedestrians entirely in *DOTS* space.
- **Hybrid mono** : physics simulation run on *Monobehaviour* scripts, but input taken from *DOTS* entities simulation.

Physics simulation type:

- **No physics** : dots physics off.
- **Unity physics** : *Unity* dots physics on.
- **Havok physics** : *Havok* dots physics on (havok physical package is required).

Cull physics : on/off culling of the physics of dynamic objects that are far from the player.

Cull static physics : on/off culling of the physics of static objects that are far from the player.

Force legacy physics : force enable *built-in physics*, otherwise *built-in physics* will be disabled when *ragdoll* is disabled.

Health system support : on/off health systems for all entities (vehicles, pedestrians, etc...).

Navigation support : on/off navigation systems for pedestrians.

Props damage system support : on/off damage systems for *props*.

Target FPS : target fps of the device.

Hide UI : on/off UI.

Show FPS : on/off fps ui panel.

8.2.2 Sound Configs

Common Sound Config



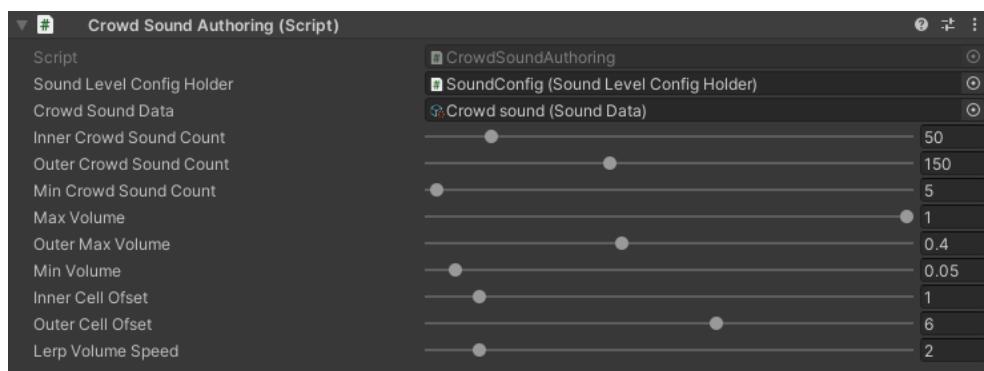
Has sounds : on/off DOTS sound systems.

Crowd sound : on/off *crowd sound* system for pedestrians.

Random horns sound : on/off horn *sound* system for traffic.

Crowd Sound Config

onfig for crowd background sound. The sound of the crowd is calculated based on of two areas: the inner circle and the outer circle. The sound in the inner circle is louder than the sound in the outer circle.



Crowd sound data : crowd *sound* data.

Inner crowd sound count : maximum volume for a given number of pedestrians in the inner circle.

Outer crowd sound count : maximum volume for a given number of pedestrians in the outer circle.

Min crowd sound count : minimum number of pedestrians to play the crowd sound.

Max volume : maximum volume level for the crowd sound.

Outer max volume : maximum volume in the outer circle.

Min volume : minimum volume level for the crowd sound.

Inner cell offset : offset of neighbouring cells relative to current cell in hashmap in the inner circle.

Outer cell offset : offset of neighbouring cells relative to current cell in hashmap in the outer circle.

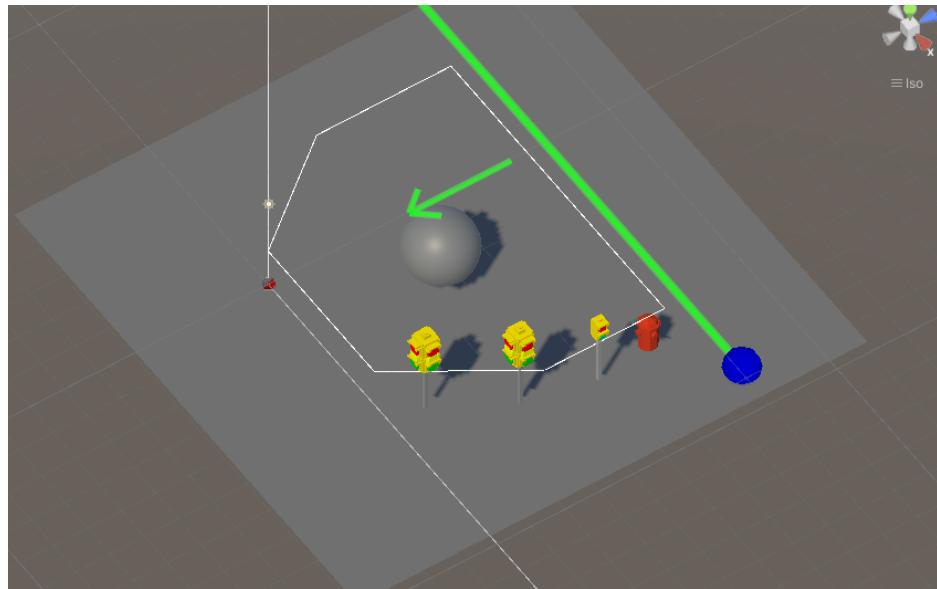
Lerp volume speed : speed of sound volume change between current value and target value.

8.3 Common Test Scenes

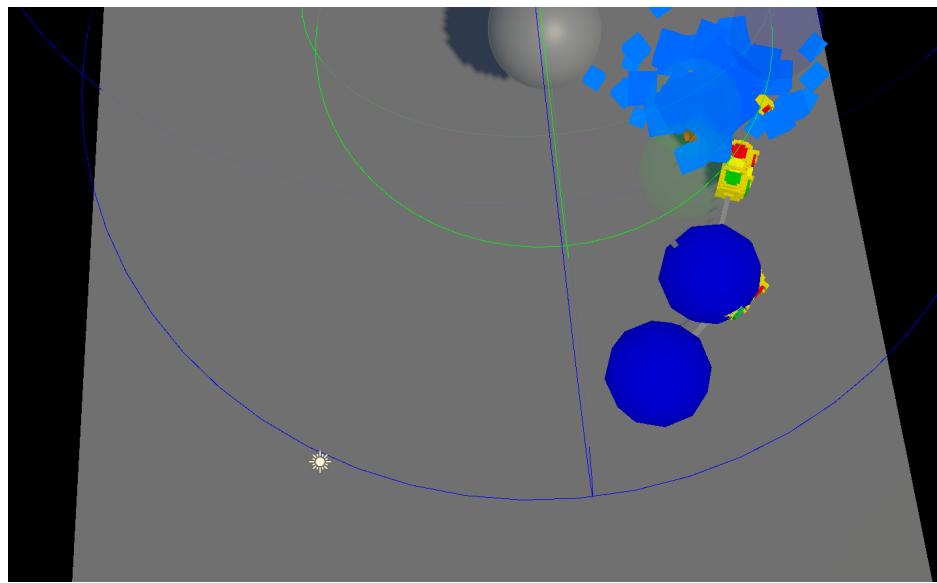
8.3.1 Props Test Scene

The scene is designed to test the reaction of the *props* to damage.

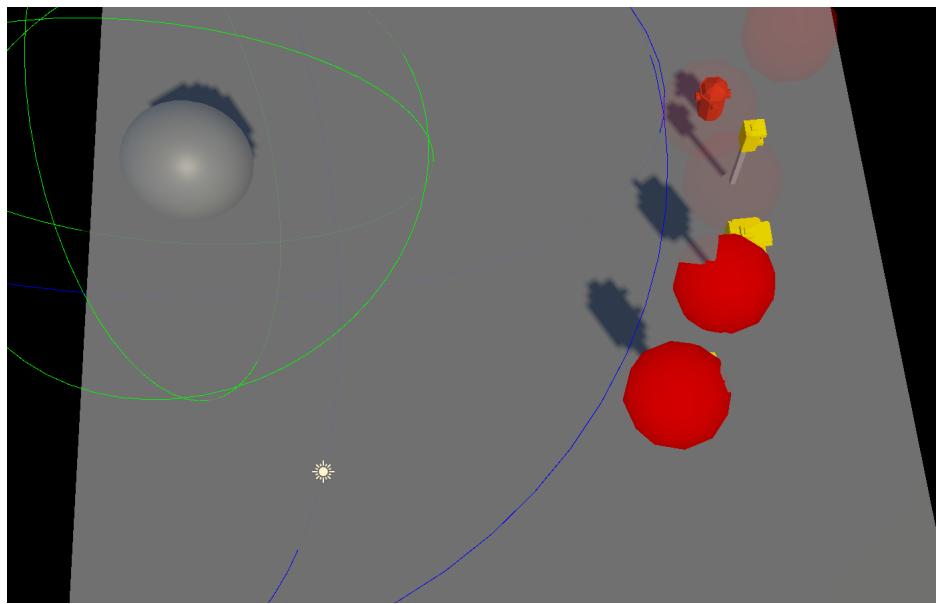
1. Source scene:



2. The ball knocks down the props (*cull states example*):



3. In the *culling state*, the props return to their original state:



8.4 Layer Info

- By default, the layers are stored in the *ProjectConstants.cs* script file.
- The project uses the following layers:

8.4.1 Road Layers

- **TrafficNode [20]** : *TrafficNode* layer [**required**].
- **PedestrianNode [21]** : *PedestrianNode* layer [**required**].

8.4.2 Object Layers

- **Ground [18]** : ground surface layer, but you can use any custom ground layer you want if you set it in the *Traffic cars* for ground detection. [*optional*]
- **Ragdoll [19]** : *ragdoll* collider layer. [*optional*]
- **StaticPhysicsShape [22]** : *Static object* layer collider layer (e.g. house, fence). [*optional, only used for DOTS scenes*]
- **Chunk [23]** : layer of *streaming* 3D chunks in the chunk subscenes created by *SubsceneChunkCreator* tool. [*optional, only used for DOTS scenes*]
- **Props [24]** : collider layer of *props* (e.g. traffic light, hydrant, mailbox). [*optional*]
- **Combined [25]** : layer of combined meshes into the one by 3rd party combining tool (*demo scene only*). [*optional*]

8.4.3 Misc Layers

- **CollidableNpc [15]** : player npc layer. *[demo scene built-in player, optional]*
- **TraficCar [16]** : *Traffic car* layer. *[demo scene only, optional]*
- **Npc [17]** : *Pedestrian* npc layer. *[demo scene only, optional]*

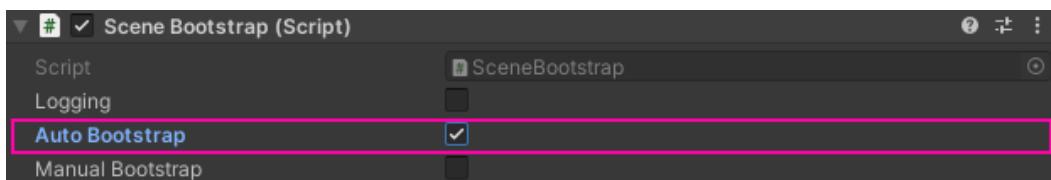
SCENE HANDLE

9.1 Bootstrap

- To start the scene automatically on the *Build* application, tick the *Auto-bootstrap* option in the *CityEntryPoint* (in the *Editor*, the scene loads by default).



CityEntryPoint.



SceneBootstrap.

- If you want to start the scene with code, assign *CityEntryPoint* with *SceneBootstrap* script to your script & use follow method, example:

```
[SerializeField] private SceneBootstrapBase sceneBootstrapBase;

private void StartInit()
{
    sceneBootstrapBase.StartInitialization();
}
```

- Use this code if you need to know when the bootstrap is complete, example:

```
[SerializeField] private SceneBootstrapBase sceneBootstrapBase;

private void Start()
{
    sceneBootstrapBase.OnComplete += SceneBootstrap_OnComplete;
}

private void SceneBootstrap_OnComplete()
{
    // Custom logic.
}
```

9.2 Scene Unload

- Before you unload the main scene, you should manually unload the entity subscene.
- Assign *EntityWorldService* to your script.



- You can use this sample code:

```
[SerializeField] private EntityWorldService entityWorldService;

private void Dispose()
{
    entityWorldService.DisposeWorld();
}
```

TEST SCENES

10.1 Traffic Test Scene

10.1.1 How To Use

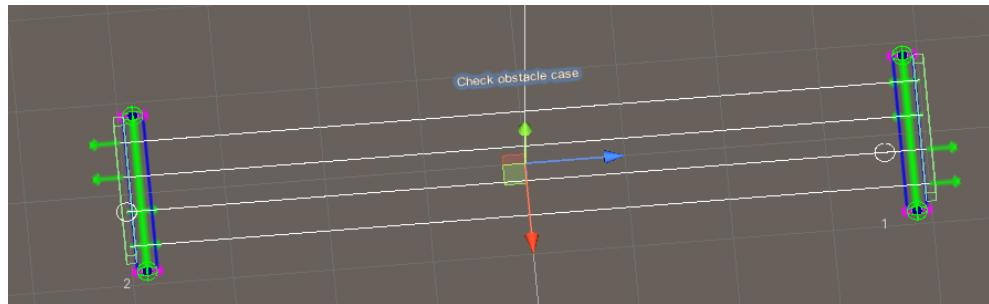
Youtube tutorial.

1. Create a *road segment*.
2. Create a parent *GameObject* and add the *TrafficCarRoadDebugger* component.
3. In the created component, press the *Show buttons*.
4. Select the in paths in the scene where you want the vehicle to spawn.
5. Adjust the normalized *path* position to set the spawn point.
6. Start the scene.
7. Click the *Spawn* button in the component.
8. Learn more about the *TrafficCarRoadDebugger* settings.

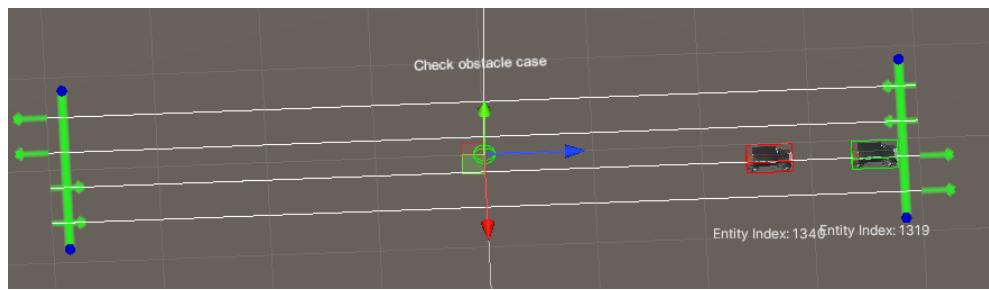
10.1.2 Test Cases

Check Obstacle

Config *obstacle* parameters.



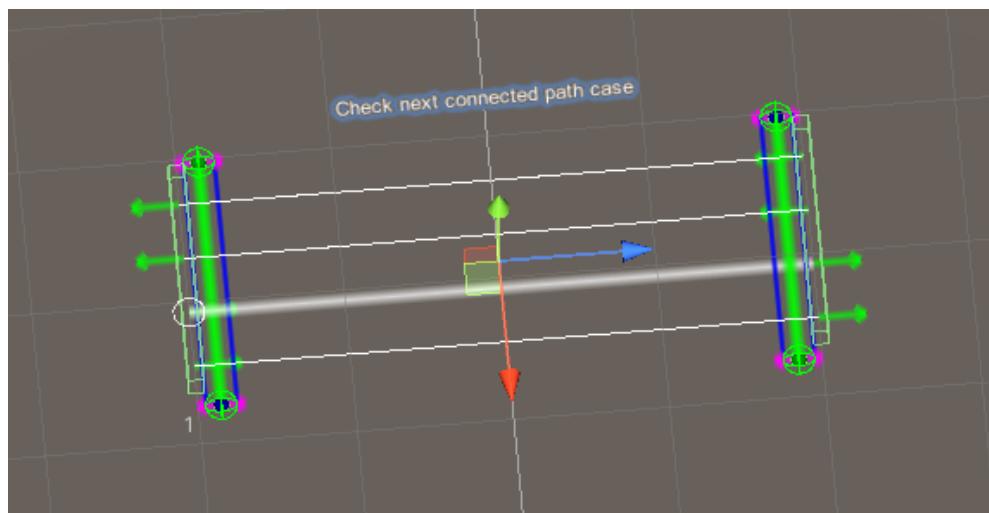
Source settings.



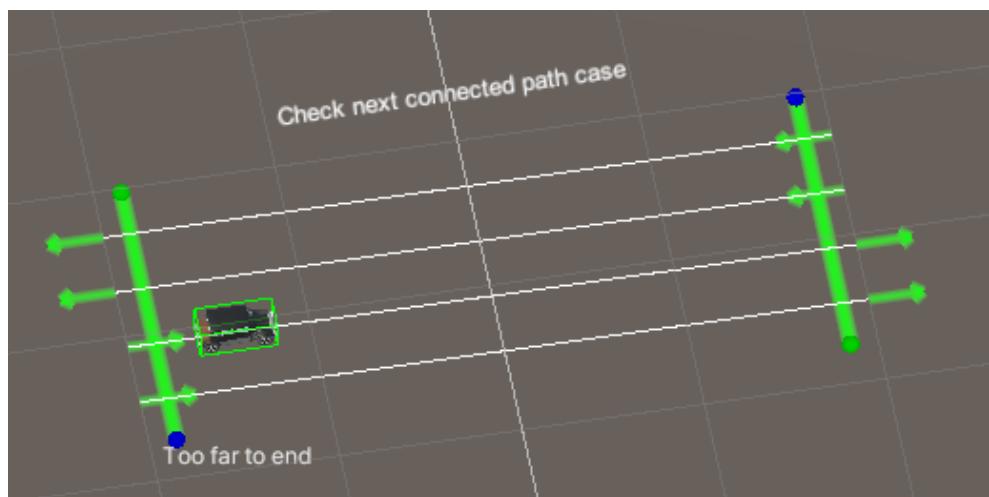
Test case result.

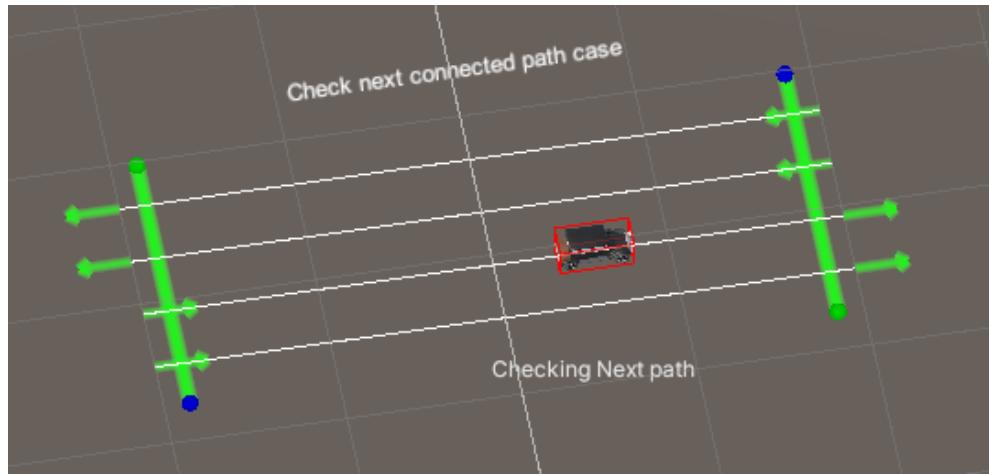
Check Next Connected Path

Config *Next connected path* parameter.



Source settings.



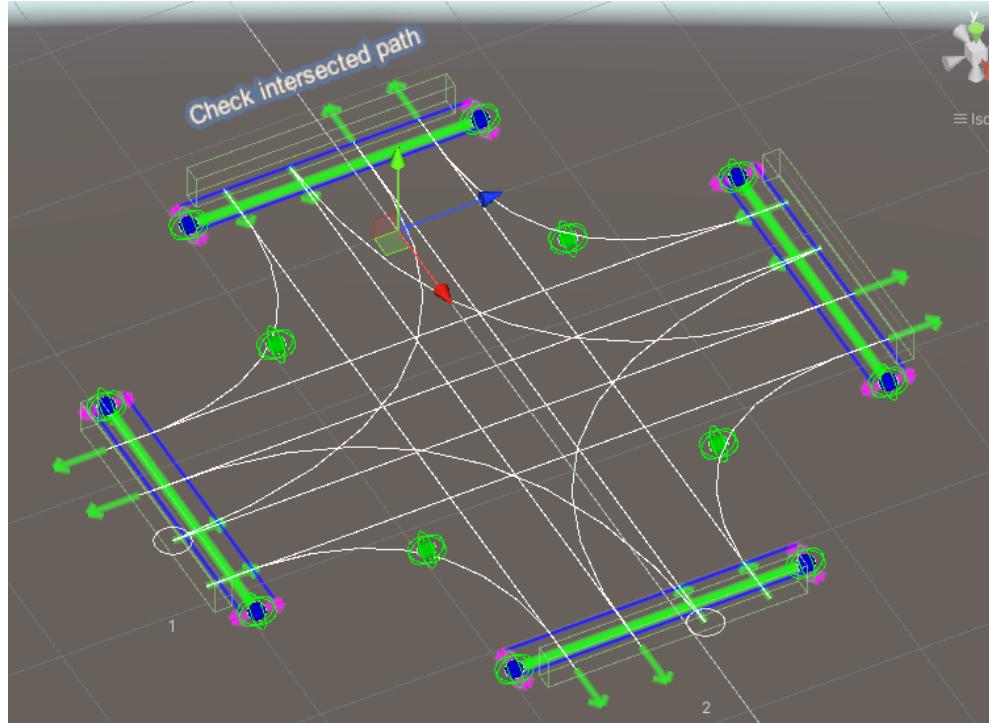


Test case result.

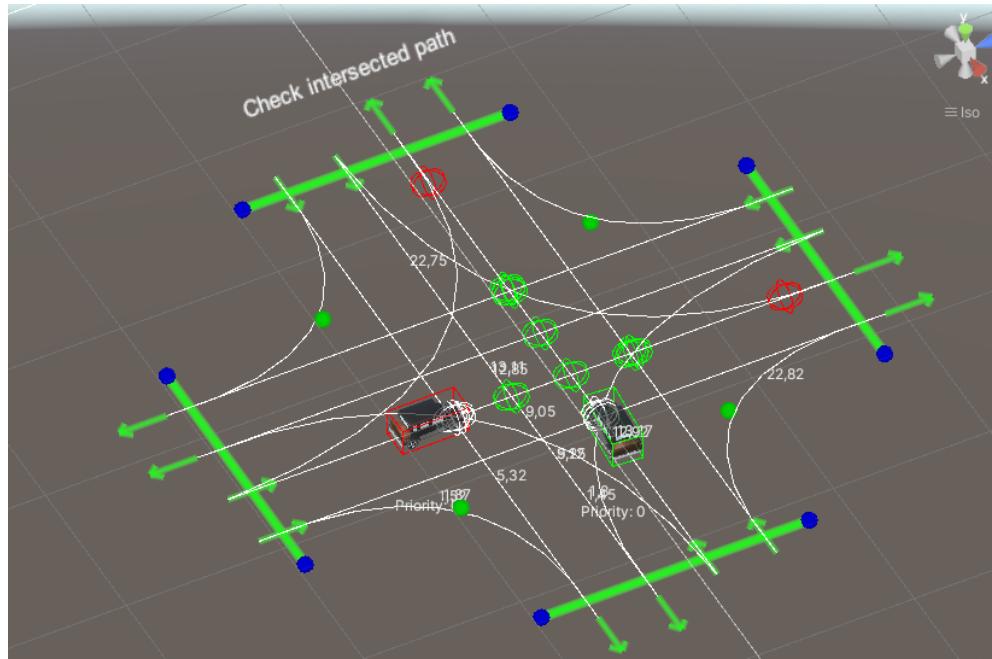
Check Intersected Path

Config *Intersected* parameters.

Two cars

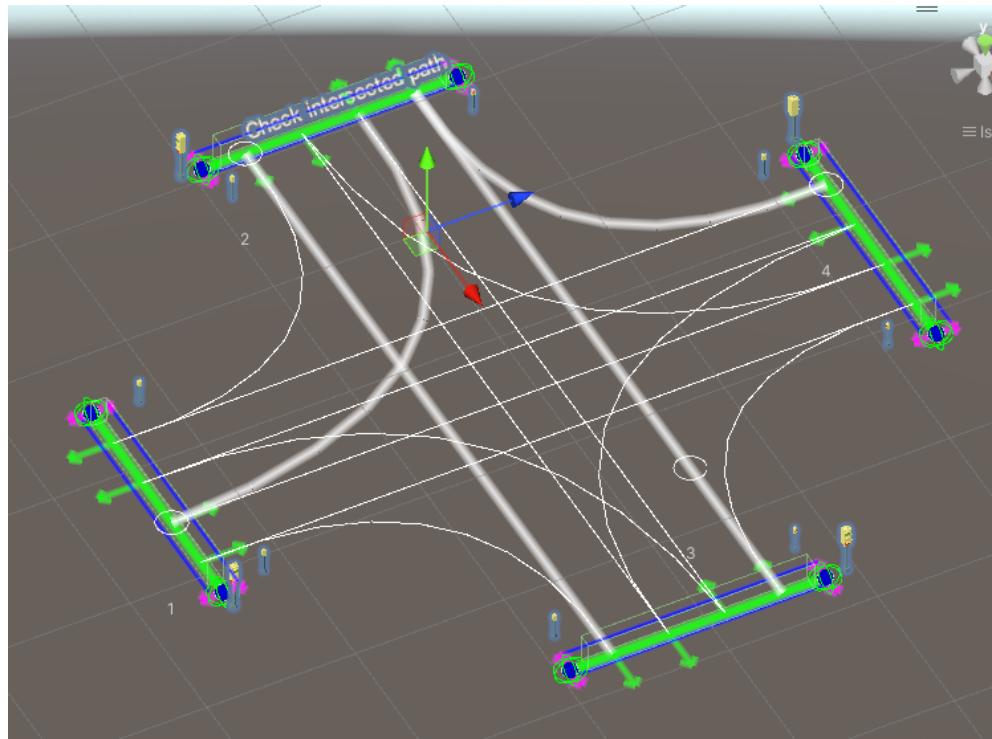


Source settings.

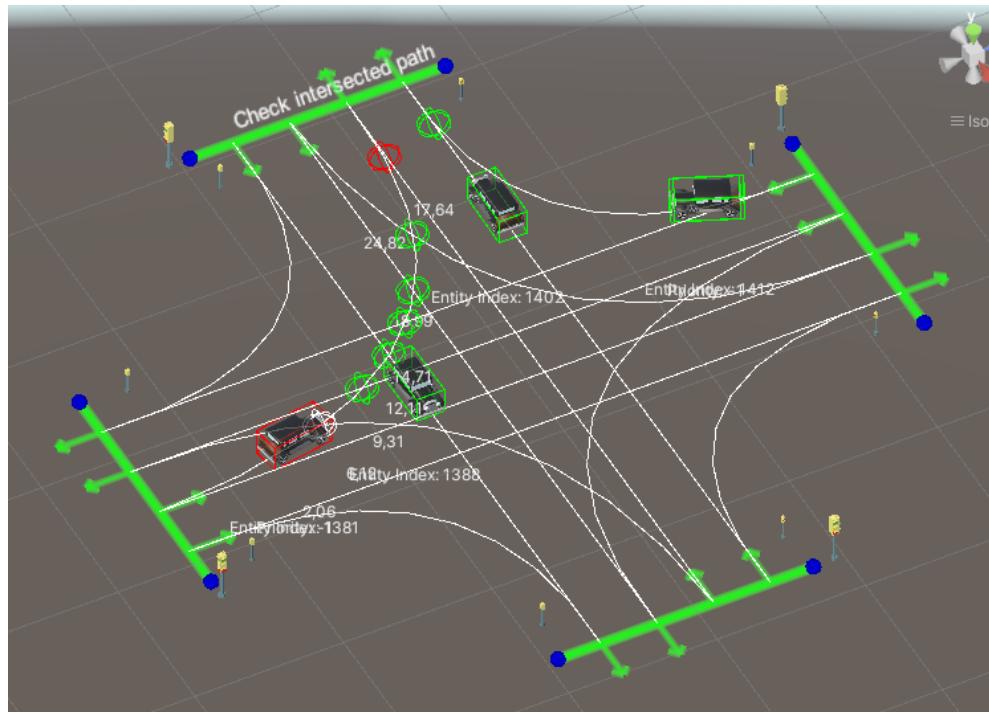


Test case result.

Multiple cars



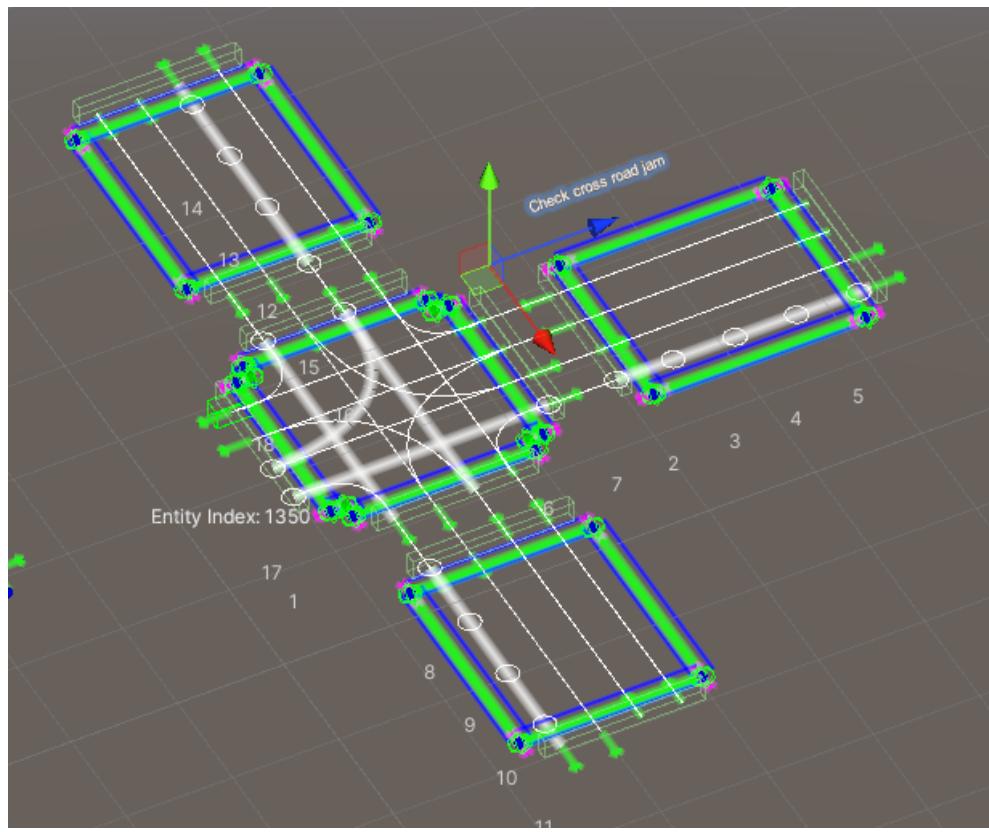
Source settings.



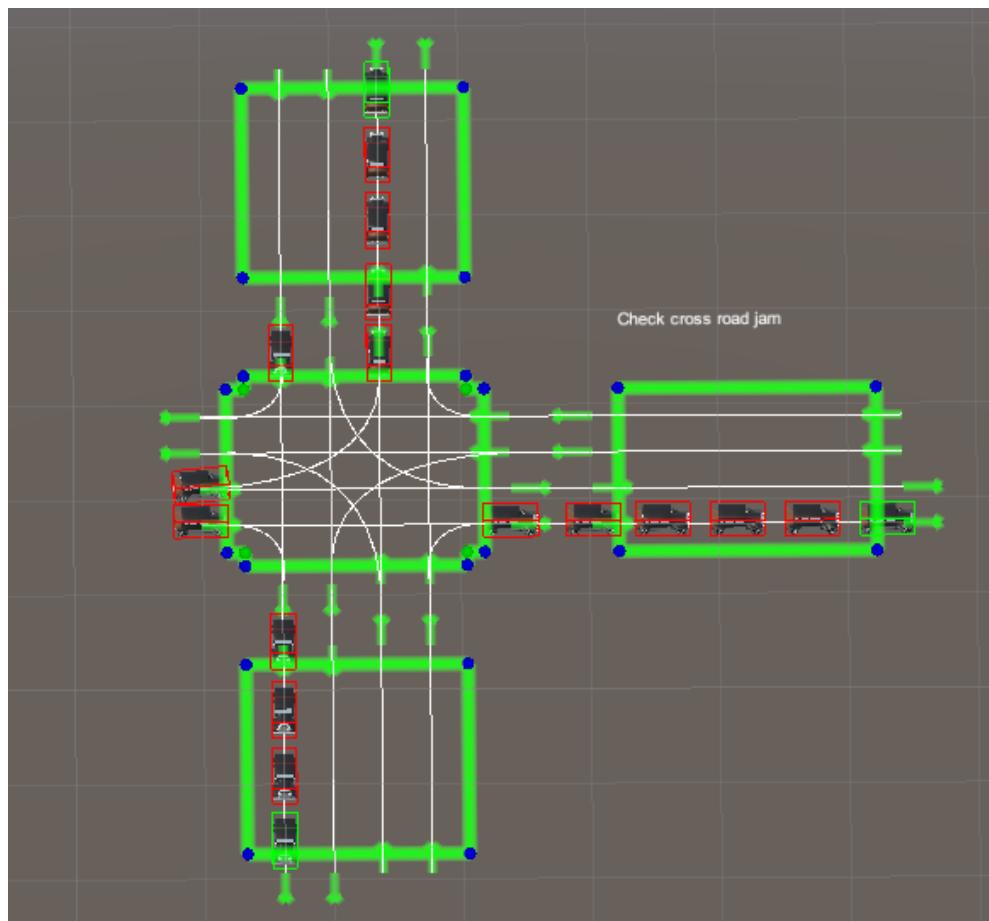
Test case result.

Check Crossroad Jam

Config *Avoid crossroad jam* parameter.



Source settings.

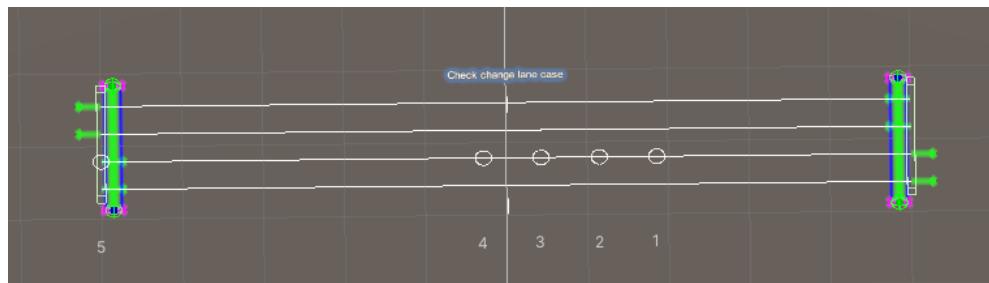


Test case result.

Check Change Lane

Config parameters.

Traffic jam in the lane

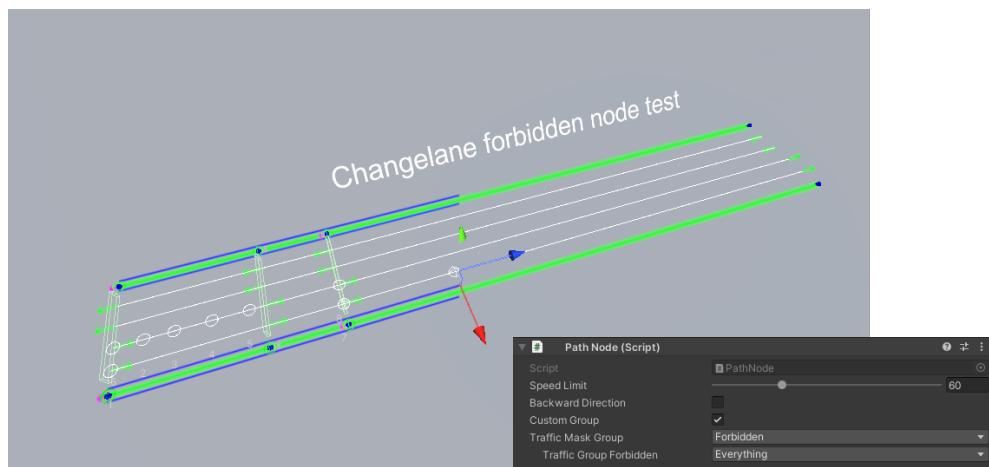


Source settings.

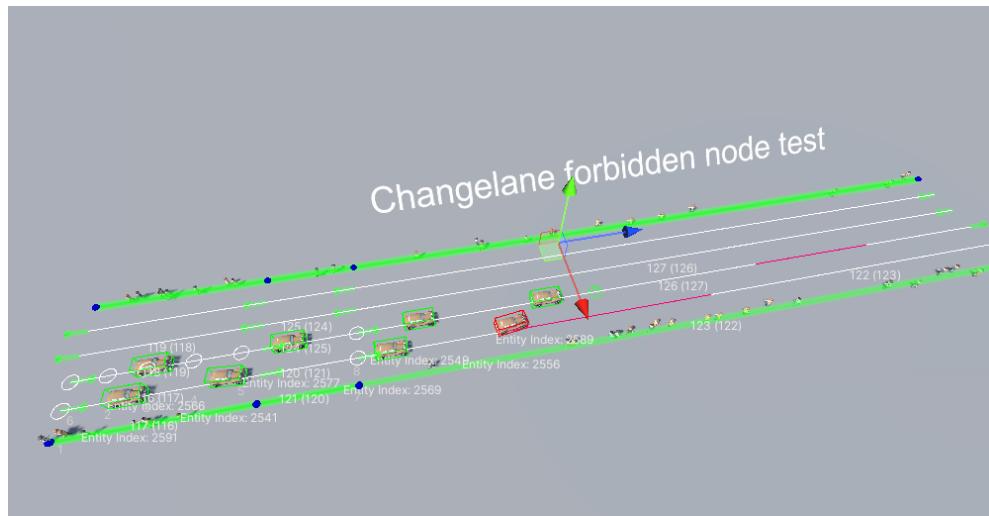


Test case result.

Traffic fobidden node test

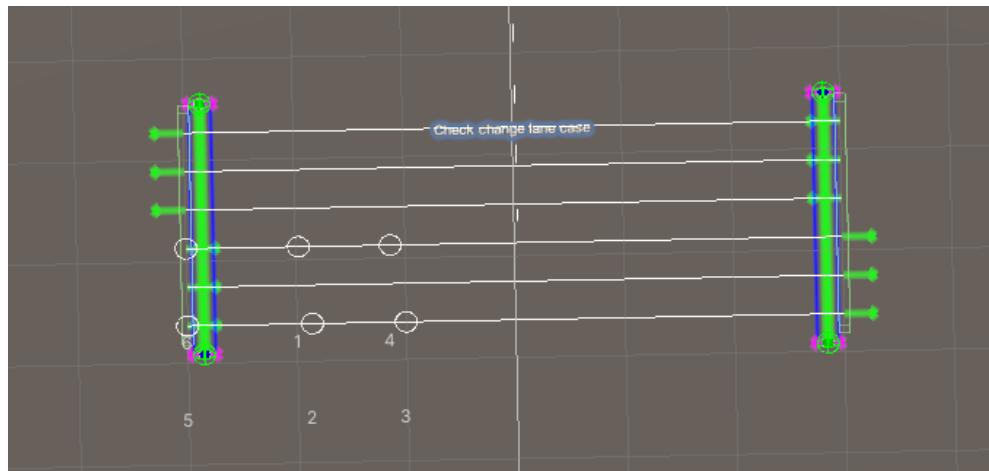


Source settings.



Test case result.

Multiple lanes test 1

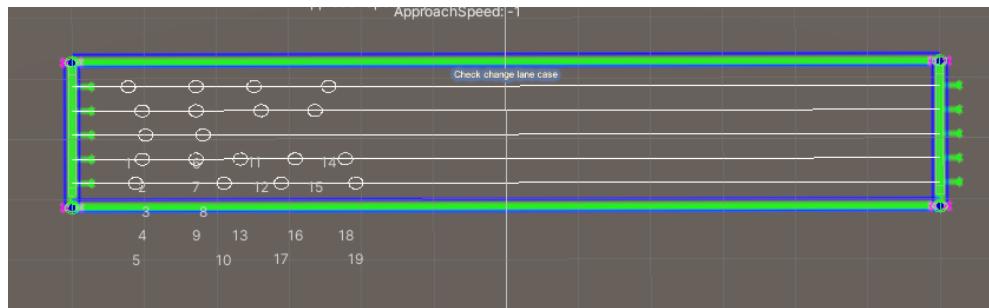


Source settings.

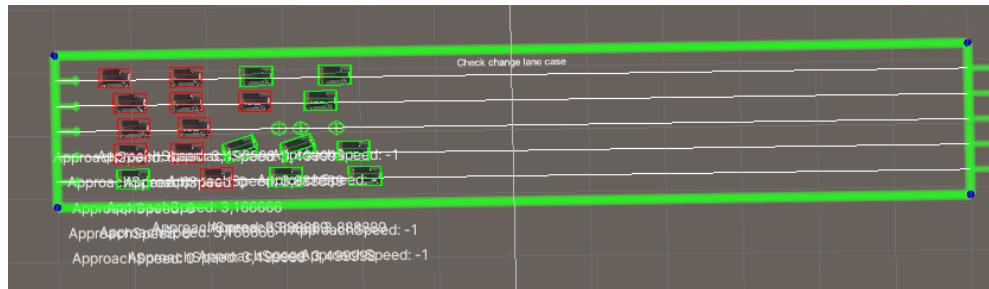


Test case result.

Multiple lanes test 2

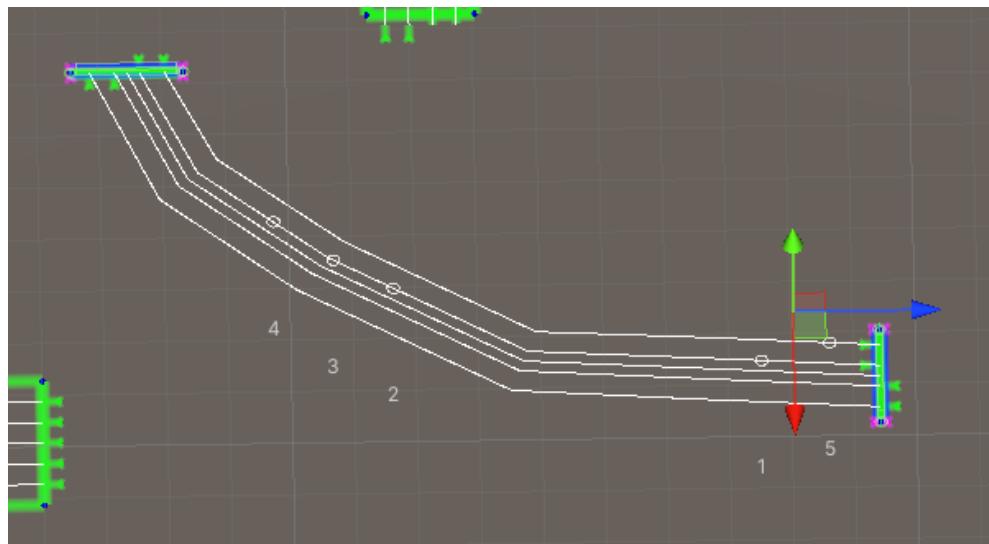


Source settings.

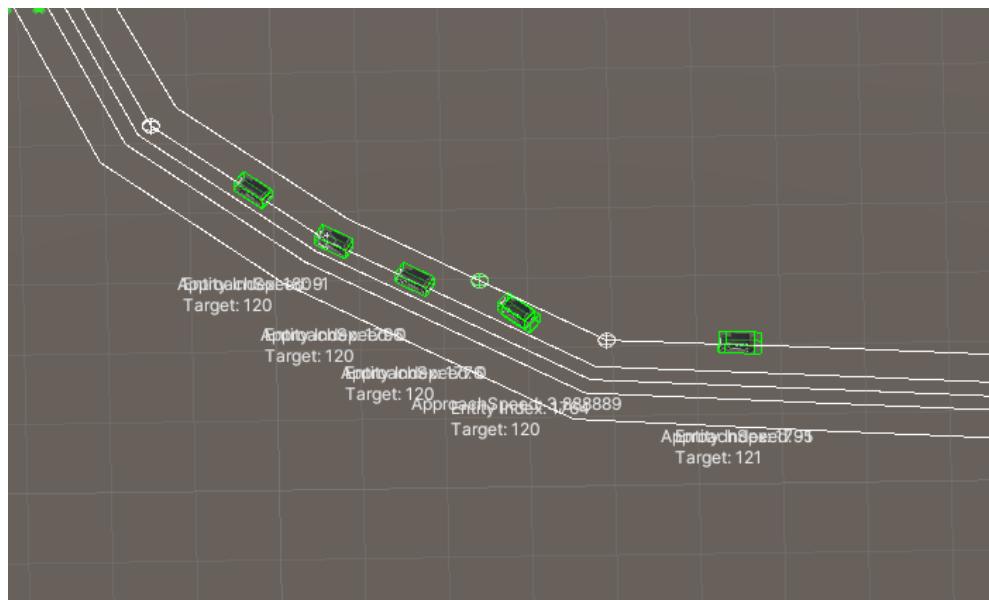


Test case result.

High speed change lane



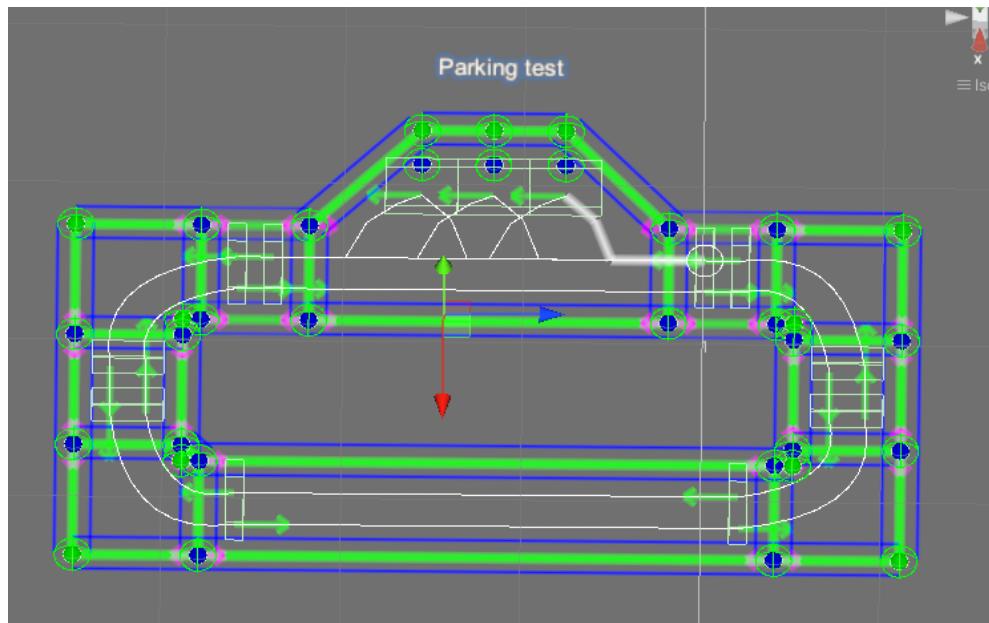
Source settings.



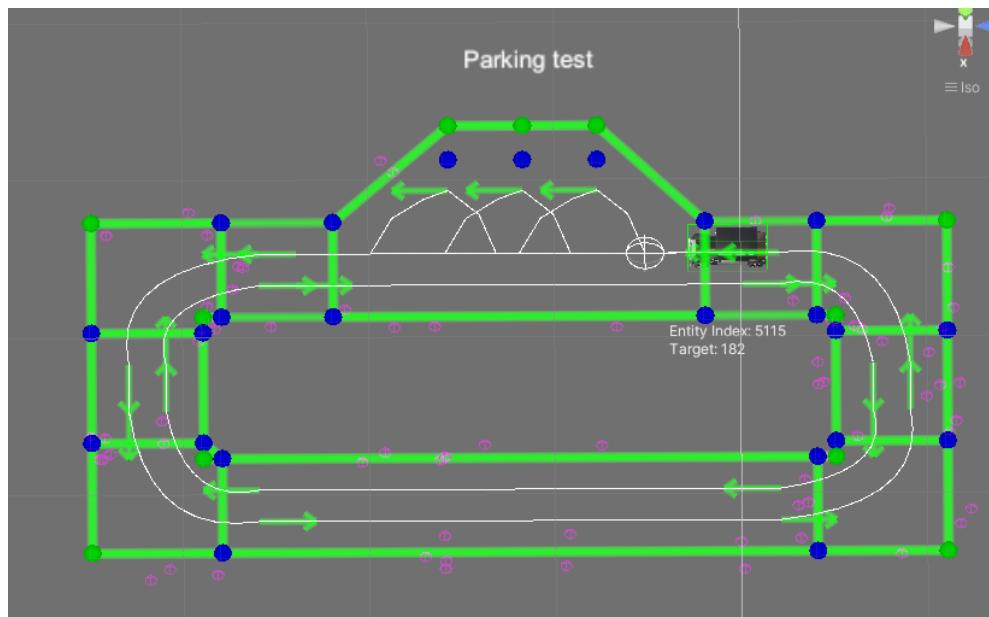
Test case result.

Check Traffic Parking

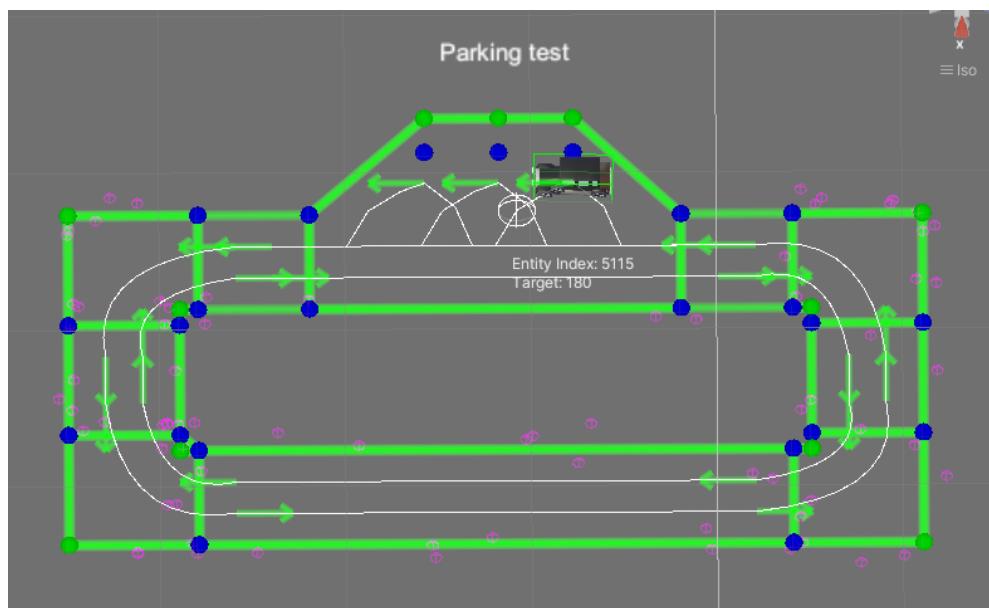
Config *parameters*.



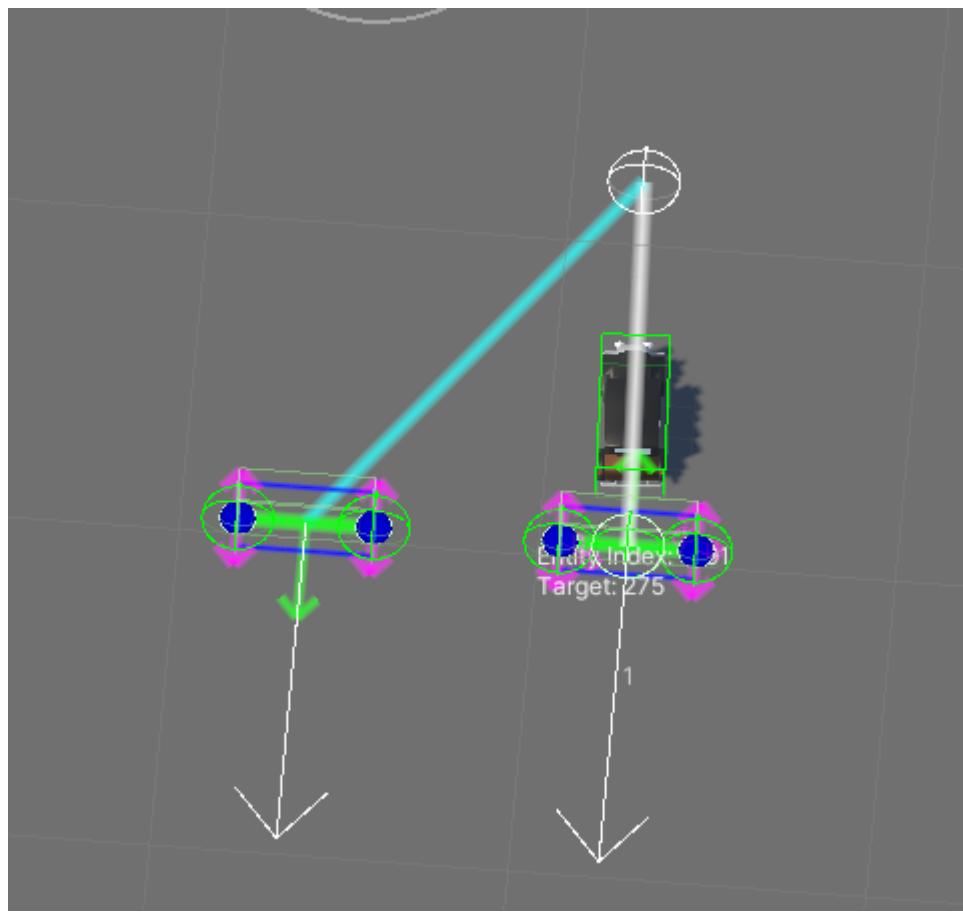
Source settings.



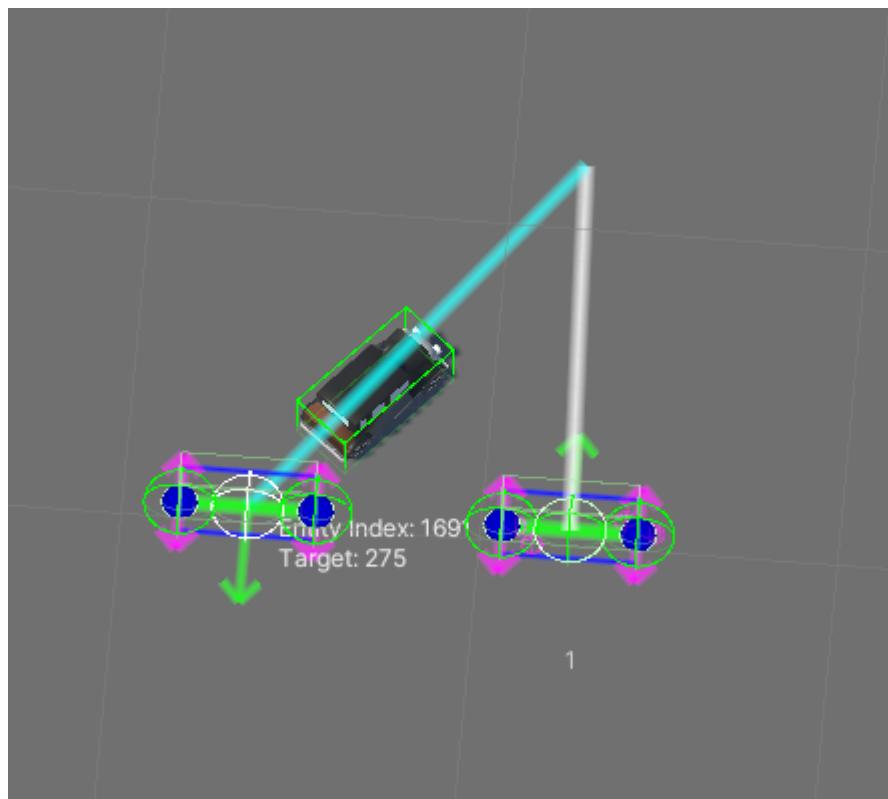
Source car.



Test case result.

Check Traffic Reverse

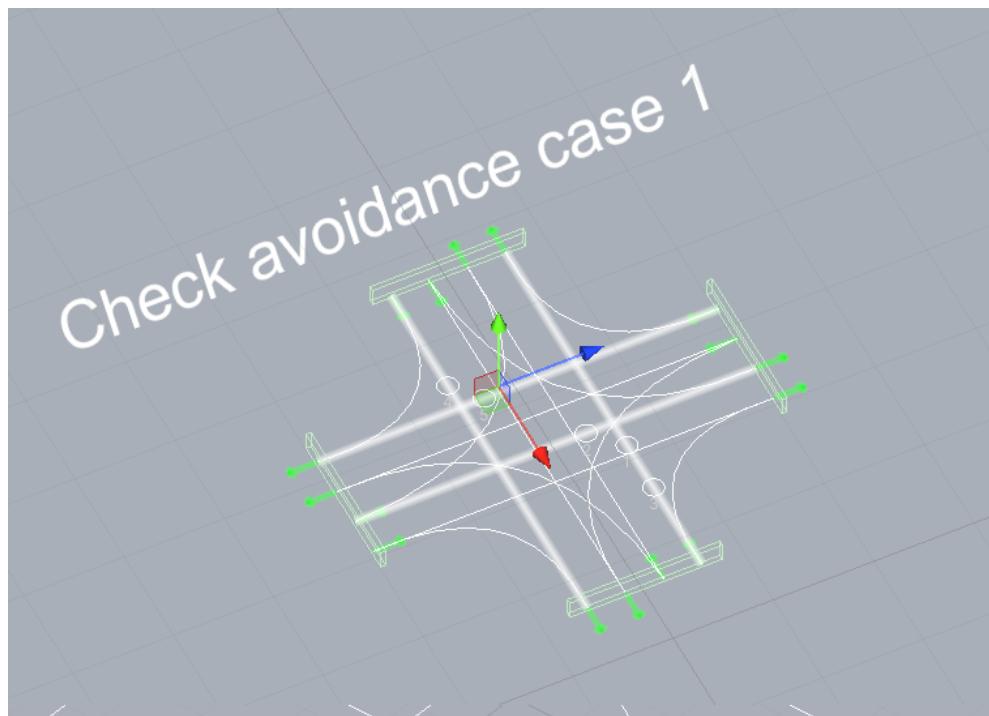
Source.



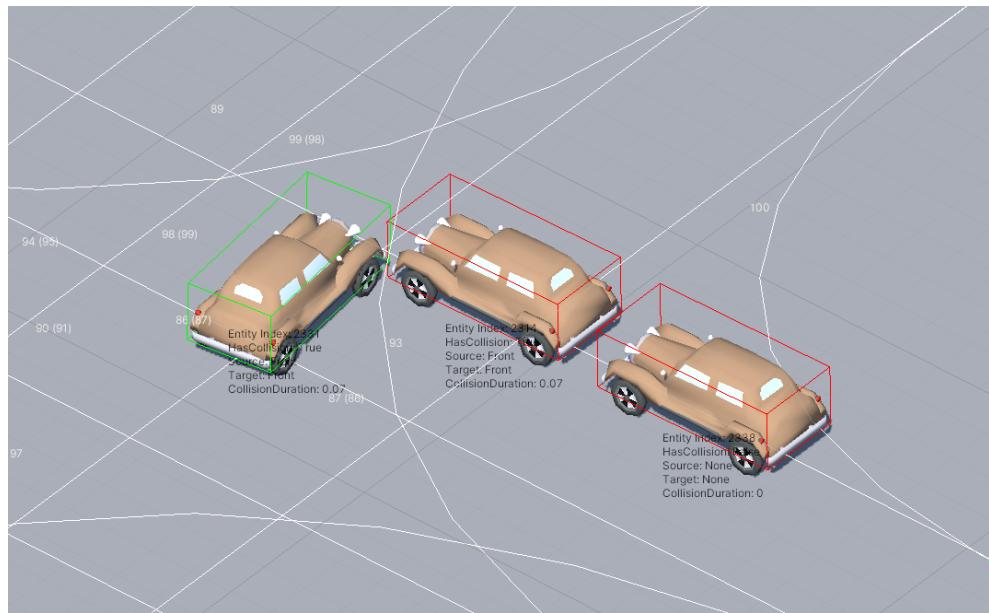
Final result result.

Check Avoidance

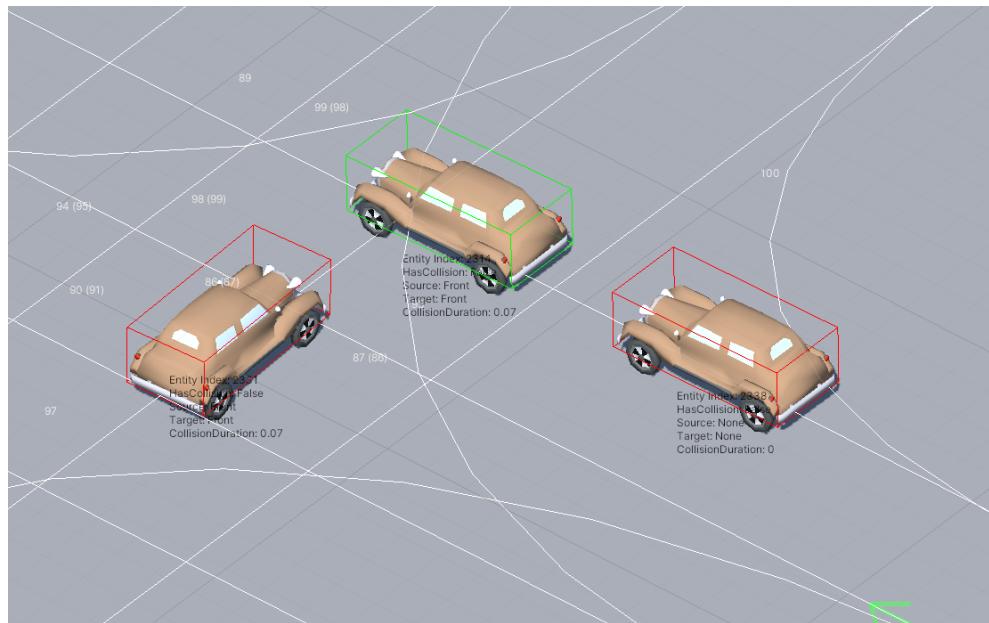
Test case of traffic avoidance config.



Source settings.

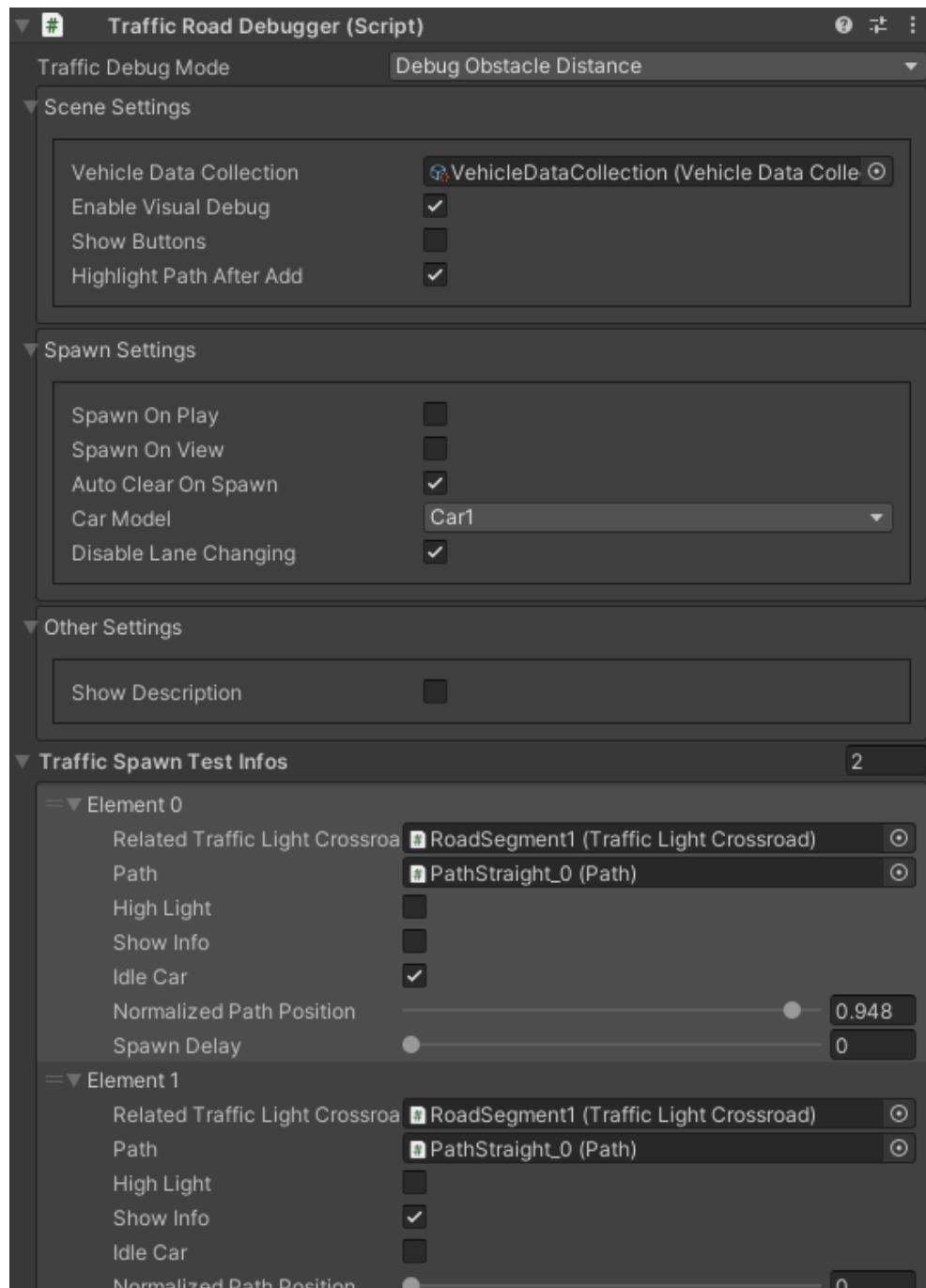


Cyclical obstacle example.



Avoiding cyclical obstacle example.

10.1.3 Traffic Road Debugger



Scene settings

Vehicle data collection : reference to the *collection* of all vehicles.

Enable visual debug : on/off visual debug in the scene.

Show buttons : show add *path* button to the component in the scene.

Highlight path after add : on/off highlight *path* after adding.

Spawn settings

Spawn on play : spawn the cars after the start of the scene.

Auto clear on spawn : previously created cars in the test case will be destroyed on a new spawn.

Spawn car model : *car model* of spawning cars.

Disable lane changing : forcibly disabling the ability for vehicles to change lanes.

Other settings

Show description : show description of test case.

Traffic spawn test entry

Related trafficlight crossroad : linked *trafficlight crossroad*.

Path : linked *path* for the spawned cars.

Highlight : on/off highlight *path*.

Show info : on/off visual info of spawned cars in the scene.

Idle car : on/off vehicle idle of the spawned vehicle.

Normalized path position : min approach speed.

Spawn delay : delayed vehicle spawn after test case spawn has started.

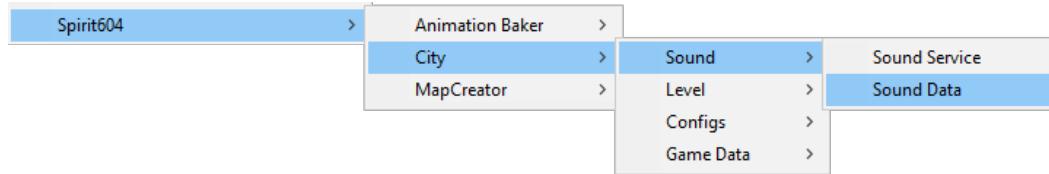
SOUND

11.1 Built-in

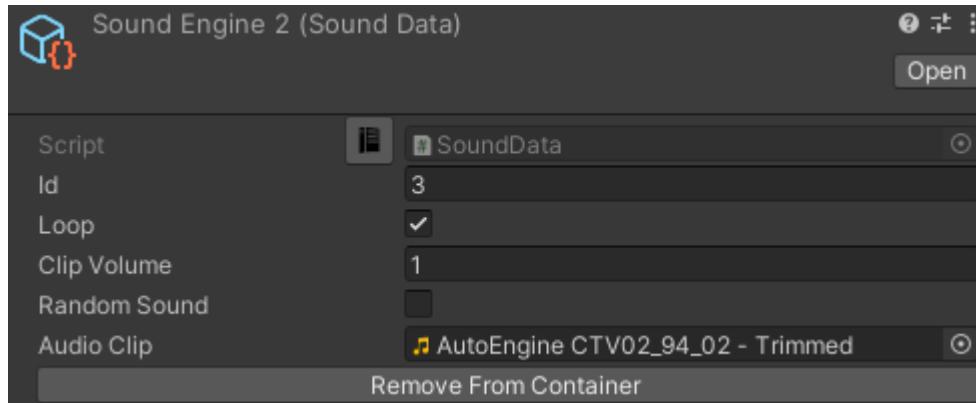
By default is used for all sounds in the project (if FMOD is not installed).

11.1.1 How To Use

1. Create *Sound Data* in the *Unity* project.



2. Assign your desired *AudioClip*.



Note: Sound ID is created automatically.

3. Press *Add To Container* button to add sound to the *Sound container*.



- Now, we can trigger the sound using the code examples below.

11.1.2 Code Examples

MonoBehaviour

```
public class ExampleSoundBehaviour : MonoBehaviour
{
    [SerializeField] private SoundData exampleSoundData;

    private void PlaySound()
    {
        SoundManager.Instance.PlayOneShot(exampleSoundData, transform.
    ↵position);
    }
}
```

Jobs

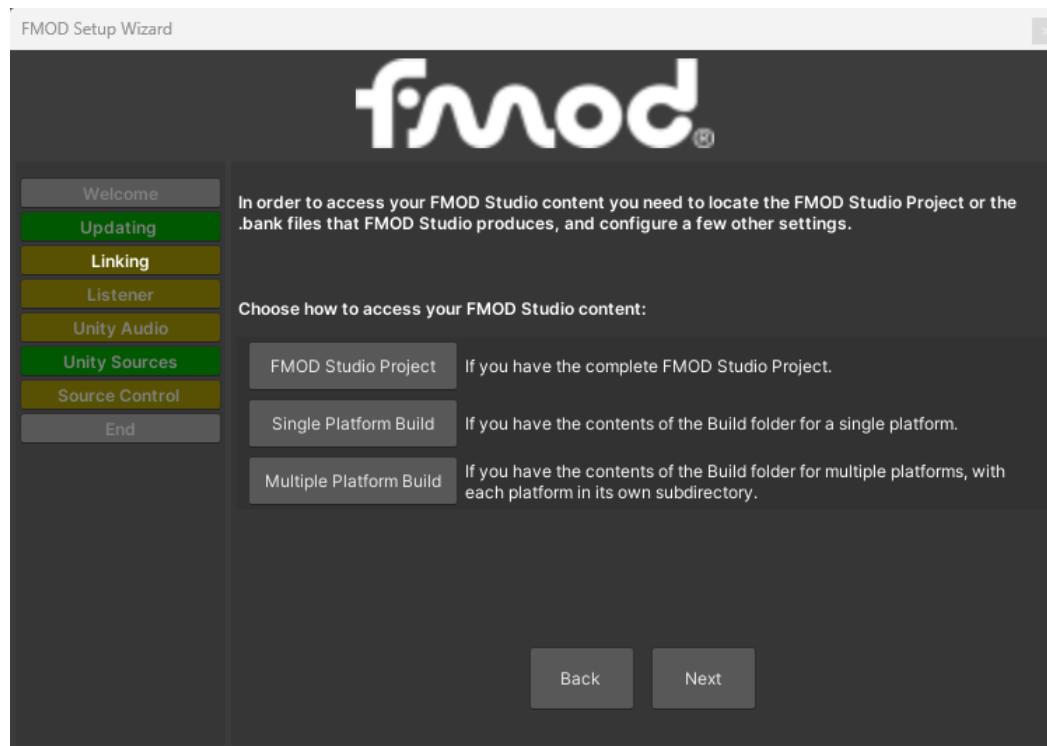
If you want to create sound in Job, *read here*.

11.2 FMOD

FMOD is used for all sounds in the project (if FMOD installed).

11.2.1 Installation

1. Download the FMOD plugin from the *Asset Store*.
2. Add the FMOD scripting define to the project.
3. Select the project FMOD path in the window that appears on the *Linking* tab (by default *DotsCity/FMOD/fmodproject/fmodproject.fspro*).



- Sign up & download the [FMOD Studio software](#) from the official site ([official guide](#)) [optional step, required if you want to add & edit sounds].

FMOD Studio Desktop application for creation of adaptive audio

Filter by major FMOD version number or [Unity Verified](#) status. See [revision history](#).

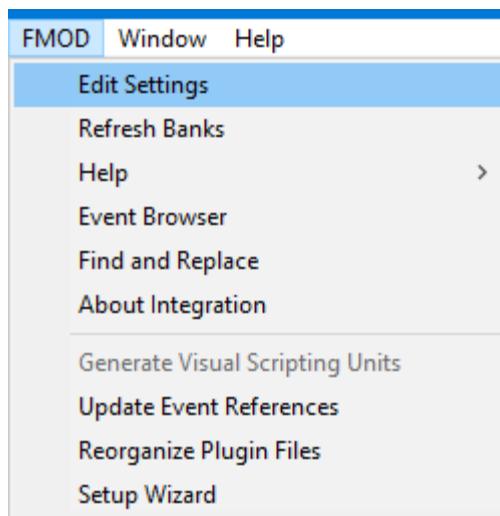
2.02 2.01 Older All [Unity Verified](#)

2.02.11 (Unity Verified)

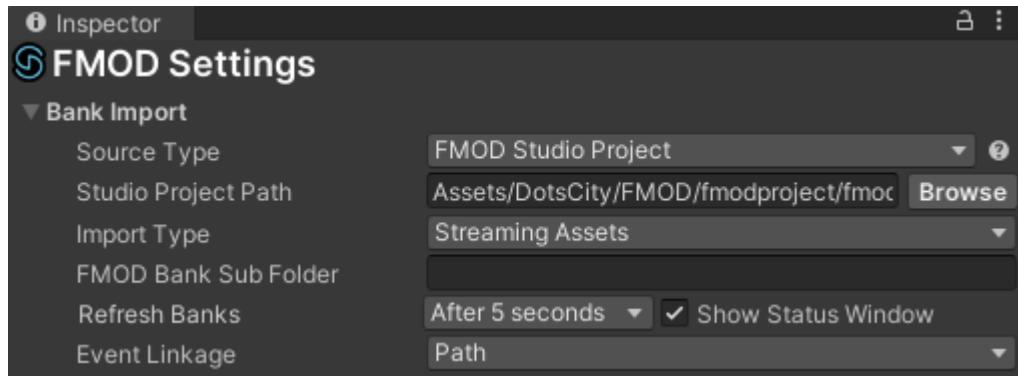
This is a [Unity Verified](#) version.

Windows (64-bit)	Download
Windows (32-bit)	Download
Mac	Download
Linux (.deb)	Download
Linux (AppImage)	Download

- Open *FMOD* settings.

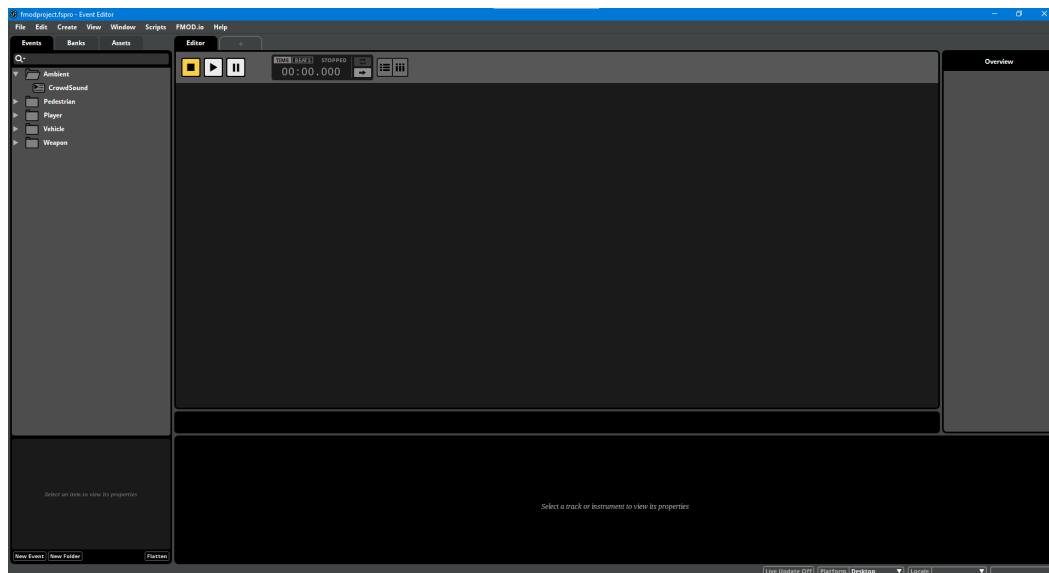


6. By default, project path: *FMOD* project path *DotsCity/FMOD/fmodproject/fmodproject.fspro*, make sure that the *FMOD* project path is set correctly.



11.2.2 How To Use

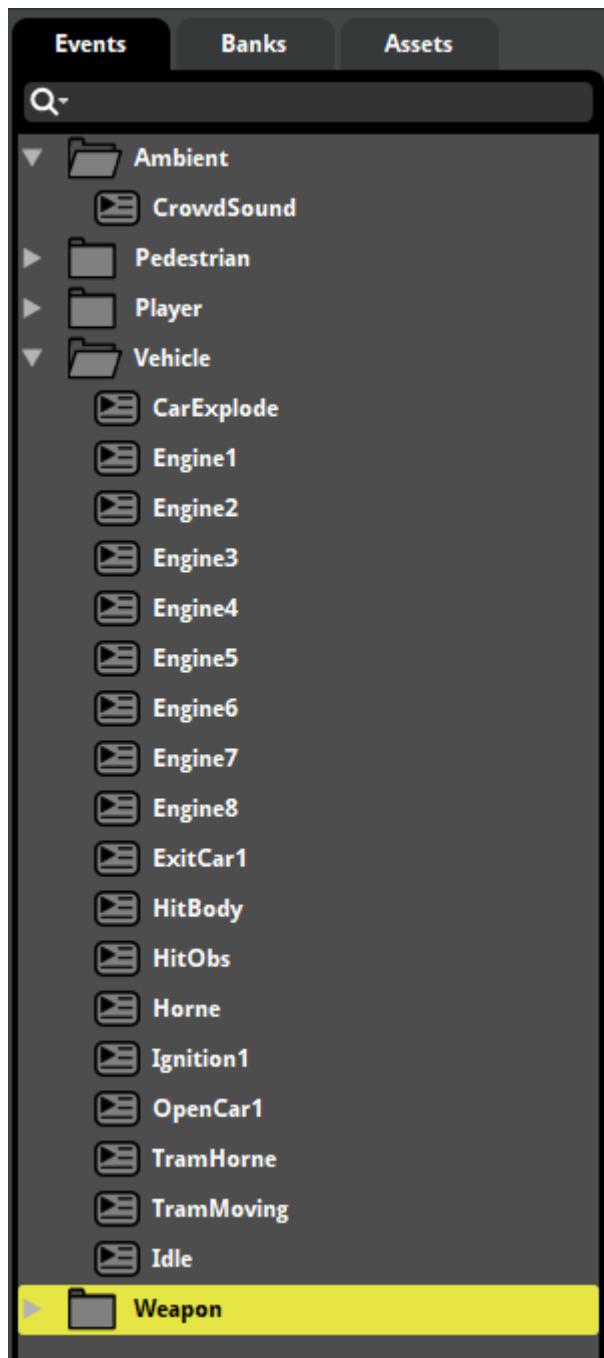
1. Open *FMOD Studio* installed on your computer.



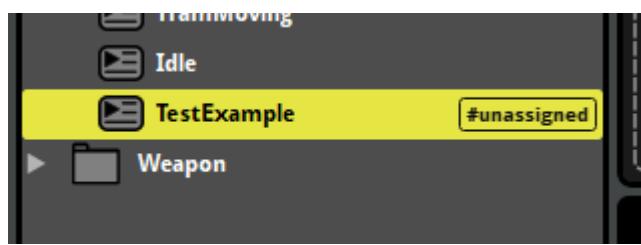
2. Open *Bank* bookmark.



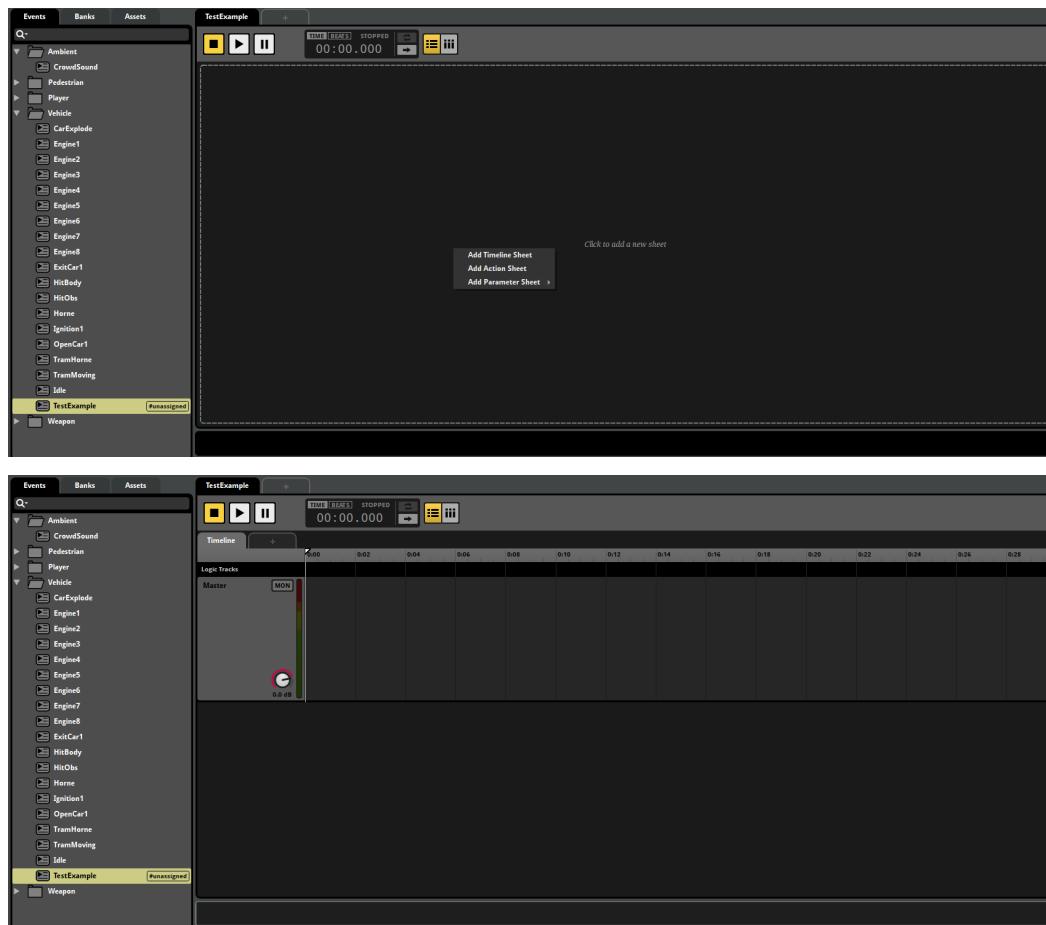
3. If you do not have an existing bank or need to create a new one, right-click in the window and press *New Bank*.
4. In the bookmark *Events* - Create or open exist folder.



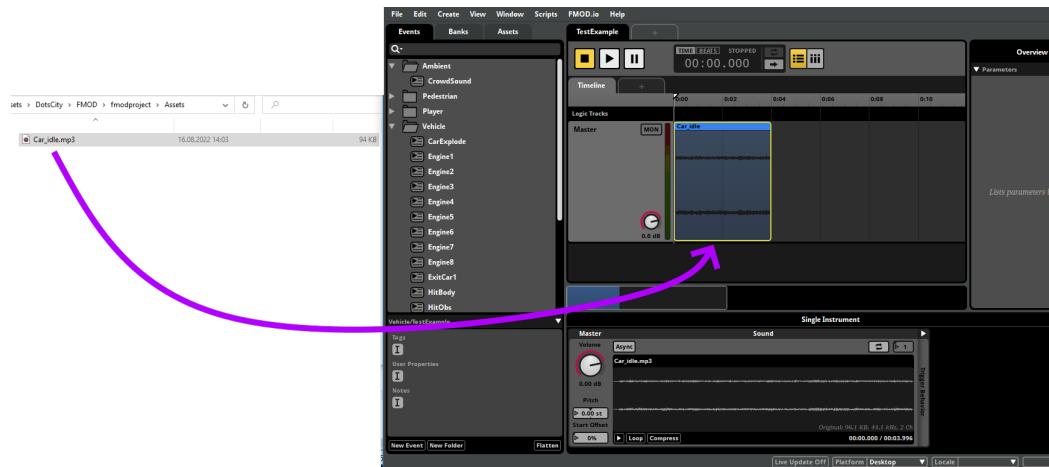
- Right-click on the created folder and press *Create Event*, rename created event.



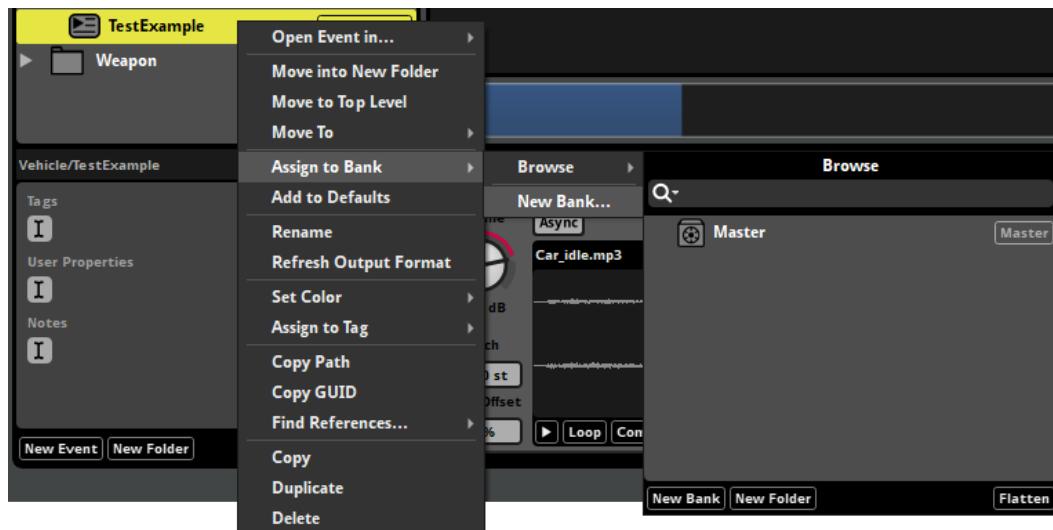
- Right-click on *Add Timeline Sheet* in the created event.



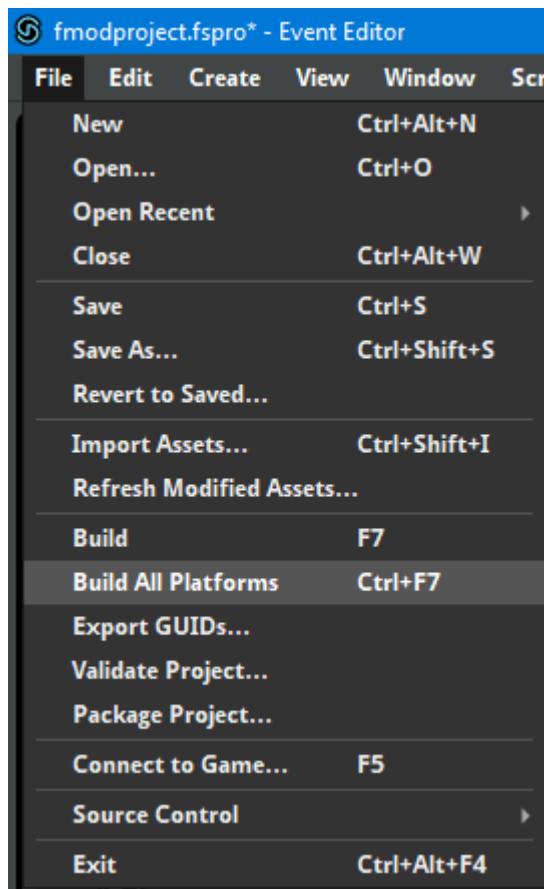
7. Drag and drop the desired sound into the timeline.



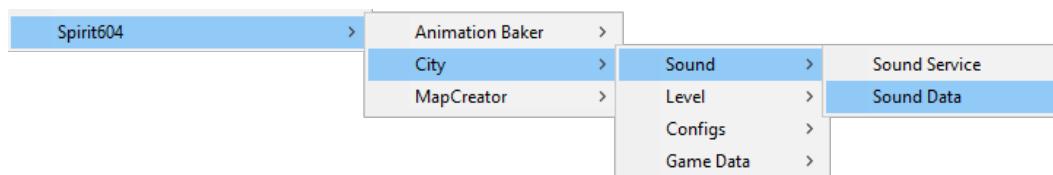
8. Customize your sound.
9. Assign the selected *FMOD event* to the *Bank*.



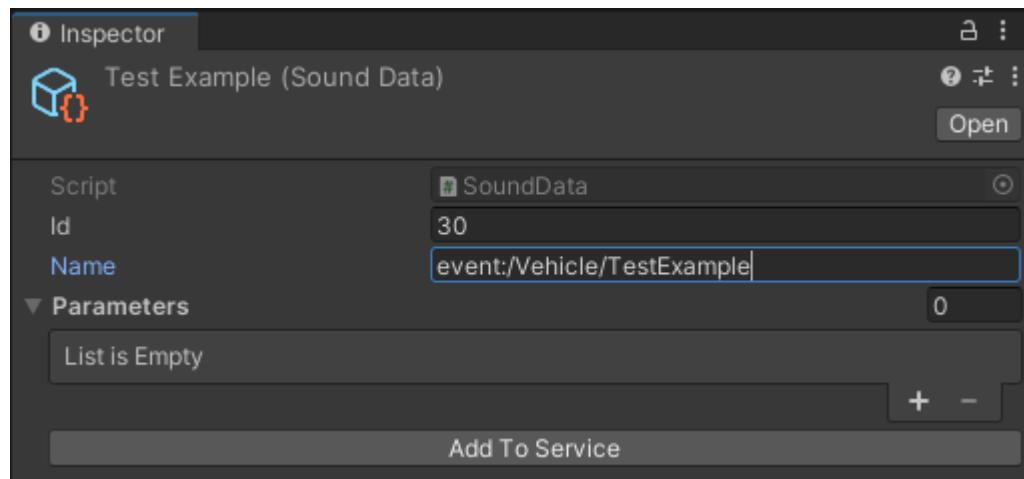
10. Build *FMOD* project.



11. Create *Sound Data* in the *Unity* project.

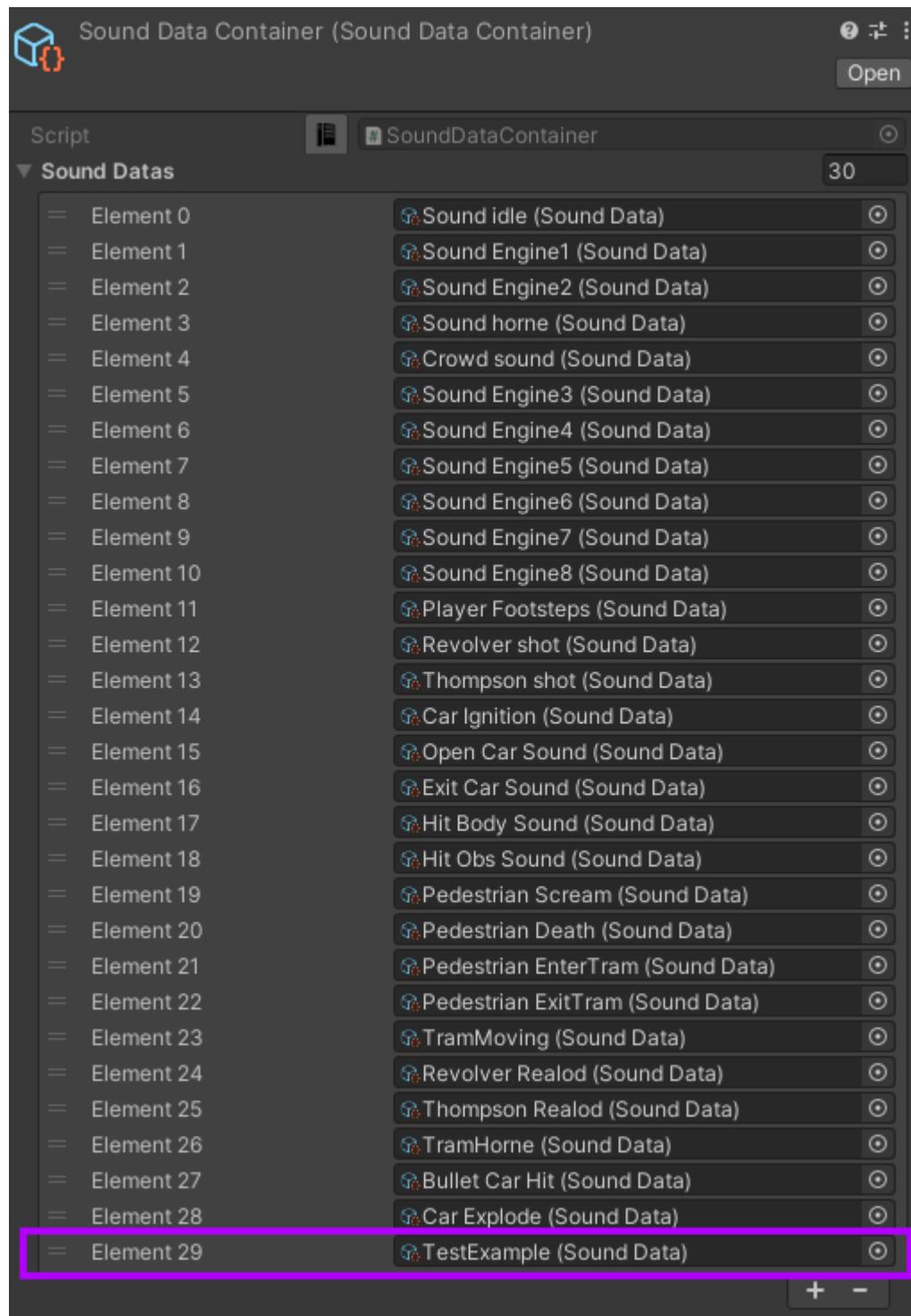


-
12. Enter trigger name *event:/Vehicle/TestExample*.



Note: Sound ID is created automatically.

13. Press *Add To Container* button to add sound to the *Sound container*.



14. Now, we can trigger the sound from the *code*.

11.3 Data

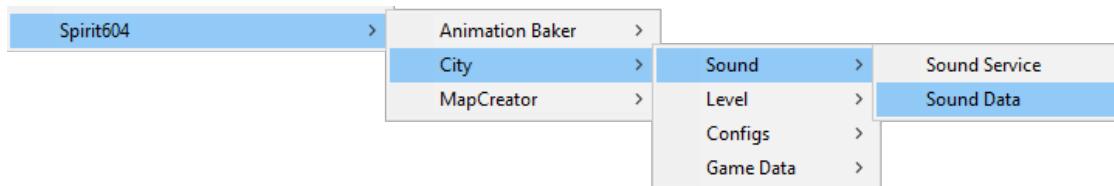
11.3.1 Sound Data

Contains data about the sound.

How To Create

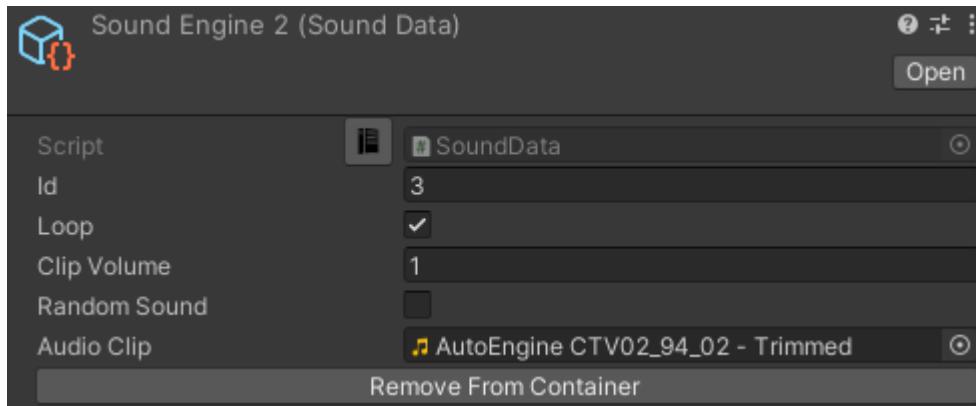
Select from the project context menu:

Spirit604/City/Sound/Sound Data



Settings

Built-in



Id : immutable ID, by which sounds are triggered in *DOTS traffic city* (ID is created automatically).

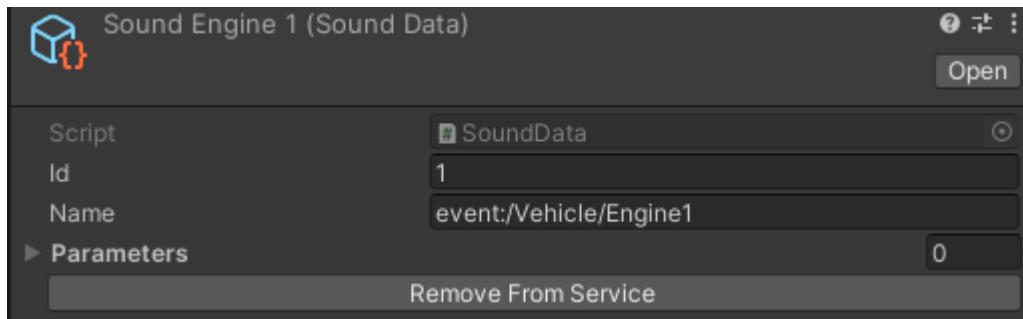
Loop : on/off sound looping.

Clip volume : volume of the audio clip.

Random sound : on/off random sound.

Audio clip : reference to `AudioClip` .

FMOD



Id : immutable ID, by which sounds are triggered in *DOTS traffic city* (ID is created automatically).

Name : [event name](#) of the sound.

Parameters : [event parameters](#) .

11.3.2 Sound Data Container

Contains data on all *sounds* in the *Unity* project.



Warning: If you do not add *sound* to the service, it cannot be activated from the code.

11.4 Code Examples

11.4.1 Sound Types

- **Default** : default sound entity.
- **One Shot** : entity played once & destroyed afterwards.
- **Tracking** : entity tracks target entity.
- **Tracking Vehicle** : entity tracks target vehicle entity.
- **Tracking And Loop** : entity tracks target entity & loop playback.

11.4.2 How To Create

EntityManager methods

```
SoundExtension.CreateSoundEntity(ref this EntityManager entityManager, int soundId,
    float volume = 1f)
// Creating a default sound entity.
```

```
SoundExtension.CreateTrackedSoundEntity(ref this EntityManager entityManager, int soundId,
    Entity parentEntity, float volume = 1f)
// Creation of a sound entity that follows a given entity.
```

```
SoundExtension.CreateChildSoundEntity(ref this EntityManager entityManager, int soundId,
    Entity parentEntity, float volume = 1f)
// Creation of a sound entity that will be a child of a given entity.
```

CommandBuffer methods

Burst compatible methods.

```
SoundExtension.CreateSoundEntity(ref this EntityCommandBuffer commandBuffer, Entity soundPrefabEntity,
    int soundId, float volume = 1f)
// Creating a default sound entity.
```

```
SoundExtension.CreateSoundEntity(ref this EntityCommandBuffer commandBuffer, Entity soundPrefabEntity,
    int soundId, float3 position, float volume = 1f)
// Create a sound entity at a specific position.
```

Create prefab query method

```
SoundExtension.GetSoundQuery(EntityManager entityManager, SoundType soundType)
// Get `EntityQuery` with the selected `Sound type`.
```

Create sound example

```
public partial class ExampleSoundSystem : SystemBase
{
    private EntityQuery soundPrefabQuery;

    protected override void OnCreate()
    {
        soundPrefabQuery = SoundExtension.GetSoundQuery(EntityManager, SoundType.
Default);
    }

    protected override void OnUpdate()
    {
        var commandBuffer = ...
        var soundPrefabEntity = soundPrefabQuery.GetSingletonEntity();

        // Pass 'commandBuffer' & 'soundPrefabEntity' into the IJobEntity or
Entities.ForEach
        // commandBuffer.CreateSoundEntity(soundPrefabEntity, soundId,
position);
        // 'soundId' can be taken from 'SoundData'
    }
}
```

Params

- soundId : id of sound taken from *sound data*.
- soundPrefabEntity : sound *prefab entity* taken from *EntityQuery*.
- position : initial position of the sound.
- volume : volume of the sound (0..1 range).

11.4.3 How To Play

```
public partial class PlayAndStopSoundExampleSystem : SystemBase
{
protected override void OnUpdate()
{

// Get world sounds
var sounds = GetComponentLookup<SoundEventComponent>(false);

Entities
```

(continues on next page)

(continued from previous page)

```
.WithBurst()
.ForEach(
    Entity entity
    in SoundHolder soundHolder) =>
{
    // Some play condition
    bool shouldPlay = true;

    // Some sound entity container component
    Entity soundEntity = soundHolder.Entity

    SoundEventComponent soundEvent = sounds[soundEntity];

    if (shouldPlay)
    {
        soundEvent.SetEvent(SoundEventType.Play);
    }
    else
    {
        soundEvent.SetEvent(SoundEventType.StopFadeout);
    }

    sounds[soundEntity] = soundEvent;

}).Schedule();
}
}
```

11.4.4 How To Destroy

Enable the *PooledEventTag* component in the *sound* entity.

11.4.5 How To Loop

1. Create a *Sound entity*.
2. Add a *LoopSoundData* component (assign a *Duration* value).

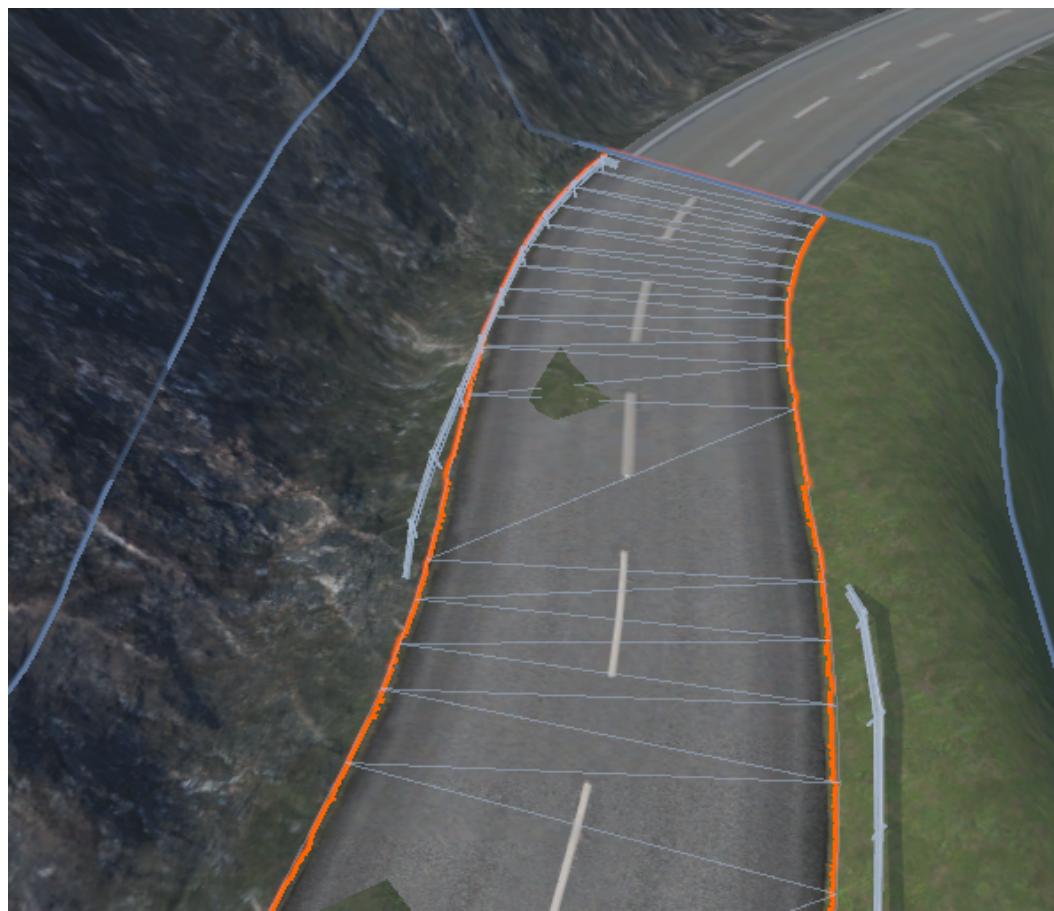
12.1 EasyRoads3D

12.1.1 How To Use

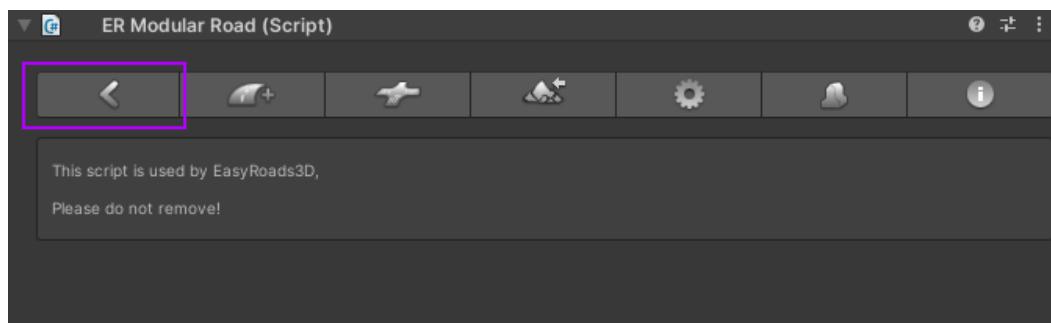
1. Buy & download, import the following asset plugin:

[EasyRoads3D](#)

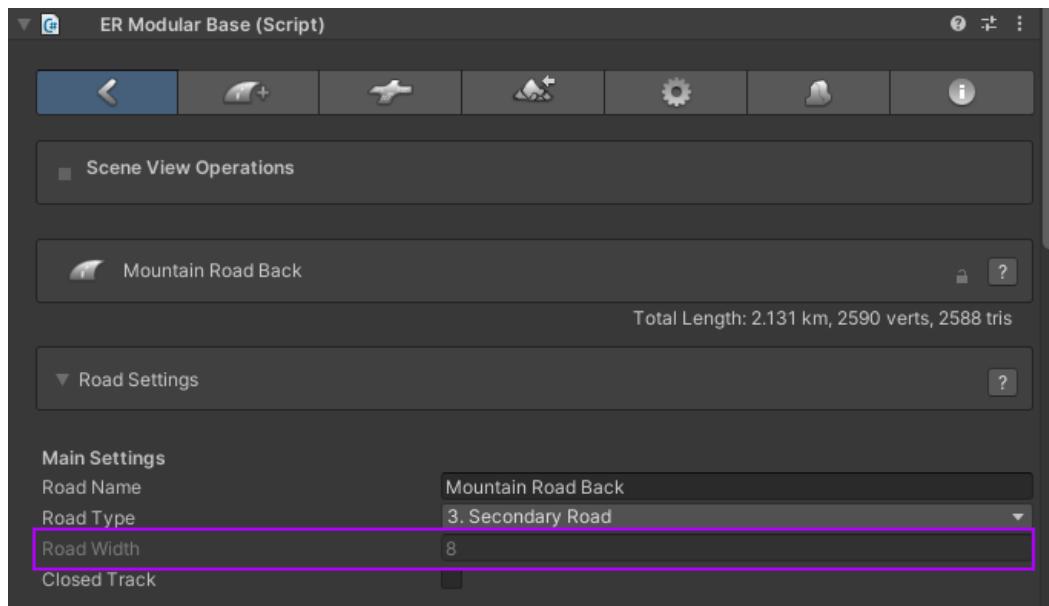
2. Open your scene.
3. Create a new game object & add *ER3D_RSGenerator* component.
4. In the *Config* field select default config or duplicate exist sample config.
5. Set *Min road length* to the desired value to ignore short roads.
6. Select any 2-lane road in the scene.



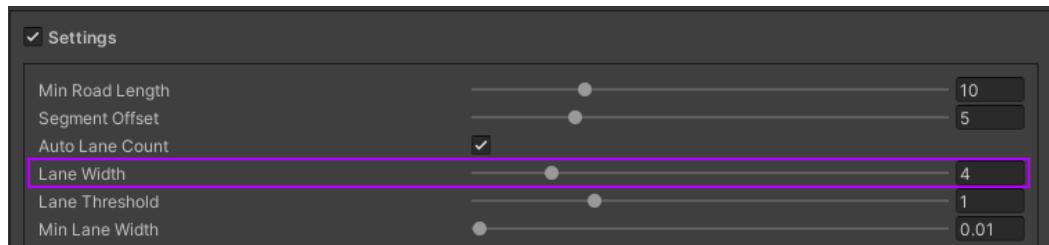
7. Press *left-arrow* button in the inspector.



8. Check *Road width* & set lane width in the *ER3D_RSGenerator* accordingly.



- For example, 8 road width & 2 lanes => lane width is 4.

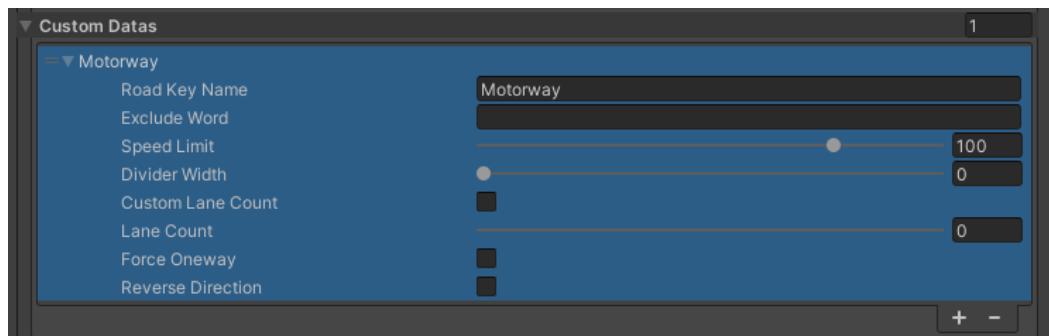


10. Set *Ignore road names* to ignore roads that aren't intended for traffic. [optional step]

11. Set *Speed limits* for crossings. [optional step]

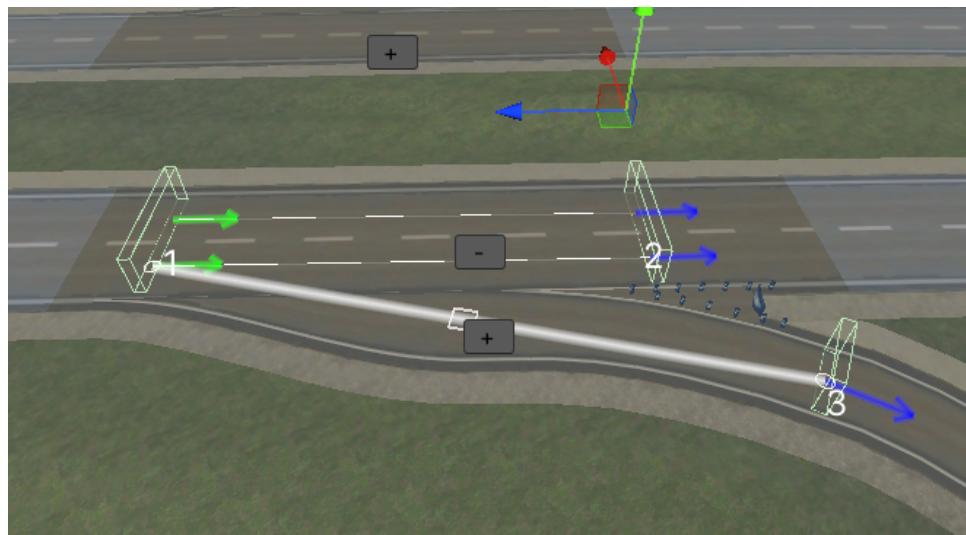


12. Add *Custom Data* if you want to override the settings for certain straight roads with specific text. [optional step]



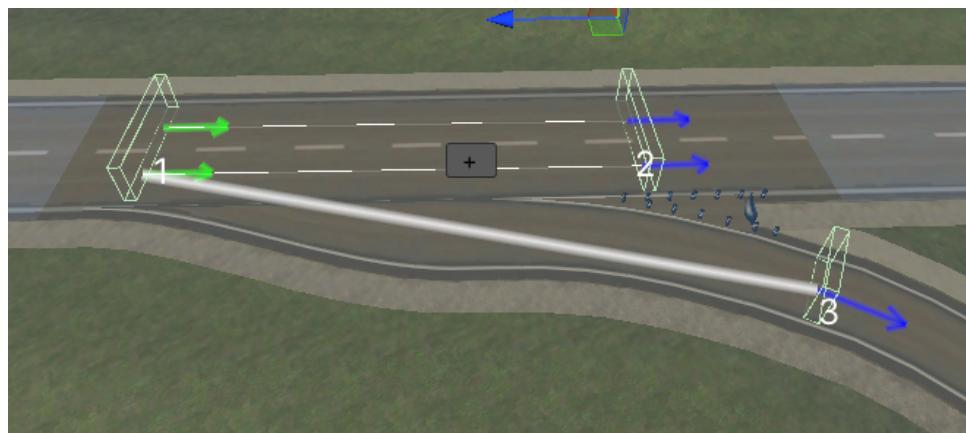
13. If your scene have custom road prefabs with *ER Crossing Prefabs* component:

- Create the *Road segment* where your custom prefab road is located.



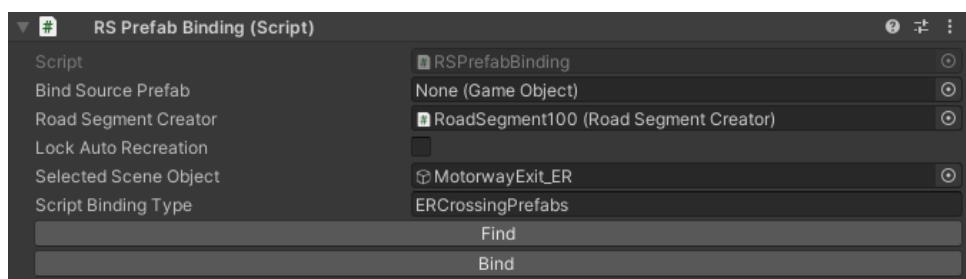
Example

2. Add the *RSPrefabBinding* component to the created *Road segment*.
3. In the *RSPrefabBinding* component, enter *ERCrossingPrefabs* text into the *Script binding type* field [one-time step].
4. Press the *Find* button.
5. Press + button on scene.

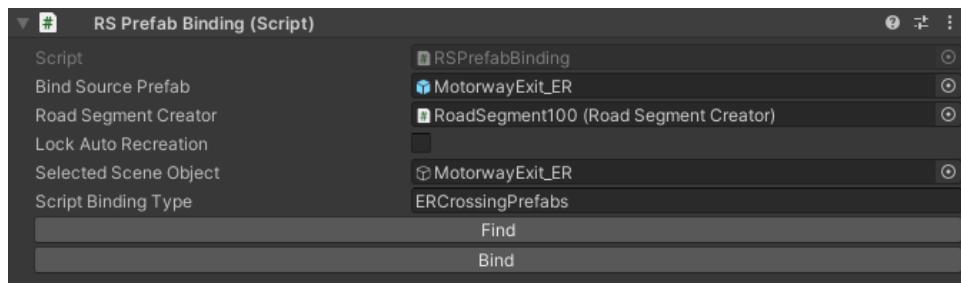


Example

6. Make sure the *Selected scene object* field have the correct scene road.



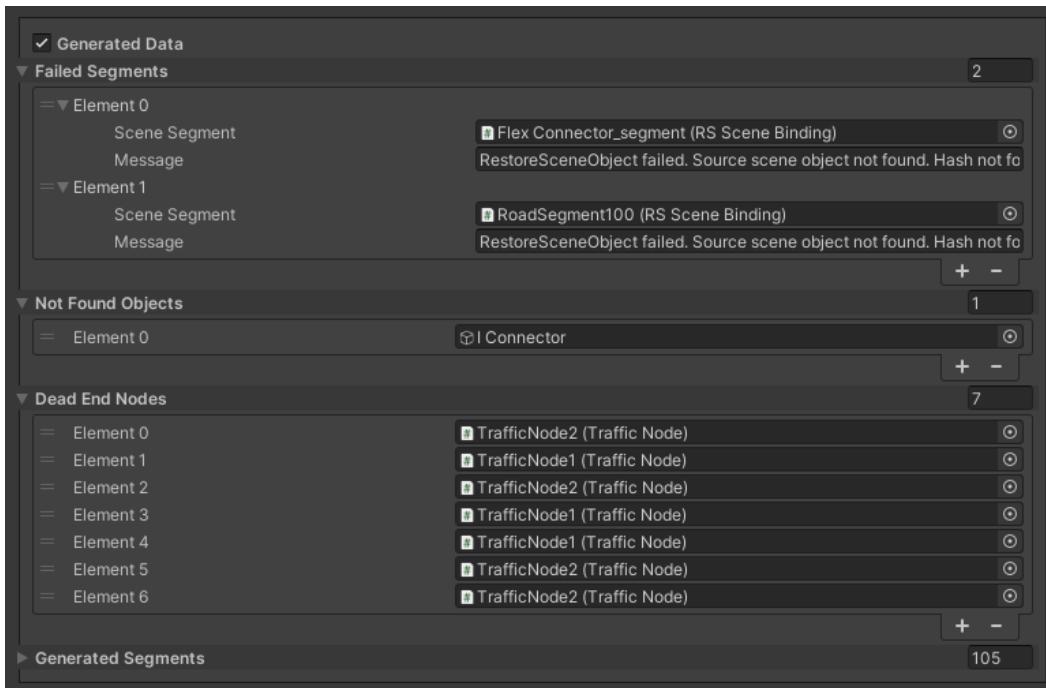
7. Press the *Bind* button.



8. Save created *Road segment* to prefabs.
9. Add created prefab to the *ER3D_RSGenerator* into the *Custom prefabs* field.



14. Press the *Generate* button.
15. Open the *Generated data* tab.



16. Check the generated data for errors & dead ends.
17. If you want to customize road & exclude the desired road from being deleted for the new generation:
 1. Select the desired *Road segment* on the scene.
 2. In the *RSSceneBinding* component, tick on the *Lock auto recreation* option.

3. If you now regenerate roads with *ER3D_RSGenerator*, this road will be saved for destruction.

12.2 Integration Cities

Integration packages allow you to quickly integrate your favourite cities into the **DOTS** stack. Buy, download and integrate cities in a few clicks in less than a minute.

Integration packages contain the following items:

- Routes for traffic are fully configured.
- Routes for pedestrians are fully configured.
- Automatic configuration of vehicles and their integration into the city.
- Automatic configuration of pedestrian and their integration into the city.

- *Polygon City*
 - *How To Use*
- *Toon City*
 - *How To Use*
- *City Pack 1*
 - *How To Use*

12.2.1 Polygon City

Synty asset store



Template example.

How To Use

1. Buy & download, import the following asset package:
[Synty Polygon City pack](#)
2. Open the demo scene: *PolygonCity/Scenes/Demo*.
3. Unpack the *DotsCity/Integrations/PolygonCity/PolygonCity_Integration* package.
4. Open the *PolygonCity/DotsTemplate/PolygonCity_Integration* asset.
5. Click *Integrate* button.
6. Launch the scene.
7. Enjoy.

12.2.2 Toon City

SICS Games asset store





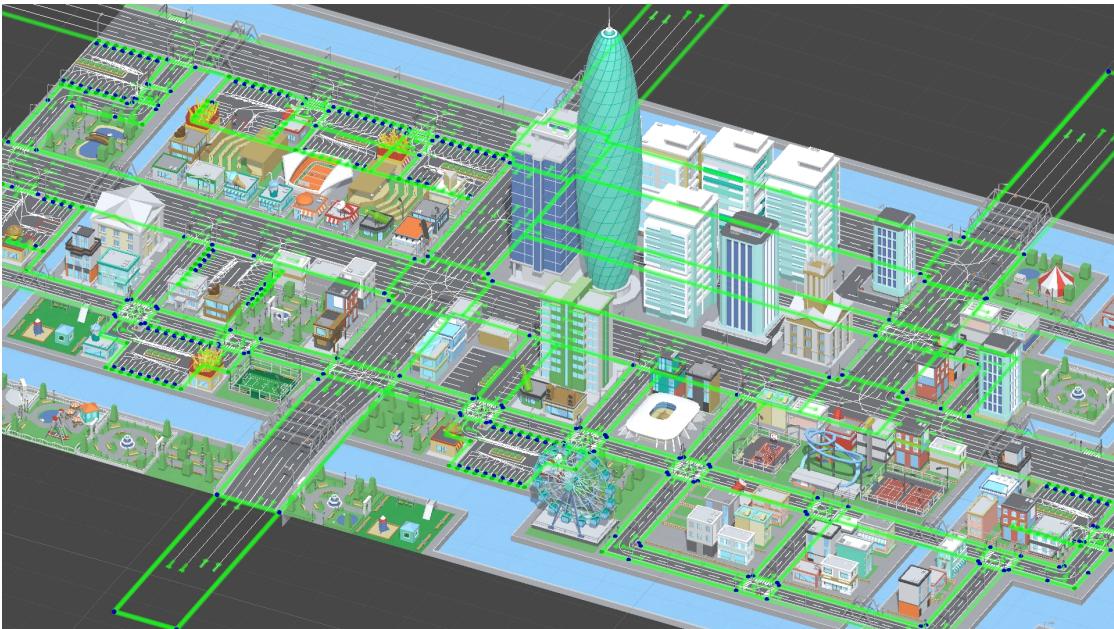
Template example.

How To Use

1. Buy & download, import the following asset package:
[SICS Games Toon City pack](#)
2. Unpack URP package *Toon City/URP_Install_Toon City_202**.
3. Open the demo scene: *Toon City/Scenes/Demo_Scene_1*.
4. Unpack the *DotsCity/Integrations/Toon City/Toon City_Integration* package.
5. Open the *Toon City/DotsTemplate/Toon City_Integration* asset.
6. Click the *Integrate* button.
7. Launch the scene.
8. Enjoy.

12.2.3 City Pack 1

[ithappy asset store](#)



Template example.

How To Use

1. Buy & download, import the following asset package:
[ithappy City pack 1](#)
2. Open the demo scene: *City_Pack_1/Scenes/City_Demonstration*.
3. Unpack the *DotsCity/Integrations/City_Pack_1/City_Pack_1_Integration* package.
4. Open the *City_Pack_1/DotsTemplate/City_Pack_1_Integration* asset.
5. Click the *Integrate* button.
6. Launch the scene.
7. Enjoy.

DEBUG

Youtube tutorial.

13.1 Traffic Debug

- *How To Open*
- *Traffic Spawn Button Helper*
- *Traffic Debugger*
 - *Debugger Type*
 - * *Default*
 - * *Target*
 - * *Approach speed*
 - * *State*
 - * *Target index*
 - * *Path index*
 - * *Speed limit*
 - * *Change lane*
 - * *Collision*
 - * *Obstacle*
 - * *No target*
 - *Settings*
- *Traffic Raycast Debugger*
 - *Example*
- *Traffic NpcObstacle Debugger*
 - *Example*
- *Traffic Public Debugger*
 - *Example*
- *Traffic Light Debugger*

13.1.1 How To Open

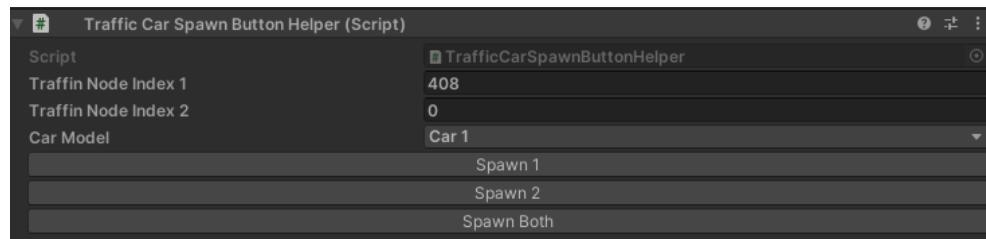
Youtube tutorial.

On the scene select:

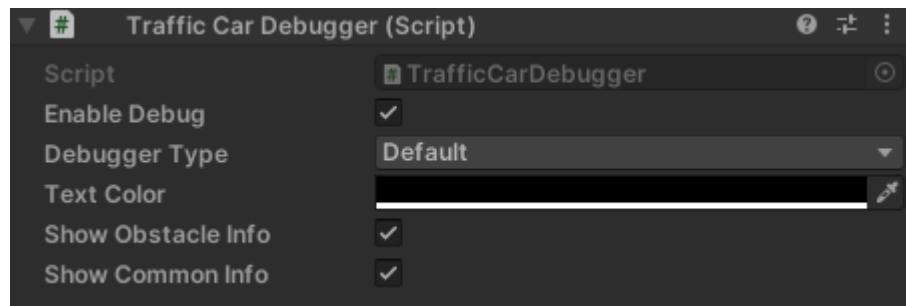
CityDebugger/TrafficDebugger/

13.1.2 Traffic Spawn Button Helper

For manual spawn of a car by the *selected index*.



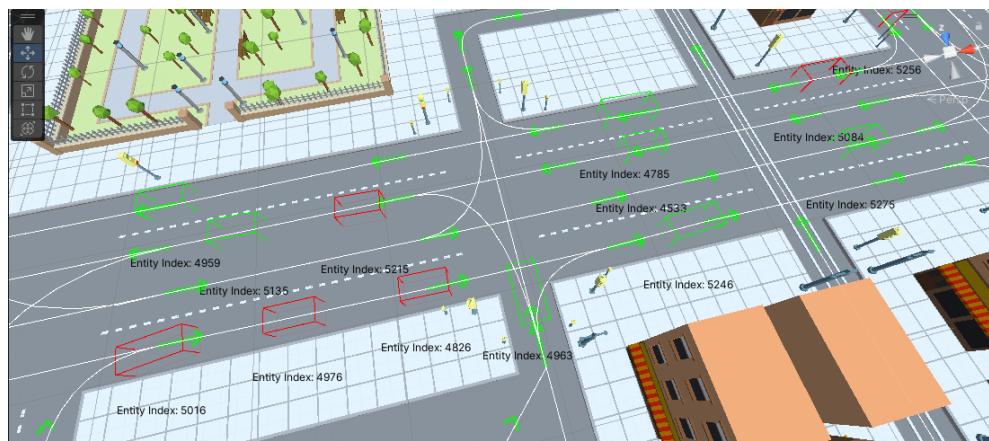
13.1.3 Traffic Debugger



Enable debug : on/off debugger.

Debugger Type

Default



Target

Shows the target position of the vehicle.

Approach speed

Shows the vehicle's approach speed.

State

Shows the state of the vehicle.

Target index

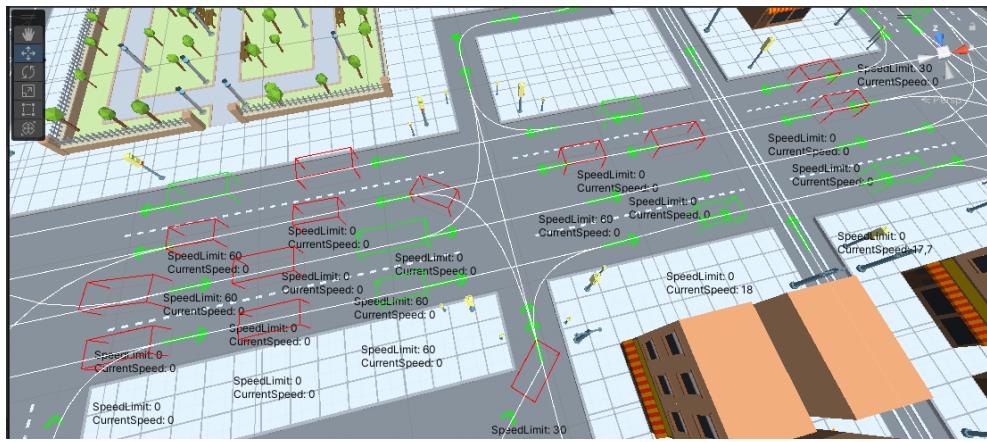
Shows the target indexes of the vehicle.

Path index

Shows the global path index of the vehicle.

Speed limit

Shows the current speed and the speed limit of the vehicle.



Change lane

Shows the change lane point on the lane in the scene.

Collision

Shows the collision direction of the vehicle.

Obstacle

Shows the obstacle entity & obstacle reason type of the vehicle.

Obstacle reason type:

- **Undefined**
- **DefaultPath** : default obstacle in the current path or next connected path of the vehicle.
- **NeighborPath** : obstacle on neighbouring paths, that starts from the current path.
- **JamCase_1** : the vehicle stays at the entrance of the crossroad & doesn't enter to avoid traffic jams.
- **FewChangeLaneCars** : obstacle when the current vehicle and the obstacle vehicle change lanes at the same time.
- **ChangingLane** : the obstacle vehicle changes lanes to the lane of the current vehicle.
- **Intersect_1_TargetCarCloseToIntersectPoint** : target obstacle vehicle too close to intersection of two paths (both vehicles are close, but the target vehicle is closer).
- **Intersect_2_TargetCarCloseToIntersectPoint** : target obstacle vehicle too close to intersection of two paths (only target vehicle is too close).
- **Intersect_3_OtherHasPriority** : vehicles meeting at an intersection of two paths have different priorities, with the higher priority vehicle passing first (unless the vehicle is too close to the intersection).
- **Intersect_4_SamePriority** : vehicles that meet at an intersection of two paths have the same priority, whichever vehicle is closer to the intersection that passes first.

No target

Shows the list of vehicles without a destination.

Settings

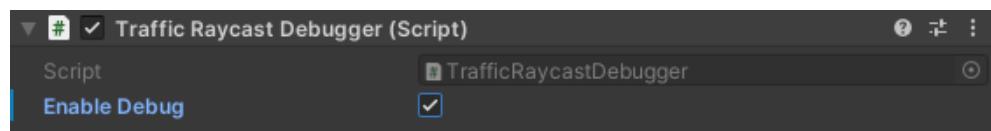
Text color : colour of scene text UI.

Show obstacle info : display obstacles for vehicles (red color vehicle has obstacle).

Show common info : show the entity index of the vehicles.

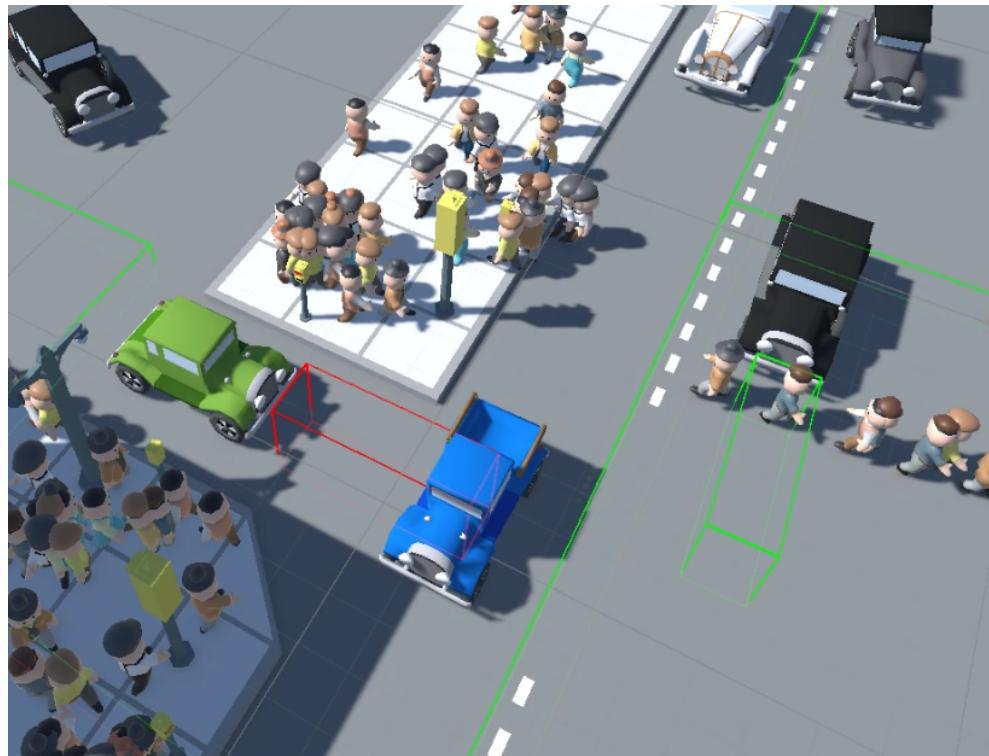
13.1.4 Traffic Raycast Debugger

Shows the raycast box of the car. (*Config*) (*info*)



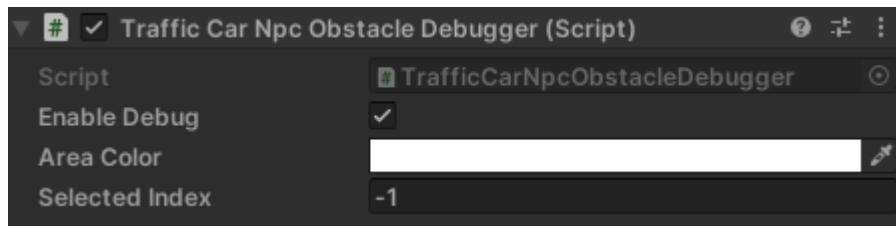
Enable debug : on/off debugger.

Example



13.1.5 Traffic NpcObstacle Debugger

Shows the calculation area and the vehicle's obstacle NPCs.

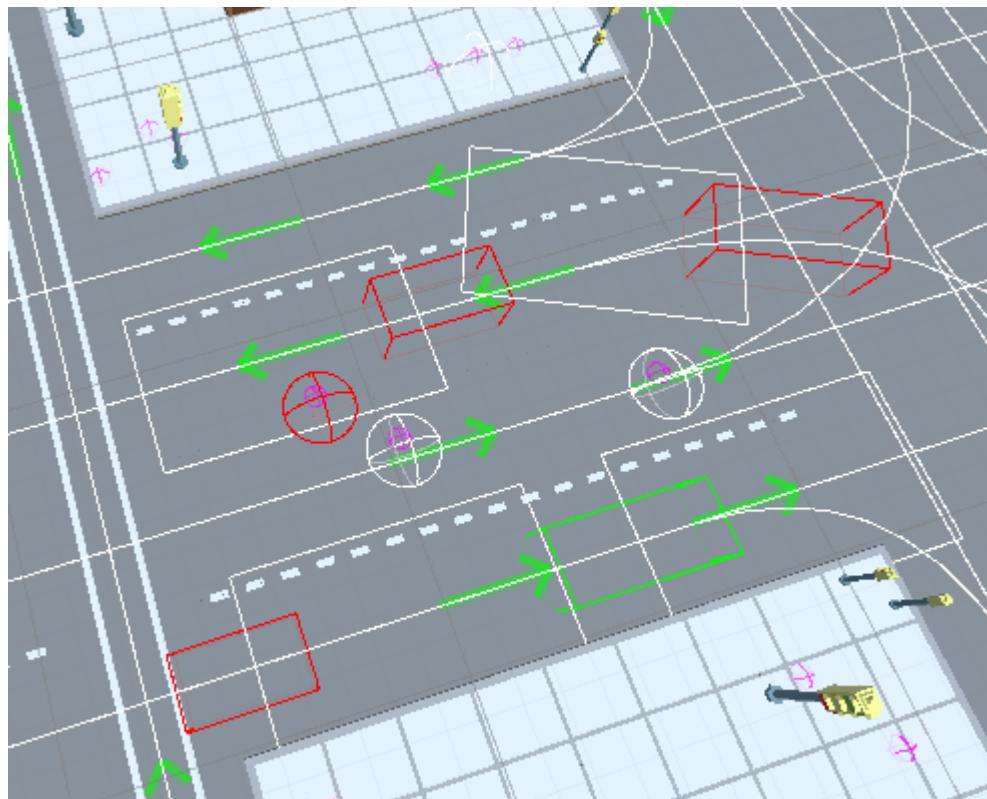


Enable debug : on/off debugger.

Area color : colour of the area where the vehicle calculates the npc obstacles.

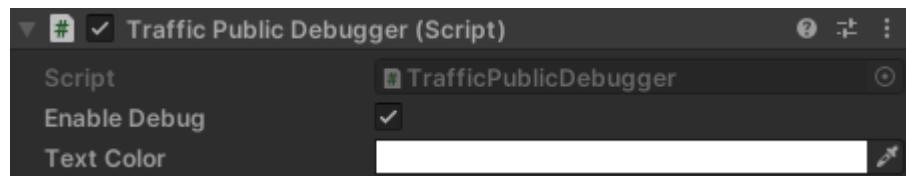
Selected index : only for this entity index will debug be enabled (-1 all entities).

Example



13.1.6 Traffic Public Debugger

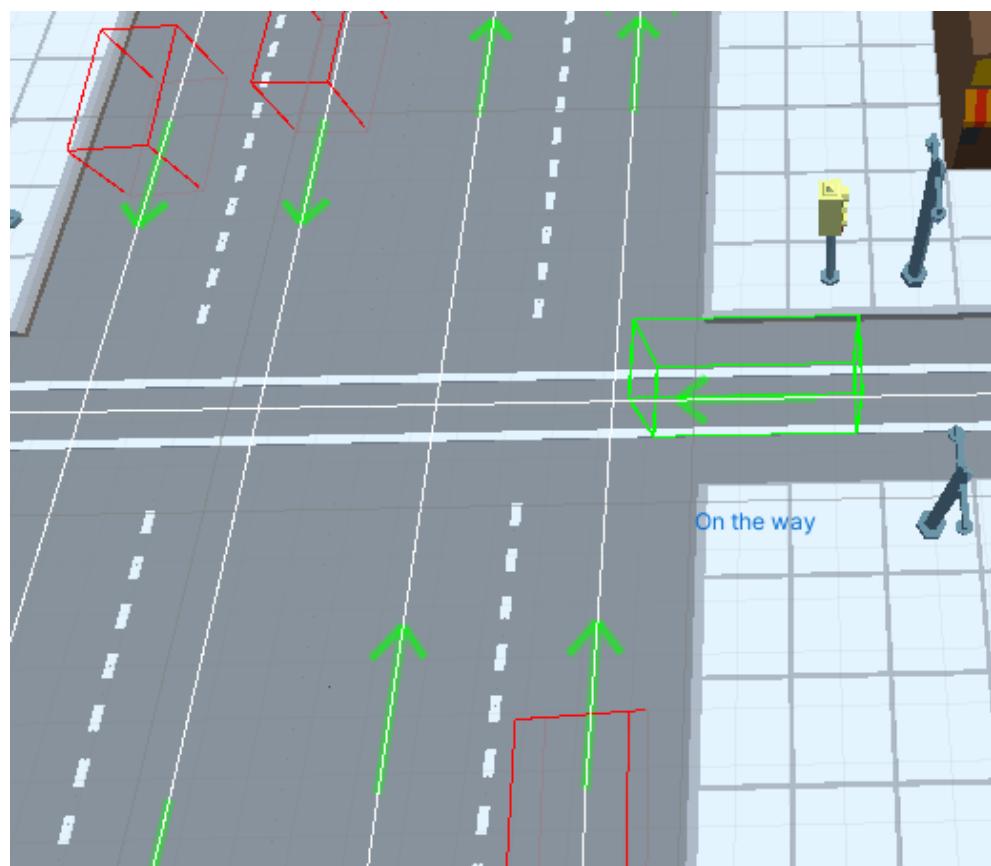
Shows *public transport traffic* data.



Enable debug : on/off debugger.

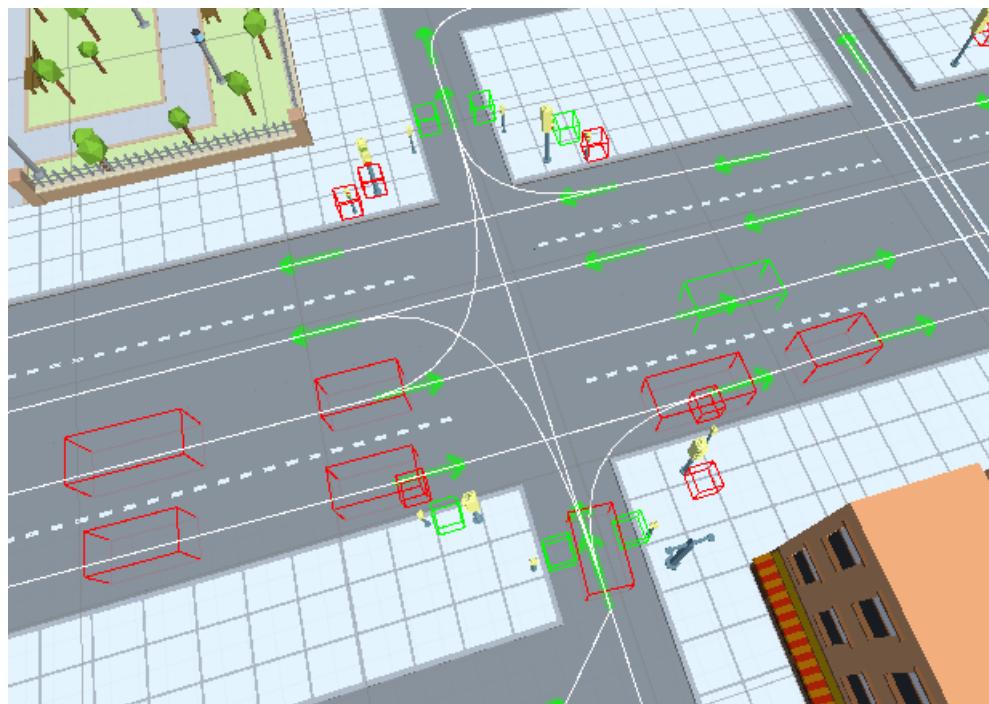
Text color : colour of scene text UI.

Example



13.1.7 Traffic Light Debugger

Shows the *state* of traffic light objects.



13.2 Traffic Node Debug

- *Traffic Node Data Viewer*
 - *How To Open*
 - *How To Filter*
 - *Settings*
- *Traffic Node Debug*
 - *Index Debug*
 - *Available Debug*
 - *Light State Debug*
 - *Capacity Debug*

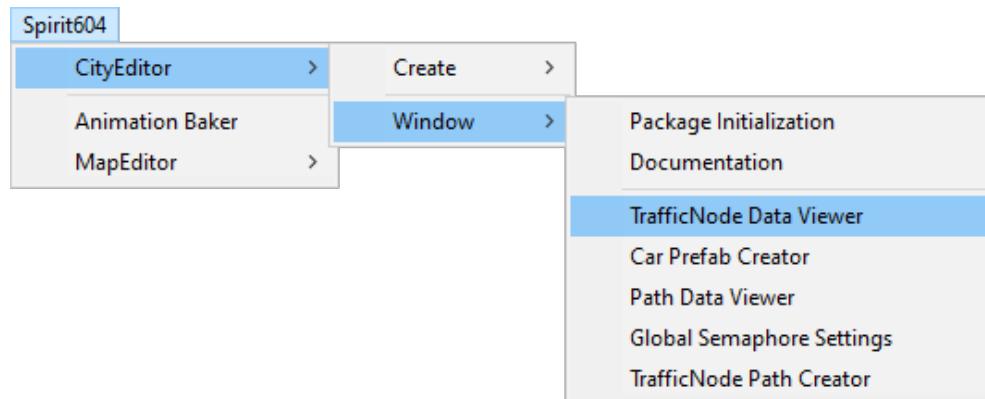
13.2.1 Traffic Node Data Viewer

Tool for finding *traffic nodes* with filtering of selected parameters that differ from prefab.

How To Open

In the *Unity* toolbar:

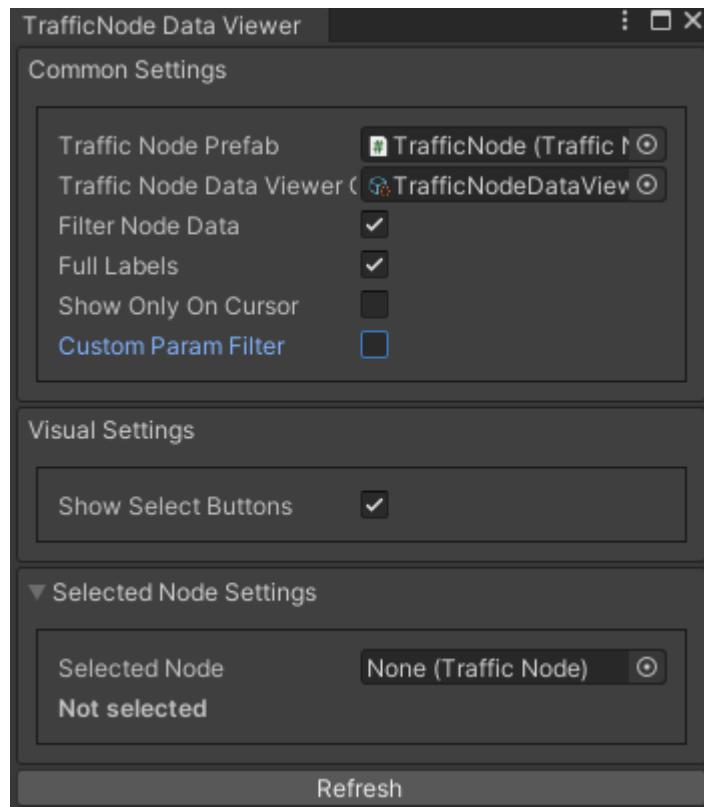
Spirit604/CityEditor/Window/TrafficNode Data Viewer



How To Filter

1. Enable *Filter node data* option.
2. If you need to filter by a selected parameter enable *Custom param filter* and add desired parameters.

Settings



Traffic node prefab : *traffic node* source prefab for comparison.

Traffic node data viewer : internal filter settings.

Filter node data : on/off filtering

Full labels : show full setting labels.

Show only on cursor : show node traffic data only when the cursor is hovering.

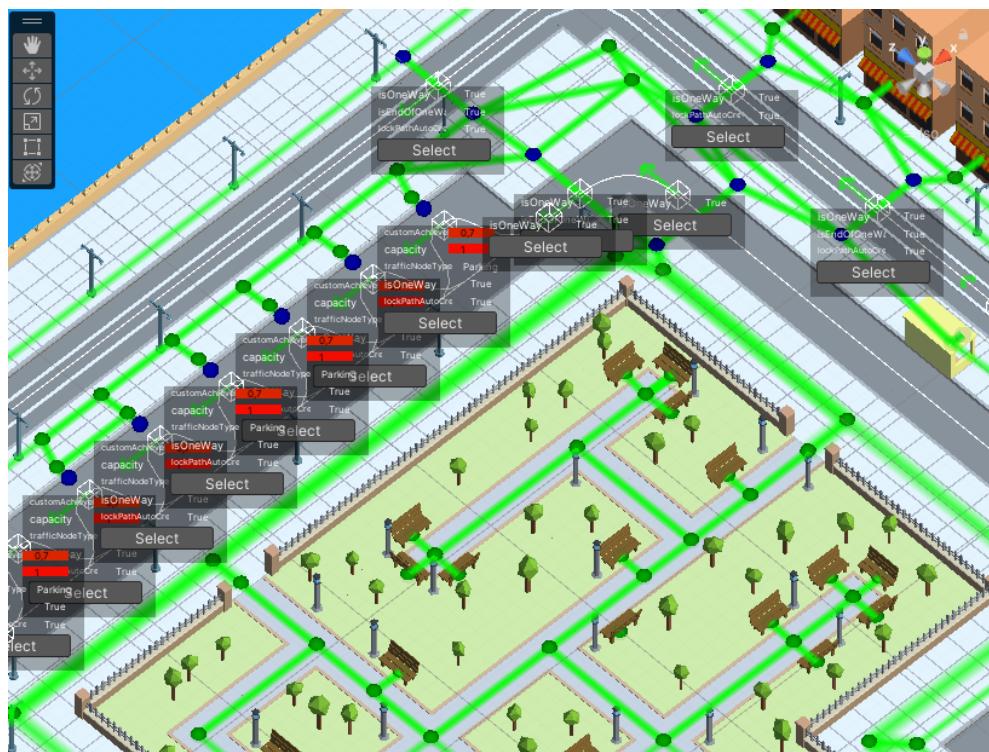
Custom param filter : node filtering by selected parameters.

Show select buttons : on/off select buttons in the scene.

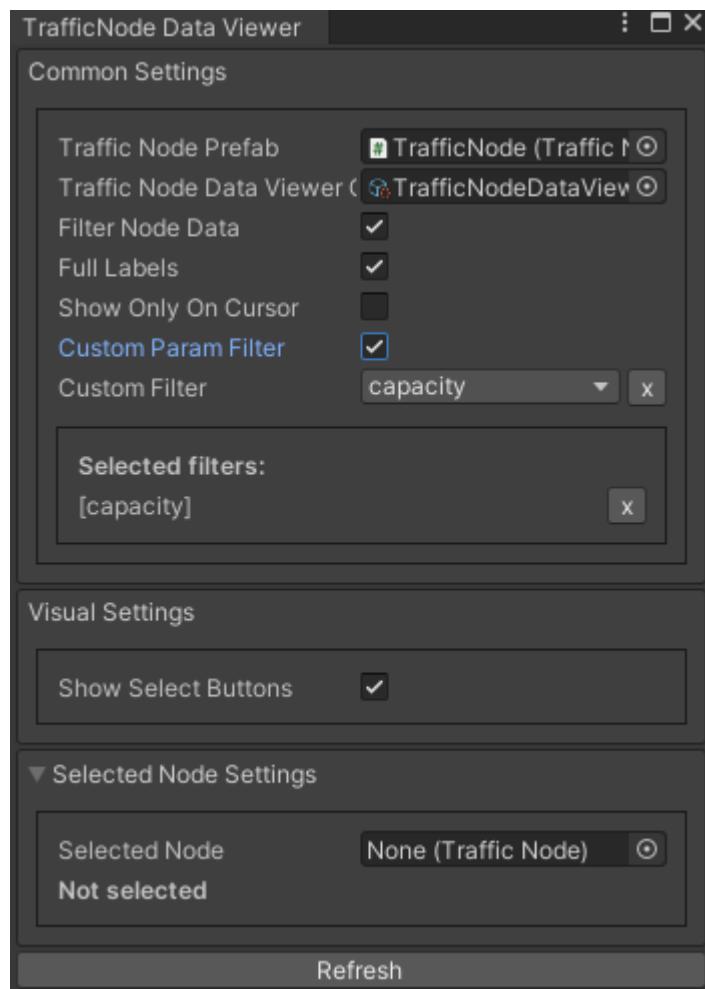
Selected node : *settings* for the selected *traffic node* (here you can quickly tweak the settings).

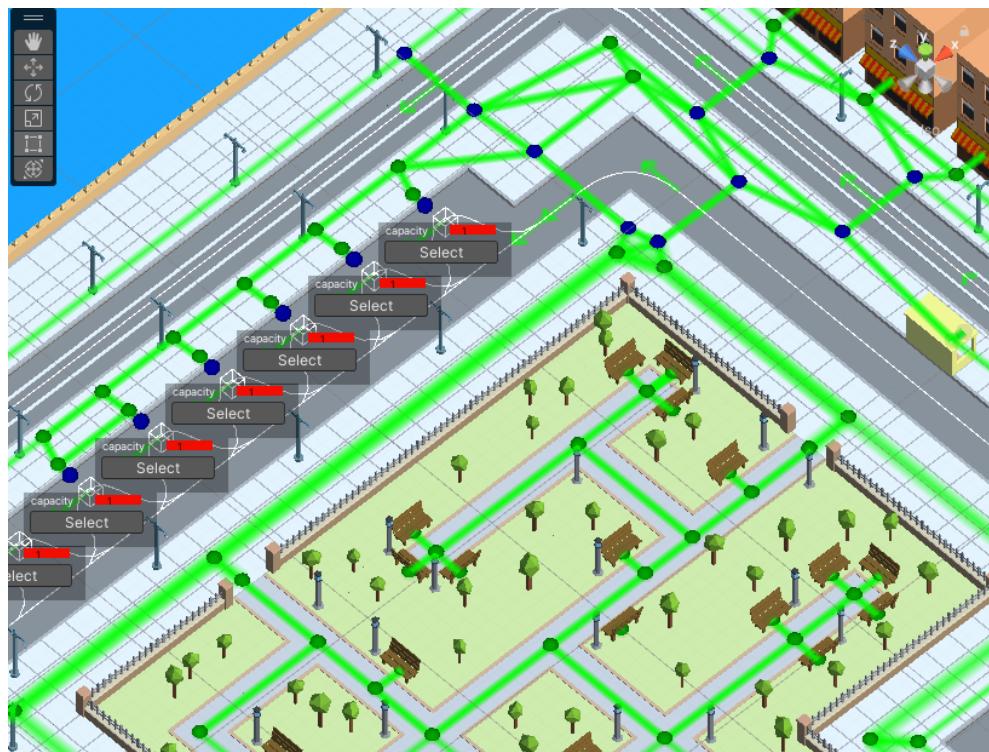


Filtering disabled.



All parameters filtering enabled example.

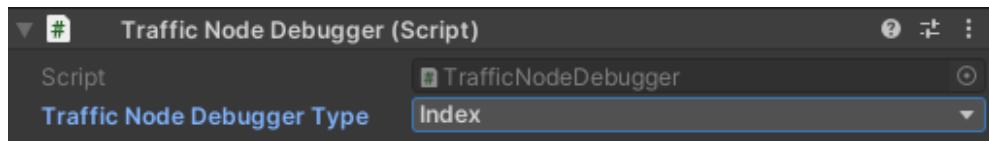




Filtering by “capacity” example.

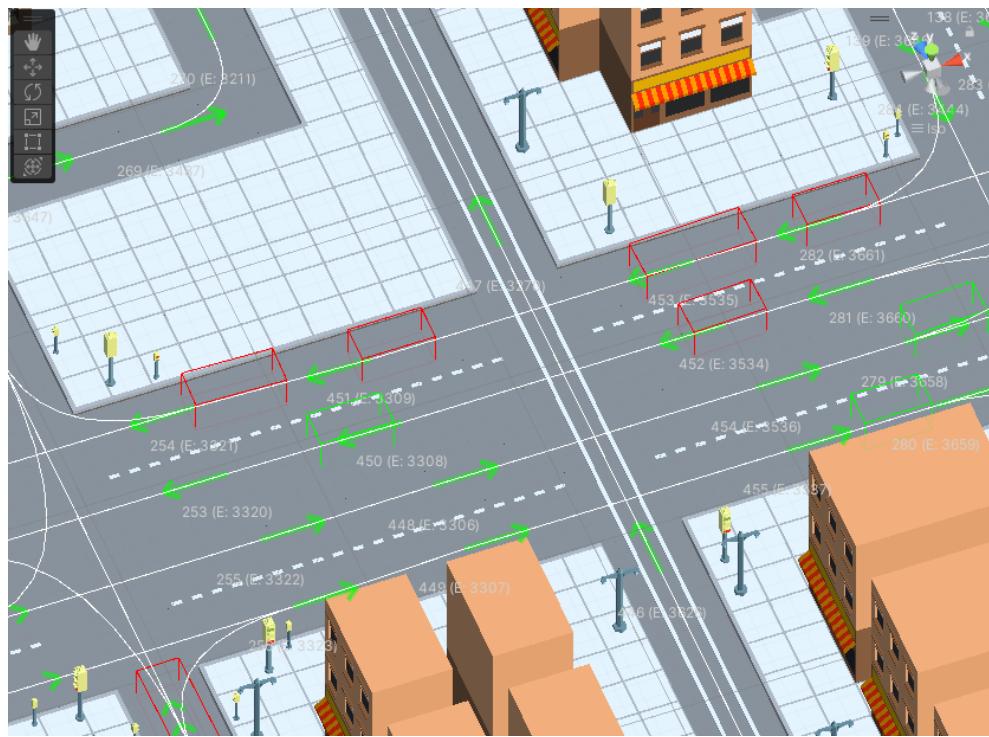
13.2.2 Traffic Node Debug

For debugging *traffic node* entities in runtime.



Index Debug

Shows the node index. For example: 450 (E: 3308) - [local *index for spawning* a vehicle] [entity index].



Available Debug

Shows the availability of a node for spawning.



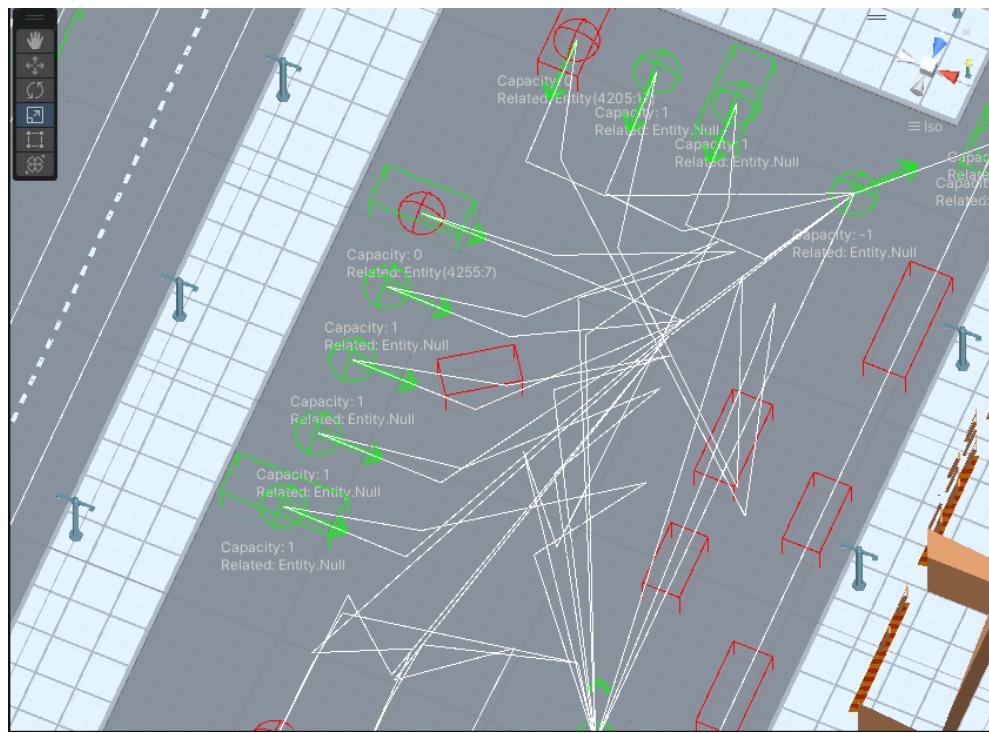
Light State Debug

Shows *light state* node traffic.



Capacity Debug

Shows the capacity of the node and the linked vehicle.



13.3 Pedestrian Debug

- *How To Open*
- *Pedestrian Debugger*
 - *Pedestrian Debugger Type*
 - * *Default*
 - * *Destination*
 - * *Destination Index*
 - * *Navigation*
 - * *Talk*

13.3.1 How To Open

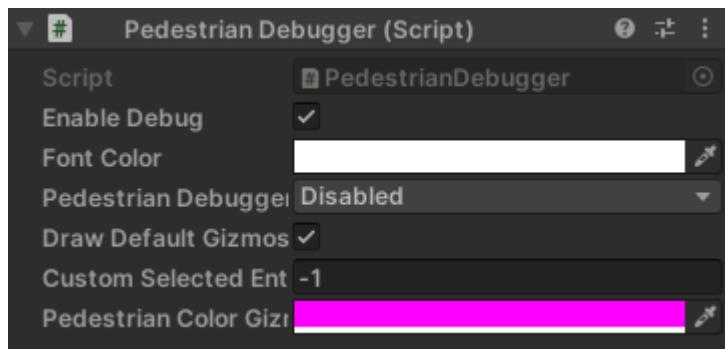
Youtube tutorial.

In the scene, select:

CityDebugger/PedestrianDebugger/

13.3.2 Pedestrian Debugger

For debugging the *pedestrian* entities.



Custom selected entity index : only the data of the selected entity index will be displayed.



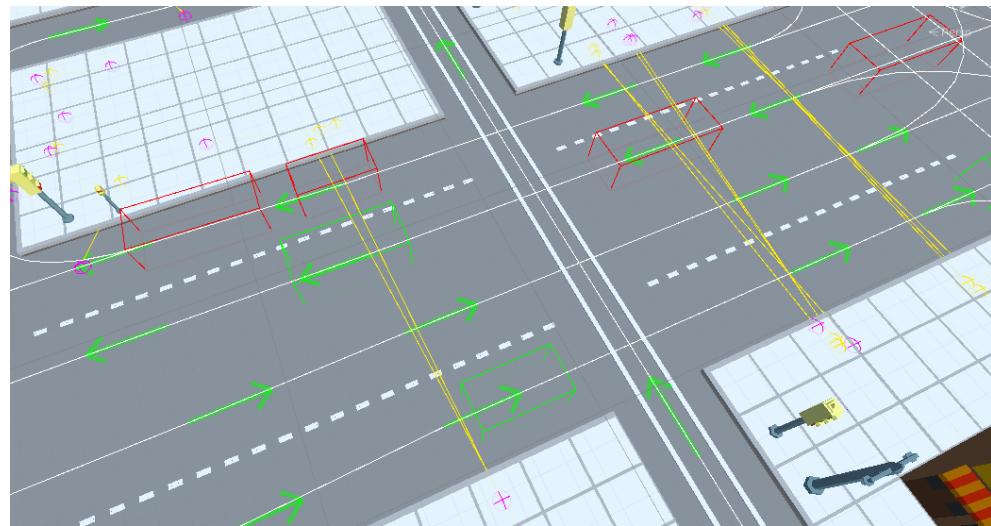
Debug info disabled example.

Pedestrian Debugger Type

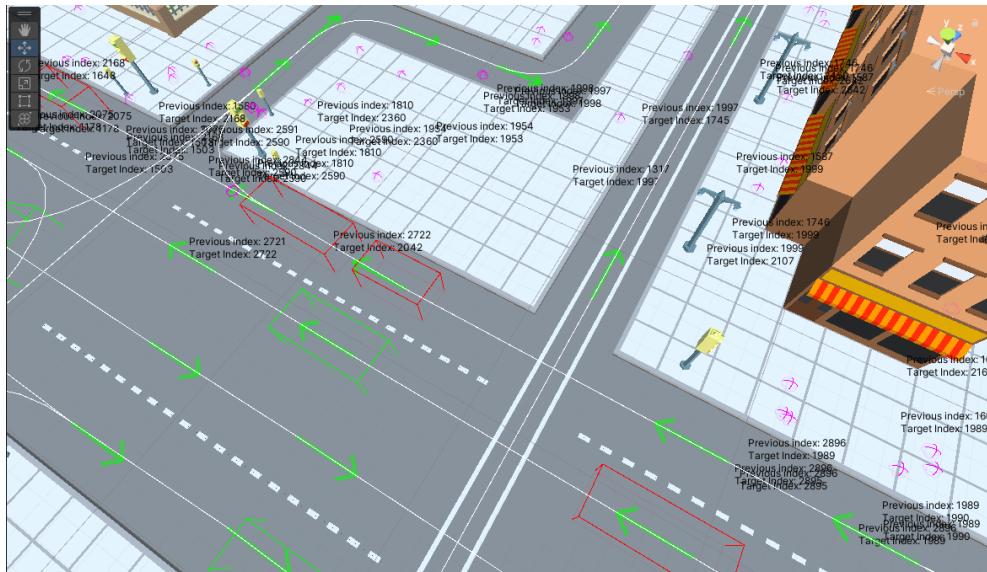
Default



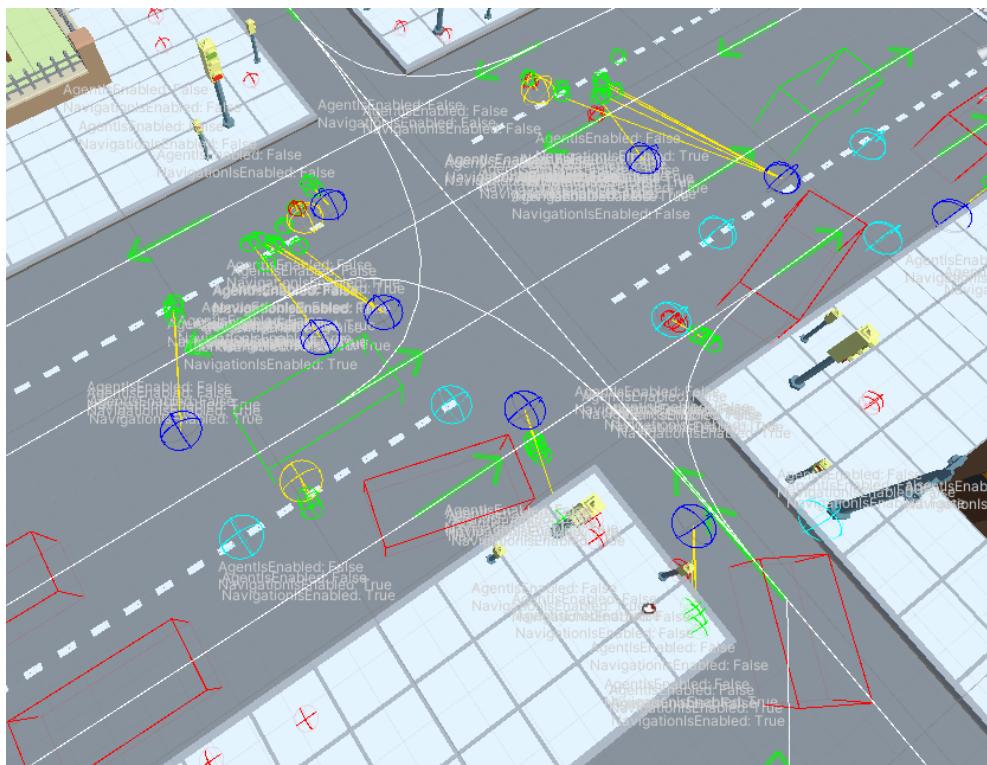
Destination



Destination Index



Navigation



Talk

Shows data on the conversation duration of pedestrians.

13.4 Pedestrian Node Debug

- *How To Open*
- *Pedestrian Node Debugger*
 - *Debugger Type*
 - * *Default*
 - * *Light info*
 - * *Crosswalk*
 - * *Other settings*

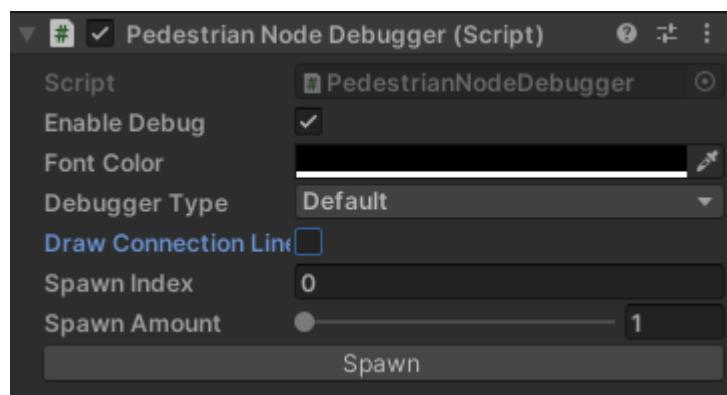
13.4.1 How To Open

Youtube tutorial.

On the scene select:

CityDebugger/PedestrianDebugger/

13.4.2 Pedestrian Node Debugger



Enable debug : on/off debug.

Debugger type

Draw connection line : on/off visual connection line in the scene.

Line width : line width of connection line.

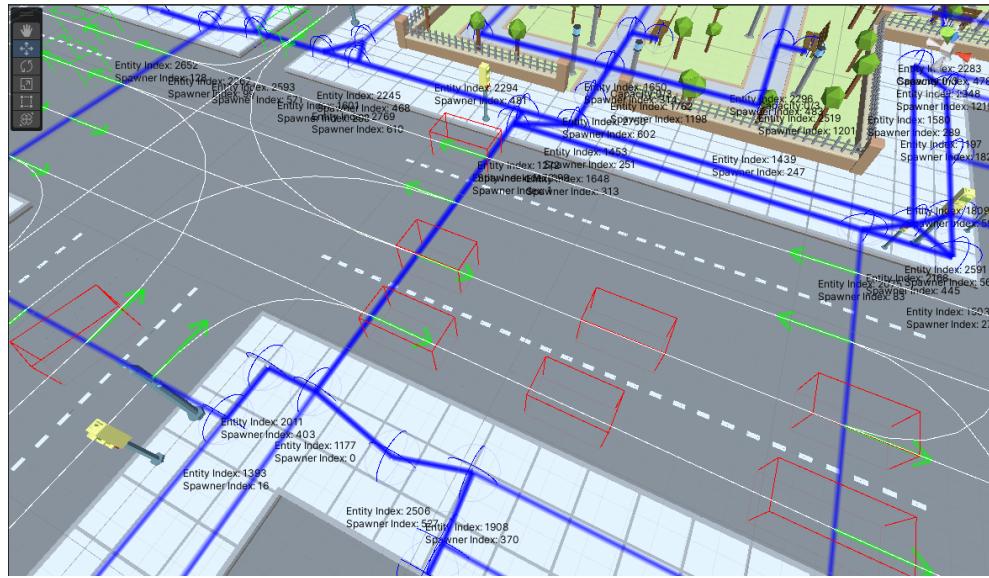
Spawn index : for manual spawn of a pedestrian by the *selected index*.

Spawn count : number to spawn pedestrians.

Debugger Type

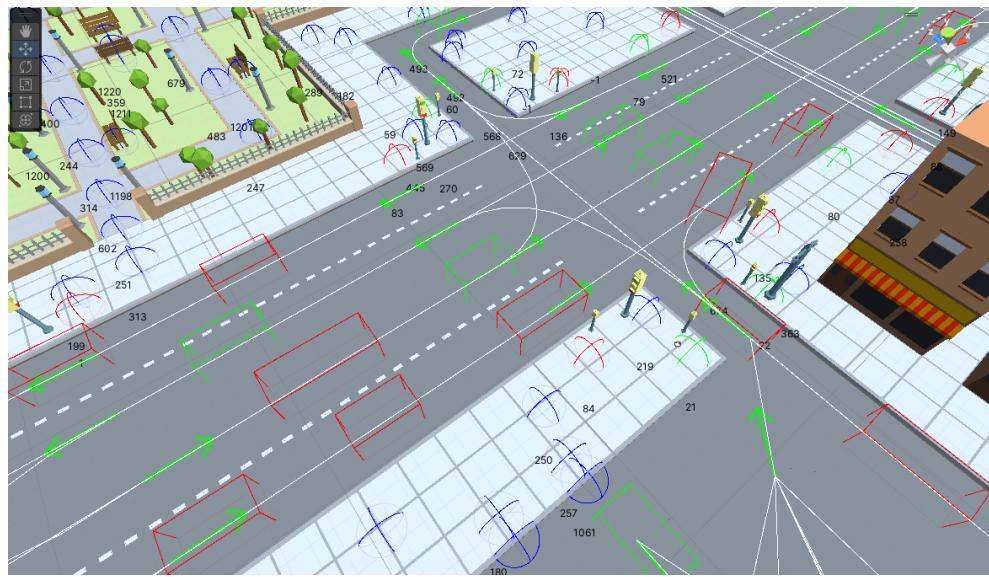
Default

Shows the entity index of the node and the spawn index for pedestrians.



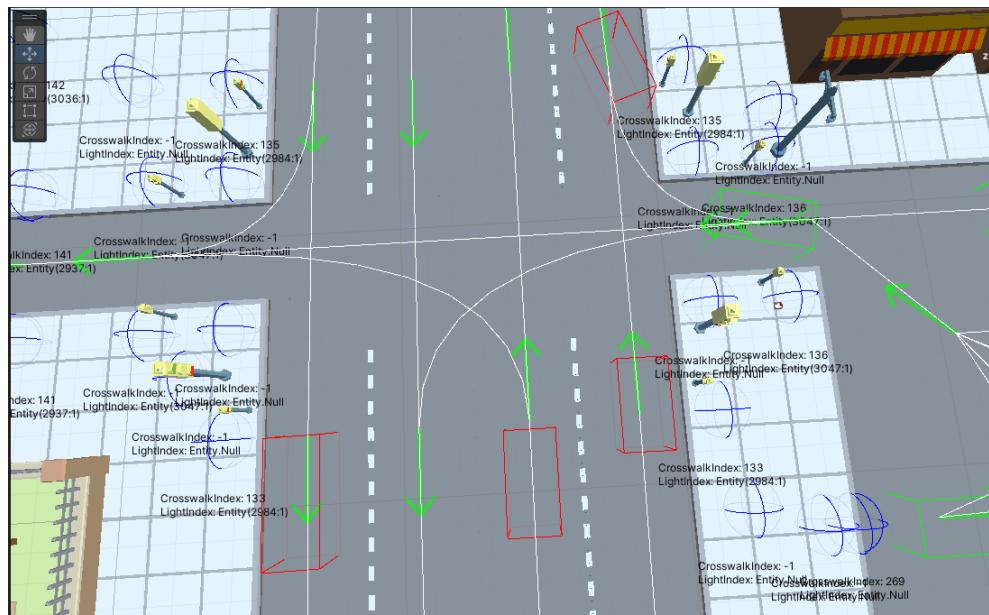
Light info

Light state of the pedestrian node.



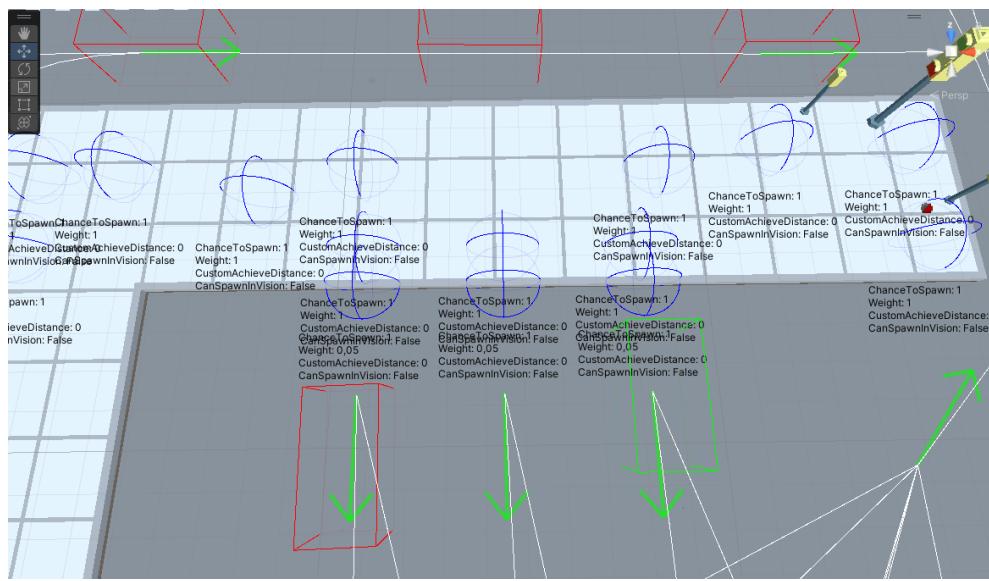
Crosswalk

Shows pedestrian crosswalks.



Other settings

Settings of the *pedestrian node*.



13.5 Path Debug

- *Path Debugger*
 - *How To Open*
 - *Settings*
- *Path Data Viewer*
 - *How To Open*
 - *How To Use*
 - *Settings*
 - *Examples*
- *Path Index Debugger*
 - *How To Open*
 - *Settings*
 - *Examples*

13.5.1 Path Debugger

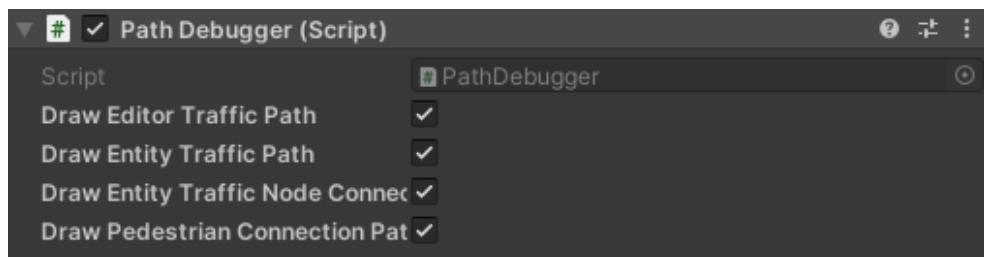
Youtube tutorial.

How To Open

In the scene, select:

CityDebugger/PathDebugger

Settings



Draw editor traffic path : on/off *path* visualisation in the scene in *Editor*.

Draw entity traffic path

[on/off entity *path* visualisation in the scene at runtime.]

- **Draw entity traffic node connection** : on/off draw *TrafficNode* entity connection debug at runtime.

Draw pedestrian connection path : on/off *PedestrianNode* connection visualisation in the scene in the *Editor*.

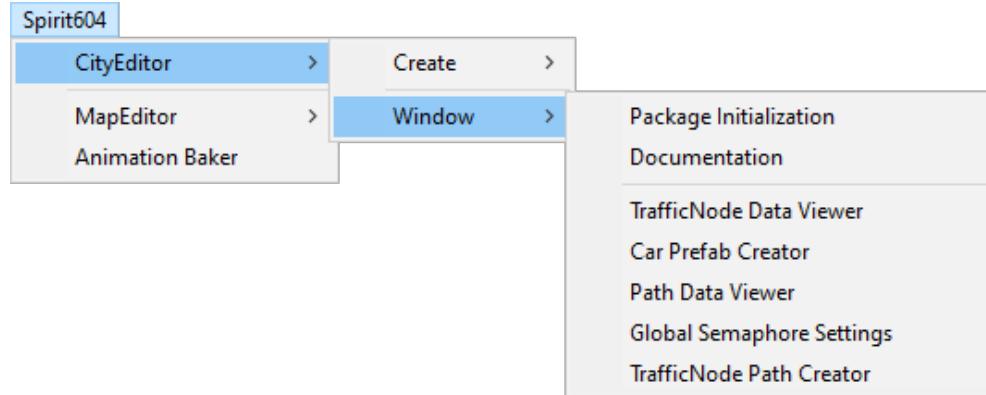
13.5.2 Path Data Viewer

Tool for quick visualisation of *path* parameters.

How To Open

In the *Unity* toolbar:

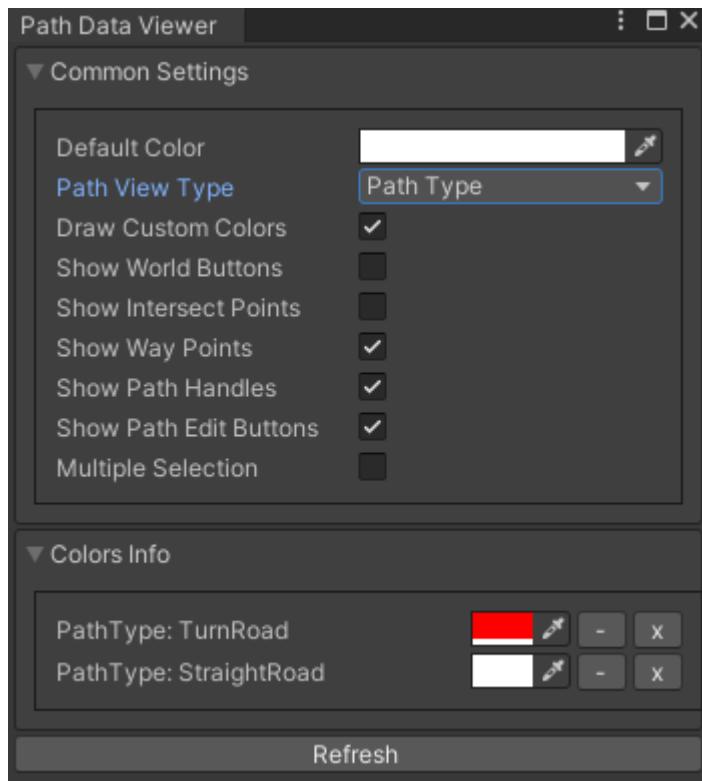
Spirit604/CityEditor/Window/Path Data Viewer



How To Use

1. Select *Path view type*.
2. In the *Colors info* tab, press - to hide the paths with the selected option.
3. Press + to display the hidden path with the selected option again.
4. Press x to reset saved color of the parameter.

Settings



Default color : default *path* color.

Path view type: selected *parameter* for visualisation (*examples*)

- **Speed limit** : speed limit of the :ref:`paths <path>`.
- **Priority** : priority of the :ref:`paths <path>`.
- **Path type** : path type of the :ref:`paths <path>`.
- **Traffic path group** : *traffic group* of the :ref:`paths <path>`.
- **Traffic path node group** : *traffic group* of the *waypoints*
- **Node direction** : node direction (forward or backward) of the *waypoints* in the *paths*.
- **Arrow light** : shows the *paths* with the assigned custom light.
- **Rail** : shows the *paths* with the *rail* parameter.

Draw custom colors : on/off custom colors of the *paths* in the scene.

Show world buttons : show world *path* selection buttons.

Show intersect point : on/off visual *intersection points* in the scene.

Show waypoints : on/off *waypoints* of the *path* in the scene.

Show path handles : on/off *path* position handles of the selected path.

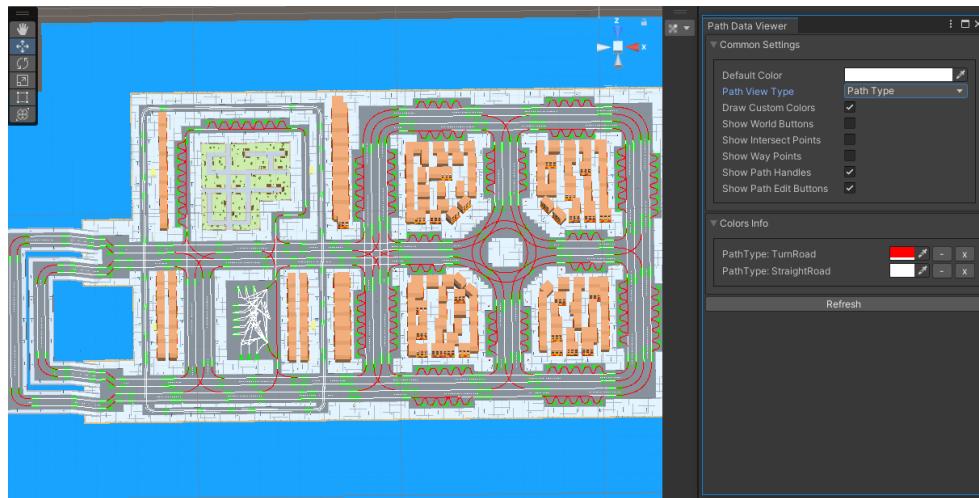
Show path edit buttons : on/off *path* edit buttons of the selected path.

Multiple selection : on/off feature to select multiple *paths* at the same time (useful for setting the same value for multiple *paths*).

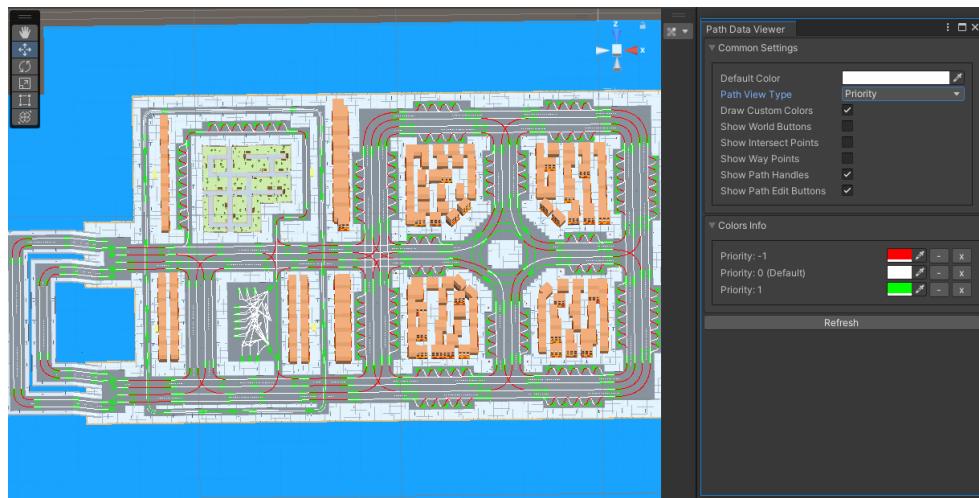
Show unselect buttons : show unselect button for already selected *paths* in multiple selection mode.

Refresh : update *path* data in the viewer.

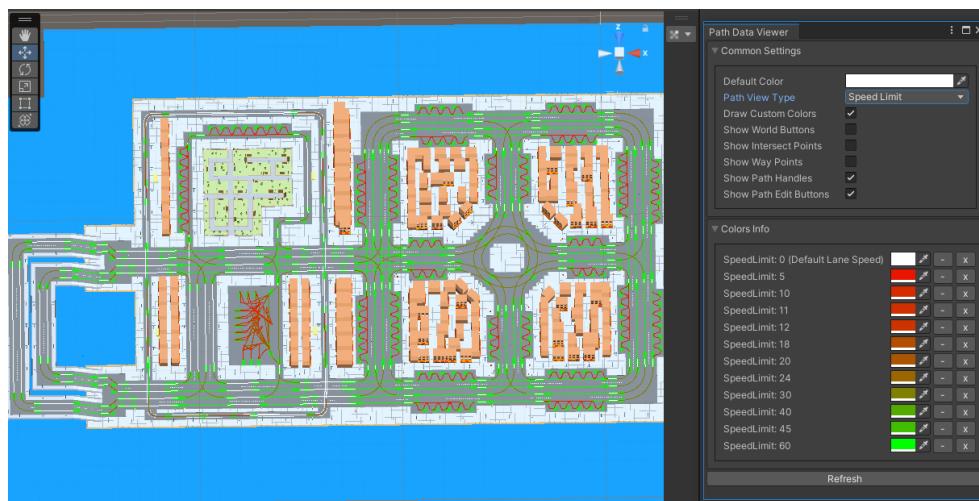
Examples



Path type example.



Priority path example.



Speed limit path example.

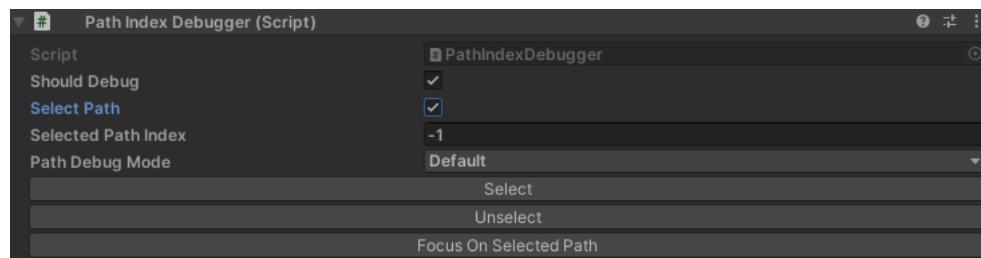
13.5.3 Path Index Debugger

How To Open

In the scene, select:

CityDebugger/PathDebugger

Settings



Should debug : on/off debugger.

Select path : on/off path selection settings.

Selected path index : display the data for the selected path (-1 path is not selected).

Path debug mode :

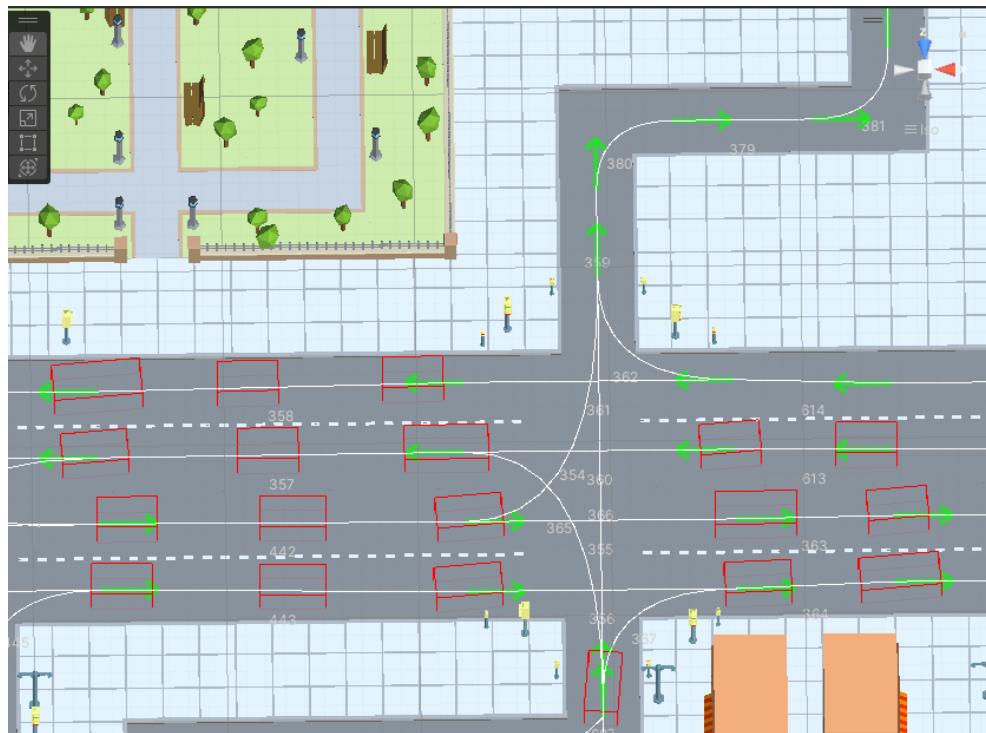
- **Default** : only the current path index is shown.
- **Parallel** : parallel path indexes.
- **Neighbor paths** : neighbor path indexes (paths that start from the same point).
- **Next connected paths** : indexes to which the current path is connected.
- **Intersected paths** : intersection paths indexes.

- **Car count** : number of cars with the current path index.

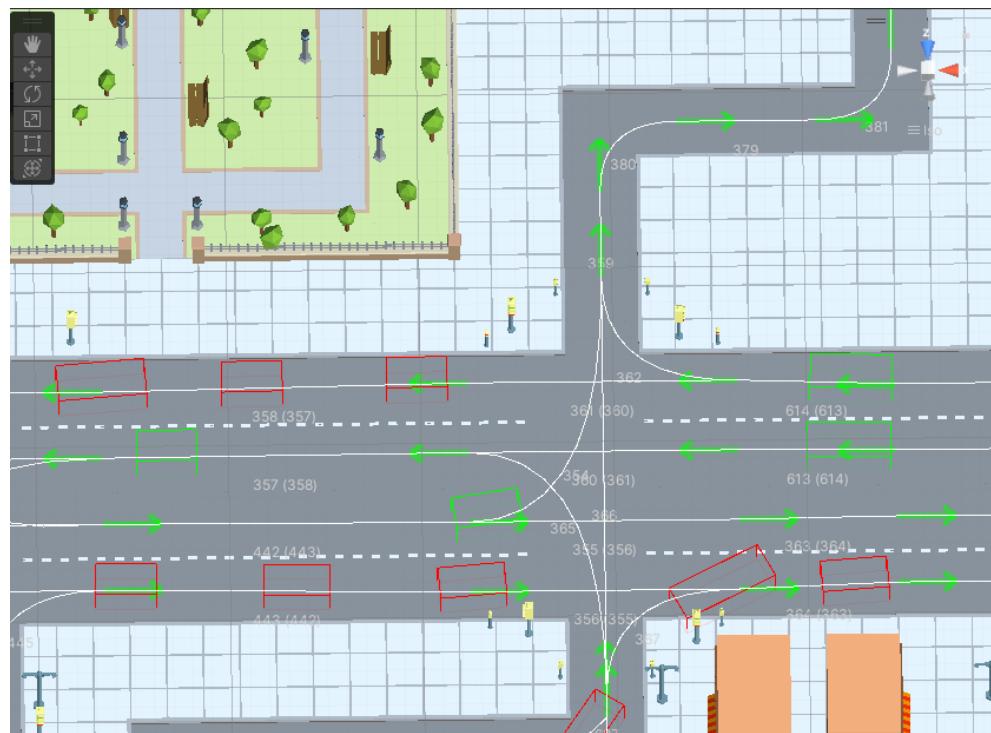
Index example:

- 543 (544, 545, 546) - current path index is 543. Other relevant path indexes, depending on the chosen *Path debug mode*.

Examples



Default “Path debug mode” example.



Parallel paths “Path debug mode” example.

13.6 Traffic Object Finder

- *Available Types*
 - *How To Open*
 - *How To Use*

Tool for finding traffic objects in the scene by *InstanceID* for troubleshooting.

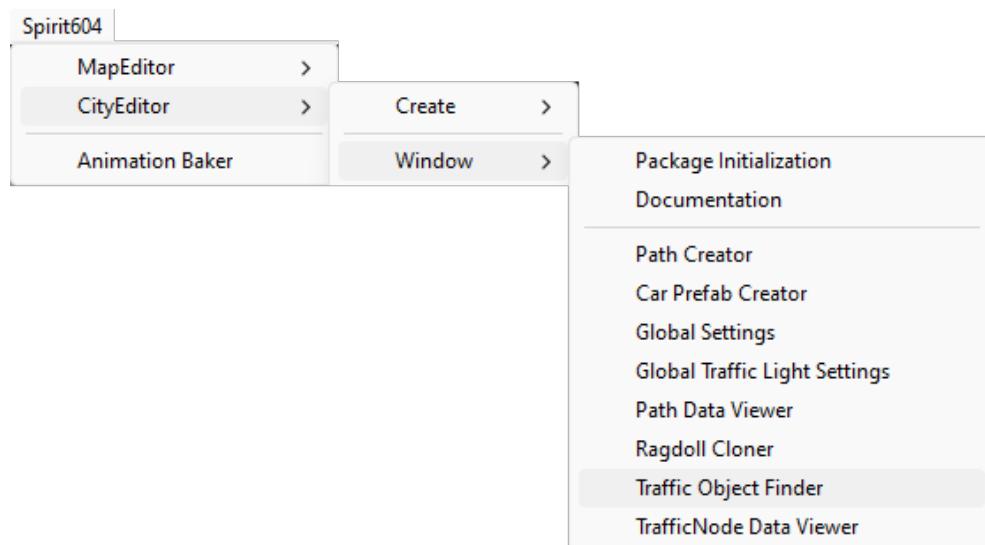
13.6.1 Available Types

- *Traffic nodes.*
 - *Path.*
 - *Pedestrian Node.*
 - *Traffic Light Object.*
 - *Traffic Light Handler.*
 - *Traffic Light Crossroad.*

13.6.2 How To Open

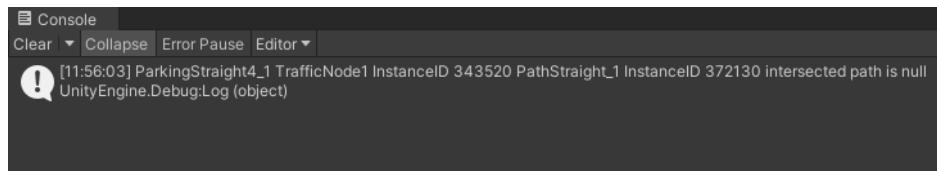
In the *Unity* toolbar:

Spirit604/CityEditor/Window/Traffic Object Finder



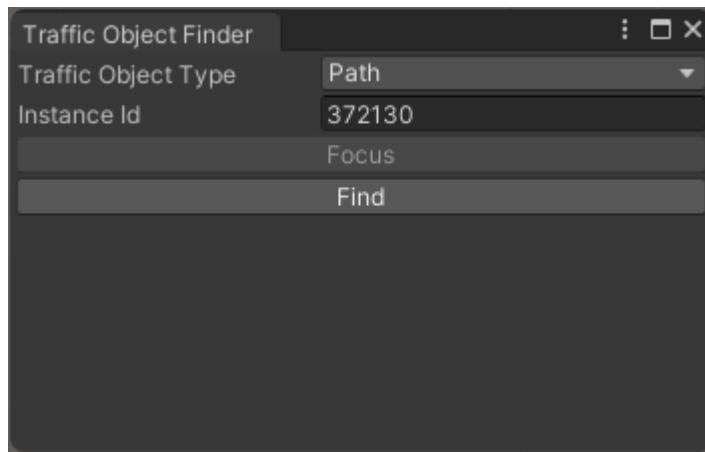
13.6.3 How To Use

1. Select *type* of traffic object.
2. Copy & paste *InstanceId* from the error message into the window.



Message example.

3. In the example, the type is *path* and the *InstanceId* is 372130.



4. Press *Find* & *Focus* buttons.

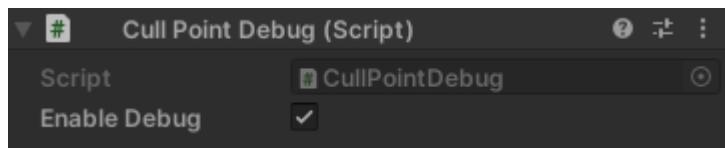
13.7 Common Debug

- *CullPoint Debugger*
- *CullState Object Debugger*
 - *Examples*
- *Hashmap Grid Debugger*
- *Section Debugger*

Youtube tutorial.

13.7.1 CullPoint Debugger

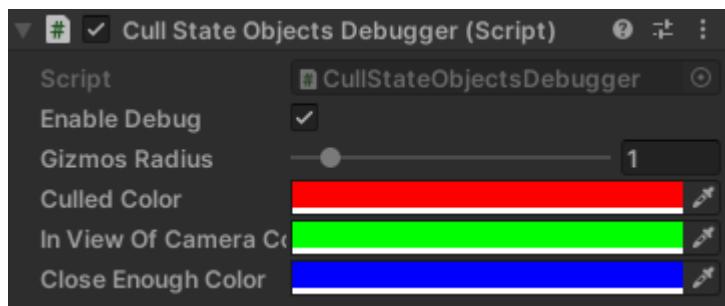
Displays the *cull point* radius (*example*).



Enable debug : on/off debugger.

13.7.2 CullState Object Debugger

Displays the *cull state* of objects in the scene (*example*).



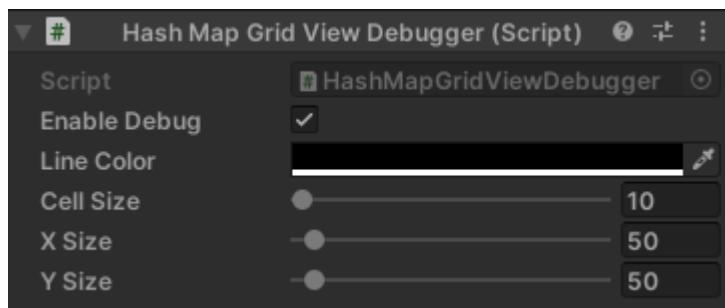
Examples



Cull point debug and cull state object debugger are enabled example (red - culled, blue - close to camera, green - in view of camera).

13.7.3 Hashmap Grid Debugger

Displays the size and position of the hashmap cell (can be useful when choosing the basic hashmap cell size in DOTS systems).



Enable debug : on/off debugger.

Line color : colour line in the scene.

Cell size : grid cell size.

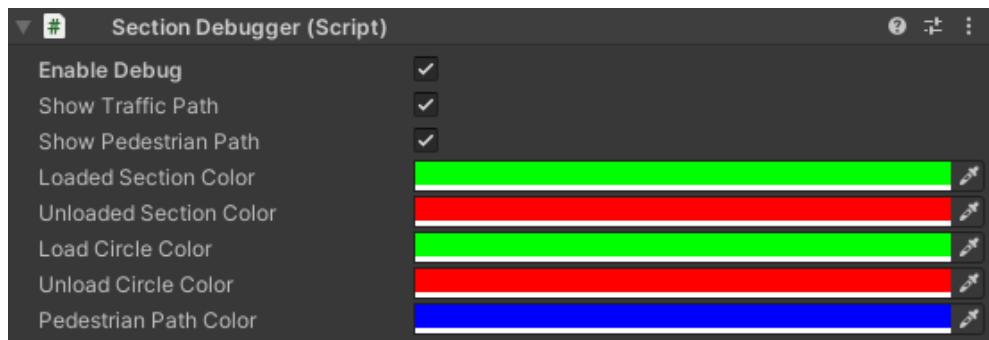
X size : X grid size.

Y size : Y grid size.

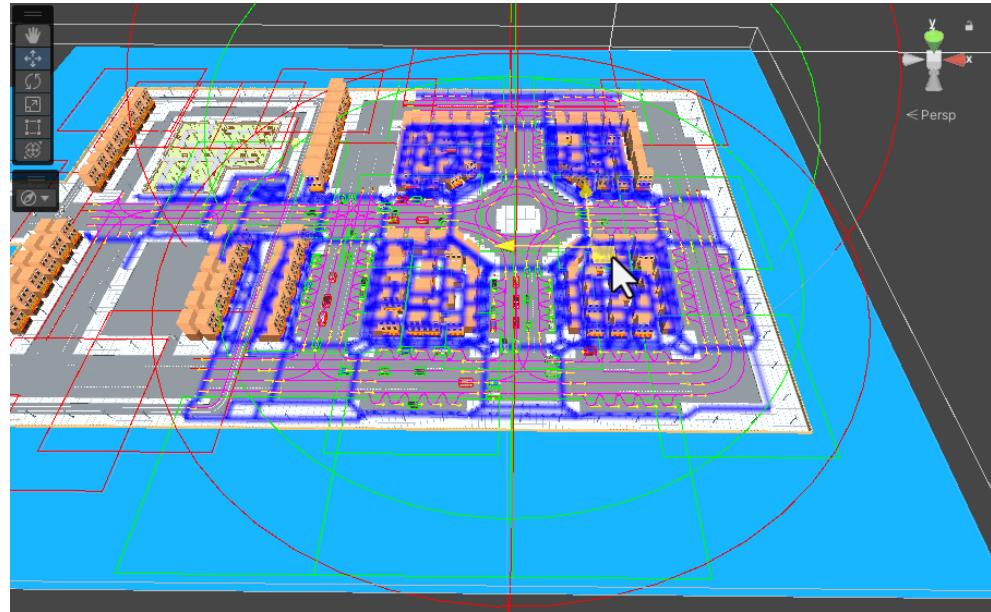


13.7.4 Section Debugger

Displays the *load section* radius, created road sections, road connections (*TrafficNodes* and *PedestrianNodes*).



- Enable debug** : on/off debugger.
- Show traffic path** : on/off display connection of *TrafficNodes*.
- Show pedestrian path** : on/off display connection of *PedestrianNodes*.
- Loaded section color** : loaded section color.
- Unloaded section color** : unloaded section color.
- Load circle color** : load circle color.
- Unload circle color** : unload circle color.
- Pedestrian path color** : connection color of *PedestrianNodes*.



Road streaming example.

CHAPTER
FOURTEEN

GLOSSARY

14.1 Common Terms

Factions are an enum type to solve the issue of friendly fire in damage systems

Factions:

- All :
- Player :
- Police :
- City :
- Mafia :

Pure entity : entities work entirely in the DOTS space.

Hybrid entity : entities that combine DOTS entities and default GameObjects (game objects are tied by position to an entity).

14.2 Traffic

Car type is used to convert prefabs of cars of the same model but different types (e.g. from a traffic car to a player's car when getting into it)

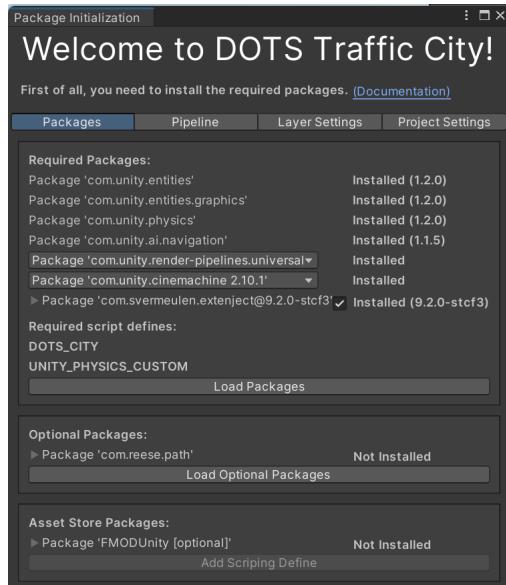
Car type:

- None
- Player
- Police
- City
- Mafia

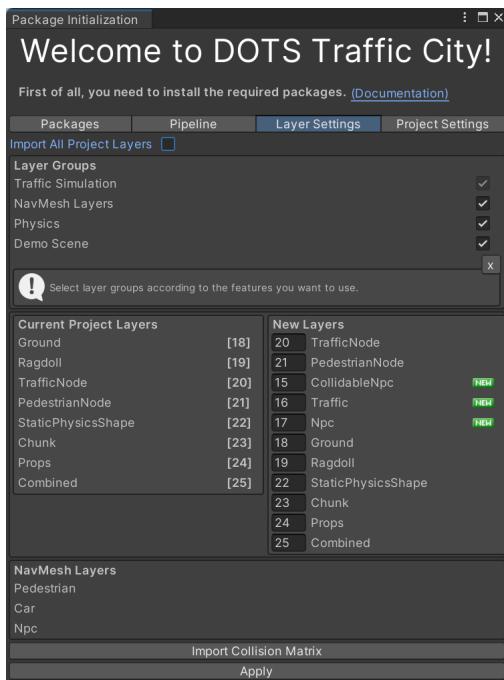
UPGRADE GUIDE

15.1 v1.0.x to v1.1.0

1. Make backup before importing the package.
2. Delete the following folders:
 - *DotsCity/Scripts/*
 - *Plugins/Spirit604/*
3. Import the package.
4. Open *Package initialization window*.



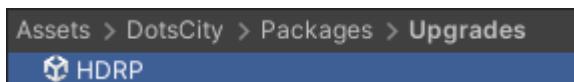
5. Select *Layer settings* tab.
6. Untick *Import all project layers* option.
7. Press *Import Collision Matrix & Apply* buttons.



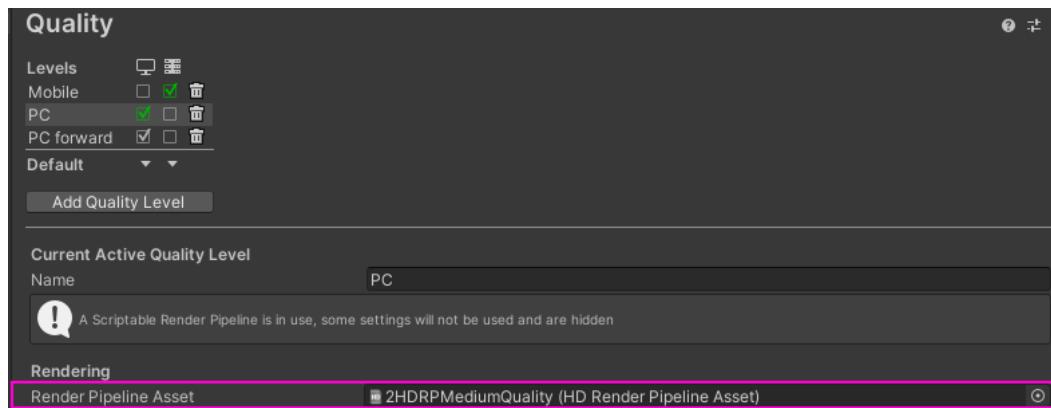
8. Remove old *Naughty attributes* package (if installed).
9. Restart the *Unity*.
10. If the scene has lost the references, close the *Unity* & delete this file:
 - *[ProjectFolder]/Library/ArtifactDB*
11. The new project is ready.

15.2 HDRP

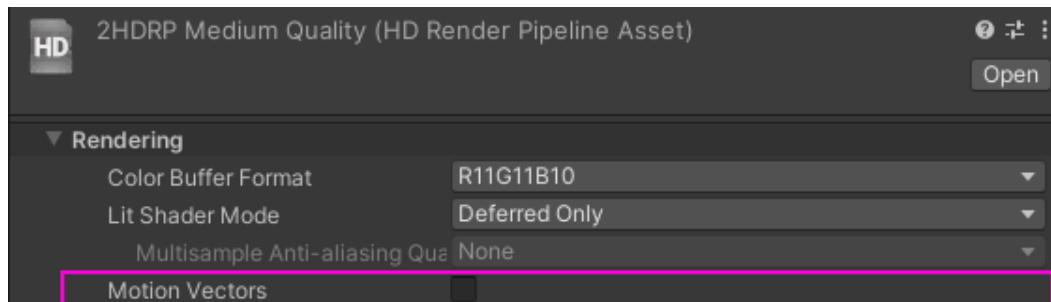
1. Download *HDRP* template project.
2. Remove *URP* packages in the *Package Manager* (if exist).
3. Unpack *HDRP* package.



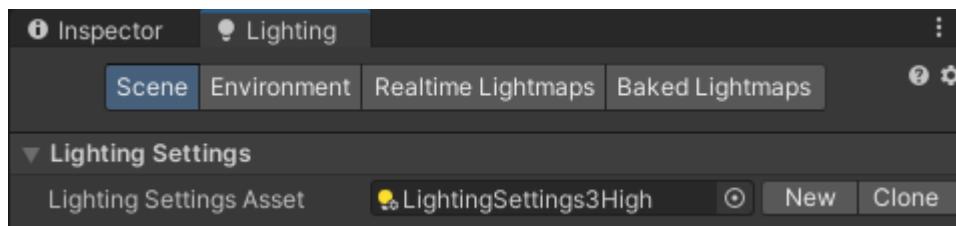
4. Open quality settings from the *Unity* toolbar:
Edit/Project settings/Quality
5. Set the *Render pipeline asset* from the *HDRP sample*.



6. Open assigned pipeline asset & disable *Motion Vectors* option.

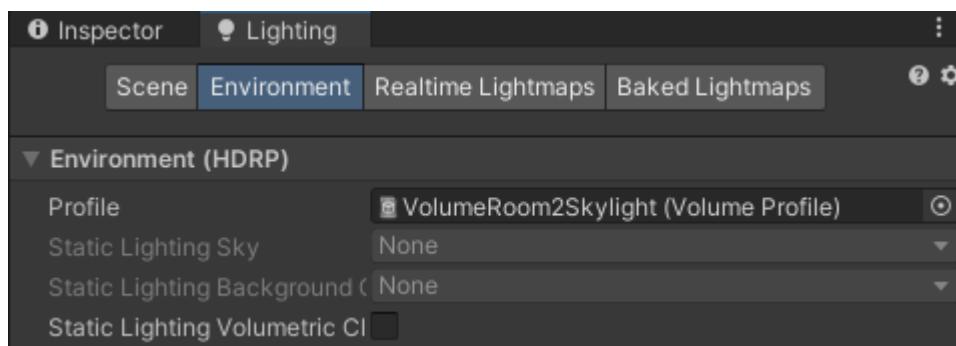


7. Set the *Light settings* for the desired scene (light settings can be taken from the *HDRP sample*).



Example.

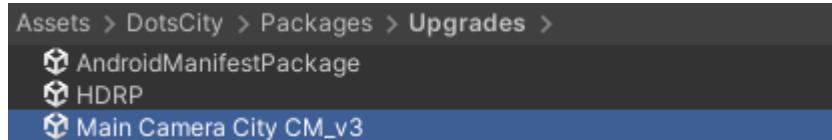
8. Set *Environment settings* for the desired scene.



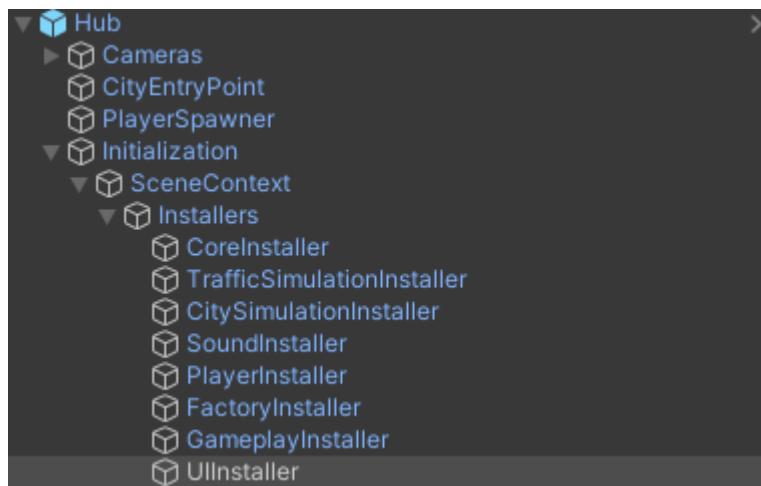
Example.

15.3 Cinemachine v3 Upgrade

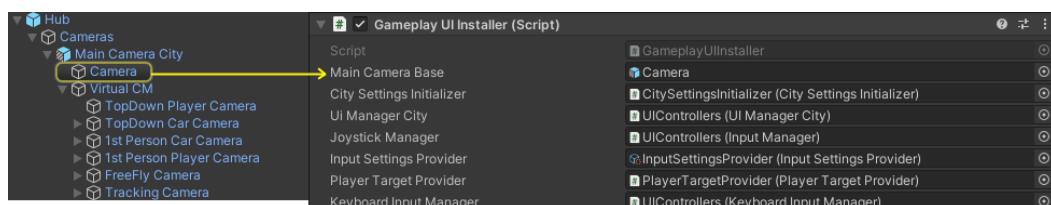
1. Make sure you have downloaded [Cinemachine v3 package](#).
2. Unpack *Main Camera City CM_v3* package.



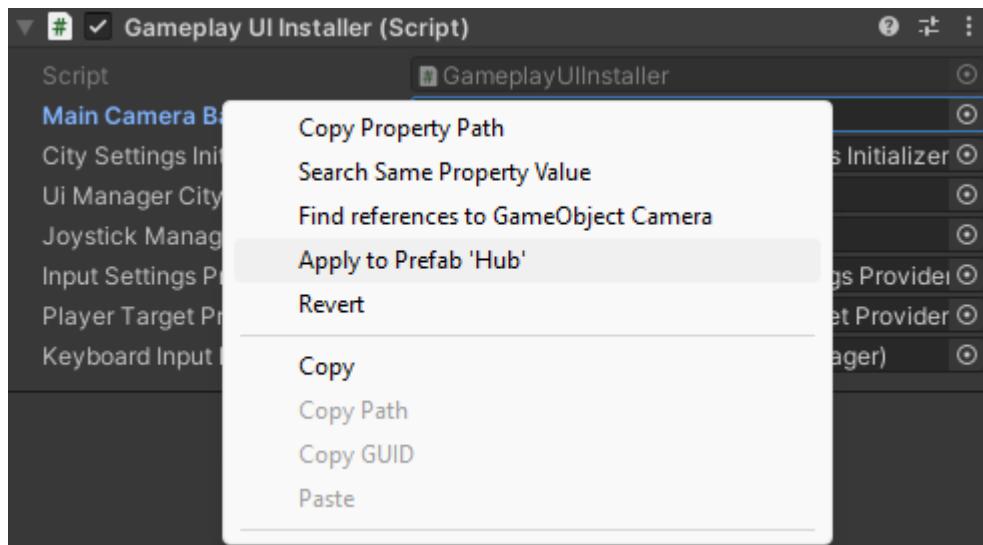
3. Select *UIInstaller*.



4. Drag & drop *Camera* object into *Main Camera Base* field in *UIInstaller*.

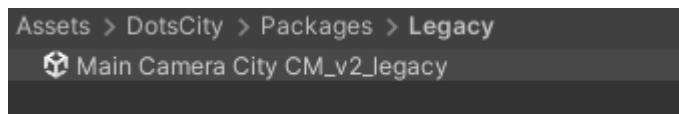


5. Right click on this field & press *Apply to Prefab Hub*.

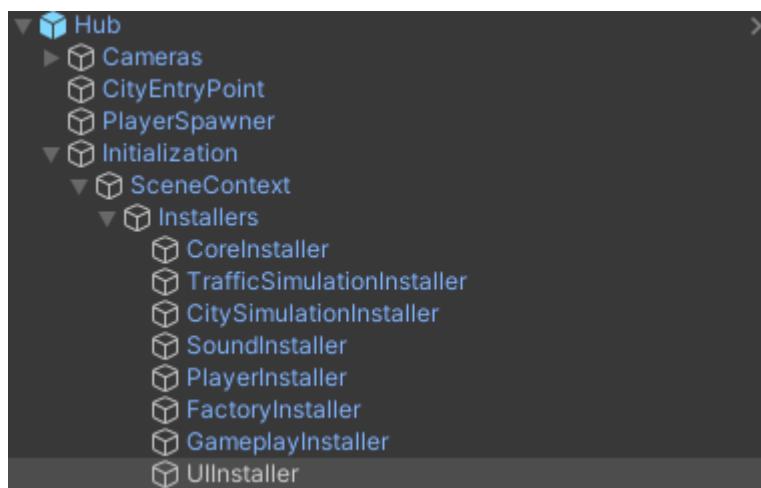


15.4 Cinemachine v2

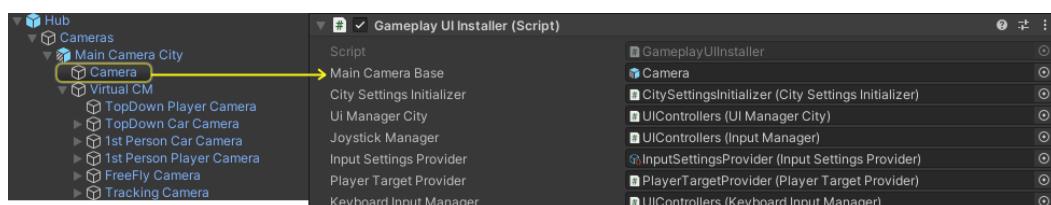
1. Unpack *Main Camera City CM_v2_legacy* package.



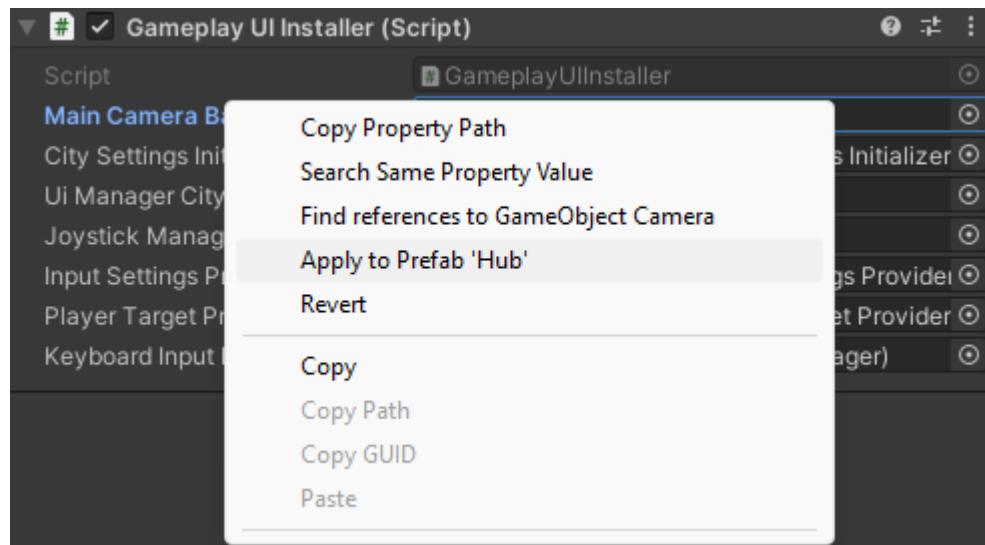
2. Select *UIInstaller*.



3. Drag & drop *Camera* object into *Main Camera Base* field in *UIInstaller*.



4. Right click on this field & press *Apply to Prefab Hub*.



KNOWN ISSUES

16.1 Unity.Physics package spikes by BuildPhysicsWorld & Solve-AndIntegrate systems on mobile devices.

16.1.1 Temporary fix until official *Unity* release

1. Close the *Unity* project.
2. Delete *Unity.Entities* & *Unity.Physics* packages from the *ProjectFolder/Library/PackageCache/* path.
3. Download fork packages *Unity.Entities* & *Unity.Physics* by [tertle](#) to *ProjectFolder/Packages/* path.
4. Open the project, if it fails, close *Unity*, clear the library folder & open again.

16.2 [FMOD] assert : assertion: 'level >= 0.0f' failed

Issue that doesn't affect on work, just ignore it for now.

16.3 Leak Detected : Persistent allocates 3 individual allocations.

Editor-related memory leaks.

The section will be updated with the most frequently asked questions.

17.1 Common Questions

17.2 Issues

17.2.1 I changed my preset, but the changes were not applied to the scene?

Try reopen the subscene by toggling the checkbox on & off.

17.3 Errors

17.3.1 No ‘git’ executable was found

How to fix

1. Download & install [git](#).
2. Make sure the system variable [PATH](#) contains the git install path.
3. Reboot your PC.

17.3.2 ChunkDataUtility.cs(1412,60): error CS0117

1. Update the Unity app to 2022.3.21 or higher.

17.3.3 error CS0006: Metadata file ‘nunit.framework.dll’ could not be found

How to fix

1. Restart the editor.
2. If the error is still appears, delete the *Library* folder from the project folder.

CHAPTER
EIGHTEEN

CHANGE LOG

18.1 [1.2.1c] - 08-11-2024

18.1.1 Fixed

- Fixed rare endless stuck traffic car when using raycast.
- Fixed collision system calculation.
- Fixed tile chunk prefab example.

18.1.2 Changed

- Added an option to change some general settings from the corresponding configs.
- *Path creator* can be used in the *Prefab stage*.
- *Global Light Settings* can be used in the *Prefab stage*.
- *Road segment* can be created in the *Prefab stage*.

18.2 [1.2.1b] - 06-11-2024

18.2.1 Added

- Added support for lane changing on run-time roads.
- Added a helper button for traffic lights that have lost their reference to a traffic light crossroad.
- Added support for *Odin Inspector*.

18.2.2 Fixed

- Fixed intersection conversion for run-time road chunks.

18.3 [1.2.1] - 04-11-2024

18.3.1 Added

- Added support for *Multi-road segments* by adding 1 *RuntimeSegment* at a time in *Runtime Road mode*.
- Added new *RuntimeTile Chunk Road demo* to demonstrate the road chunks added at runtime.
- Added a new option to leave the car idle for a certain amount of time when parking if pedestrian is disabled for the scene.
- Parking can be used on sloping surfaces.
- Added auto-curve type detection for *Path creator*.

18.3.2 Fixed

- Fixed *path* intersection calculation for custom shape surface.
- Minor fix *Car prefab creator* text pattern search for wheels.
- Fixed *Animation Baker* baking with single texture atlas for multi-mesh characters.
- Fixed a problem with the *Local Avoidance* switch multi-targeting in a short amount of time.

18.3.3 Changed

- Improved obstacle detection in intersecting & neighbouring cases.

18.4 [1.2.0] - 28-10-2024

18.4.1 Added

- Runtime graph creation.
- Added new *RuntimeTile Demo* scene.
- Traffic light *API* to get traffic light state from monobehaviour script.
- Added option to manually handle traffic light state.
- New train system.
- Custom train system support.
- Custom train demo scene.
- Added the ability to split *external traffic routes* into smaller ones to better balance spawning.
- Added custom settings for pedestrian nodes for selected routes.
- Added the ability to split pedestrian routes into smaller ones to better balance spawning.

- New *TriggerLight* type for *TrafficNode*, which triggers selected traffic light when traffic car enters this node.
- The *Traffic Road Debugger* can be used at runtime to manually spawn vehicles in custom scenarios.
- Added saving of *Road Segment Creator* settings so that a new road segment is created using the previously saved settings.
- Added a handy duplicate feature for existing connected *Road Segment Creator* to create clean duplicates without existing connected paths.
- Added a sample custom player to interact with the custom car & pedestrian.
- Added one-way roads for pedestrian nodes.
- Added manual sync button for all configs.

18.4.2 Fixed

- Fixed car creation offset for car parts when the car parts are not the parent of the car body.
- Fixed wheel detection during car creation in some cases.
- Fixed adding trigger area tag to non-pedestrian entities.
- Fixed *Auto-crossroad* generation when the custom segment contains one-way paths.
- Fixed Path creator detects wrong connection sides in some cases.
- Fixed steering input can be incorrectly calculated in some cases.

18.4.3 Changed

- Now the road *Graph* is created at runtime when the scene starts.
- *Cinemachine v3* used by default.
- Traffic light states for each traffic light handler are now stored in the dynamic buffer.
- Improved randomization of initial pedestrian spawn.
- Added traffic light debugging for paths with custom lights.

18.5 [1.1.0g] - 19-09-2024

18.5.1 Fixed

- Crossroad jam obstacle fix.
- Fixed sound pooling when vehicle is destroyed.
- Fixed lane change potential obstacle stuck when multiple cars are changing to the same lane.
- Fixed avoidance of mono cars when trying to change lanes.
- Fixed custom traffic light for specific path.
- Fixed initial *HDRP* installation conflict with *Cinemachine v3* package.

18.6 [1.1.0f] - 10-09-2024

18.6.1 Changed

- Improved *NPC* obstacle detection.

18.6.2 Fixed

- *TrafficNpcCalculateObstacleSystem* debug race condition fixed.
- Anti-roll fix for *Arcade Vehicle Controller*.
- Fixed warning messages.
- Fixed potential config sync failure in some cases.
- Fixed missing reference in the *PolygonCity*.
- Fixed *EasyRoads3D* exception when crossing has 1 connecting road.

18.7 [1.1.0e] - 16-08-2024

18.7.1 Added

- Auto-crosswalk connection in the *Road Parent*.
- Auto-connection distance in the *Road Parent*.
- Added new road warning messages.
- New *Agents Navigation* config.
- New agent hybrid component.

18.7.2 Fixed

- Fixed move handle for moving two or more road segments.
- Crowd sound system dependency fix.
- Fixed *Ragdoll* not being pooled.

18.7.3 Changed

- Improved *Road Parent* UI.

18.8 [1.1.0d] - 12-08-2024

18.8.1 Added

- Interpolation of the car view for culled mono physics cars.
- New collision stuck avoidance system for *Hybrid mono* cars.

18.8.2 Fixed

- Agents Navigation integration editor error fix.
- Minor player arcade car prefab fix.
- Traffic node viewer fix.

18.8.3 Changed

- Improved transition between physics & no physics arcade cars.

18.9 [1.1.0c] - 09-08-2024

18.9.1 Added

- New auto-sync config option between MainScene & Subscene.
- Traffic node gizmos settings.
- New pure city stress scene.

18.9.2 Fixed

- Minor script fix for Unity 2023.2.
- Fixed potential config corruption for builds.
- Fixed stress scene demo exit error.
- Arcade vehicle controller wheel position fix.

18.9.3 Changed

- Minimum *Unity* version 2022.3.21.
- Improved arcade sample cars.

18.10 [1.1.0b] - 06-08-2024

18.10.1 Added

- Added *CarModelRuntimeAuthoring*, *BoundsRuntimeAuthoring*, *VelocityRuntimeAuthoring* entity runtime components.

18.10.2 Fixed

- Fixed compatibility with Entities 1.3.0.
- Fixed initial entity scale for runtime entities with *CopyTransformFromGameObject* component.
- Fixed bootstrap if user tries to start bootstrap twice.
- FMOD minor script fix.
- Nav agents dependency fix.

18.11 [1.1.0] - 05-08-2024

18.11.1 Added

- Full Hybrid mode support:**
 - New *monobehaviour compatible* traffic.
 - New hybrid NPCs compatible with any custom character controller.
 - New hybrid traffic lights.
- New *EasyRoads3D* integration.
- New *Agents Navigation* integration.
- New *API* for custom spline roads generation.
- New *EntityWeakRef* class to link Monobehaviour script & traffic & pedestrian node entities.
- New player traffic control feature.
- New runtime entity hybrid workflow for runtime gameobjects.
- New hybrid GPU mode that allows you to mix hybrid animator models for near and GPU animation for far at the same time.
- New universal animation handling API for GPU & hybrid animator entities.
- Limit texture baking for *Animation Baker*.
- Multi texture container for *Animation Baker*.
- Added chasing cars feature.
- Path Waypoints can be traffic node functionality.
- Added endless streaming for *Custom straight* road.
- Added *Auto-crossroad* option for *Custom segment* for custom shape crossroads.
- Pedestrian node generation along *Custom straight* road.

- *Custom straight* can be converted into the *Custom segment* road.
- Crosswalk connection for *Custom segment*.
- Added left-hand traffic option.
- Custom cull state config calculation for specific entities.
- New camera view based culling calculation method.
- New spawn culling layer adjustment for traffic & pedestrians.
- New traffic node display for right, left lanes in segments & path spawn nodes.
- Traffic & pedestrian node debug in *Editor* mode.
- New project initialization window.
- Added support for Unity's built-in audio engine.
- Added *HDRP* support.

18.11.2 Fixed

- Fixed traffic spawning in culled areas.
- Fixed custom physics vehicle could jump after restoring physics at runtime in some cases.
- Fixed a potential crash when user undoing changes *Custom straight* roads.
- Fixed obstacle detection for neighbouring paths.
- Fixed *Player spawner* not spawning in some cases when adding the new *ID* for player NPCs.
- Player spawn no longer throws an exception if it doesn't exist.
- Fixed *Input* for *Player car* in *Editor* when *Android* build is selected.
- Fixed road segment merge.

18.11.3 Changed

- Major refactoring of the project to make it more modular.
- Now the project can be used for traffic simulation only, without player & extra features.
- Project no longer overwrites the settings by default.
- FMOD no longer required package.
- Removed *Naughty attributes* dependency.
- *Zenject* can be an optional dependency.
- Now all sound data is stored in *SoundDataContainer* scriptable object.
- Min *Burst* version 1.8.16 for *Unity* 2022.3.31 or higher.

18.12 [1.0.7d] - 06-06-2024

18.12.1 Added

- Create & connect *Pedestrian nodes* in the prefab scene.
- Added gradle config for Android for Unity 6.
- Added support *Cinemachine 3.0+*.

18.12.2 Fixed

- Fixed Unity package dependency resolving for the first time can cause endless script compilation.
- *Custom straight* road may have null traffic nodes due to initial creation in some cases.
- Fixed *Custom straight* road oneway path generation with multiple lanes.
- Fixed *Custom segment* path surface snapping.
- Fixed *Pedestrian node creator* losing sceneview focus, causing the hotkey for it to be disabled.
- Animation baker minor UI fixes & improvements.

18.13 [1.0.7c] - 31-05-2024

18.13.1 Fixed

- Fixed package initilization window doesn't load in some cases.
- Fixed package initilization window appears randomly on Mac OS.

18.14 [1.0.7b] - 29-05-2024

18.14.1 Added

- Auto bootstrap option for single scene.
- Bootstrap logging.
- Entity road drawer for the editor time.

18.14.2 Fixed

- Car prefab creator ID duplicate error.
- Script defines after the project update.
- Input in the custom vehicle test scene.

18.15 [1.0.7] - 24-05-2024

18.15.1 Added

- New auto-spline option for *Bezier* curves in the *Path Creator*.
- New *extrude lane* option for *Custom segment* road in the *RoadSegmentCreator*.
- New divider line for *Traffic nodes* & *Custom straight* roads.
- New components to interact with *Hybrid pedestrians* from *MonoBehaviour's*.
- Custom ragdoll user's support for *Hybrid pedestrians*.
- New custom IDs for vehicles in the *Car Prefab Creator*.
- New car model selection list for the *player spawner* when the player is spawned in the car.
- User's *custom camera* integration.

18.15.2 Fixed

- Fixed *Pedestrian node* connection on custom terrain shapes in the *Pedestrian node creator*.
- Fixed auto-switch type for oneway paths in the *Path Creator*.
- Player spawn, if the player originally spawned in the car.
- Fixed a potential *Type mismatch* error for animation clips in *Animation Baker* which could cause the UI to break.
- Fixed a potential *Nan* position for pedestrian in the *Antistuck system*.
- Fixed traffic spawner for the path with *0* index.
- Fixed compatibility with Unity 2023.2.

18.15.3 Changed

- *Pedestrian node* scene filtering updates when node settings are changed in the *Pedestrian node creator*.
- *PedestrianReferences* component renamed to *PedestrianEntityRef*.

18.16 [1.0.6] - 22-04-2024

18.16.1 Added

- New connection type for *Path Creator*.
- New *traffic light* customizations for Road Segment Creator tool.
- New *crosswalk node shape* option for *Road Segment Creator*.
- New state utils methods for pedestrian.

18.16.2 Fixed

- Fixed path connection for Path Creator in some cases
- Fix for traffic light duplication when editing a road segment in the subscene.

18.16.3 Changed

- UX improvement for Path Creator.

18.17 [1.0.5] - 15-04-2024

18.17.1 Added

- New *multi-mesh* customization support for GPU animations.
- New custom *attachments* support for GPU animations.
- New custom GPU animation *option* for selected pedestrians.
- Integration for custom *player vehicle controller* plugin which controlled by MonoBehaviour script [**experimental**].

18.17.2 Fixed

- Animation GPU baking with animated parent.
- Fixed physics surface cloning tool in some cases.
- Traffic spawn fix in some cases.
- Fixed obstacle detection for reverse or arc paths.
- Static physics culling.

18.17.3 Changed

- Traffic lights are disabled by default for straight road templates.
- Removed obsolete options for Car Prefab Creator.

18.18 [1.0.4] - 04-04-2024

18.18.1 Added

- New align custom straight road feature *along the surface*.
- New animation baker clip *binding*.

18.18.2 Fixed

- Path recalculation for custom straight roads.
- Re-creation of the road segment with custom user orientation.
- Fix waypoint info display for road segment in some cases.

18.18.3 Changed

- Improved *snapping* for custom road segments.

18.19 [1.0.3b] - 01-04-2024

18.19.1 Fixed

- First init editor hotfix.
- Path baking validation fix.

18.20 [1.0.3] - 29-03-2024

18.20.1 Added

- Added GPU animation *transition preview*.
- New optimized shaders for crowds.
- GPU data preparation for LODs.
- New user-friendly animation shader control.

18.20.2 Changed

- Update to entities 1.2.0
- GPU animation baking and playback algorithm for better memory texture layout.
- Improved GPU transition animations.

18.21 [1.0.2] - 25-03-2024

18.21.1 Added

- New movement randomization speed for pedestrians.

18.21.2 Fixed

- A rare build crash caused by the area trigger system.
- Fixed the pedestrian physics runtime option in the build.
- Mobile input for build.

18.22 [1.0.1b] - 22-03-2024

18.22.1 Fixed

- Traffic mask settings editor fix.
- Script refactoring.

18.23 [1.0.1] - 20-03-2024

18.23.1 Fixed

- Missing script hotfix.

18.24 [1.0.0] - 19-03-2024

- Initial release.