# FOREVER

User Manual and API Reference

## Contents

# 1. Getting Started

Forever is a complete solution for creating endless runner games or procedurally generated linear games (non-infinite) such as racing (closed loop) and skiing (A-B) games in both 3D and 2D.

The system is based around the concepts of level segments – game objects which are designed by hand and then instantiated and extruded during runtime. The generated segments in a level can be a fixed amount or can be an indefinite amount with new segments being created as the player proceeds and old ones – removed.

The entire system is designed with ease of use and performance in mind while keeping the flexibility and the possibility for extensions high. By default, all extrusion operations are done in a separate thread so that the game experience remains smooth.

Forever is based off another Dreamteck plugin called Dreamteck Splines and if both plugins are installed, Forever offers the possibility to also bend splines if a level segment contains such – this is handy for creating enemies, constrained to paths within a segment or even procedural meshes within the generated procedural segments. In order to make use of the Dreamteck Splines support in Forever, the “DREAMTECK_SPLINES” scripting define needs to be added inside the Project Settings / Player settings.

## 1.1. How Everything Works

During runtime, level segment prefabs are spawned and stitched together in order to form the level. The order of spawning can be random or user-defined. Each segment is instantiated (Object Pooling is not used because the segments are modified) and destroyed when no longer needed. The level segments are created by hand in the Unity editor like any other object.

A spline path is generated for each spawned segment and each segment is extruded along its own path.

How the path is generated is defined by special scripts called Path Generators. The path generators hold the logic for path generators. Forever comes with a couple of Path Generator scripts but users can easily create their own path generation logic by inheriting from the **LevelPathGenerator** class.

## 1.2. Package Contents

- Forever
  - Editor
  - Gameplay
  - Level Generator
  - Level Segments
  - Examples.unitypackage
- Splines
- Utilities

The Splines and Utilities folders are required by Forever and their contents should not be removed.

Inside the Forever folder, there is a unitypackage file containing example projects. They are a good starting point for beginners and if examples are not needed, then this package can be safely deleted.

The Gameplay folder contains logic for running along the generated levels. The classes inside are self-dependent so if the built-in runner logic is not needed for the purpose of the project, this folder can also be removed safely. However, it is recommended that at least the ProjectedPlayer.cs file is kept in the project because it acts as a communicator component between the Level Generator and the player – it lets the Level Generator know where the player is so that new segments can be generated and old ones – removed. In conclusion – even if a custom player script is used to traverse the levels, the logic of the ProjectedPlayer.cs is a must have in order for the Level Generator to work properly.

## 1.3. The Level Segment

The Level Segments are the building blocks of each procedural level that Forever generates. They are simply prefabs which have the Level Segment component attached to them.

During runtime, the Level Generator spawns these prefabs and builds the entire level out of them.

A Level Segment can either be Extruded or Custom. Extruded segments will be bent along the generated random path while custom segments will not be altered. A level could be built entirely from Custom or Extruded segments or a combination of the two.

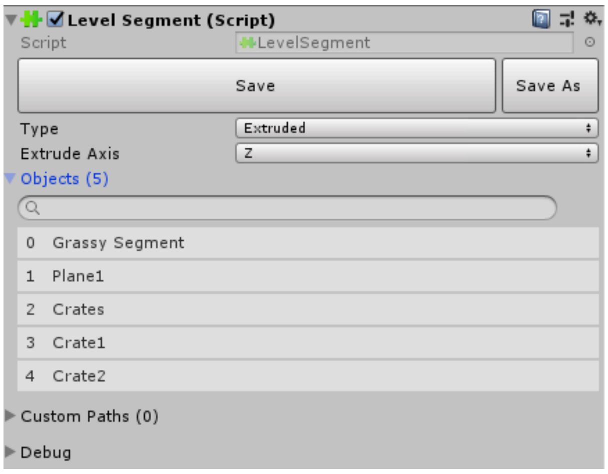
Each Level Segment holds definitions for how each of its children is handled during the extrusion process. Objects can be offset, rotated and scaled based on the generated path, or they can be ignored completely.

### 1.3.1. Creating Level Segments

To create a Level Segment, create a Game Object in the scene and add the Level Segment component to it. Inside the inspector two buttons will appear: SAVE AND SAVE AS BUTTONS IMAGE

Level Segment prefabs should be created and applied using these two buttons instead of clicking the “Apply” button in the prefab menu at the top. If a Level Segment prefab is saved like a regular prefab, it will not be able to pack the needed information for the generation and this may result in an error during runtime.

### 1.3.2. Basic Segment Setup



Type: Extruded
Extrude Axis: Z
Objects (5)

0 Grassy Segment
1 Plane1
2 Crates
3 Crate1
4 Crate2

When a segment is created, it first needs to be set up properly before it can be used in the level generation process.

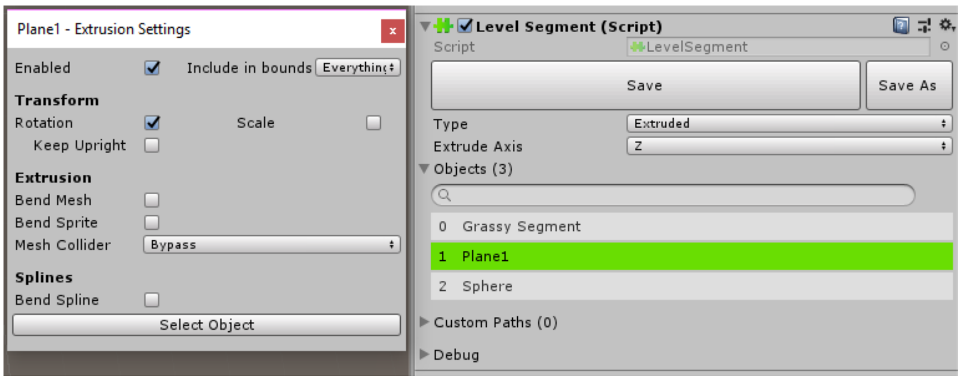
The Type of the segment is the first most important setting that needs to be made. By default, segments are set to “Extrude” which means that the segment will be bent along the generated level path.

All segments in a **level sequence** need to be able to snap seamlessly with each other and extruded level segments need to be designed in a straight line – the bending and turning will come from the level generation during runtime.

After adding a Level Segment component to an object, it will also automatically create a Rigidbody and set it to kinematic. This is done to improve the performance of the level generation and the **Floating Origin** work.

### 1.3.2.1. Extruded Segment Type

When "Extrude" is selected, an object list will appear inside the inspector. This is the list of all game objects in the level segment. Clicking on each one will bring up the Extrusion Settings window:



Plane1 - Extrusion Settings

Enabled ✓ Include in bounds: Everything

**Transform**
Rotation ✓ Scale ☐
Keep Upright ☐

**Extrusion**
Bend Mesh ☐
Bend Sprite ☐
Mesh Collider: Bypass

**Splines**
Bend Spline ☐

Select Object

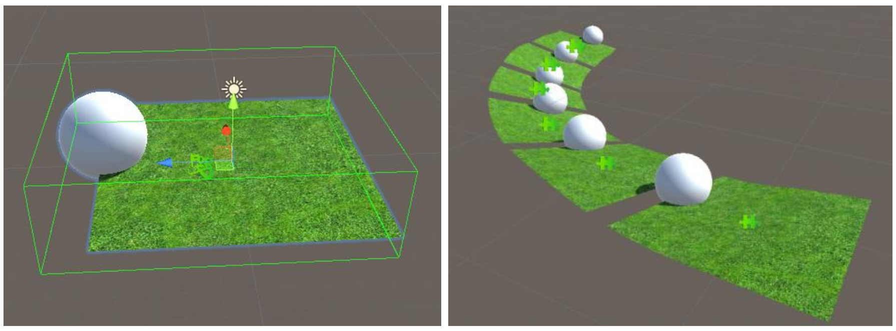
Level Segment (Script)
Script: LevelSegment
Save | Save As
Type: Extruded
Extrude Axis: Z
Objects (3)
0 Grassy Segment
1 Plane1
2 Sphere
Custom Paths (0)
Debug

From there, the object can be set up for the extrusion process. Multiple objects can be selected by holding down L. Shift or Control/Command.

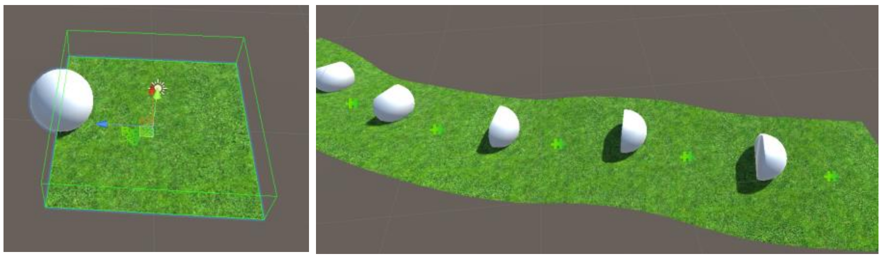
- Enabled – Should the object be included in the extrusion process?
- Include in Bounds – What part of the object to include inside the bounds calculation.
- Rotation – Should rotation be applied?
- Keep Upright – If true, the object will be rotated but will be also leveled along a specific vector.
- Scale – Should scale be applied?
- Bend Mesh – Should the mesh of the object be bent along the spline?
- Mesh Collider – How should mesh colliders be handled?
- Bend Sprite (Experimental) – attempts to bend 3D sprites.

Extruded segments will have their bounds drawn inside the editor. Editor. The bounds must cover the segment's main geometry completely in order for the segment to be generated correctly. When an object's mesh is set to bend and the object itself is included in the bounds, the bounds will automatically expand to fit the mesh. However, if the level segment doesn't have a single big mesh that covers the whole segments, the bounds could be expanded artificially using empty game objects and putting them at the far ends of the segment.

Where the bounds end defines where the next segment will start so having bounds which are extending outside of the visual area will create gaps:



In the same time, if an object's mesh is set to extrude but the object itself is configured to not be included inside the bounds, upon extrusion, the object's mesh will be flattened by the bounds:

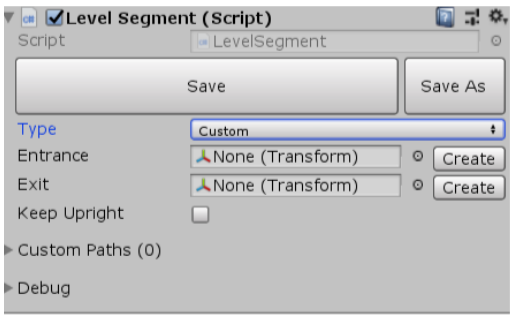


This is why it is important to include the meshes inside the bounds for objects that are meant to bend.

It is, however, useful to leave a mesh out of the bounds calculation and still have it bent. Such is the case for detail meshes like grass and other vegetation which might protrude out of the segment bounds a little. Having the grass excluded from the bounds calculation will make sure it is tightly packed with everything else and no gaps between segments appear.

Extruded segments also have the Axis setting which can be set to either X, Y or Z. The axis defines along which axis the segment will be extruded. For 3D games, the Z axis is usually the one used and for 2D games, the X axis is used. The Y axis could work for an endless runner game where the player is constantly falling down or climbing a tower.

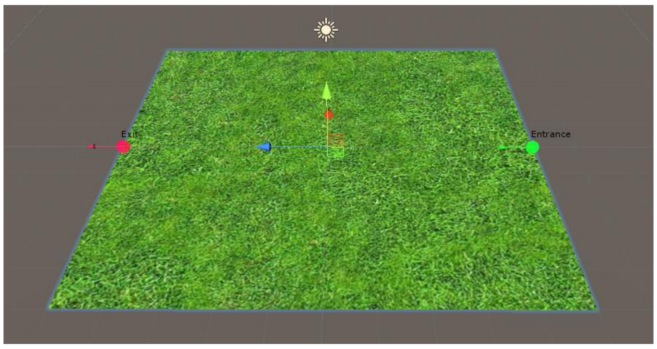
### 1.3.2.2. Custom Segment Type


Level Segment (Script)
Script: LevelSegment
Save | Save As
Type: Custom
Entrance: None (Transform) | Create
Exit: None (Transform) | Create
Keep Upright: ☐
Custom Paths (0)
Debug

The custom segments are segments which are simply spawned and not modified. In Custom mode the object list is gone and in its place, there are two Transform reference fields:

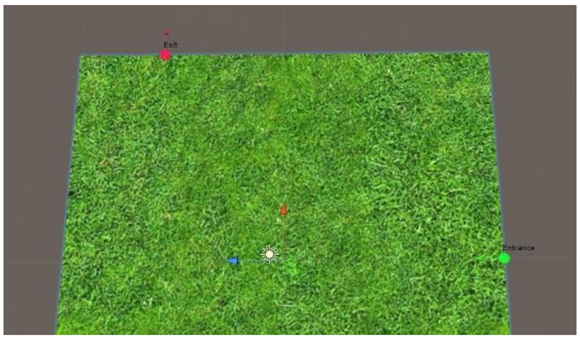
- Entrance – The point of entry
- Exit – The point of exit

Each custom segment must have those two assigned. When generating the level, the Level Generator will align the custom segments based on where their Entrance transform is positioned and how it is oriented. The Exit transform is used to tell the Level Generator where and how to place and orient the segment that comes after that. The proper placement for these two objects would be at the edges of the segment where it needs to connect with the other segments.



The Keep upright option is useful for when the level segment needs to be leveled perfectly regardless of the orientation of the previous segment. Otherwise, the custom segment will be oriented to match the previous segment's direction. Usually it is recommended to use "Keep Upright" for custom segments.

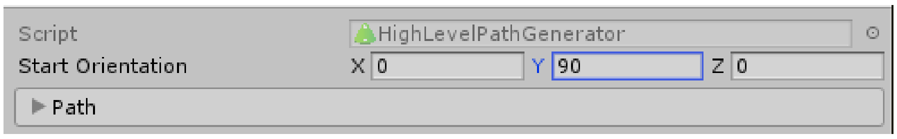
Custom segments do not necessarily have to go in a straight line. The Entrance and the Exit can point in different direction. In fact, custom segments are very useful when creating 90-degree turns. Having non-straight segments, however, can cause a couple of issues.



If three 90-degree turns (turning in the same direction) spawn one after each other, it is possible that the segments will intersect with existing previous segments. In order to prevent that either the sequences have to be set up in such a way that such segments spawn rarely or a custom segment shuffle approach must be used (see Segment Shuffle).

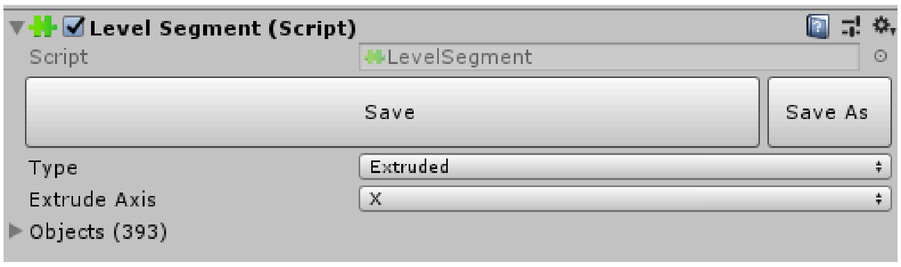
### 1.3.3. Setting Up Extruded Segments In 2D

When working in 2D, there are a few things that need to be considered. First of all, the path needs to be set up to generate along the X axis. This can be done either by rotating the Level Generator so that its local Z axis aligns with the world X axis (rotate it 90 degrees along the Y axis) or by setting the start orientation of the path generator to 0, 90, 0:



Script: HighLevelPathGenerator
Start Orientation: X 0 Y 90 Z 0
Path

Next, each 2D segment that is using Extrusion needs to be configure to extrude along the X axis:



Level Segment (Script)
Script: LevelSegment
Type: Extruded
Extrude Axis: X
Objects (393)

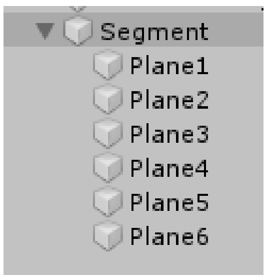
Sprite renderers can be set up to bend along the paths too. Select an object from the objects list and toggle “Bend Sprite” in the Extrusion Settings window. This will convert the Sprite Renderer into a configuration of a Mesh Filter and a Mesh Renderer that uses the sprite material to display the sprite.

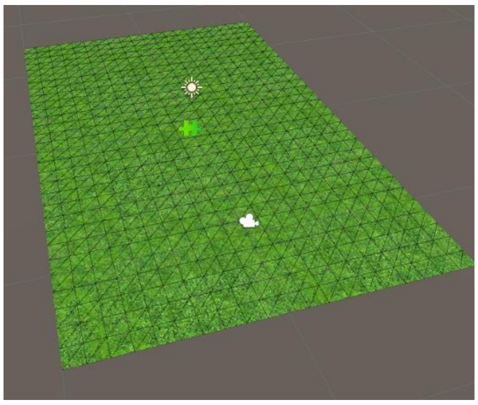
Currently only Sprite Renderers can be set to bend along the path. Tilemaps and Sprite Shape Renderers are still unsupported. Polygonal collider bending support will come in version 1.01.

For Custom type segments, there are no additional steps that need to be made – 2D segments of custom type will work out of the box.

### 1.3.4. Example Setup for Extruded Segments

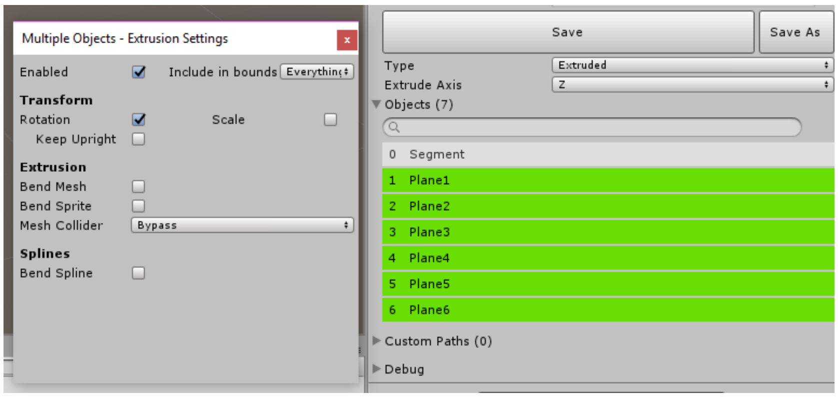
- Create an empty Game Object and name it “Segment”.
- Create a couple of Plane objects. Snap them together and parent them under the empty Game Object:


```
▼ Segment
    Plane1
    Plane2
    Plane3
    Plane4
    Plane5
    Plane6
```



- Add the Level Segment component to the Segment object.
- Inside the Level Segment component, make sure that the type is set to “Extruded” and the Axis is set to Z.

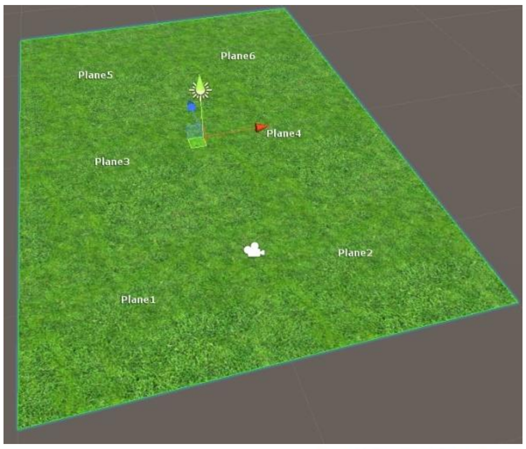
- Expand the Objects foldout, click on Plane1 and then, holding shift, click on Plane6 inside the Level Segment object list:


```
Multiple Objects - Extrusion Settings                    x
Enabled        ☑   Include in bounds  Everything
Transform
Rotation       ☑        Scale   ☐
  Keep Upright ☐
Extrusion
Bend Mesh      ☐
Bend Sprite    ☐
Mesh Collider  Bypass
Splines
Bend Spline    ☐
```

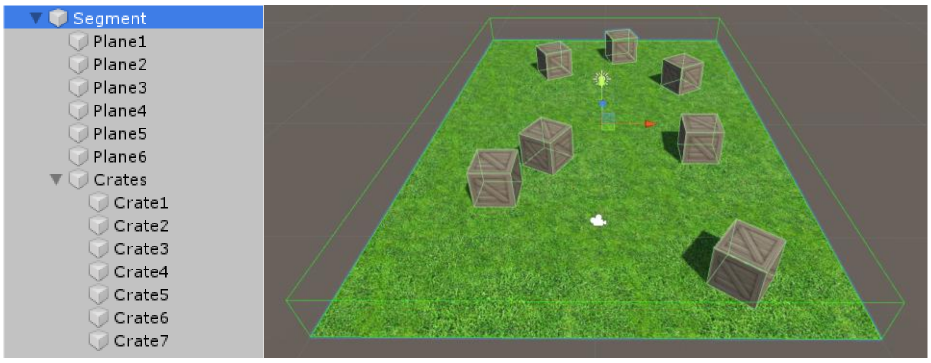
```
Save                                   Save As
Type              Extruded
Extrude Axis      Z
▼ Objects (7)
0  Segment
1  Plane1
2  Plane2
3  Plane3
4  Plane4
5  Plane5
6  Plane6
▶ Custom Paths (0)
▶ Debug
```

- Inside the extrusion settings window, check “Bend Mesh” and make sure that “Include in Bounds” has the “Mesh” flag checked so that the meshes of the planes are included in the bounds. If the meshes are properly included, the entire segment will have a green outline:

  These are the calculated bounds of the segment.

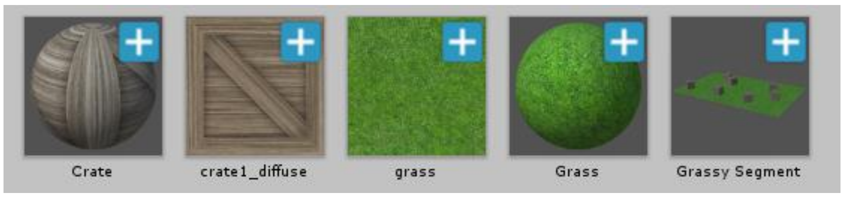


- Set the “Mesh Collider” field to “Copy”. This will make sure that the mesh colliders of the Plane objects will copy the extruded mesh and the collision will correspond to the visuals.
- Add other level design objects under the Segment game object



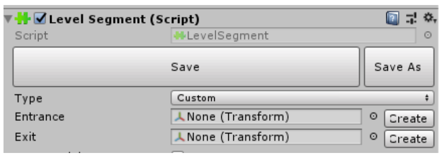
In this case, crates are added to the segment. The crates should not have their meshes bent so “Bend Mesh” is not checked for the crates.

- Click the “Save” button inside the Level Segment component inspector and specify a place inside the project where the segment should be saved.
- A Level Segment prefab will be created which can now be used inside the Level Generator.



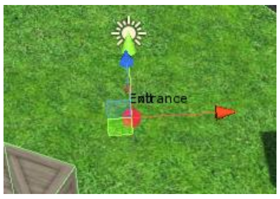
## 1.3.5. Example Setup for Custom Segments

- Duplicate the existing level segment prefab from the previous example and place an instance of it inside the scene.
- Set the type of the Segment to “Custom” from the Level Segment component inspector.



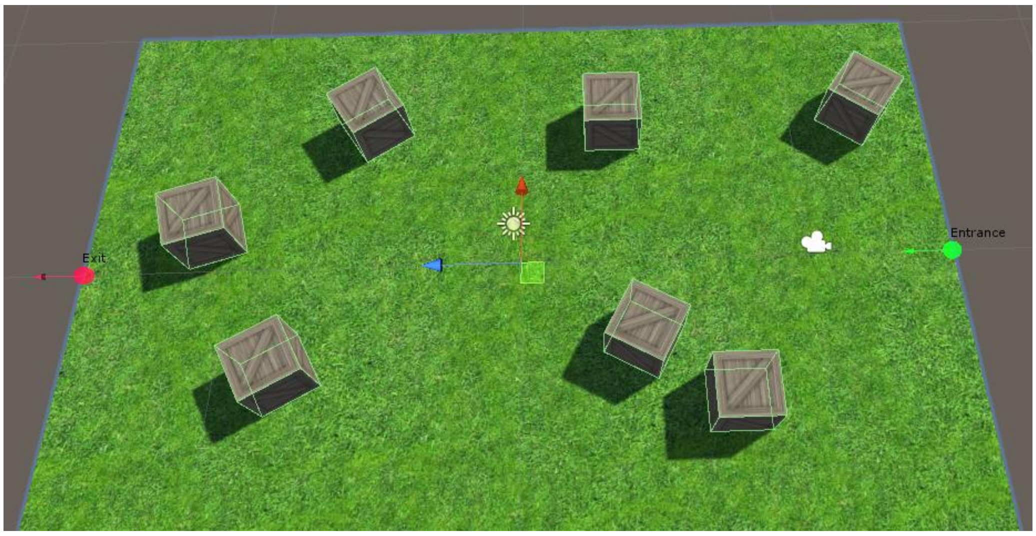
Script: LevelSegment
Type: Custom
Entrance: None (Transform)
Exit: None (Transform)

- Click both “Create” buttons next to the “Entrance” and “Exit” reference fields to create an entry an exit point for the custom segment. This will create two new children at the 0,0,0 local coordinates of the segment



- Select the Entrance object from the hierarchy and snap it to the start of the segment by holding V and moving the position handle.
- Do the same for the Exit object.

- Upon selecting the segment game object again, the Entrance and Exit objects will be visualized with green and red.



- Toggle “Keep Upright” if desired and Save the segment

### 1.3.6. Editing Existing Level Segments

To edit an existing Level Segment prefab, simply put it in the scene and click “Save” after editing is done.

#### 1.3.6.1. Editing Level Segment Prefabs in Unity 2018.3 +

If using Unity 2018.3 or above, the level segment prefabs can be edited inside the prefab mode safely. Once the Level Segment prefab has been created using the “Save As” button, it is safe to edit inside the prefab mode and packing operations will be performed automatically upon exiting prefab mode.

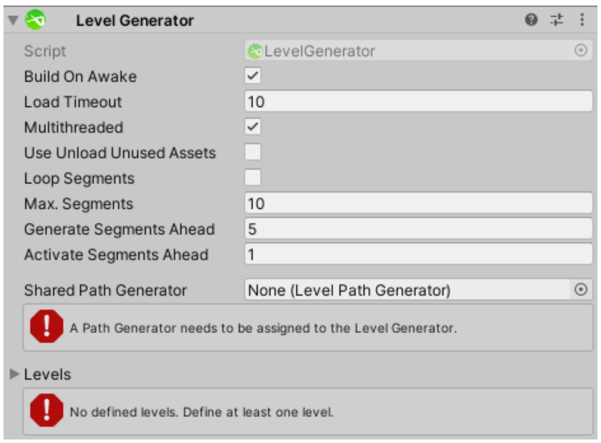
## 1.4. The Level Generator

The Level Generator is the main brain behind the generation of the levels. It is a component that uses the Singleton design pattern (meaning that there should be only one active Level Generator per scene) and it is recommended to be placed in a separate Game Object, dedicated only to it. All generated level segments during runtime will become children of that object.

To create a Level Generator, add the Level Generator component to an empty game object in the scene.

The Inspector of a newly created Level Generator will show two errors:

- A Path Generator needs to be assigned
- No defined levels.



Level Generator

| | |
|---|---|
| Script | LevelGenerator |
| Build On Awake | ✓ |
| Load Timeout | 10 |
| Multithreaded | ✓ |
| Use Unload Unused Assets | ☐ |
| Loop Segments | ☐ |
| Max. Segments | 10 |
| Generate Segments Ahead | 5 |
| Activate Segments Ahead | 1 |
| Shared Path Generator | None (Level Path Generator) |

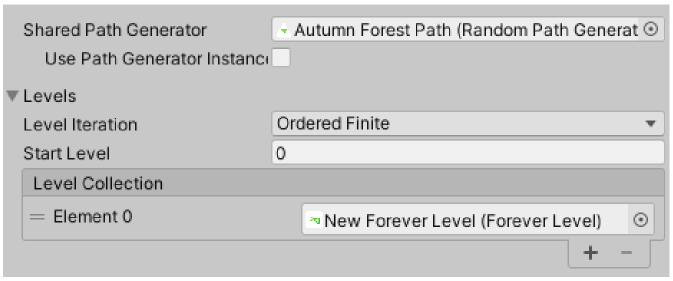
A Path Generator needs to be assigned to the Level Generator.

Levels

No defined levels. Define at least one level.

The Level Generator needs to have a Path Generator assigned to it so by assigning an existing Path Generator asset (see [The Path Generator]), this error will be fixed.

The Level Generator is based on levels. Each level contains a collection of sequences which contain a collection of level segments. In order for the Level Generator to work, it needs to have at least one level defined. To create a new level asset, select **Assets/Create/Forever/Level.** In the Level Generator component, expand the Levels foldout and link the newly created level asset into the Level Collection list:



| | |
|---|---|
| Shared Path Generator | Autumn Forest Path (Random Path Generat |
| Use Path Generator Instanc | ☐ |

Levels

| | |
|---|---|
| Level Iteration | Ordered Finite |
| Start Level | 0 |

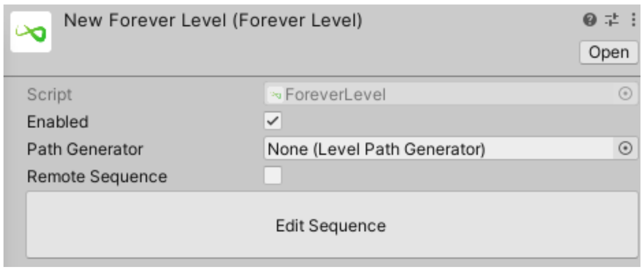
Level Collection

| | |
|---|---|
| Element 0 | New Forever Level (Forever Level) |

### 1.4.1. Managing Levels

Levels in Forever are comprised of sequences. Each sequence is a collection of level segments. After the given sequence has finished spawning segments, the level will move onto the next existing sequence until all sequences are done.

A level needs to have at least one sequence in order to be able to spawn Level Segments. To open the sequence editor for a level, select the level asset in the project and click the "**Edit Sequence**" button in the inspector.



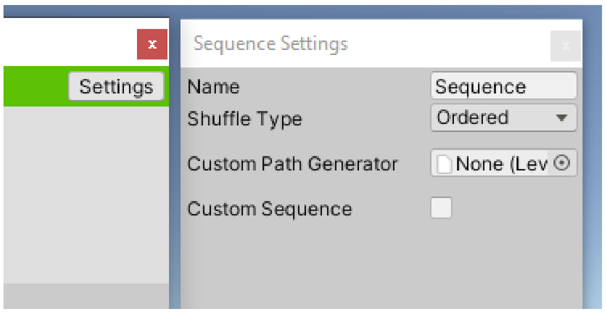
New Forever Level (Forever Level)

| | |
|---|---|
| Script | ForeverLevel |
| Enabled | ✓ |
| Path Generator | None (Level Path Generator) |
| Remote Sequence | |

#### 1.4.1.1. Creating and Managing Sequences

To create a new sequence inside a given level, click the "Add Sequence" button in the Level Window. A new empty sequence will be created. To add level segments to the sequence, drag and drop the prefabs inside the sequence. Note that only prefabs with the Level Segment component can be added to sequences.

When a sequence has Level Segments assigned to it, these level segments can be re-arranged by dragging and dropping them. This is useful for when the sequence type is set to "Ordered".



Sequence Settings

| | |
|---|---|
| Name | Sequence |
| Shuffle Type | Ordered |
| Custom Path Generator | None (Lev |
| Custom Sequence | |

Each sequence can be configured via the **Settings** button at the top right corner.

There are three types of sequences:

- Ordered
- Random by Chance
- Shuffled
- Custom

**Ordered** will spawn all segments inside the sequence in the order they are added and then the level will move onto the next sequence.

**Random by Chance** will spawn N random segments (N - defined by the Count property).

**Shuffled** will create a randomly shuffled list of the segments and will spawn all segments without repeating

**Custom** allows for the developer to write their own algorithm for picking segments (see Segment Shuffle).

Right-clicking the green header of a sequence will show a context menu with options for management like renaming, reordering and removing.

**Note – a sequence can be set to loop forever if set to "Random" and the count of the segments is set to 0. In this case, the sequence will never finish unless interrupted with a script**.

### 1.4.2. Level Generator Properties

- **Build on Awake**

As the name suggests, if this is toggled, the level generator will start building the level as soon as it becomes active in the scene. If not toggled, then the level generation can be started by calling **LevelGenerator.instance.StartGeneration();**

- **Load Timeout**

How long should the Level Generator wait for a level to load. This is used for edge cases where there is a problem loading the level scene.

- **Multithreaded**

Should the generation process be carried out on another thread? Usually this should be set to True except for WebGL applications where multithreading is not supported. Having the generation logic run on a separate thread will greatly improve the performance.

- **Use Unload Unused Assets**

If set to true, Resources.UnloadUnusedAssets() will be called after each level is done generating. This can cause performance hiccups however. Forever has a built-in resource management system that is faster and could be used instead.

- **Loop Segments**

Use this option when creating looped tracks with the Custom Path Generator. It will tell the runners and projected player that the level should start over once the last segment is reached.

- **Max. Segments**

Defines the maximum number of level segments that can exist in the level. Once started, the Level Generator will keep adding new segments ahead until the Max. Segments number is reached. At this point, the Level Generator will start removing old segments to keep the count of the segments in check.

- **Generate Segments Ahead**

Defines how many level segments should be generated ahead of the current segment. This is needs to be coherent with the view distance in the game. If the segments in the project are big (long), then generation could add just a couple segments ahead, otherwise, more segments will be needed.

- **Activate Segments Ahead**

Defines how many of the generated segments head should also be activated. When a segment is generated, its colliders get deactivated in order to save performance. When a segment is near the player, it can be activated so that collision starts to work. This value needs to be between 1 and "**Generate Segments Ahead**".

- **Level Iteration**

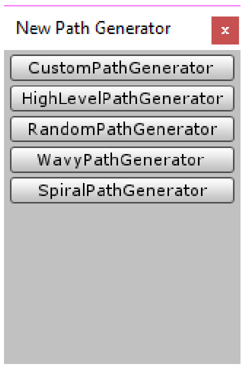
Defines how the Level Generator will iterate through the linked levels inside the Level Collection.

- **Ordered Finite** – goes through all levels in order and stops generating after the last level
- **Ordered Clamp** – goes through all levels in order and repeats the last level indefinitely
- **Ordered Loop** – goes through all levels in order and loops back when the list ends
- **Random** – picks random levels to generate indefinitely
- **Single Repeat** – will generate the level defined by "Start Level" indefinitely
- **Single Finite** – will generate the level defined by "Start Level" a single time and will stop

## 1.5. The Path Generator

The Path Generators are scriptable objects which are created as assets inside the Project and hold logic for path generation. In order to generate any level, there needs to be a Path Generator assigned to the Level Generator. The Path Generators generate paths for each segment separately in the form of Spline control points. The generated paths in Forever are interpolated using either the Bezier spline function or are just linear.

To create a Path Generator, right click inside the Project panel -> Create/Forever/Path Generator. This will bring up an editor window with the available Path Generators.



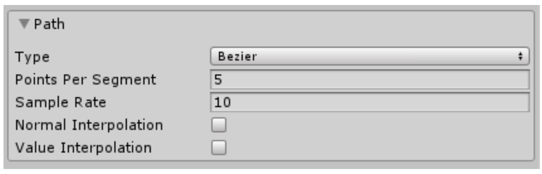
More path generators can be created by deriving from the LevelPathGenerator class (see Extending the Functionality for more information). As soon as a new Path Generator class is created, it will be automatically indexed inside the New Path Generator window.

Once a Path Generator is created, it can be configured by selecting it in the Project panel and tweaking its values in the inspector.

Path Generators can be switched at any time during the level generation and the level path will proceed to generate according to the current Path Generator that is in use.

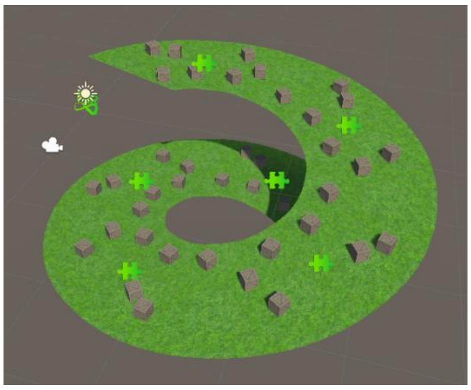
### 1.5.1. Setting up a Path Generator

All path generators come with a core setting for the generated path. This defines what the generated path per each segment will be – not the entire level path.



▼ Path

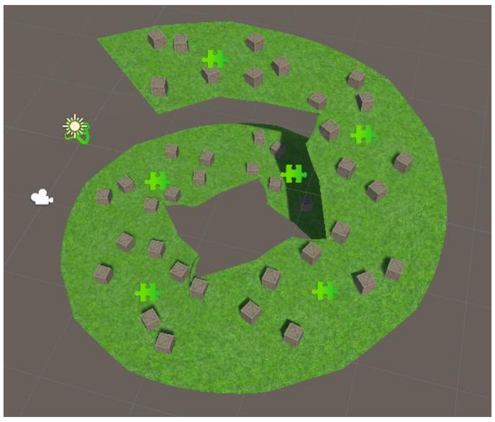
| | |
|---|---|
| Type | Bezier |
| Points Per Segment | 5 |
| Sample Rate | 10 |
| Normal Interpolation | ☐ |
| Value Interpolation | ☐ |



Since the generated paths in Forever are actually splines, two algorithms are supported – Bezier and Linear. All Path Generators in Forever are written to generate Bezier tangents for the paths but if the Type of the path is set to "Linear" the Bezier interpolation will not be used. For example, this is how the Spiral path generator works when the generated path is set to Bezier:

And if one of the generated segments is selected, a debug visualization will display how the point's tangents are generated:

If the path is set to Linear, the result of the generation will bypass the tangents:



It does not look good in the case of the spiral, but there are cases where this effect could be sought after.

The Points Per Segment field defines how many path points will be generated per segment. More points are better for a more detailed generation but the average project should not require more than 3 to 5 points per segment.

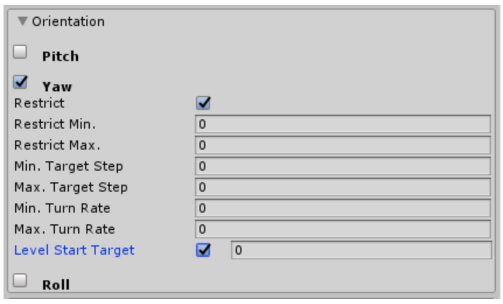
The Sample Rate defines the rate at which the generated spline will be sampled between each two points. 10 means 10 samples between points and with a segment with three points, that would mean a total of 20 samples per segment. Increasing the samples would increase the smoothness of extrusion but usually 10 samples should be enough.

The Normal and Value interpolations define how path Normals, Colors and Values will be interpolated from point to point. By default, a linear interpolation is used but in order to increase smoothness, curves can be used to define easing.

### 1.5.2. Random Path Generator

This is the most versatile path generator that comes by default with the Forever package. It can be used for the generation of a wide range of paths and along with directions, it can also randomize path sizes, colors and normals.

The first settings panel for the Random Path Generator is the Orientation panel. This is where the general path behavior is defined – where the path can go and how it behaves. This is defined by the pitch, yaw and roll properties. By default, all three are disabled which will produce a path going in a straight line. When one is enabled, additional fields for setup are shown. All values inside the orientation panel are in signed degrees.



▼ Orientation

☐ **Pitch**

☑ **Yaw**

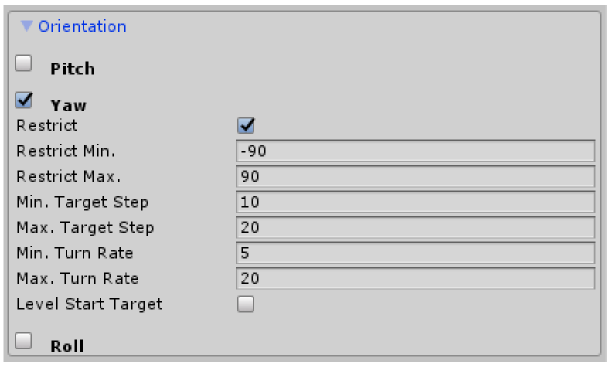
| | | |
|---|---|---|
| Restrict | ☑ | |
| Restrict Min. | 0 | |
| Restrict Max. | 0 | |
| Min. Target Step | 0 | |
| Max. Target Step | 0 | |
| Min. Turn Rate | 0 | |
| Max. Turn Rate | 0 | |
| Level Start Target | ☑ | 0 |

☐ **Roll**

When the orientation of the Random Path Generator reaches the target angle, the algorithm finds a new target angle to move towards. This new target angle is defined by the restrictions (if enabled) and the target step angles which define the minimum and maximum random angle to add to the current angle to produce the new target.

- Restrict – should the orientation of the path be restricted?
- Restrict Min. – minimum allowed angle (-45 for example)
- Restrict Max. – maximum allowed angle (60 for example)
- Min. Target Step – minimum angle to change the target with (should be positive)
- Max. Target Step – maximum angle to change the target with (should be positive)
- Min. Turn Rate – minimal angle to turn the path towards the target angle with (should be positive)
- Max. Turn Rate – maximum angle to turn the path towards the target angle with (should be positive)
- Level Start Target – target angle for when the level loads

Here is an example setup for the path generator along with the produced result:

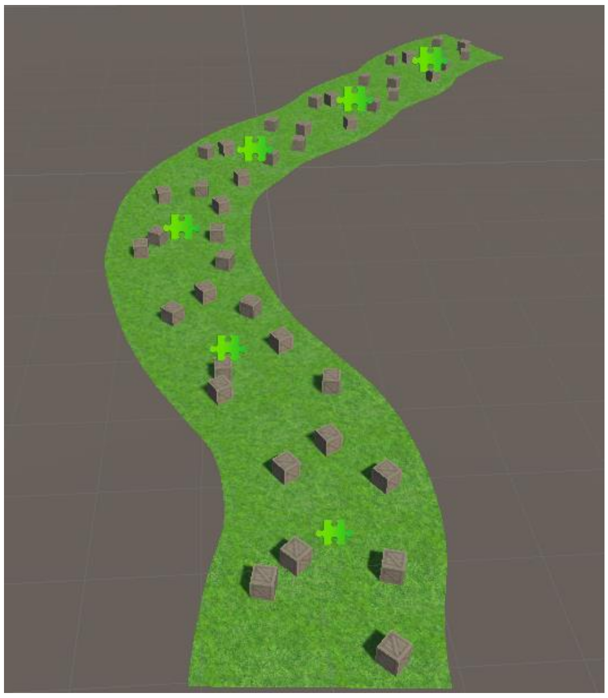


Orientation

- [ ] **Pitch**
- [x] **Yaw**

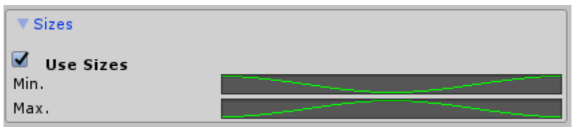
| | |
|---|---|
| Restrict | ✓ |
| Restrict Min. | -90 |
| Restrict Max. | 90 |
| Min. Target Step | 10 |
| Max. Target Step | 20 |
| Min. Turn Rate | 5 |
| Max. Turn Rate | 20 |
| Level Start Target | ☐ |

- [ ] **Roll**



The Sizes panel provides settings for randomizing the sizes of the generated path points. Path point sizes affect how the geometry is generated and if scaling is enabled for objects, the objects' transforms will be scaled too.

The Sizes panel has two curve fields for the minimum and maxium size. The curves define how the points will be scaled generating from the start towards the end of the segment. This is an example setup:

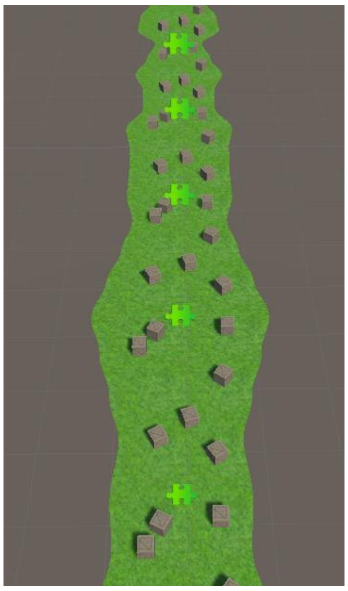


Sizes

- [x] **Use Sizes**

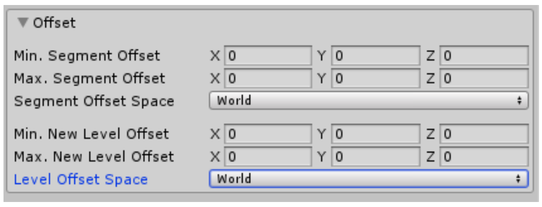
Min.

Max.



It is a good idea to enable value interpolation in the Path panel settings when using sizes and colors and use an ease-in, ease-out curve.

The Colors panel works the same way as the sizes panel but alters the point colors. It uses two gradients as opposed to two curves. In order to see applied colors, the extruded objects should also be set up to receive color from the Extrusion Settings window and should use a shader which supports vertex colors.



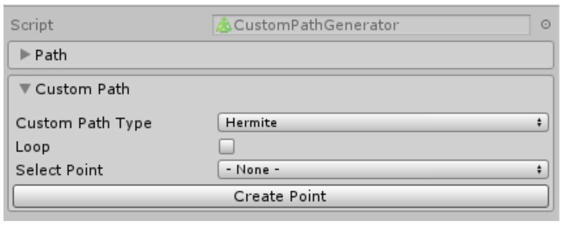
▼ Offset

| | | | |
|---|---|---|---|
| Min. Segment Offset | X 0 | Y 0 | Z 0 |
| Max. Segment Offset | X 0 | Y 0 | Z 0 |
| Segment Offset Space | World | | |
| Min. New Level Offset | X 0 | Y 0 | Z 0 |
| Max. New Level Offset | X 0 | Y 0 | Z 0 |
| Level Offset Space | World | | |

The Offset panel ads offsets to the generated segments. There are two types of offsets – segment offset which applies an offset to each segment and new level offset which applies an offset to the first segment from each new level. The space property defines how the offset is calculated. World will add a world offset while Self will add an offset that is local to the path.

### 1.5.3. Custom Path Generator

The Custom Path Generator is the most unique path generator that comes out of the package. First of all, it is meant for finite levels. So in order to use it, the Level Generator needs to be set to “Finite”.



Script CustomPathGenerator

▶ Path

▼ Custom Path

| | |
|---|---|
| Custom Path Type | Hermite |
| Loop | ☐ |
| Select Point | - None - |

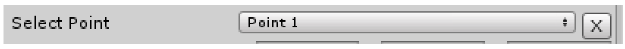
Create Point

This generator uses a user-defined path which is then broken into pieces and fed to all segments. The user-defined path is a spline which can either be Hermite, Bezier, BSPline or Linear. The path can either be created in the editor, defined by code during runtime or a combination of the two – user-defined with runtime randomization.

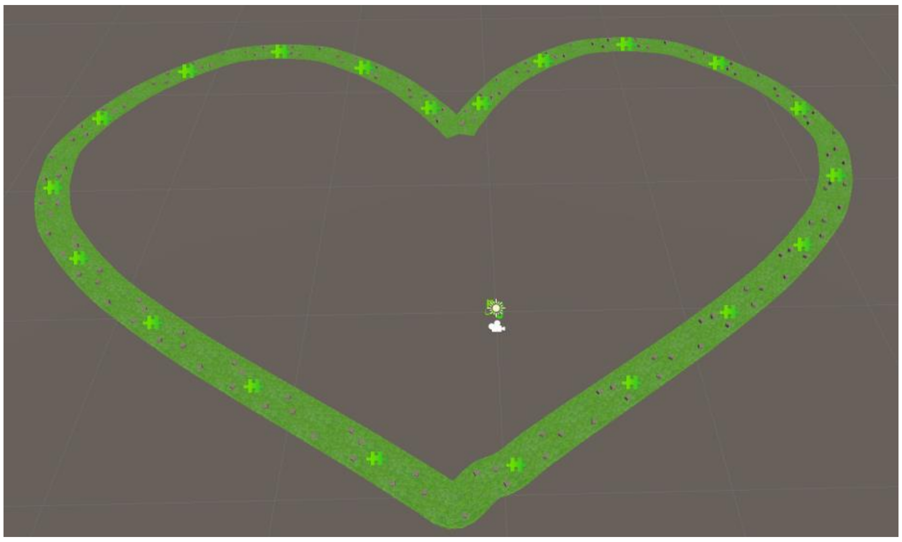
To create a path for the Custom Path Generator, select the generator in the project tab and focus the origin of the scene in the scene view. Click the “Create Point” button twice to create two spline points and move the second one away in the scene view.

Created points can be selected by clicking on them in the scene view or using the dropdown menu inside the custom path panel.

If the path has 4 or more created points, then it can be set to loop. Looped paths are useful for the generation of racing tracks.

Deleting points is done using the X button next to the selected point in the inspector 

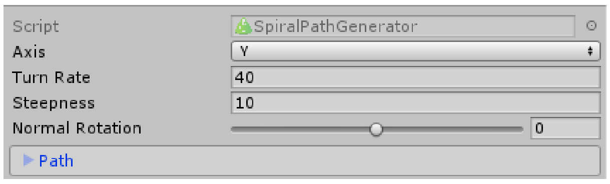
Select Point: Point 1 X



Segments, generated using the Custom Path Generator will get stretched along the generated spline path so it is important to set the segment count in the Level Generator properly.

### 1.5.4. Spiral Path Generator

The Spiral Path Generator does what its name suggests – it generates a spiral path. It is simple to set up as it only requires a few properties:



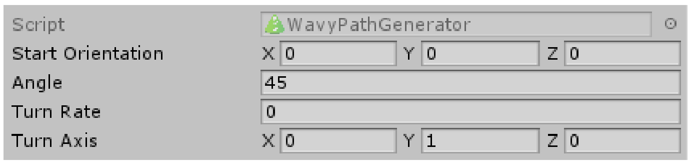
```
Script            SpiralPathGenerator
Axis              Y
Turn Rate         40
Steepness         10
Normal Rotation   0
▶ Path
```

- Axis – The axis, around which the spiral will revolve
- Turn Rate – turn rate of the axis per generated spline point. More points per segment will make a sharper turn
- Steepness – how steep the spiral is in degrees

- Normal Rotation – rotates the normal of the points around the spiral. For example, in order to produce a wall-ride type of game, set the normal rotation to either -90 or 90 and the entire surface will rotate

### 1.5.5. Wavy Path Generator

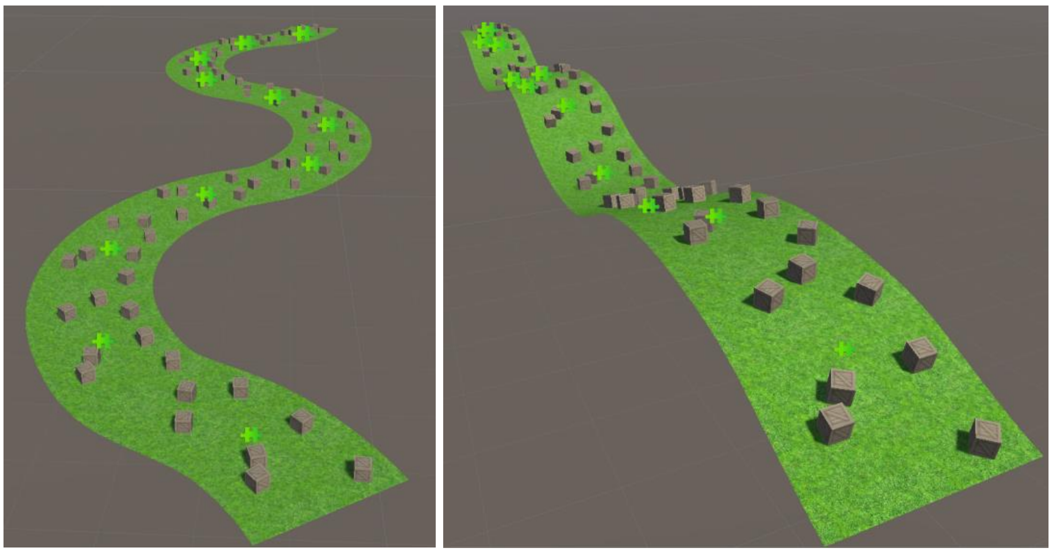
The Wavy Path Generator generates a zig-zag type of path going in one general direction.



| | |
|---|---|
| Script | WavyPathGenerator |
| Start Orientation | X 0 Y 0 Z 0 |
| Angle | 45 |
| Turn Rate | 0 |
| Turn Axis | X 0 Y 1 Z 0 |

The Angle property defines how much the path will be deviating from the initial direction and the Turn Rate defines at what rate this will be happening. Higher turn rate means sharper turns.

The Turn Axis property defines the axis around the turns will be made. A turn axis of 0,1,0 means the path will make turns around the Y axis and will be flat.

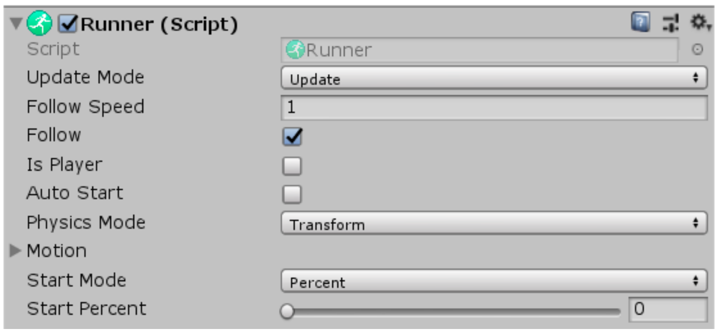


## 1.6. Gameplay

Forever provides a package of behaviors for following the generated level segments out of the box. These are components which take advantage of the system's API and make it easy to set up a working prototype within minutes with only some simple coding needed to control the offset of the player while following.

### 1.6.1. The Runner Component

There are three Runner components in Forever – Runner, LaneRunner and CustomLaneRunner. Both LaneRunner and CustomLanerRunner derive from Runner and add lane switching functionality.



Runner (Script)

| | |
|---|---|
| Script | Runner |
| Update Mode | Update |
| Follow Speed | 1 |
| Follow | ✓ |
| Is Player | ☐ |
| Auto Start | ☐ |
| Physics Mode | Transform |
| ▶ Motion | |
| Start Mode | Percent |
| Start Percent | 0 |

To create a Runner, add the Runner component to the Player object or another existing object in the scene. The Runner component is found in the component menu as "Basic Runner".

All Runners have a Follow Speed property as well as a "Follow" toggle which defines if the Follower should run the logic for following.

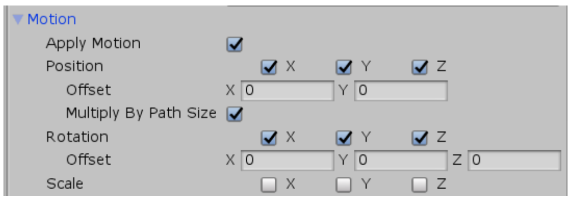
Only the Player object should have the **Is Player** property toggled as it defines whether or not the Runner should notify the Level Generator of its presence. The Runner which has the Is Player property set to true, will cause the Level Generator to create new segments as it follows.

Physics Mode defines the target which will receive the follow result – Transform, Rigidbody or Rigidbody2D

Start Mode defines where the runner will start along the entire available generated level. Available modes are Percent, Distance and Project.

- Percent defines a percent [0-1] along the level where the Runner will start.
- Distance calculates the distance in world units from the start of the generated level to the point where the Runner should start
- Project will attempt to find the closest available point along the generated level and start the Runner from there

The Motion foldout controls how the running is applied to the target. If Apply Motion is not toggled, the follow result will not be applied to the target at all. However, the following logic will still run and the Runner's public result property which holds the follow information will update. This is useful if applying the motion needs to happen in another component.



Motion

| | | | |
|---|---|---|---|
| Apply Motion | ✓ | | |
| Position | ✓ X | ✓ Y | ✓ Z |
| Offset | X 0 | Y 0 | |
| Multiply By Path Size | ✓ | | |
| Rotation | ✓ X | ✓ Y | ✓ Z |
| Offset | X 0 | Y 0 | Z 0 |
| Scale | ☐ X | ☐ Y | ☐ Z |

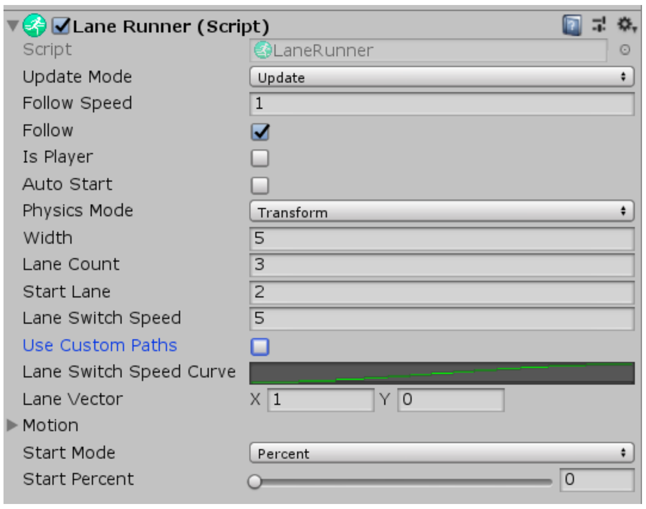
There is a position offset and a rotation offset property. Both work locally to the current follow result meaning that X is always perpendicular to the current segment path and Y always aligns with the path normal.

To implement the player's sideways movement, these two offsets need to be modified with a script.

### 1.6.2. Laner Runner

The Lane Runner is an abstraction of the Runner and adds lane switching functionality without the need of additional programming.

The Lane Runner adds additional parameters for defining the lanes. It uses the **Width** property to define the total span of the lanes and the **Lane Count** property to define how many lanes there are. For example, if the Width is set to 3 and the Lane Count is set to 3, this means that there will be three lanes, each one with the width of 1 unit. Start Lane defines which of the lanes will be the player's starting lane and Lane Switch speed along with the Lane Switch Speed Curve define the speed at which the Runner will switch between the lanes.



Lane Runner (Script)

| | |
|---|---|
| Script | LaneRunner |
| Update Mode | Update |
| Follow Speed | 1 |
| Follow | ✓ |
| Is Player | |
| Auto Start | |
| Physics Mode | Transform |
| Width | 5 |
| Lane Count | 3 |
| Start Lane | 2 |
| Lane Switch Speed | 5 |
| Use Custom Paths | |
| Lane Switch Speed Curve | |
| Lane Vector | X 1 Y 0 |
| Motion | |
| Start Mode | Percent |
| Start Percent | 0 |

The Lane Runner has a property called the **Lane Vector**. It is a 2D Vector which defines the direction of the lanes in relation to the generated path. A lane vector of (1, 0) defines horizontal lanes and a vector of (0, 1) defines vertical lanes.

If **Use Custom Paths** is toggled, the Lane Runner will look for custom paths, defined in the current Level Segment and will use them instead of the artificial lanes. The percent along the custom paths will be mapped to the percent along the current segment [0-1] so if the custom paths have a more irregular shape, the movement along them using the Lane Runner might look unnatural. If the Level Segment does not have custom paths defined, then it will fall back to using artificial lanes.

To switch between lanes during runtime, the "**lane**" property of the Lane Runner needs to be set [1 – Lane Count]. The Lane Runner will automatically switch to the target lane.

### 1.6.3. Custom Lane Runner

The Custom Lane Runner is entirely meant to follow custom paths (see Custom Level Segment Paths). It requires the Level Segments to have at least one defined custom path and will follow with a uniform speed.

This Runner is most suitable where complex level design is required and each segment has unique passages which need to be traced accurately.

### 1.6.4. The Player Projector Component

The Player Projector is a simple component which does not affect the object's behavior in any way. Its purpose is to continuously project the player along the generated level and notify the Level Generator if new segments need to be created.

If a free player movement (for example physics or character controller) is what the project is going for, this component should be attached to the Player object.

Note that this component should not be attached to player objects which have the Runner components – the Runner components notify the Level Generator automatically.

### 1.6.5. Floating Origin

In games, if an object moves too far away from the origin of the scene (in other words, its coordinate values are big numbers), things can start to jitter. This happens because of the floating point limitation – the bigger the floating point number is, the less digits it can have after its decimal point.

To counter this issue, games use a trick called "Floating Origin". If the player along with the camera moves outside of a given coordinate range, all objects in the game are offset so that the camera's coordinates become 0,0,0. The player does not notice anything because the whole game world moves with them but the coordinates of the player and its camera are reset.

Forever offers a simple solution to that problem – the Floating Origin component. The Floating Origin should be used in infinite runner games to prevent the aforementioned issue. It automates the entire process and is integrated with the Forever, sending offset events to all segments to notify them to update their values.

To use the Floating Origin, add it to the Level Generator object and add a reference to the player camera. If the camera is not referenced, the Floating Origin will attempt to find it automatically. Everything which needs to be offset, needs to be put inside the Level Generator (usually this is the Player and the Camera).

## 1.7. Basic API Use

In order to use the Forever API, the Forever namespace needs to be included:

```
using UnityEngine;
using Dreamteck.Forever;
```

This will give access to all Forever components like the Level Generator, Level Segment, the Runners and the Builders.

### 1.7.1. Using the Level Generator in Code

The Level Generator is a singleton and can be reached from anywhere using **`LevelGenerator.instance`.**

There are two main basic commands for the Level Generator:

**`LevelGenerator.instance.StartGeneration();`**
and **`LevelGenerator.instance.Clear();`**

**StartGeneration** will start the generation sequence if Build on Awake is not set. Clear will stop the generation process and remove all generated segments.

**`LevelGenerator.instance.ready`** can be used to check if the initial generation phase of the Level Generator is done – in other words if the game is ready to start.

**`LevelGenerator.instance.segments`** is a list of all currently existing level segments in the order of generation – the first being the oldest one and the last being the newest one.

**`LevelGenerator.instance.levels`** is the list of all defined levels.

**`LevelGenerator.instance.currentLevel`** is a reference to the current level used by the Level Generator. An example use of this would be to skip sequences. If the current level has a single sequence which is set to be endless, the sequence can be stopped like this:

```
LevelGenerator.instance.currentLevel.sequenceCollection.sequences[0].Stop()
```

Then the next sequence / level can start.

#### 1.7.1.1. Evaluating the Generated Level

The generated level path can be evaluated at any given point using the Evaluate method. In order to do so, the Dreamteck.Splines namespace needs to be included. In order to get the Evaluation result, a SplineResult object needs to be defined first. Then this object needs to be passed to the evaluate method.

The Evaluate method uses a percent [0-1] which defines the point along the generated path that will be evaluated, 0 being the beginning of the path and 1 being the ending (the last current segment).

```
    SplineResult evalResult = new SplineResult();
    protected override void Start()
    {
        LevelGenerator.instance.Evaluate(0.5, evalResult);
        Debug.Log("Evaluated the middle of the segment - > " +
evalResult.position);
    }
```

The SplineResult object contains the position, direction, normal, size, color and percent along the level generated path.

#### 1.7.1.2. Finding the Closest Point Along the Level

It is possible to find the nearest point along the level to an existing point in world space. The Project method of the LevelGenerator does exactly that and the result of the projection is a SplineResult object, like with the Evaluate method.

```
LevelGenerator.instance.Project(worldPosition, evalResult);
Debug.Log("Closest point along the path is at percent " +
evalResult.percent + " and at position " + evalResult.position);
```

Furthermore, this result can be used to find the level segment that contains this projected point using the GlobalToLocalPercent method. This is a method which a global percent [0-1] along the entire generated level into a segment index and a percent [0-1] local to the segment with that index:

```
SplineResult evalResult = new SplineResult();
LevelGenerator.instance.Project(worldPosition, evalResult);
int segmentIndex;

LevelGenerator.instance.GlobalToLocalPercent(evalResult.percent, out
segmentIndex);
Debug.Log(LevelGenerator.instance.segments[segmentIndex].name);
```

#### 1.7.1.3. Find Point Along *Distance*

Since the entire level path is a collection of separate spline paths, this means that it is not possible to easily calculate the evaluation percent for a point that is located N units away from the point at a given percent along the path. The length of the splines does not correspond to the percent values. Both a 2-meter path and a 10-meter will evaluate between [0-1] so if there are shorter and longer segments, it gets very hard to find the proper distance to move along the spline with.

Fortunately, Forever offers the **Travel** method. It takes a starting point along the entire path (percent [0-1]) and a distance in world units to travel along the paths. The returned result is the percent of the new point:

```
SplineResult evalResult = new SplineResult();
double percent = LevelGenerator.instance.Travel(0.0, 10f,
Spline.Direction.Forward);
LevelGenerator.instance.Evaluate(percent, evalResult);
```

### 1.7.2. Using the Level Segment in Code

The Level Segment component comes with a lot of the methods that the LevelGenerator has in terms of path operations. The Evaluate, Project and Travel methods are both present and work in the same way the ones in LevelGenerator do. The LevelGenerator ones, however, operate only on the path of the given segment so the [0-1] range encapsulates only the path of the given segment.

```
SplineResult evalResult = new SplineResult();
LevelSegment segment = LevelGenerator.instance.segments[0];
segment.Evaluate(0.5, evalResult);
```

For Level Segments which have custom paths defined, it is also possible to perform the same operations on any given custom path. Custom paths can be accessed through the **customPaths** array:

```
LevelSegment segment = LevelGenerator.instance.segments[0];
segment.customPaths[0].Project(worldPoint, evalResult);
```

If neither the Runner components nor the ProjectedPlayer component is used for the player object, the Enter method of a segment needs to be called when a player enters the segment. This is very important because this lets the Level Generator know where the player is and if new segments need to be created / old - destroyed.

```
SplineResult evalResult = new SplineResult();
LevelSegment lastSegment;
private void Update()
{
      if (LevelGenerator.instance == null) return;
      if (!LevelGenerator.instance.ready) return;
      LevelGenerator.instance.Project(transform.position,
evalResult);
      int segmentIndex = 0;
LevelGenerator.instance.GlobalToLocalPercent(evalResult.percent, out
segmentIndex);
            LevelSegment current =
LevelGenerator.instance.segments[segmentIndex];
      if(current != lastSegment)
      {
          current.Enter();
          lastSegment = current;
      }
}
```

### 1.7.3. Using the Runners in Code

Referencing a Runner is done the same way any other component is referenced in Unity - `Runner runner = GetComponent<Runner>();` Once referenced, all the properties that are available in the inspector are also available in code under the same names. For example, `runner.followSpeed = 5f;`

For Runners which are not set to automatically, start, the StartFollow() method needs to be called. This method has two overrides:

- `StartFollow()` – will automatically start following by using the defined Start Mode
- `StartFollow(LevelSegment segment, double percent) –` will start following the given segment at the given local percent

To bind player input to the basic runner (the Runner component), assign a Vector2 value to the runner's motion.offset and rotationOffset properties:

```
runner.motion.offset = new Vector2(strafe, 0f);
runner.motion.rotationOffset = new Vector3(0f, 0f, roll);
```

#### 1.7.3.1. LaneRunner

The Lane Runner introduces the lane property. Changing the lane property will change the lane of the runner.

This is an example of controlling the lane property using the arrow keys:

```
LaneRunner runner = GetComponent<LaneRunner>();
if (Input.GetKeyDown(KeyCode.LeftArrow)) runner.lane--;
if (Input.GetKeyDown(KeyCode.RightArrow)) runner.lane++;
```

#### 1.7.3.2. CustomLaneRunner

The Custom Lane Runner component also has the **lane** property which points to the current custom path that the runner should follow. The lane property of the Custom Lane Runner, however, might change automatically if the runner enters a segment with less lanes than the previous and the lane property is set to a higher value than the amount of lanes.

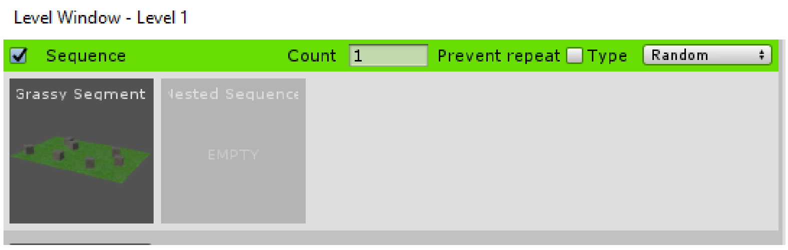
# 2. Advanced Use

This chapter covers usage beyond the basic setup.

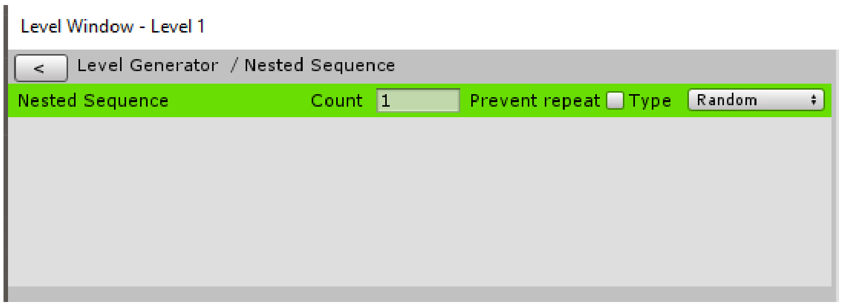
## 2.1. Nested Sequences

The level sequences in Forever can contain nested sequences along the linked segments. A nested sequence is a sequence within a sequence. Whenever the algorithm lands on a nested sequence, it will start executing it and will keep executing it until the sequence is done. Nested sequences are useful for example in cases where segments are randomized but there should be a couple of segments which always need to be spawned in a given order.

To create a nested sequence, open the level editor window and right click inside an existing sequence. Then choose “New Nested Sequence”. A new, empty nested sequence will appear.



This empty sequence now needs to have segments assigned into it. To do so, right click on the nested sequence and select “Edit”. This will open up the nested sequence inside the sequence editor. A sequence path will appear at the top of the editor window which will point out which sequence is currently being edited.



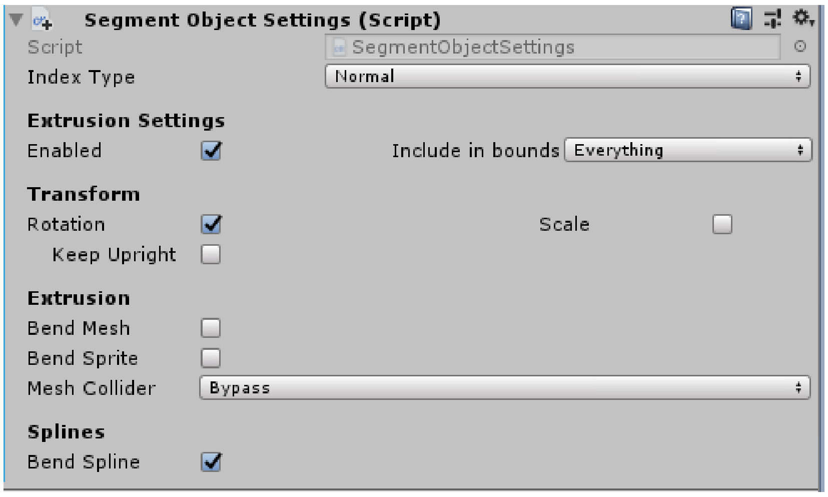
After a sequence is set up, clicking the back button at the top left will return the view back to the parent sequence.

There is no limit to how many nested sequences there can be in one another but in most cases one level of nesting is not required.

## 2.2. The Segment Object Property Component

For extruded segments, each object needs to be configured in the inspector in order to make sure that it is properly included in the bounds and the right components are bent. However, when the same prefab object is included in different Level Segments and the same extrusion rules have to apply to it, having to set up extrusion properties for each instance of that prefab will be time consuming.

This is why Forever includes the Segment Object Settings component. It is a component which can be added to any object and will let the user define how the object will be handled by the extrusion process.


Segment Object Settings (Script)
Script: SegmentObjectSettings
Index Type: Normal

Extrusion Settings
Enabled: ✓
Include in bounds: Everything

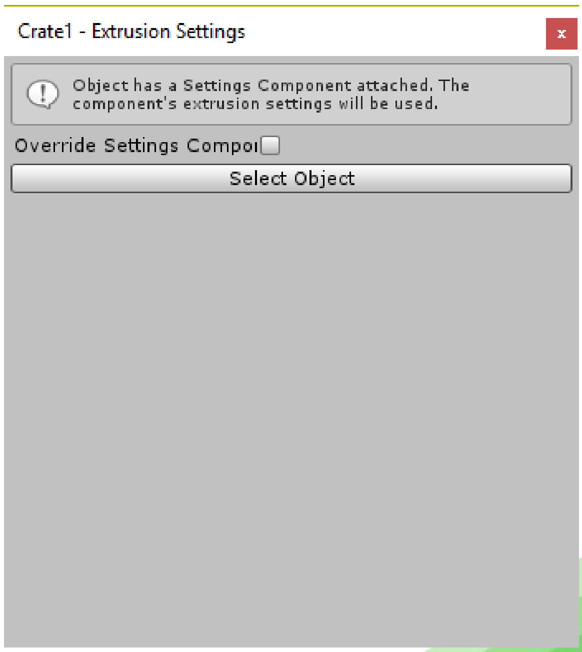
Transform
Rotation: ✓
Scale: ☐
Keep Upright: ☐

Extrusion
Bend Mesh: ☐
Bend Sprite: ☐
Mesh Collider: Bypass

Splines
Bend Spline: ✓

When the Segment Object Settings component is added to an object, it will display the same UI that the Extrusion Settings window does. These settings will be automatically assumed by all Level Segments this object gets added to.

When an object with the Segment Object Settings component is opened in the Extrusion Settings window of a level segment, a message will appear notifying that the settings from the Segment Object Settings component are being used. This can be overridden by toggling the “Oberride Settings Component” checkbox:


Crate1 - Extrusion Settings

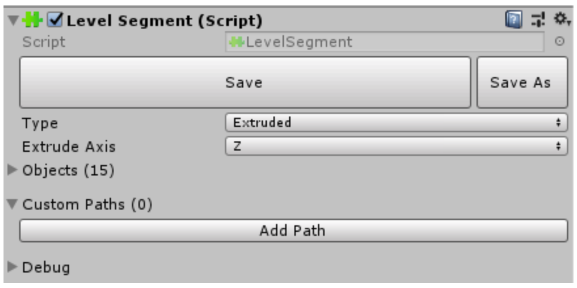
Object has a Settings Component attached. The component's extrusion settings will be used.

Override Settings Compo ☐

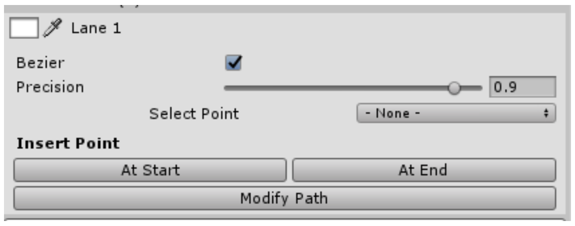
## 2.3. Custom Level Segment Paths

Custom runner paths can be set up for each level segment. The custom paths are user-defined spline paths that can be used for running along the segment or evaluation. They can be used by both extruded and custom segments.

To create a custom path, expand the "Custom Paths" foldout of any given level segment and click "Add Path".


Level Segment (Script)
Script: LevelSegment
Type: Extruded
Extrude Axis: Z
Objects (15)
Custom Paths (0)

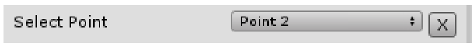
This will automatically create a new custom path based on the segment type and extrusion axis. Clicking on the path name will expand its properties and will enable editing inside the scene.


Lane 1
Bezier: ✓
Precision: 0.9
Select Point: - None -

When selected, a path and its points will be visualized in the scene view. Clicking on path points inside the scene view will select them and enable editing for them. Selecting a point can also be done through the inspector by using the "Select Point" dropdown menu.

The custom paths can either be Bezier or Linear. By default, Bezier is set to true but disabling it will convert the path into a more simple, linear path.

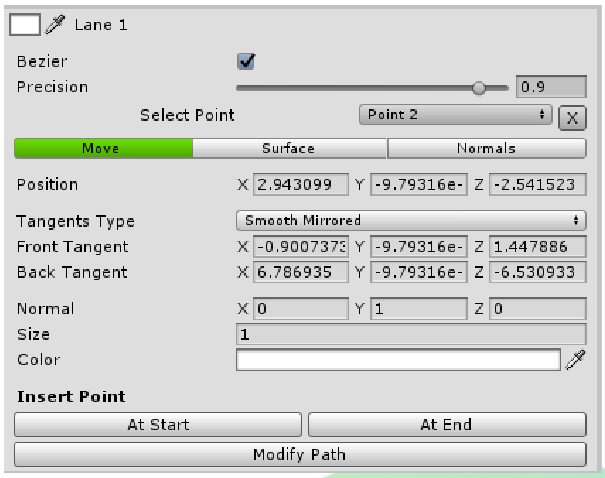
Adding new points to the custom path is done using the two buttons of the bottom of the path panel – Insert Point: "At Start" and "At End". These buttons insert new spline points after the first and before the last point of the spline.

Removing points is done using the "X" button next to the point selection menu when a point is selected: 
Select Point: Point 2

A path cannot have less than two control points so the delete button will not be available when only two points exist in the path.

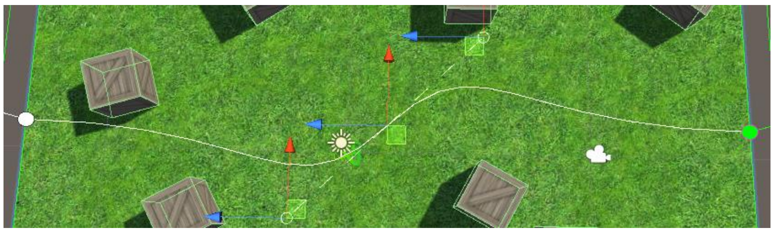
### 2.3.1. Editing Custom Path Points

When a custom path point is selected, its properties will be exposed in the inspector. The values of the points are in world coordinates so it does matter where the level segment's transform is positioned and oriented. However, editing the values might not be an efficient way of setting up the paths. Instead, paths can be edited directly inside the scene.

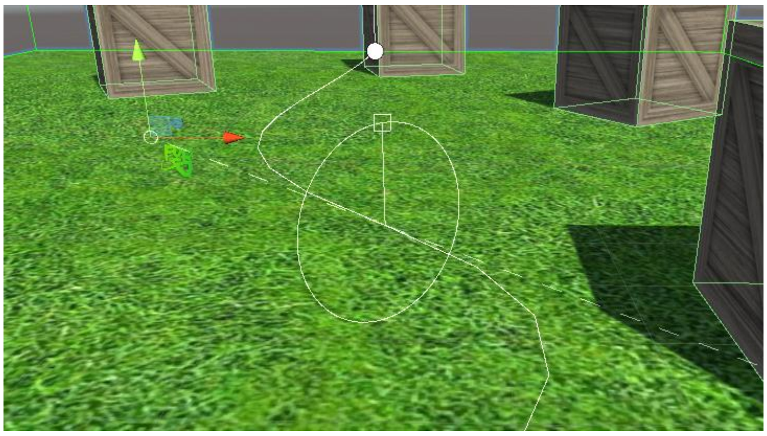

Lane 1
Bezier: ✓
Precision: 0.9
Select Point: Point 2
Position: X 2.943099 Y -9.79316e- Z -2.541523
Tangents Type: Smooth Mirrored
Front Tangent: X -0.9007373 Y -9.79316e- Z 1.447886
Back Tangent: X 6.786935 Y -9.79316e- Z -6.530933
Normal: X 0 Y 1 Z 0
Size: 1
Color:

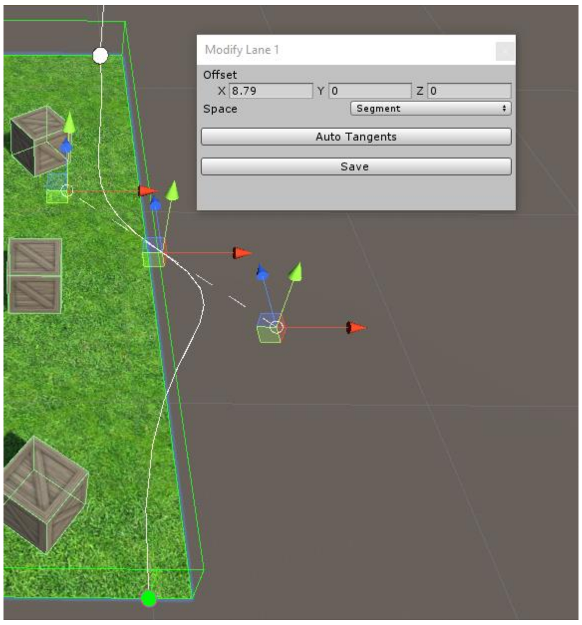
There are three editing tools for the custom paths:

- Move – provides a simple move handle which allows for editing the position and the tangents' positions of each point.



- Surface – It uses raycasting to place the points along the surface of colliders. To use it, click and drag the point's handle over any surface.
- Normals – provides a convenient handle for editing the path normals.





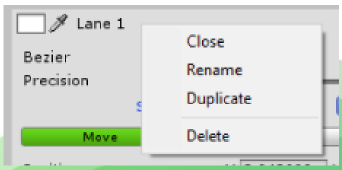
Offset X 8.79 Y 0 Z 0
Space Segment

In addition to these three tools, there are bulk operations which can be applied to the paths. They are found inside the Modify Path window which can be accessed through the "Modify Path" button in the path inspector.

Currently there are two operations available – mass offset and auto tangents.

### 2.3.2. Managing the Custom Paths

To manage a custom path, right click on its name for options.



## 2.4. Builders

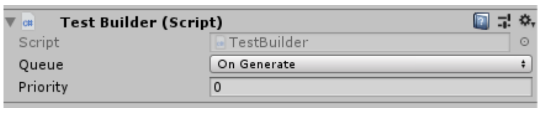
The Builders are a set of special components which are meant to work in conjunction with the Level Segment component. These are MonoBehaviours which get called from the Level Segment upon level generation and can be used to randomize the level segment layout, spawn a certain game object or further generate procedural geometry.

The Builders do not use the Update, FixedUpdate and LateUpdate methods. Instead, they have Build and BuildAsync.

When creating builders one of these two methods (or both) has to be overridden in order to run builder logic.

The Builders have two properties:

- Queue
- Priority


Test Builder (Script)

|  |  |
|---|---|
| Script | TestBuilder |
| Queue | On Generate |
| Priority | 0 |

The Queue property defines at which point during the generation will the builder be executed. **On Generate** will run the builder as soon as the level segment is out and extruded and **On Activate** will run the builder when the segment gets activated.

Priority defines the order of execution for the builder in the given queue. The higher the priority value is, the later the builder will be executed. This is useful for situations where there are builders, dependent on other builders.

This is an example of a builder which has 50% chance of disabling the object it is attached to.

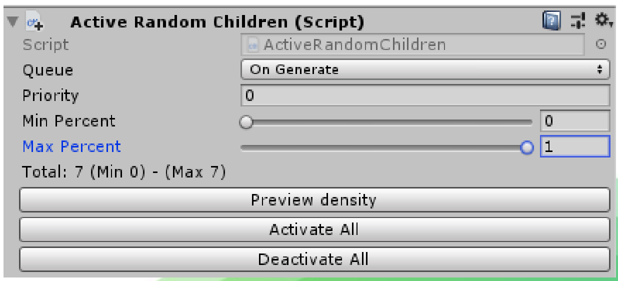
```
using UnityEngine;
using Dreamteck.Forever;

public class TestBuilder : Builder
{
    protected override void Build()
    {
        base.Build();
        if (Random.Range(0, 100) < 50) gameObject.SetActive(false);
    }
}
```

Forever 1.00 comes with two builders out of the box and more will be added in later versions.

### 2.4.1. Active Random Children

This Builder will deactivate a random amount of its children upon generation. The amount of child objects which will remain active is defined by the Min Percent and Max Percent sliders. The percent of active children after building will be a value between those two.


Active Random Children (Script)

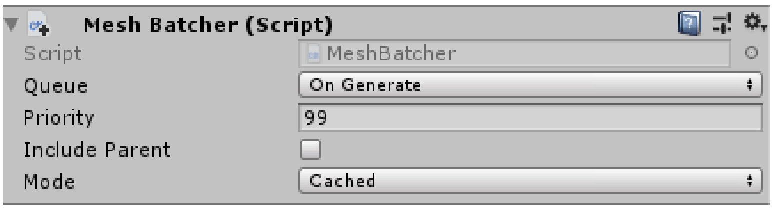
|  |  |
|---|---|
| Script | ActiveRandomChildren |
| Queue | On Generate |
| Priority | 0 |
| Min Percent | 0 |
| Max Percent | 1 |

Total: 7 (Min 0) - (Max 7)

### 2.4.2. Mesh Batcher

The Mesh Batcher Builder does what the name suggests – it looks for meshes in its children which are using the same material and batches them upon generation. This is useful for mobile games where draw calls need to be reduced.

In order for it to work, it requires another component to be attached to all children objects which will be batched – the Batchable. The Mesh Batcher will only register the Batchable objects in its children, leaving out everything else.


Mesh Batcher (Script)

|  |  |
|---|---|
| Script | MeshBatcher |
| Queue | On Generate |
| Priority | 99 |
| Include Parent | ☐ |
| Mode | Cached |

This Builder has two modes:

- Cached – will cache all builders during editing so that it doesn't have to look for them during runtime
- Dynamic – will not cache any builders and will look for them during runtime – useful when objects are Instantiated dynamically using other builders.

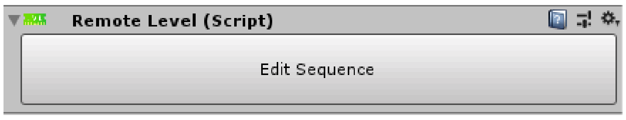
The Mesh Batcher does not include inactive objects in the operation.

## 2.5. Remote Levels

If the game has a lot of levels, each of which using unique resources it might become a problem at some point. Since a lot of assets will have to be loaded at once, the RAM usage might increase significantly as well as the load time.

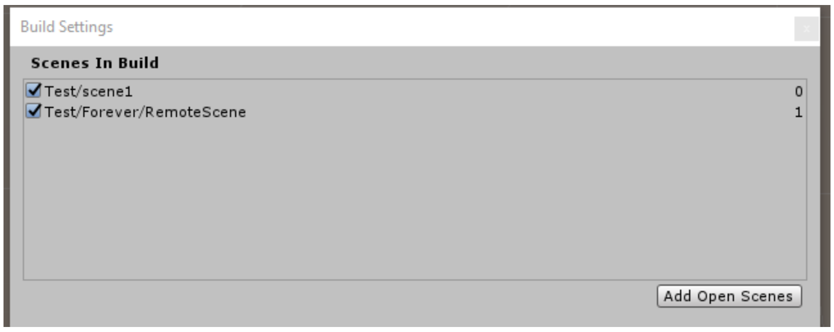
This is why Forever has something called Remote Levels. This is a feature which allows level information to be exported to different scenes and during runtime, each level will load its scene in the background when needed. After the level has been played, the level scene is unloaded once again, freeing up memory.

To create a remote level, create a new scene, delete all objects inside and create a new empty object at 0,0,0. Then add the Remote Level component to it.



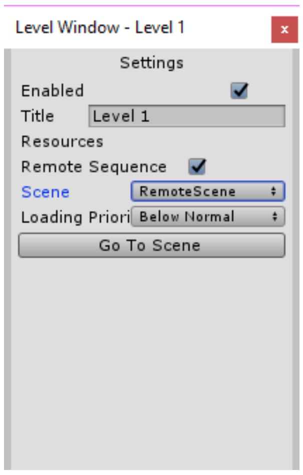
Click “Edit Sequence” and the sequence editor will open. Assign all level segments inside the sequence editor and deselect the Remote Level object. At this point, the Remote Level will automatically extract all unique assets from the assigned segments and will spawn them into the scene. This is required so that Unity can load all assets in the background and stuttering is evaded when spawning segments for the first time.

The remote level scene must be saved **and added to the build queue.**


Build Settings

Scenes In Build

| | |
|---|---|
| ✓ Test/scene1 | 0 |
| ✓ Test/Forever/RemoteScene | 1 |


Level Window - Level 1

Settings

| | |
|---|---|
| Enabled | ✓ |
| Title | Level 1 |
| Resources | |
| Remote Sequence | ✓ |
| Scene | RemoteScene |
| Loading Priori | Below Normal |

Going back to the main scene with the Level Generator object, select the Level Generator and open the level which will be linked to this newly created scene. Inside the level window, toggle "Remote Scene" – this will make the window much smaller as the sequence editor will disappear. From the dropdown menu select the scene, containing the remote level and select a thread priority for the loading process (usually Below Normal is good).

When the game runs, this level will now load the linked scene additively as soon as it is needed. When the last segment from this level is destroyed, the scene will be unloaded in the background.

All objects that are created inside the remote scene will be automatically disabled as soon as the scene loads so there is no need to worry about them showing in up in the game.

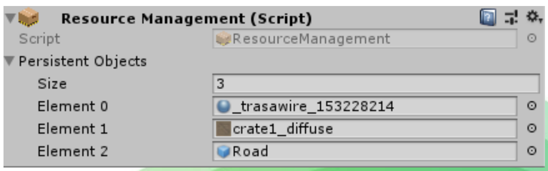
## 2.6. Resource Management

Forever cleans up after itself by destroying all generated objects as soon as they are not used anymore. And with remote level loading, level resources are unloaded per level. However, there is one other method which can be used to unload unused assets and it is the Resource Management system.

To use the Resource Management feature, simply add the Resource Management component to the Level Generator object. No further setup is required – everything will start working automatically.

The Resource Management system looks for assets that are no longer in use and unloads them as soon as the last Level Segment that is using them is gone. This feature should be used with caution because it is only concerned with the resources used in the level generation itself. If a mesh/material/sound is used in a level segment but it is also used outside of the level generation loop, upon the destruction of this segment this resource will be unloaded.

There is a workaround to the Resource Management unloading unwanted resources and it is the "Persistent Objects" array. Adding any objects to this array will make sure that none of the resources inside get unloaded.


Resource Management (Script)

| | |
|---|---|
| Script | ResourceManagement |
| Persistent Objects | |
| Size | 3 |
| Element 0 | _trasawire_153228214 |
| Element 1 | crate1_diffuse |
| Element 2 | Road |

## 2.7. Disabling Multithreading

Multithreading is used to run the calculations for segment extrusion. If for any reason multithreading is not desired, it can be disabled by adding the **Segment Extruder** component to the Level Generator object and disabling multithreading in the inspector.

The Segment Extruder component is automatically added during runtime and by default it is set to use multithreading. If, however Segment Extruder is added in the editor, this option can be overridden.

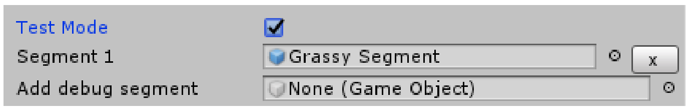
## 2.8. Test & Debugging

Forever provides features to help developers test and debug the level generation.

### 2.8.1. Testing New Segments

There is a way to test new level segments without having to create levels and inserting them into sequences.

To enter test mode, toggle the "Test Mode" checkbox in the Level Generator inspector. This will enable a game object field for adding a new Level Segment. Dragging level segments into this field will add them to the test array. When the game plays, the level generator will only pick random segments from the test array and will spawn them instead of going through the levels.



| | |
|---|---|
| Test Mode | ✓ |
| Segment 1 | Grassy Segment |
| Add debug segment | None (Game Object) |

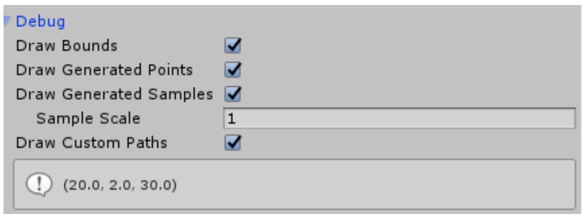
Removing segments from that array is done by either clicking the "x" button or setting their field to None.

Note that only prefabs with the Level Segment component can be added to the test array.

### 2.8.2. Debugging Level Segments

The Level Segment component provides visual information about its properties inside the scene view both during edit mode and runtime. The visual information is only drawn when the segment is selected and can be toggled on or off.

To define what gets drawn, expand the "Debug" foldout at the bottom of the Level Segment component's inspector.

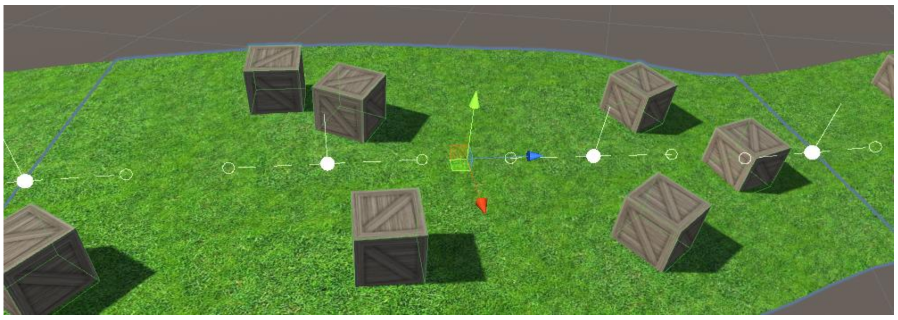


Debug

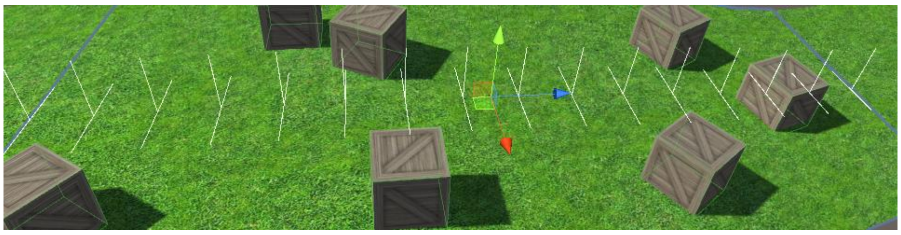
| | |
|---|---|
| Draw Bounds | ✓ |
| Draw Generated Points | ✓ |
| Draw Generated Samples | ✓ |
| Sample Scale | 1 |
| Draw Custom Paths | ✓ |

(20.0, 2.0, 30.0)

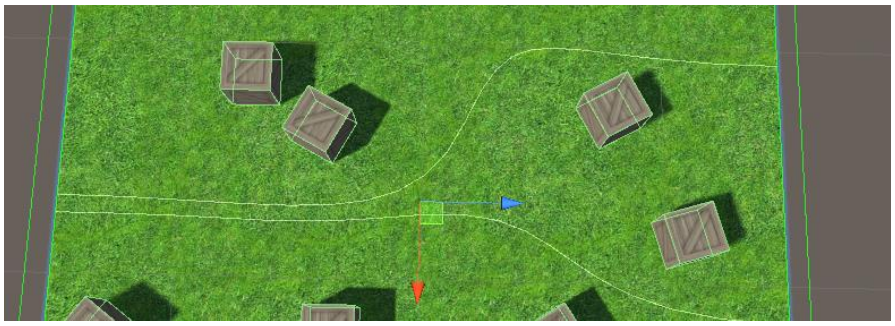
- Draw Bounds – should the segment bounds be drawn when in Extrude mode? The bounds size is also displayed at the bottom of the inspector.
- Draw Generated Points – should the points from the generated segment path be drawn when the segment is selected during runtime?



- Draw Generated Samples – should the cached result of the segment's path evaluation be displayed during runtime?



- Sample Scale – multiplier for the size of the cached samples
- Draw Custom Paths – Should the custom segment paths be always drawn together?



These properties are serialized which means they will be saved with the segment. Each segment can have its own variation of the debug properties.

# 3. Extending the Functionality

Forever is designed for flexibility and variety so one of the main goals of the plugin is to have easily expandable functionality. Expanding the functionality does require C# and OOP knowledge as it is done via inheritance.

To derive from any of Forever's classes, the Dreamteck.Forever and namespace needs to be included first: `using Dreamteck.Forever;`

## 3.1. Writing Custom Path Generators

Since the paths that are being generated are Bezier splines which also use normals to define the orientation of the generated geometry, writing a proper robust algorithm for the generation requires some basic linear algebra knowledge. Fortunately, there are two types of path generators – the **LevelPathGenerator** and the **HighLevelPathGenerator**. The LevelPathGenerator class is the base class for all path generators and the HighLevelPathGenerator is an abstraction for it which reduces some of the hassle of defining the paths' points and makes it easier to construct a nice path generating behavior. The LevelPathGenerator class deals with the spline points themselves and provides the developer with full control over how spline points are generated.

### 3.1.1. How Path Generators Work?

The path generators have a public method called **GeneratePath**. This method is what the Level Generator calls in order to generate paths for each level segment and it is non-overridable. This is what happens inside this method:

1. The segment for which the path is generated is passed and examined. If this segment belongs to a new level, the path generator is notified via the OnNewLevel method
2. The segment is checked for custom path rules – these are rules which override the work of the path generator – for example, a rule that the path for this segment is always flat – without denivelation. (see **Custom Path Rules**)
3. A point array for the path is created and is passed to the **OnBeforeGeneration** method. This method is by default empty and can be used to perform bulk operations on all points prior to the actual generation process. It is technically possible to generate the entire path in this method.
4. Each point is passed to the **GeneratePoint** method. This method is used to define the point's properties.
5. The point array is passed to one final method – **OnPostGeneration** which is used to perform one last bulk operation on the entire point array in order to finalize it. This method is also empty by default and can also be used to generate the entire path if the other methods are not used.

If a Custom Rule is present on the segment, the points will also be passed to the custom rule's instances of the same methods – OnBeforeGeneration, GeneratePoint and OnPostGeneration after the methods are executed inside the Path Generator.

### 3.1.2. Inheriting the LevelPathGenerator

When inheriting the LevelPathGenerator class, another namespace needs to be included: `using Dreamteck.Splines;`

```
using UnityEngine;
using Dreamteck.Forever;
using Dreamteck.Splines;

public class MyPathGenerator : LevelPathGenerator
{
    public override void Initialize(LevelGenerator input)
    {
        base.Restart(input);
        //Any code for initializing the generation process goes here
        //Resetting variables
        //Getting references, etc.
    }

    public override void Continue(LevelPathGenerator previousGenerator)
    {
        base.Continue(previousGenerator);
        //Code for transferring internal data from another generator to this
one
        //Used during runtime when the LevelGenerator's pathGenerator
property is assigned
        //This will allow the next generated path to continue from the same
position and orientation
        //Example: lastPointPosition = ((MyPathGenerator
)previousGenerator).lastPointPosition;
    }

    protected override void OnNewLevel()
    {
        base.OnNewLevel();
        //Called when a segment from a new level is created
        //Code for increasing difficulty values or resetting goes here
    }

    protected override void OnBeforeGeneration(SplinePoint[] points)
    {
        base.OnBeforeGeneration(points);
        //Bulk operation on all points prior to the actual generation
        //This method is not used in most cases
    }

    protected override void GeneratePoint(ref SplinePoint point, int
pointIndex)
    {
        base.GeneratePoint(ref point, pointIndex);
        //Generating a single point
        //point.SetPosition(...);
        //point.SetTangent(...);
        //point.normal = ...;
    }

    protected override void OnPostGeneration(SplinePoint[] points)
    {
        base.OnPostGeneration(points);
        //Finalizing bulk operation on all generated points
        //Example: AutoTangents(ref points, 0);
    }

    protected override void OnOriginOffset(Vector3 direction)
    {
        base.OnOriginOffset(direction);
        //Offset any world Vector3 values that this generator uses by
"direction"
        //For example, if the latest point's position is cached in a private
Vector3 field called "lastPosition":
        //lastPosition -= direction;
    }
}
```

LevelPathGenerator-derived classes have access to some protected members which can come in handy for the path generation.

- **levelGeneratorTransform** – a reference to the Level Generator's transform from which the path generation starts
- **segment** – the Level Segment the path is being generated for
- **level** – the level of the Level Segment
- **isNewLevel** – is it the first segment of a new level?
- **lastPoint** – a copy of the last generated path point

### 3.1.3. Inheriting the HighLevelPathGenerator

The HighLevelPathGenerator is different than the LevelPathGenerator in a couple key different ways.

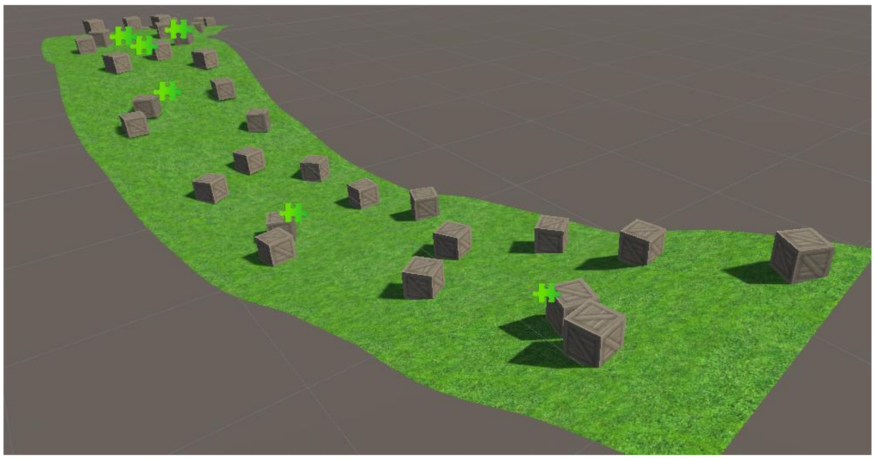
First of all, instead of using the SplinePoint struct, it uses a simpler one called Point. The Point struct does not have tangents and a normal. Instead it has a rotation property. However, all points are by default set to automatically generate their tangents and normals. This is useful because for simpler generation behaviors, all that is needed is the position of the points to be set.

The HighLevelPathGenerator uses an internal Vector3 property called orientation. This is the angle of the direction in which the point will be generated. An orientation of 0,0,0 for each point will produce a straight path. It is not mandatory to use orientation but it enables the usage of the GetPointPosition method which uses the orientation in conjunction with the previous point to generate a new position for the current point.

This is an example of a simple random path generator:

```
using UnityEngine;
using Dreamteck.Forever;

public class MyPathGenerator : HighLevelPathGenerator
{
    protected override void GeneratePoint(ref Point point, int pointIndex)
    {
        base.GeneratePoint(ref point, pointIndex);
        orientation += new Vector3(Random.Range(-10f, 10f), Random.Range(-
10f, 10f), 0f);
        point.position = GetPointPosition();
    }
}
```



## 3.2. Writing Custom Builders

Writing a custom builder is simply a matter of inheriting the Builder class and overriding the Build and/or the BuildAsync methods. Both methods will fire once when their que comes. Build will execute the entire code immediately while BuildAsync is a coroutine.

```
using UnityEngine;
using Dreamteck.Forever;
using System.Collections;

public class CustomBuilder : Builder
{
    protected override void Build()
    {
        base.Build();
        //Builder logic
    }

    protected override IEnumerator BuildAsync()
    {
        yield return new WaitForSeconds(1f);
        //delayed logic
    }
}
```

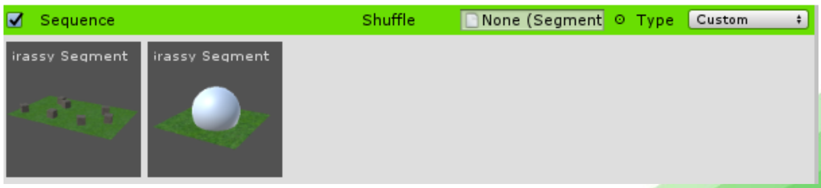
The Reset method can be used to set a default value for the builder's priority:

```
    private void Reset()
    {
        priority = 10;
    }
```

## 3.3. Segment Shuffle

The Segment Shuffle is a custom rule for picking segments inside a level sequence.

To assign a Segment Shuffle object to a sequence, set its type to custom and then assign the object in the "Shuffle" field:



☑ Sequence Shuffle None (Segment Type Custom

To create a Segment Shuffle object, a new class needs to be created and derived from the SegmentShuffle class. Once a SegmentShuffle class exists, a SegmentShuffle object can be created by going into Create/Forever/Segment Shuffle.

When inheriting from the SegmentShuffle class, there is only one method that needs to be overridden – the Get method. In its arguments, it has a reference to the current sequence along with an index for the current segment. The index is the number of the segment that is being generated in this run.

There is an internal property called "_isDone" which needs to be set to true when the shuffle is done so that the sequence can be finalized. If _isDone is not set to true, the sequence will loop indefinitely.

```
using Dreamteck.Forever;

public class CustomShuffle : SegmentShuffle
{
    public override Level.SegmentDefinition Get(Level.SegmentSequence sequence,
int index)
    {
        if (index == sequence.segments.Length - 1) _isDone = true;
        return sequence.segments[(sequence.segments.Length - 1) - index];
    }
}
```

This is an example of a reverse-ordered shuffle which will start from the last segment and end with the first segment from the sequence definition.

_isDone needs to be set prior to returning the chosen segment.

## 3.4. Custom Path Rules

Custom Path Rules are MonoBehaviours which override the path generation of the segment they are attached to. They are useful when certain level segments need to have a specific path that is different from the rest of the segments. Using a Custom Path Rule is done by **adding it to the root of the Level Segment prefab**.

The logic of the Custom Path Rules is called through the current working Path Generator. After each OnBeforeGenerate, OnGeneratePoint and OnPostGeneration (see Writing Custom Path Generators for more information).

In order to create a Custom Path Rule, a new class needs to be created and derived from **CustomPathRule**.

```
using UnityEngine;
using Dreamteck.Forever;
using Dreamteck.Splines;

public class FlatPathRule : CustomPathRule
{
    public override void OnGeneratePoint(SplinePoint point, SplinePoint previous,
int pointIndex, int pointCount)
    {
        Vector3 pos = point.position;
        pos.y = previous.position.y;
        point.SetPosition(pos);
        pos = point.tangent;
        pos.y = previous.position.y;
        point.SetTangentPosition(pos);
```

```
            point.normal = Vector3.up;
       }
 }
```

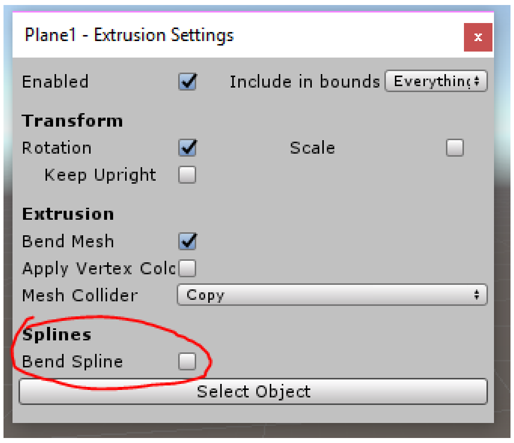
In this example, the generated points are flattened along the Y axis and the point normals are set to 0,1,0. The horizontal turns of the path will be preserved but the path will have no denivelation.

# 4. Forever and Dreamteck Splines

Forever has support for the Dreamteck Splines plugin. Spline objects can be added to the Level Segments and can be extruded along the level paths.

In order to use Dreamteck Splines with Forever, make sure that the latest version of Dreamteck Splines is installed in the project. Versions of Dreamteck Splines, older than 1.0.96 will not work and will create conflicts.

Once Dreamteck Splines is installed, the object property setup for Level Segment objects should start offering the "Bend Spline" checkbox:



Plane1 - Extrusion Settings

Enabled ✓ Include in bounds Everything

**Transform**

Rotation ✓ Scale ☐

Keep Upright ☐

**Extrusion**

Bend Mesh ✓

Apply Vertex Colc ☐

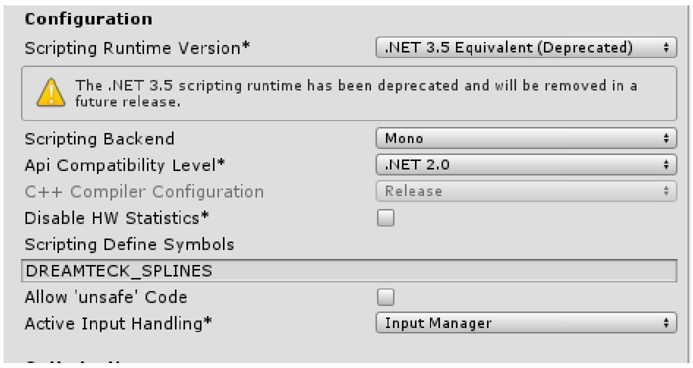
Mesh Collider Copy

**Splines**

Bend Spline ☐

Select Object

If this is not the case and Dreamteck Splines is confirmed to be installed, then the "DREAMTECK_SPLINES" scripting define might be missing from the project. To add it, go to Edit/Project Settings/Player and navigate to the "Scripting Define Symbols" field:



**Configuration**

Scripting Runtime Version* .NET 3.5 Equivalent (Deprecated)

The .NET 3.5 scripting runtime has been deprecated and will be removed in a future release.

Scripting Backend Mono

Api Compatibility Level* .NET 2.0

C++ Compiler Configuration Release

Disable HW Statistics* ☐

Scripting Define Symbols

DREAMTECK_SPLINES

Allow 'unsafe' Code ☐

Active Input Handling* Input Manager

If the field is not empty, separate the defines with a semicolon:

**MY_CUSTOM_DEFINE;DREAMTECK_SPLINES**

# 5. API Reference

## 5.1. SplinePoint

Representation of a control point. SplinePoint is used to define Splines. Passing a new array of points to a spline changes the spline.

### 5.1.1. Public Enumerations

| | |
|---|---|
| Type {Smooth, Broken, FreeSmooth} | Smooth mirrors the tangents, Broken makes the tangents independent and FreeSmooth is a combination of the two |

### 5.1.2. Public Properties

| | |
|---|---|
| SplinePoint.Type type | The type of the point |
| Vector3 position | The position of the point |
| Color color | The color of the point |
| Vector3 normal | The normal direction of the point |
| float size | The size of the point |
| Vector3 tangent | The first tangent position for Bezier splines |
| Vector3 tangent2 | The second tangent position for Bezier splines |

### 5.1.3. Public Methods

| | |
|---|---|
| void SetPosition(Vector3 pos) | Sets the position of the point and moves its tangents too |
| void SetTangentPosition(Vector3 pos) | Sets the tangent position of the point |
| void SetTangent2Position(Vector3 pos) | Sets the second tangent’s position of the point |

### 5.1.4. Static Methods

| | |
|---|---|
| SplinePoint Lerp(SplinePoint a, SplinePoint b, float t) | Interpolation between two spline points |

### 5.1.5. Constructors

| | |
|---|---|
| SplinePoint(Vector3 p) | Creates a new smooth point |
| SplinePoint(Vector3 p, Vector3 t) | Creates a new smooth point |
| SplinePoint(Vector3 pos, Vector3 tan, Vector3 nor, float s, Color col) | Creates a new fully defined smooth point |

| | |
|---|---|
| SplinePoint(Vector3 pos, Vector3 tan, Vector3 tan2, Vector3 nor, float s, Color col) | Creates a new fully defined smooth point |

## 5.2. SplineResult

When a spline is evaluated, multiple values are returned. This is the result of spline evaluation.

### 5.2.1. Public Properties

| | |
|---|---|
| Vector3 position | The position of the evaluation result |
| Vector3 normal | The normal of the evaluation result |
| Vector3 direction | The direction of the evaluation result |
| Color color | The color of the evaluation result |
| float size | The size of the evaluation result |
| double percent | The time (0-1) the spline was evaluated at |

### 5.2.2. Read-Only Properties

| | |
|---|---|
| Quaternion rotation | Returns Quaternion.LookRotation(direction, normal) |
| Vector3 right | Returns a perpendicular vector to direction and normal |

### 5.2.3. Public Methods

| | |
|---|---|
| void Lerp(SplineResult b, double t) | Interpolates between the current values and b's values |
| void Lerp(SplineResult b, float t) | Interpolates between the current values and b's values |
| void CopyFrom(SplineResult input) | Copies the values of the input SplineResult object |

### 5.2.4. Static Methods

| | |
|---|---|
| SplineResult Lerp(SplineResult a, SplineResult b, double t) | Interpolates between two spline results |
| SplineResult Lerp(SplineResult a, SplineResult b, float t) | Interpolates between two spline results |

| | |
|---|---|
| void Lerp(SplineResult a, SplineResult b, double t, SplineResult target) | Interpolates between two spline results and stores the result in the target spline result |
| void Lerp(SplineResult a, SplineResult b, float t, SplineResult target) | Interpolates between two spline results and stores the result in the target spline result |

## 5.3. Spline

A spline in world space. This class stores a single spline and provides methods for evaluation, length calculation, raycasting and more.

### 5.3.1. Example

```
using UnityEngine;
using System.Collections;
using Dreamteck.Splines; //Include the Splines namespace

public class SimpleSplineController : MonoBehaviour {
       void Start () {
        //Create a new B-spline with precision 0.9
        Spline spline = new Spline(Spline.Type.BSpline, 0.9);
        //Create 3 control points for the spline
        spline.points = new SplinePoint[3];
        spline.points[0] = new SplinePoint(Vector3.left);
        spline.points[1] = new SplinePoint(Vector3.up);
        spline.points[2] = new SplinePoint(Vector3.right);
        //Evaluate the spline and get an array of values
        SplineResult[] results = new SplineResult[spline.iterations];
        spline.Evaluate(results);
        //Display the values in the editor
        for (int i = 0; i < results.Length; i++)
        {
            Debug.DrawRay(results[i].position,            results[i].normal,
results[i].color);
        }
    }
}
```

### 5.3.2. Public Properties

| | |
|---|---|
| SplinePoint[] points | The control points of the spline |
| Spline.Type type | The type of the spline which defines what interpolation should be used. |
| double precision | The approximation rate (0-0.9999) of the spline |
| AnimationCurve customValueInterpolation | Custom curve for size and color interpolation between points |
| AnimationCurve customNormalInterpolation | Custom curve for normal interpolation between points |

### 5.3.3. Read-Only Properties

| | |
|---|---|
| bool  isClosed | Whether or not the spline is closed |
| double moveStep | The step size of the percent incrementation when evaluating a spline (based on percision) |
| int iterations | The total count of samples for the spline (based on the precision) |

### 5.3.4. Public Methods

| | |
|---|---|
| SplineResult Evaluate(double percent) | Whether or not the spline is closed |
| void Evaluate(ref SplineResult[] samples, double from = 0.0, double to = 1.0) | The step size of the percent incrementation when evaluating a spline (based on percision) |
| Vector3 EvaluatePosition(double percent) | The total count of samples for the spline (based on the precision) |
| void EvaluatePositions(Vector3[] positions, double from = 0.0, double to = 1.0) | Evaluates the spline using its precision and writes the result positions to the array. |
| float CalculateLength(double from = 0.0, double to = 1.0, double resolution = 1.0) | Calculates the length of the spline |
| double Project(Vector3 point, int subdivide = 4, double from = 0.0, double to = 1.0) | Projects a point on the spline. Returns evaluation percent. |
| bool Raycast(out RaycastHit hit, out double hitPercent, LayerMask layerMask, double resolution = 1.0, double from = 0.0, double to = 1.0, QueryTriggerInteraction hitTriggers = QueryTriggerInteraction.UseGlobal) | Casts rays along the spline against all colliders in the scene |
| bool RaycastAll(out RaycastHit[] hits, out double[] hitPercents, LayerMask layerMask, double resolution = 1.0, double from = 0.0, double to = 1.0, QueryTriggerInteraction hitTriggers = QueryTriggerInteraction.UseGlobal) | Casts rays along the spline against all colliders in the scene and returns all hits. Order is not guaranteed. |
| double Travel(double start, float distance, Direction direction) | Returns the percent from the spline at a given distance from the start point |
| void Close() | Closes the spline (requires the spline to have at least 4 points) |

| | |
|---|---|
| void Break() | Breaks the closed spline |
| void Break(int at) | Breaks the closed spline at a given point |
| void ConvertToBezier() | Converts the spline from Hermite to Bezier without losing its initial shape. |

#### 5.3.5. Constructors

| | |
|---|---|
| Spline(Type t) | Creates a spline of a given type |

### 5.4. LevelSegment

A MonoBehaviour component for defining level segments. It holds information for

#### 5.4.1. Public Enumerations

| | |
|---|---|
| Type { Extruded, Custom } | Extruded type means the segment will be extruded along a path and a Custom type will not modify the segment |
| Axis { X, Y, Z } | The Axis along which the extrusion takes place |
| EvaluateMode { Cached, Accurate } | Evaluation mode for the path operations. Cached uses the cached samples and Accurate calculates new values on the fly |
| GenerationState { Idle, Free, Extruding, Preparing, Destroying } | Generation state description for the level generation process. |

#### 5.4.2. Public Delegates

| | |
|---|---|
| void SegmentEnterHandler(LevelSegment segment) | A handler for when a segment is entered. |
| delegate void EmptyHandler() | An empty handler |

#### 5.4.3. Events

| | |
|---|---|
| SegmentEnterHandler onSegmentEntered | Static event, called when a segment is entered. |
| EmptyHandler onActivate | Called when the segment is activated |
| EmptyHandler onEnter; | Called when the segment is entered |

### 5.4.4. Read-Only Properties

| | |
|---|---|
| bool activated | Is the segment activated? |
| bool isReady | Is the segment extruded and ready to be played? |

### 5.4.5. Public Properties

| | |
|---|---|
| Type type = Type.Extruded | The type of the segment. |
| Transform customEntrance | A Transform entrance for the entry point of a custom level segment |
| Transform customExit | A Transform entrance for the exit point of a custom level segment |
| bool customKeepUpright | Should this segment be kept upright if custom? |
| LevelSegmentCustomPath[] customPaths | The array of custom paths, defined for this level segments |
| ObjectProperty[] objectProperties | The array, containing all registered children and their extrusion properties if the type is set to Extrude |
| int customMainPath | Index, pointing to the custom path that is considered to be the main path for the segment. (For example, the middle one – 1) |
| Axis axis | Axis of extrusion |
| SplineResult[] samples | The array of path samples used by the Evaluate method and etc. |
| Spline spline | The spline, constructed for this level segment (this is the extrusion path). |
| int sampleRate | Number of spline samples between two control points |
| LevelSegment next | Reference to the next Level Segment |
| LevelSegment previous | Reference to the previous Level Segment |
| Level level | Reference to the level this segment belongs to |

| | |
|---|---|
| bool stitch | Should the segment be stitched seamlessly to the previous one when the type is set to Extrude? |

### 5.4.6. Public Methods

| | |
|---|---|
| TS_Bounds GetBounds() | Whether or not the spline is closed |
| void Destroy() | The step size of the percent incrementation when evaluating a spline (based on percision) |
| void DestroyImmediate() | The total count of samples for the spline (based on the precision) |
| void AddCustomPath(string name) | Add a new custom path with a name |
| void RemoveCustomPath(int index) | Remove a custom path at index |
| void Evaluate(double percent, SplineResult result, EvaluateMode mode = EvaluateMode.Cached) | Evaluate the level segment's path and return a result. |
| Vector3 EvaluatePosition(double percent, EvaluateMode mode = EvaluateMode.Cached) | Evaluate the level segment's path and return only the position. |
| double Travel(double start, float distance, Spline.Direction direction, out float traveled, EvaluateMode mode = EvaluateMode.Cached) | Returns the percent from the level segment's path at a given distance from the start point |
| void Project(Vector3 point, SplineResult result, double from = 0.0, double to = 1.0, EvaluateMode mode = EvaluateMode.Cached) | Finds the closest result to a world point along the level segment's path |
| void Enter() | Enter the segment and let the Level Generator know. |

## 5.5. LevelSegment.LevelSegmentCustomPath

A class that holds the definition and cache for a custom path inside the Level Segment. During runtime, operations like Evaluate, Travel and Project can be performed on it.

### 5.5.1. Public Properties

| | |
|---|---|
| string name | The name of the path |

| | |
|---|---|
| Color color | The path color |
| SplinePoint[] localPoints | The path's points, local to the Level Segment's root |

### 5.5.2. Read-Only Properties

| | |
|---|---|
| Spline spline | A reference to the path's spline |
| SplineResult[] samples | The cached spline samples of the path |

### 5.5.3. Public Methods

| | |
|---|---|
| void Evaluate(double percent, SplineResult result) | Evaluate the path at a given percent and write to the SplineResult object |
| Vector3 EvaluatePosition(double percent) | Evaluate the path at a given percent and return only the position |
| double Travel(double start, float distance, Spline.Direction direction, out float traveled) | Find the percent along the path that is *distance* away from the starting percent |
| void Project(Vector3 point, SplineResult result, double from = 0.0, double to = 1.0) | Find the closest result to a world point along the path. |
| void Transform() | Transform all local points and write the results into the paths' spline |
| void InverseTransform() | Inverse transform the path's spline and write the results into the local points array |
| void CalculateSamples() | Calculate the samples and cache them |
| LevelSegmentCustomPath Copy() | Duplicate the path |

## 5.6. LevelPathGenerator

The base class for generating paths. It is not used directly by the Level Generator and needs to be inherited. It provides the framework for generating the paths.

### 5.6.1. Enumerations

| | |
|---|---|
| PathType { Bezier = Spline.Type.Bezier, Linear = Spline.Type.Linear} | The type of the path – Bezier or Linear |

### 5.6.2. Public Properties

| | |
|---|---|
| PathType pathType | The type of the path |
| int sampleRate | How many samples per point |
| int controlPointsPerSegment | How many control points per segment will be generated? |
| bool customNormalInterpolation | Should custom normal interpolation be used? |
| AnimationCurve normalInterpolation | The custom normal interpolation curve |
| bool customValueInterpolation | Should custom size and color interpolation be used? |
| AnimationCurve valueInterpolation | The custom value interpolation curve |

### 5.6.3. Public Methods

| | |
|---|---|
| void GeneratePath(LevelSegment inputSegment) | Generate the path for the given segment |

## 5.7. HighLevelPathGenerator

An abstraction of the LevelPathGenerator class allowing for simpler path generation code. It is not used directly and provides an abstraction of the LevelPathGenerator framework.

### 5.7.1. Public Properties

| | |
|---|---|
| bool useCustomNormalDirection | Should the normals of all generated points be overridden? |
| Vector3 customNormalDirection | The custom normal direction to override with |
| Vector3 startOrientation | Start orientation of the path generation |

## 5.8. RandomPathGenerator

A Path Generator class used for random path generation. It provides a lot of settings to help cover a wide range of use cases.

### 5.8.1. Public Properties

| | |
|---|---|
| bool usePitch | Should pitch be randomized? |
| bool useYaw | Should yaw be randomized? |

| | |
|---|---|
| bool useRoll | Should roll be randomized? |
| bool useColors | Should colors be randomized? |
| bool useSizes | Should sizes be randomized? |
| bool restrictPitch | Should pitch angles be restricted within a certain range? |
| bool restrictYaw | Should yaw angles be restricted within a certain range? |
| bool restrictRoll | Should roll angles be restricted within a certain range? |
| Vector3 minOrientation | The minimum allowed angles when restriction is on |
| Vector3 maxOrientation | The maximum allowed angles when restriction is on |
| Vector3 minRandomStep | Minimum target angle change |
| Vector3 maxRandomStep | Maximum target angle change |
| Vector3 minTurnRate | Minimum angle move rate |
| Vector3 maxTurnRate | Maximum angle move rate |
| Vector3 startTargetOrientation | The initial target angle when generation starts |
| bool useStartPitchTarget | Should pitch use the startingOrientation? |
| bool useStartYawTarget | Should yaw use the startingOrientation? |
| bool useStartRollTarget | Should roll use the startingOrientation? |
| Gradient minColor | Minimum gradient for color randomization |
| Gradient maxColor | Maximum gradient for color randomization |
| AnimationCurve minSize | Minimum curve for size randomization |
| AnimationCurve maxSize | Maximum curve for size randomization |
| Vector3 minSegmentOffset | Minimum offset between segments |
| Vector3 maxSegmentOffset | Maximum offset between segments |
| Space segmentOffsetSpace | The transformation space of the offset coords |

| | |
|---|---|
| Vector3 newLevelMinOffset | Minimum offset between segments of different levels |
| Vector3 newLevelMaxOffset | Maximum offset between segments of different levels |
| Space levelOffsetSpace | The transformation space of the offset coords |

## 5.9. CustomPathGenerator

A Path Generator class that uses a pre-defined path as a base and generates the level path along that path.

### 5.9.1. Public Properties

| | |
|---|---|
| bool loop | Should the path be looped (requires at least 4 control points)? |
| SplinePoint[] points | The control points of the path |
| Spline.Type customPathType | The type of the path |

## 5.10. SpiralPathGenerator

A simple Path Generator class that generates a path in a spiral.

### 5.10.1. Enumerations

| | |
|---|---|
| enum Axis { X, Y, Z, NegativeX, NegativeY, NegativeZ} | The axis for the spiral generation. |

### 5.10.2. Public Properties

| | |
|---|---|
| Axis axis | The axis for the spiral generation. |
| float turnRate | Spin rate per generated path point |
| float steepness | Steepness angle of the spiral |
| float normalRotation | Surface direction rotation around the spiral path |

## 5.11. WavyPathGenerator

A simple Path Generator class that produces a wavy path.

### 5.11.1. Public Properties

| | |
|---|---|
| float angle | The maximum deviation angle for the wave |
| float turnRate | The turn rate towards the deviation |
| Vector3 turnAxis | The turn axis around the path will turn |

## 5.12. LevelGenerator

The Level Generator component responsible for the entire level generation process. It holds a reference to all defined levels and a list of all currently present segments in the scene.

### 5.12.1. Enumerations

| | |
|---|---|
| Type { Infinite, Finite } | The type of the level generator – infinite or finite |
| LevelIteration { OrderedClamp, OrderedLoop, Random, None } | Iteration mode for the generator's levels |

### 5.12.2. Events

| | |
|---|---|
| LevelEnterHandler onLevelEntered | Called when a new level is entered by the player |
| LevelEnterHandler onLevelLoaded | Called when a new level is loaded by the generator |
| LevelEnterHandler onWillLoadLevel | Called when a new level is about to be loaded by the generator |

### 5.12.3. Public Properties

| | |
|---|---|
| static LevelGenerator instance | The current active instance of the Level Generator (LevelGenerator.instance) |
| int generateSegmentsAhead | How many segments to generate ahead of the player when in Infinite mode? |
| int activateSegmentsAhead | How many segments to activate ahead of the player when in Infinite mode? |
| int maxSegments | Maximum segments that can exist in the scene in Infinite mode |

| | |
|---|---|
| bool buildOnAwake | Should the Level Generator start building as soon as it is awake? |
| Type type | The type of the Level Generator |
| int finiteSegmentsCount | The segment count when in Finite mode |
| bool finiteLoop | Is the level path looped in Finite mode (e.g. racing track) |
| Level[] levels | The array of defined levels |
| LevelIteration levelIteration | How levels will be iterated through? |
| int startLevel | The starting level of the Level Generator |

### 5.12.4. Public Methods

| | |
|---|---|
| void StartGeneration() | Enable the generator and set it to start working |
| void Clear() | Disable the generator and clear all created segments |
| void Restart() | Restart the generator |
| LevelSegment GetSegmentAtPercent(double percent) | Get the level segment at the current percent [0-1] |
| LevelSegment FindSegmentForPoint(Vector3 point) | Find the segment the given world point is closest to |
| void Evaluate(double percent, SplineResult result) | Evaluate the entire level and write the result into the SplineResult object |
| Vector3 EvaluatePosition(double percent) | Evaluate the entire level and return only the position along the path |
| void Project(Vector3 point, SplineResult result) | Project a world point onto the level path |
| double Travel(double start, float distance, Spline.Direction direction) | Find the percent along the entire level that is *distance* away from the starting percent |
| void CreateNextSegment() | Force the level generator to generate another segment |
| void DestroySegment(int index) | Destroy an existing level segment by index |

## 5.13. Level

A definition of a level in the game. It holds a Sequence Collection which in terms holds one or more sequences inside and each sequence holds a collection of level segments.

### 5.13.1. Public Properties

| | |
|---|---|
| bool enabled | Is this level enabled? |
| string title | The title of the level |
| SegmentSequenceCollection sequenceCollection | The sequence collection |
| bool remoteSequence | Is this level using a remote sequence? |
| string remoteSceneName | The scene name to load for the remote sequence |
| ThreadPriority loadingPriority | Loading priority for the remote sequence |

### 5.13.2. Read-Only Properties

| | |
|---|---|
| bool isReady | Is the level loaded and ready? |
| SegmentSequence[] sequences | A shorthand for the sequences of the level (equal to sequenceCollection.sequences) |

### 5.13.3. Public Methods

| | |
|---|---|
| void Initialize() | Initialize the level for generation |
| bool IsDone() | Are all sequences inside the level done? |
| void SkipSequence() | Skip the current sequence |
| void GoToSequence(int index) | Skip the current sequence and go to another sequence directly |
| LevelSegment InstantiateSegment(Vector3 position, Quaternion rotation) | Instantiate the next segment from the current sequence |
| Level Duplicate() | Return a deep copy of the level |

## 5.14. Level.Sequence

A collection of level segments.

### 5.14.1. Enumerations

| | |
|---|---|
| Type { Ordered, Random, Custom } | How the sequence shuffles the segments |

### 5.14.2. Public Properties

| | |
|---|---|
| bool enabled | Is the sequence enabled? |
| string name | The name of the sequence |
| SegmentDefinition[] segments | The segments of the sequence |
| int spawnCount | How many segments should the sequence spawn in random mode? |
| Type type | The type of the sequence |
| SegmentShuffle customShuffle | Custom segment Shuffle object for Custom sequences |
| bool preventRepeat | Should the sequence prevent repeating segments in random mode? |

### 5.14.3. Public Mehtods

| | |
|---|---|
| void Initialize() | Initialize the sequence for generation |
| void Stop() | Stop the sequence |
| SegmentDefinition NextDefinition() | Pull the next segment out of the sequence |
| bool IsDone() | Is the sequence done? |
| SegmentSequence Duplicate() | Create a deep copy of the sequence |

## 5.15. Level.SegmentSequenceCollection

A collection of sequences.

### 5.15.1. Public Properties

| | |
|---|---|
| SegmentSequence[] sequences | The sequence array of this collection |

## 5.16. Level.SegmentDefinition

A definition for a level segment used inside the Level.Sequence. It holds a reference to a level segment prefab or another sequence (nested sequence).

### 5.16.1. Public Properties

| | |
|---|---|
| GameObject prefab | The sequence array of this collection |
| bool nested | Is this definition a nested sequence? |
| SegmentSequence nestedSequence | The nested sequence for the definition if nested |

### 5.16.2. Public Methods

| | |
|---|---|
| LevelSegment Instantiate(Vector3 pos, Quaternion rot) | Instantiate the segment prefab |
| SegmentDefinition Duplicate() | Return a deep copy of the definition |

### 5.16.3. Constructors

| | |
|---|---|
| SegmentDefinition() | Create as new empty definition |
| SegmentDefinition(string nestedName) | Create a definition with a nested sequence |
| SegmentDefinition(GameObject input) | Create a definition with a level segment prefab |

## 5.17. Runner

The basic Runner component used to traverse the generated levels.

### 5.17.1. Enumerations

| | |
|---|---|
| UpdateMode { Update, FixedUpdate, LateUpdate } | In which method should the update logic run? |
| PhysicsMode { Transform, Rigidbody, Rigidbody2D } | What to apply the running result to? |
| StartMode { Percent, Distance, Project } | How does the Runner find its starting point? |

### 5.17.2. Public Properties

| | |
|---|---|
| double startPercent | The starting percent along the level path if StartMode is set to Percent |
| UpdateMode updateMode | |
| float startDistance | The starting distance along the level path if StartMode is set to Distance |

| | |
|---|---|
| StartMode startMode | The start mode |
| float followSpeed | The follow speed of the runner |
| bool follow | Should the following logic be executed? |
| bool isPlayer | Is this the player object? |
| PhysicsMode physicsMode | The Physics mode of the runner |

#### 5.17.3. Read-Only Properties

| | |
|---|---|
| MotionModule motion | The motion module of the runner which defines how transformations are applied |
| SplineResult result | The current result of the runner along the level path |

#### 5.17.4. Public Methods

| | |
|---|---|
| void StartFollow() | Start following using the defined start mode |
| void StartFollow(LevelSegment segment, double percent) | Start following using the given segment at the given percent |

## 5.18. LaneRunner : Runner

An abstraction of the Runner component, which introduces a simple lane switching behavior which is easy to use in code.

#### 5.18.1. Public Properties

| | |
|---|---|
| float width | The width of all lanes |
| int laneCount | Lane count |
| int startLane | The start lane [1-laneCount] |
| float laneSwitchSpeed | How fast does the runner go from lane to lane |
| bool useCustomPaths | Should the runner use custom segment paths when available? |
| AnimationCurve laneSwitchSpeedCurve | Easing for the lane switching speed |
| Vector2 laneVector | Lane vector which defines the direction of the road. 1, 0 is a horizontal road. |
| int lane | The current lane [1-laneCount] |

## 5.19. CustomLaneRunner : Runner

An abstraction of the Runner component which allows for following custom segment paths with uniform speed.

| | |
|---|---|
| int lane | The current lane |
| float laneSwitchSpeed | How fast does the runner traverse from lane to lane? |

## 5.20. ProjectedPlayer

A passive component which continuously projects the player’s position onto the generated level and lets the Level Generator know where the player is at any given time.

### 5.20.1. Enumerations

| | |
|---|---|
| UpdateMode { Update, FixedUpdate, LateUpdate } | In which method should the update logic run? |

### 5.20.2. Public Properties

| | |
|---|---|
| UpdateMode updateMode | Which method to run the update logic in? |
| float updateTime | Update interval in seconds (0 runs every frame) |

### 5.20.3. Read-Only Properties

| | |
|---|---|
| SplineResult result | The current result |

# 6. Support

## 6.1. Contact Support

For any issues, questions and feature requests, you can contact Dreamteck Support at support@dreamteck.io or write to support using our web form - https://dreamteck.io/team/contact.php Please, make sure to select "Support" from the dropdown menu in the form. Otherwise our team might take long to respond.

Or join the Dreamteck Discord server to chat with other developers who use our plugins. Upon joining, make sure to mention @Dreamers and let us know that you are a developer. You will be assigned the developer role which will give you access to the Forever channel where you can ask questions about Forever. Please note that asking for Forever support in other channels might result in the message getting deleted.

Join our Discord server: https://discord.gg/bkYDq8v

## 6.2. Frequently Asked Questions

Q: Is Forever suitable for mobile games?
A: Yes, Forever is optimized and tested on cell phones from the years 2014-2019 – no visible spikes in performance have been noticed. However, it is still possible to have some spikes on some budget tier devices.

Q: Does Forever work on Unity 5.6 or older?
A: We haven't tested it on 5.6 and have no reason to think it does.

Q: Does Forever build for all platforms?
A: Yes, Forever isn't using platform-specific code. If a building error occurs, please contact support so we can resolve the issue.

Q: Do I need programming experience to use Forever?
A: For the most part, no. The basic Forever features like level generation and running work out of the box. However, some basic programming will be required to bind the player input to the runner components (see **Using Runners in Code** for examples).

Q: Can I create levels with junctions with Forever?
A: Forever only generates one path but it is possible to do so:
- Create a junction opening segment where the terrain branches
- Create random segments that are branched and fit the opening segment
- Create a closing segment which merges the branched segments
- Add the segments to a nested sequence

Q: Can I create a game entirely out of custom segments and not use extrusion?
A: Yes, this is absolutely possible but the level generator will still require a path generator to be assigned. A suitable solution for this is to just create a simple "High Level Path Generator" and assign it. If all segments are set to custom, a path will not be generated at all.