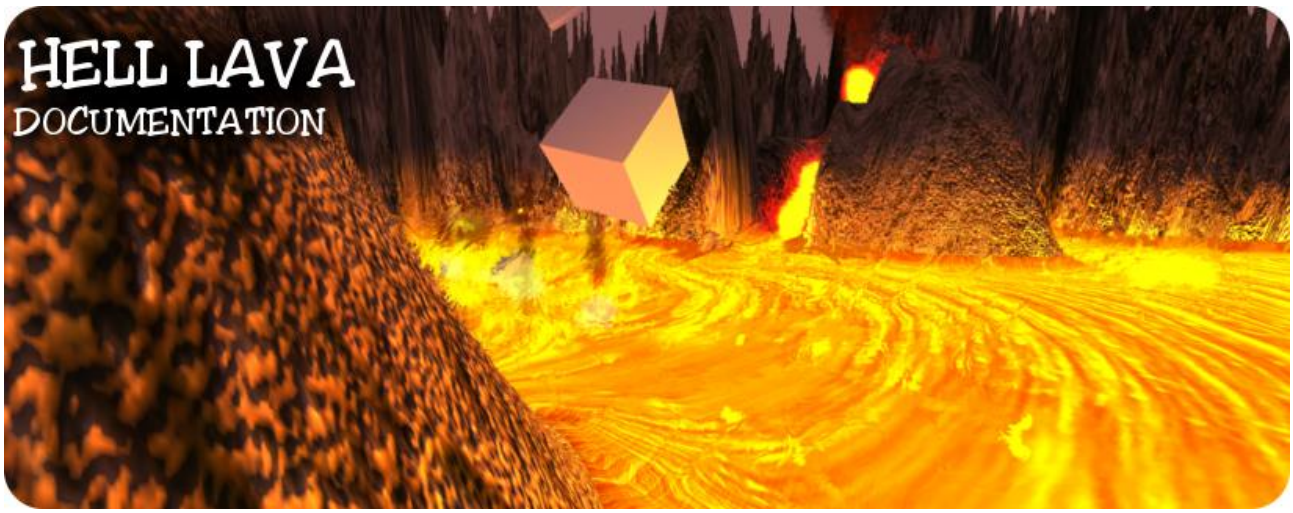


Version: 1.4.5
Supported Unity versions: 2020.3.2f1
e-mail: contact@dragons-diamond.com

I – Introduction

Hell Lava is Unity package, containing lava environment creator. Hell Lava package include following functionality: mesh surface generator, scene configuration, lava lighting system, interaction with objects, damage system, lava sound system and lava environment (projectiles, lavafalls, sparks).

Hell Lava package content:

>Hell Lava
- *>Editor*
  - **LavaCreator.cs** // lava environment creator script
  - **LavafallEditor.cs** // custom inspector for lavafall script
  - **LavaSplitPlugin.cs**// lava split tool engine for Lava Creator
- *>Example*
  - *>Scene scripts* *//those script are needed only to handle example scene*
    - **CubeHandler.cs** // handles the cube behavior when in the lava
    - **ExampleHandler.cs** // handles the first run of example scene
    - **FallingCubes.cs** // scripts that generates cubes above the lava
    - **Fly.cs** // individual motor for character controller, that allows flying
    - **FPSDisplay.cs** // display FPS measurement on screen
    - **Health.cs** // health script for objects
    - **MouseLook.cs** // simplified version of MouseLook script in C#
    - **SkyboxHandler.cs** // handles skybox rotation
    - **UIHandler.cs** // shows info on player screen
  - **cube.prefab** // simple cube from example scene to show lava interactivity
  - **example.unity** // example scene
  - **First Person Controller.prefab** // player prefab for example scene
  - **Terrain.asset** // terrain data for example scene
  - **TerrainLayer.terrainlayer** // terrain layer data for example scene
- *>Resources*
  - *>Materials*
    - **distortMaterial.mat** // material for the heat distortion effect
    - **lavaMaterial.mat** // material for lava surface set1
    - **lavaMaterial2.mat** // material for lava surface set2
    - **lavaMaterial3.mat** // material for lava surface set3
    - **lavaRiverMaterial.mat** // material for visual element of lavafall
    - **lavaShotMaterial.mat** // material for lava projectiles
    - **lavaSplashMaterial.mat** // material for lava splash particles
    - **rockMaterial.mat** // material for rocks
    - **smokeMaterial.mat** // material for lava fumes feature
    - **sparkMaterial.mat** // material for lava sparks feature
    - **UIFlamesMaterial.mat** // material for UI flames image
  - *>Meshes* *// in this folder will be saved generated lava surface meshes and their data*
    - **Lava_mesh.asset** // mesh data for lava surface from example scene
    - **Lava_data.asset** // lava surface data for lava surface from example scene
    - **LavaTop_mesh.asset** // mesh data for top lava surface from example scene
    - **LavaTop_data.asset** // lava surface data for top lava surface from example scene
  - *>Prefabs*
    - **distortion.prefab** // distortion heat prefab
    - **lavafall.prefab** // general lavafall
    - **lavaRiver.prefab** // visual element generated by **lavafall.prefab**
    - **lavaShot.prefab** // lava projectile prefab
    - **lavaSplash.prefab** // lava splash prefab when object touch lava surface
    - **playerPack.prefab** // prefab that need to be added to player object, which

lava ambient component and FireUI handler
- **pointLight.prefab** // lava lighting is created from many **pointLight.prefab**
- **sparkles.prefab** // lava sparks prefab
- **steamSoundEnter.prefab** // lava sound instantiate when object enter the lava
- **steamSoundExit.prefab** // lava looped sound when object stays in the lava
- **steamSoundLooped.prefab** // lava sound instantiate when object exit the lava

>*Sounds*
- **lavaAmbient.wav** // lava ambient sound
- **lavaEnter.wav** // lava sound played when object just entered the lava
- **lavaExit.wav** // lava sound played when object just left the lava

>*Textures*
>*Skybox*
- **skyboxBack.png** // example scene skybox texture – back
- **skyboxBottom.png** // example scene skybox texture – bottom
- **skyboxFront.png** // example scene skybox texture – front
- **skyboxLeft.png** // example scene skybox texture – left
- **skyboxRight.png** // example scene skybox texture – right
- **skyboxTop.png** // example scene skybox texture – top

- **distortion — normal.png** // normal map for distortion effect
- **flameTexture.png** // texture for UI flames
- **lavaRiverTexture.psd** // texture for visual element of lavafall
- **lavaShoreTexture2.png** // lava surface shore texture for set2
- **lavaShoreTexture3.png** // lava surface shore texture for set3
- **lavaShoreTexture3.png — normal.png** // lava surface shore normal map for set3
- **lavaShoreTexture.png** // lava surface shore texture for set1
- **lavaShotTexture.psd** // lava projectiles texture
- **lavaSplashTexture.png** // lava splash texture
- **lavaSurfaceTexture2.png** // lava surface main texture for set2
- **lavaSurfaceTexture2.png — normal.png** // lava surface shore normal map for set2
- **lavaSurfaceTexture3.png** // lava surface main texture for set2
- **lavaSurfaceTexture3.png — normal.png** // lava surface shore normal map for set3
- **lavaSurfaceTexture.png** // lava surface main texture for set1
- **lavaSurfaceTexture.png — normal.png** // lava surface shore normal map for set1
- **rockTexture.png** // texture for rocks
- **rockTexture — normal.png** // normal map for rocks
- **smokeTexture.tif** // texture for lava fumes feature
- **sparkTexture.psd** // texture for lava sparks feature
- **terrainTexture.png** // texture for terrain from example scene
- **terrainTexture — normal.png** // terrain normal map from example scene

- **GlobalData.asset** // storage for the Hell Lava global data
- **HeatDistort.shader** // shader for heat distortion effect
- **LavaShader.shader** // shader for lava surface
- **UIFlames.shader** // shader for UI flames waving

>*Scripts* *// Hell Lava collection of scripts*
>*class* *// class scripts*
- **GlobalData.cs** // represents object, that is in GlobalData
- **InLava.cs** // represents object, that is in lava
- **MeshData.cs** // represents lava surface

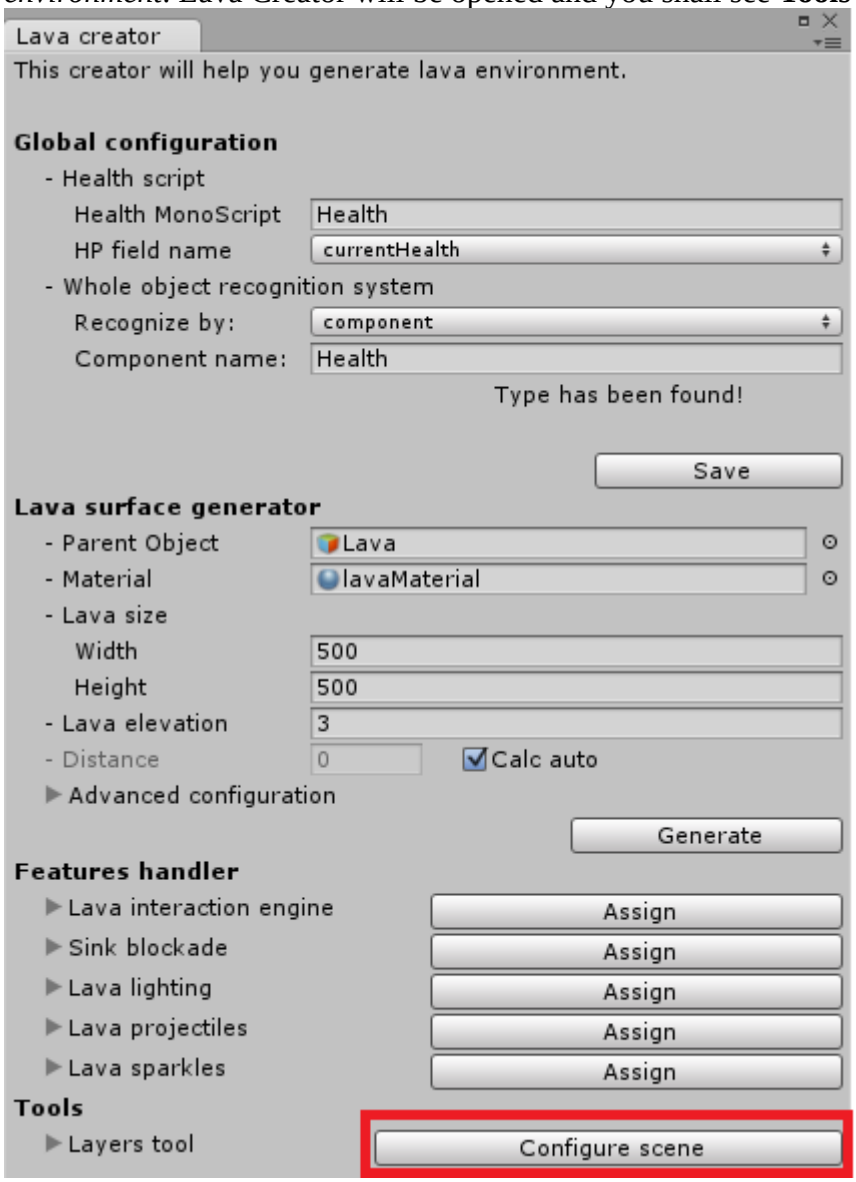
- **FireUI.cs** // fire UI animation feature handler
- **IgnoredObject.cs** // marks ignored object
- **LavaAmbient.cs** // lava ambient sound handler
- **Lavafall.cs** // lavafall engine handler

- **LavafallRiver.cs** // lavafall visualization handler
- **LavafallShadow.cs** // lavafall trigger handler
- **LavaShot.cs** // lava projectiles behavior handler
- **LavaTrigger.cs** // lava interaction with object handler
- **ProjectilesSpawner.cs** // randomly generates lava projectiles at game beginning
- **Scrolling.cs** // make lava moving
- **SoundDestroyer.cs** // destroys game object if audio is not playing
- **SoundGenerator.cs** // script that generates lava loop sound
- **SparkleSpawner.cs** // randomly generates lava sparkles particles at game beginning
- **SplashHandler.cs** // handles lava splash
- **SurfaceHandler.cs** // handles lava surface behavior

II – Prepare your project

Hell lava is very easy in configuration. It require to add a few layers, but whole process is automated and you don't have to worry about that. Just keep in mind that you need in your project **5 empty layers.** Whole configuration is processed on lava surface generation or when adding some features. It's mean that if you launch example scene first time just after package decompress, your project won't be configured (in console information about that will be displayed).

To run example scene, just go to upper menu *GameObject*, select *Create Other* and choose *Lava environment*. Lava Creator will be opened and you shall see **Tools** section and **Layers tool.**



Just press **Configure scene** next to **Layers tool** label. Configuration will add 5 layers to your project, set dependence between them and assign created layers to existing objects in scene and to Hell Lava resources in Assets.

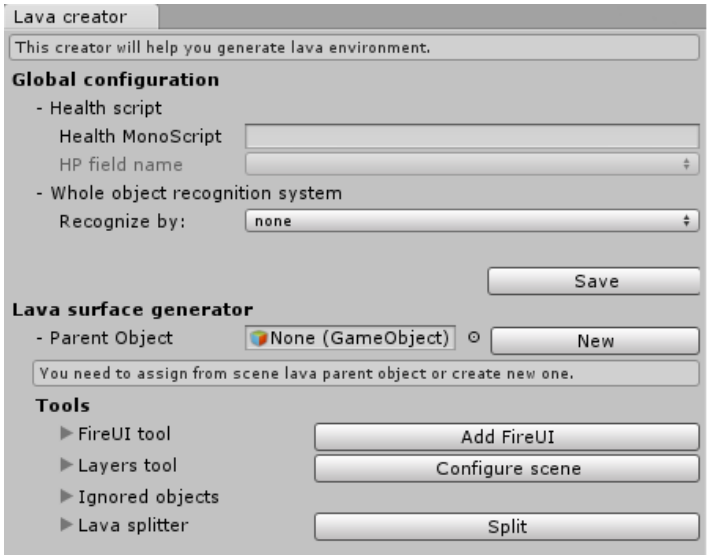
More about **Layers tool** you can read in chapter **V – Tools.**

III – Hell Lava creator

Hell Lava creator is a main tool, that allow you to generate your own lava environment. Creator is available from *GameObject* upper Unity menu at *Create Other>Lava environment*. Source script can be accessed from *Hell Lava>Editor>LavaCreator.cs*. Script was written in C#.

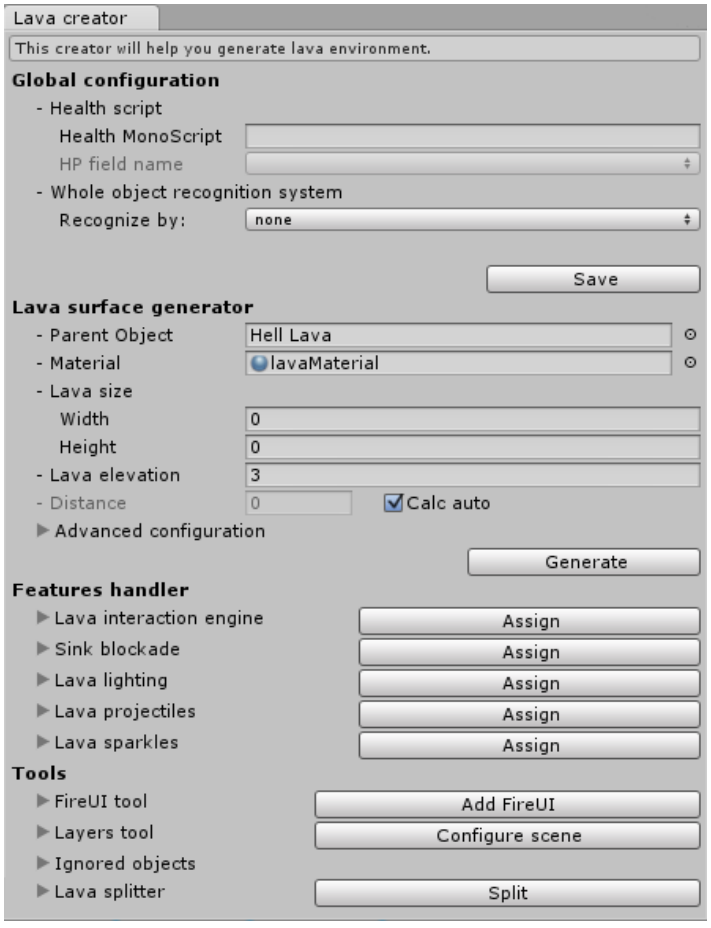
Description

After creator launch, you will see this:



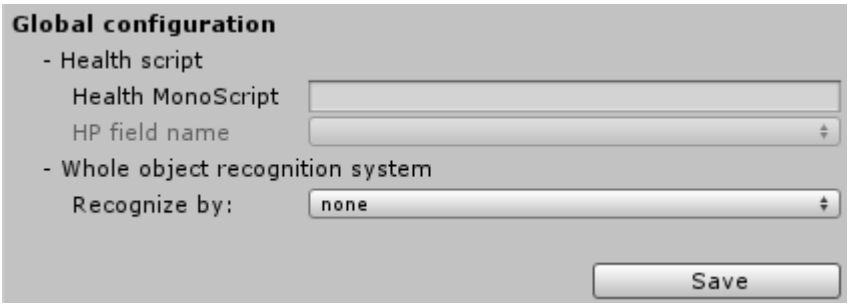
Creator is working on specific game object. One way to create new lava environment is to just drag empty game object from hierarchy to Parent Object field, but much easier is just click "**New**" button. It will create empty game object on scene and pass it reference to Parent Object.

Above action will affect in unlocking all creator functionality:



Creator is divided in four parts:
- Global configuration,
- Lava surface generator,
- Features handler,
- Tools.

Global configuration

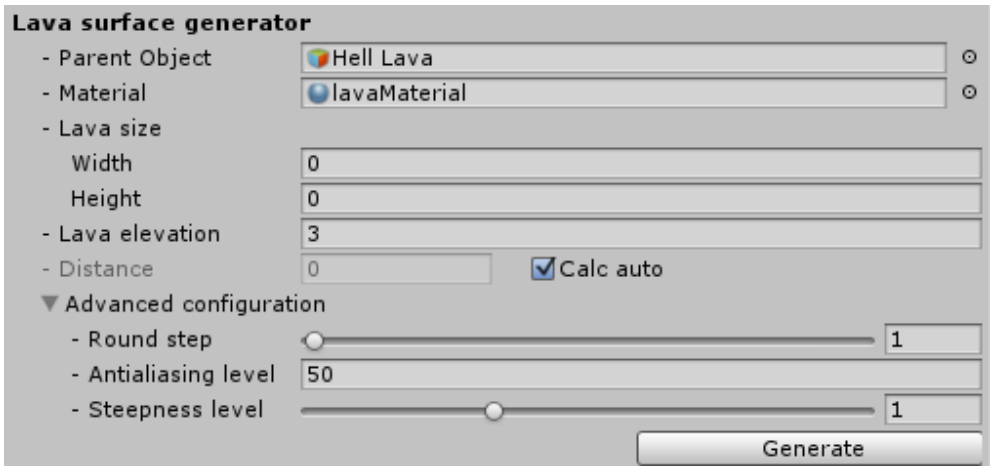


In global configuration, user can set settings, that will affect all lavas environments, that was added to the scene. Variables values are stored in *GlobalData.asset* and are saved only after **Save** button pressed. Variables that are storage are:

- Health script – this field is not required to lava environment operating, but without it, damage system won't work. **Health script** have two fields: **Health MonoScript** and **HP field name.** If **Health MonoScript** is empty, **HP field name** will remain blocked. To unlock it, just type in **Health MonoScript** field name of your health script (case sensitive!). After that, you will be able to choose from **HP field name** what variable is responsible for storing current object's HP.

- Whole object recognition system – for interaction system, it is important to recognize "what" is object. It can be complex monster with many children and colliders, or just a single rock with only one primitive collider. We don't want to remove object health for every child and at the same time thread every collider separately when enter or exit the lava. Hell Lava system will recognize object local root by selected mode. You can choose:
– *none* – whole object won't be searched.
– *component* – whole object will be found by typed component name.
– *layer* – whole object will be found by selected layer.
– *tag* – whole object will be found by given tag.
– *name* – whole object will be found if its name will be found whole or part of given string.

Lava surface generator



This is the part, where creator generates lava surface mesh ONLY. At this point, no lava functionality will be added.

If this is first time, when you selected game object to create lava surface, creator will load default data. Note, that configured variables in **global variables** will be saved to file (in Hell Lava>Resources>Meshes) and used next time when you drag this game object to Parent Object field. **WARNING! Data won't be saved on Unity exit.**

- Material – material object, that will be assigned to generated mesh. Hell lava have already prepared material with texture, that can be found in Hell lava> Resources>Materials>lavaMaterial. This field is loaded automatically.

- Lava size – **width** and **height** determines lava size. This two variables cannot be less or equal zero. Start position is taken from **Parent Object**'s position.

- Lava Elevation – it is a variable, determining how high should lava mesh supposed to be lifted at border with encountered colliders. This variable can be also negative, so you can get concave shore.

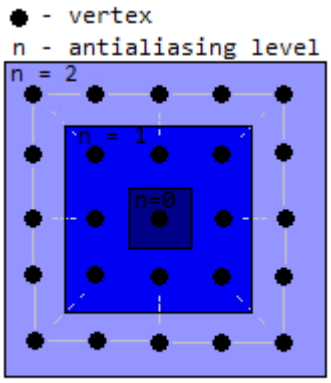
- Distance – distance between two vertices. Lower mean better resolution of lava mesh, but can't be less than 0. If **calc auto** is checked, this value will be calculated automatically.

Advanced configuration allows you to make changes in lava creator during generation.

- Round step – for every creating vertex is checked if it is close to any collider. This is done by shooting ray around examined vertex at a certain angle. **Round step** is determining that "certain angle". Less mean more precision but it takes more time to generate lava surface. Default is 1 and usually it lasts, but if you are building large surface and the distance between vertices is big, you should set this variable to lower values. Less value not always guarantees better effect, and always will greatly slow down generation process!

- Antialiasing level – determine antialiasing level for smoothing lifted edges.

Example of antialiasing effect

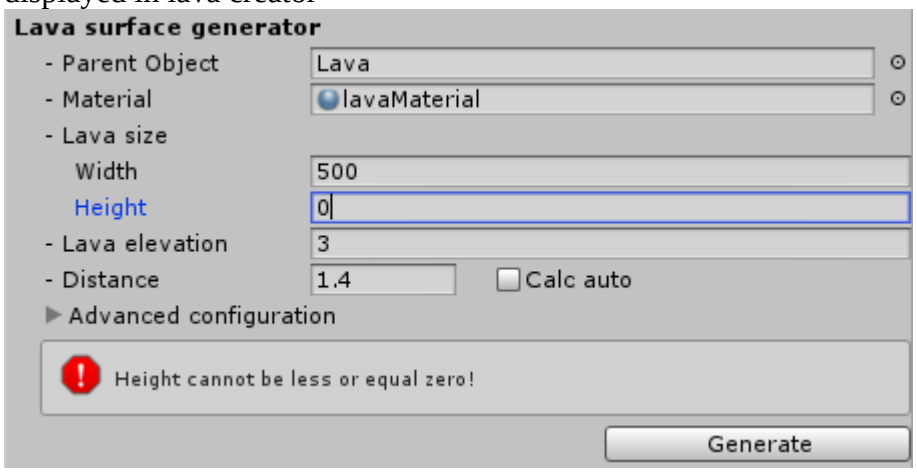


Antialiasing system level visualisation

Antialiasing system takes vertex that is lifted, and smooths vertices around. **Antialiasing level** determine how far this "around" should be. Default is level 50 and it mean, that for every lifted vertex, all its neighbors, located in area based on square with sides equal to 50 with it in center, will be subjected to the antialiasing process. Bigger value not always guarantees better effect, and always will slow down generation process. This value cannot be less than 0.

- **Steepness level** – Determine antialiasing level for smoothing lifted edges.

- **Generate** button – This will launch whole process of generating lava surface. Depending of your processor, this can take up to 5 minutes. On Intel Core i5-3210M CPU @ 2.50GHz it takes 1 minutes 27 sec to generate lava surface 500x500 with default settings. New mesh will be saved to Assets>Hell Lava>Resources>Meshes, with name [**Parent Object** name]+"*_mesh.asset*" along with its data called [**Parent Object** name]+"*_data.asset*". Next time, when you drag game object on which was created lava environment, creator will load from file **global variables** data. If you try to generate lava surface without valid data, information about it will be displayed in lava creator



Features handler

In this part we can add to our lava environment almost all features, that Hell Lava offers. Just click "Assign" and feature will be installed in your lava. For all features, configuration is already set, but if you want, you can change it. Description of each one feature and its configuration is in chapter **IV – Features.**

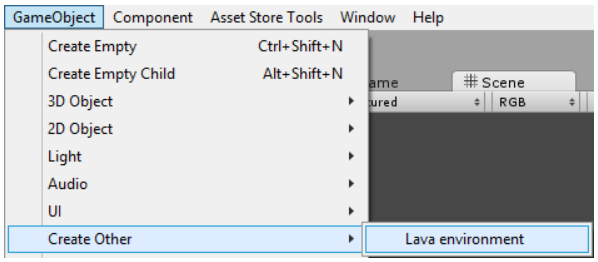
Tools

Here, you can use available for Hell Lava tools, that will save your time and help manage lava environment. More information you can find in chapter **V – Tools.**

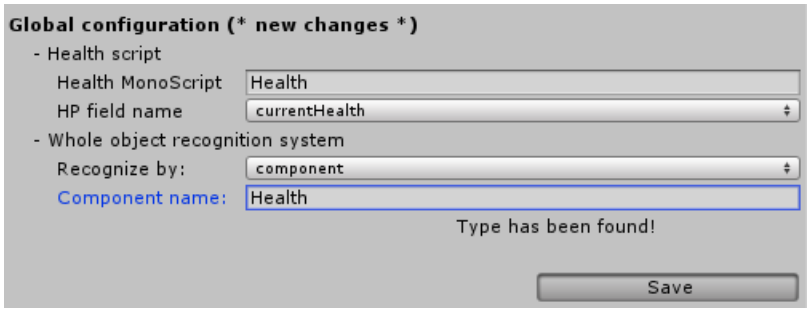
Usage example

Here you will learn how to generate your first lava environment, using Hell Lava creator.

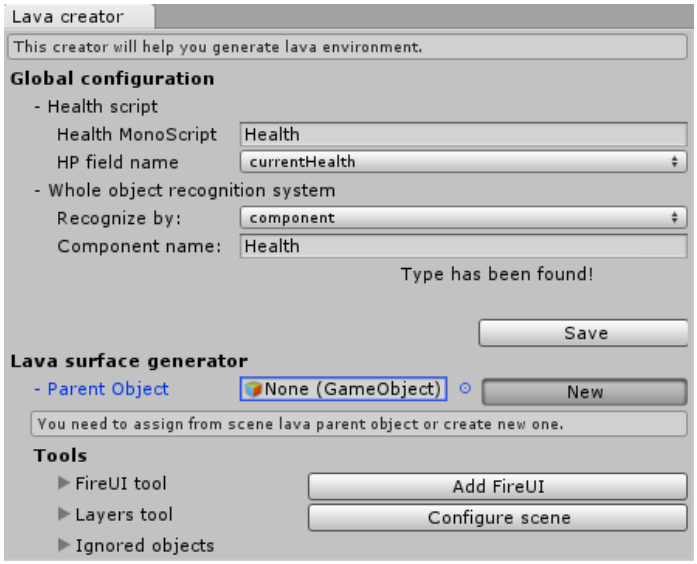
1. Go to GameObject>Create Other>lava environment.



2. Set global configuration (**not required**).
- Type name of your Health script and select field, that storage HP value.
- Select recognize mode and fill required data.
- Press "**Save**" button.

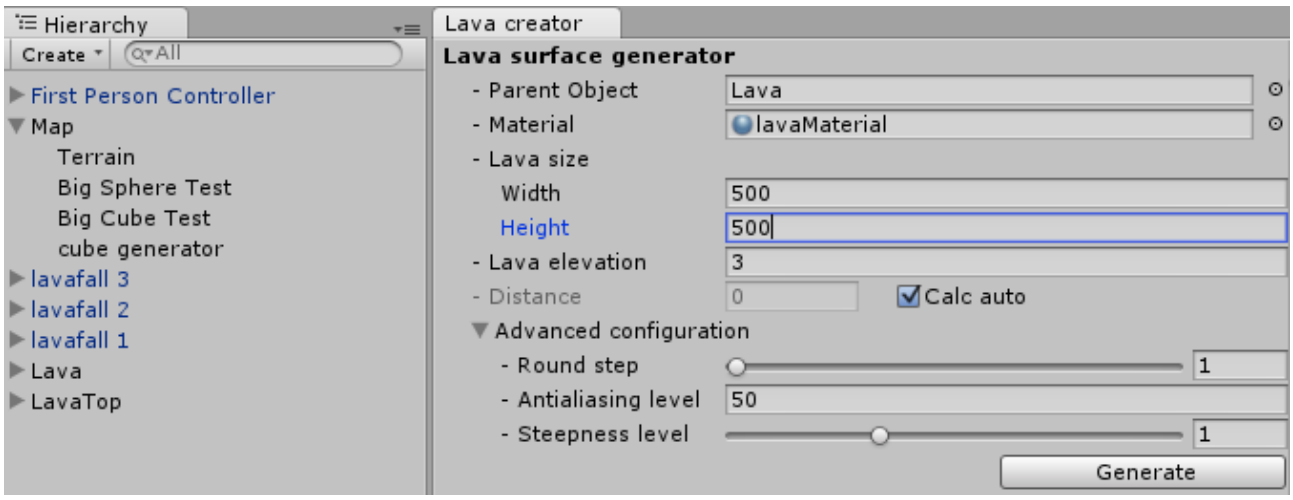


3. Press "**New**" button.



4. Assign data to Lava configuration.

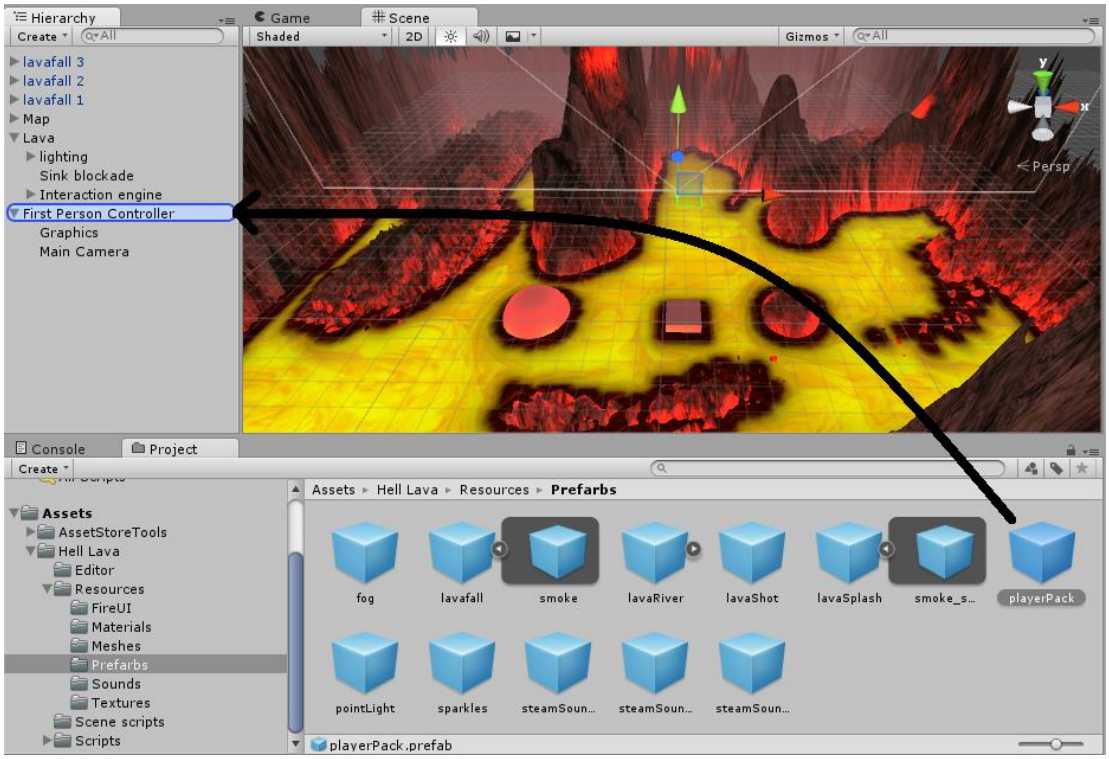
- Lava size – type size of the lava surface you want (**required**).



Everything else change as you like. More description in previous chapter.

5. Click **"Generate"** button and wait until process is finished.
6. Assign features from creator.

7. To your player game object add **playerPack** prefab, from Hell Lava>Resources>Prefabs as child. Only then Hell Lava will recognize object in lava as player object.



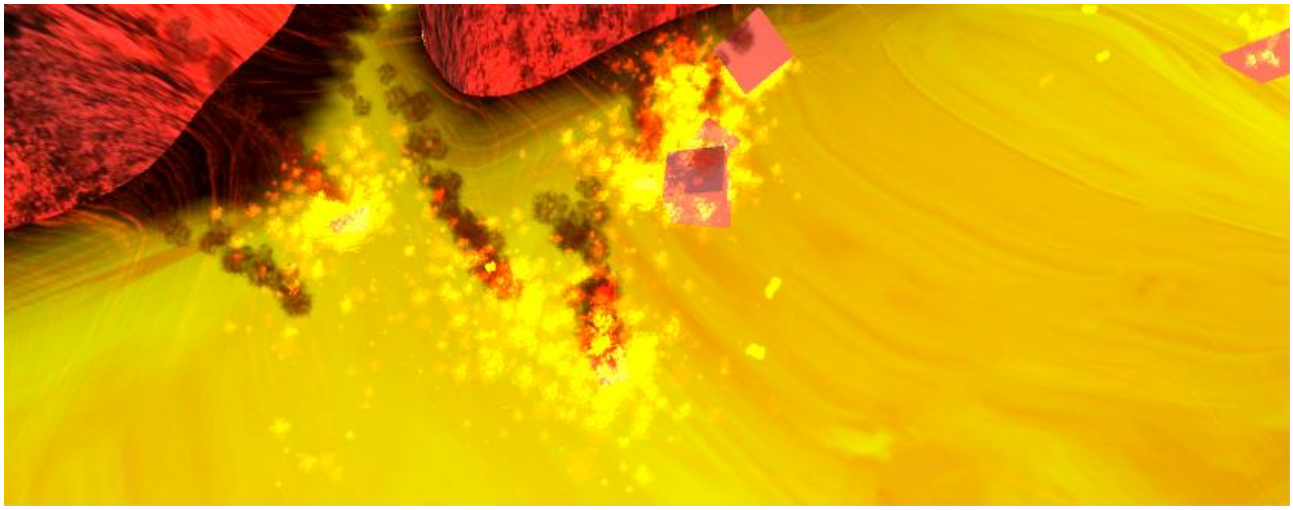
8. To add lavafalls, just drag from Hell Lava>Resources>Prefabs prefab named lavafall anywhere you want in scene. Next you may want to assign *IgnoreTrigCol* layer to just created game object. You can do it from inspector or using Layers tool from Lava creator. System will automatically find lavafalls and assign *IgnoreTrigCol* layer to them. *IgnoreTrigCol* is added after pressing "**Generate**" button.

9. Launch game and enjoy!

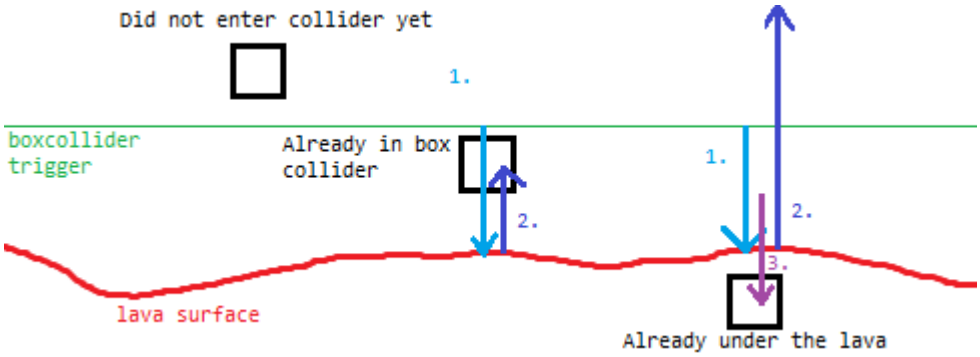
IV – Features

Hell Lava brings to your game many capabilities to diversify gameplay. Supplied by us Hell Lava features provide the ability to adapt lava environment to any situation you like.

Lava interaction engine



Since Unity 5.0.0 (where physx 3.3 was introduced), complex meshes cannot be used as triggers. To solve this problem, we had to create our own trigger system, basing on raycasts. This feature will add a box collider, that will fit to whole lava surface. Entering to box collider will trigger sample process, that will determine if object is in lava, under lava or above lava, returning also information like immersion if in lava or distance to lava surface if above or completely under surface.



1. At the beginning, one ray is shot from position where X and Z is taken from object position, and Y is from highest lava surface point, (boxcollider trigger line). This will return lava height at objects position.

2. Next ray will be shot up from point determined in first ray until object is hit. From it we can determine distance between object and lava. If cast fails, it means that object is completely under lava.

3. If second ray fails, third ray will be cast. Shot start XZ position is taken from objects position, and Y is calculated from lava height at objects position plus its collider height. From this raycast, we can determine immersion depth.

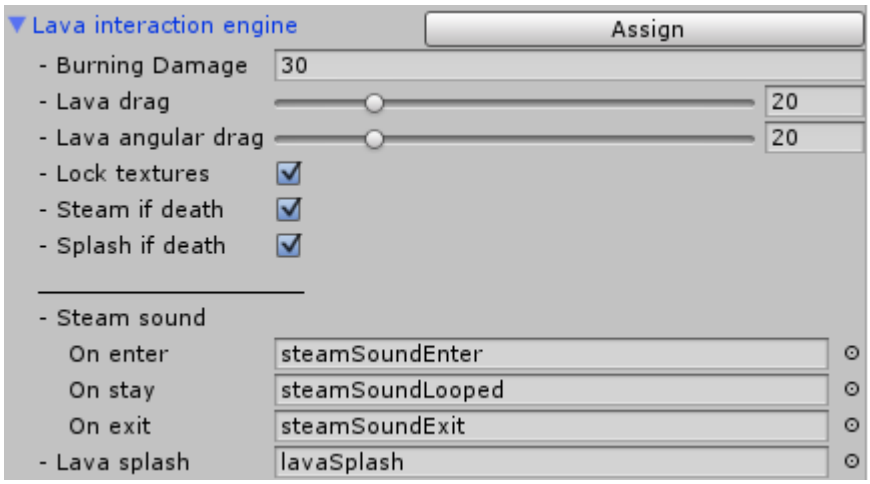
Full description of this mechanism can be read in chapter **VI – Programmer guide** part **Interaction Engine – in lava objects management**

If system determine, that object is in lava, interaction engine feature will provide effects like:
- Lava splash
- Steam sound

- Increased displacement
- Health damage

Lava interaction engine will be added to parent object with next components:
– **Mesh collider** with lava surface mesh assigned to **Mesh** field (will be updated automatically if new lava surface is generated),
– **LavaTrigger** script with filled variables,
– **Rigidbody**, set to be kinematic.

Interaction engine will have it's own child, named **Trigger**, witch attached box collider to it. Box collider will be automatically configured, so it will fit to whole lava surface. Also, this feature will provide functions to help manage objects in lava.
(More info in chapter **VI – Programmer Guide** part **Interaction Engine – in lava objects management**)



Burning Damage – an amount of damage, that is inflicted to object in one second staying in lava. It does not mater if object is complex or with only one collider. As soon as one of the character's object collider touch the surface of lava, damage will be dealt to whole object. If another collider of the same character join to lava bath, recognize system will found out that character is already in lava and won't remove health again.

Lava drag – value that will be assigned to rigidbody drag, when object enter the lava trigger to simulate lava displacement. Default is 20.

Lava angular drag – value that will be assigned to rigidbody angular drag, when object enter the lava trigger to simulate lava displacement. Default is 20. If object exit the lava trigger, and is above of it surface, object's rigidbody **drag** and **angular drag** will be set to its original values.

IMPORTANT! For lava projectiles drag won't be changed.

Lock textures – locks FireUI textures if player die in the lava. Flames wont disappear even if player leave lava after his death. Default is true.

Steam if death – if true, still instantiate constantly looped steam, even if object have 0 or less HP. Default is true.

Splash if death – if true, still instantiate constantly splash, even if object have 0 or less HP. Default is true.

Steam sound – Steam sound is a game object with attached audio source component, closed to prefab. Hell Lava have three types of steam sound: **On enter, On stay** and **On exit**. Depending on one of those three situations, specific prefab will be instantiate (*steamSoundEnter.prefab*, *steamSoundLooped.prefab* or *steamSoundExit.prefab*). **Steam Sound** have some rules, according to

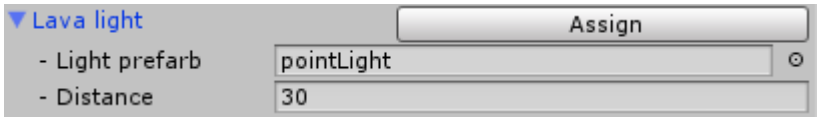
which sound will or won't be instantiate:
- When first contact with lava (onTriggerEnter), *steamSoundEnter.prefab* will be instantiated one time for whole object and attached to object as children. Children will be deleted when sound is over.
- When last contact with lava (onTriggerExit), *steamSoundExit.prefab* will be instantiated one time for whole object and attached to object as children. Children will be deleted when sound is over.
- When object stays in lava *steamSoundLooped.prefab* will be instantiated one for whole character game object and attached to object as children. Steam loop sound will stop playing when object is above or under lava surface. This sound is generated by script *SoundGenerator.cs*.

Lava splash – Lava splash is a particle system closed in *LavaSplash.prefab* prefab game object. It will be instantiated when object touch lava surface and destroyed if whole object is under or above the lava surface. Lava splash will be spawned always on the top of the lave surface, at whole game character position.

Sink blockade

This feature assign as lava children game object named *"collider"* with attached mesh collider, that will prevent object from sinking below **Block at height** level. Added game object layer will be set to *LavaCollider* and every object on *IgnoreCollider* or *IgnoreTrigCol* layer will ignore this collider. Sink blockade will be added with attached **Mesh collider** component having lava surface mesh assigned to **Mesh** field (will be updated automatically if new lava surface is generated).

Lava lighting



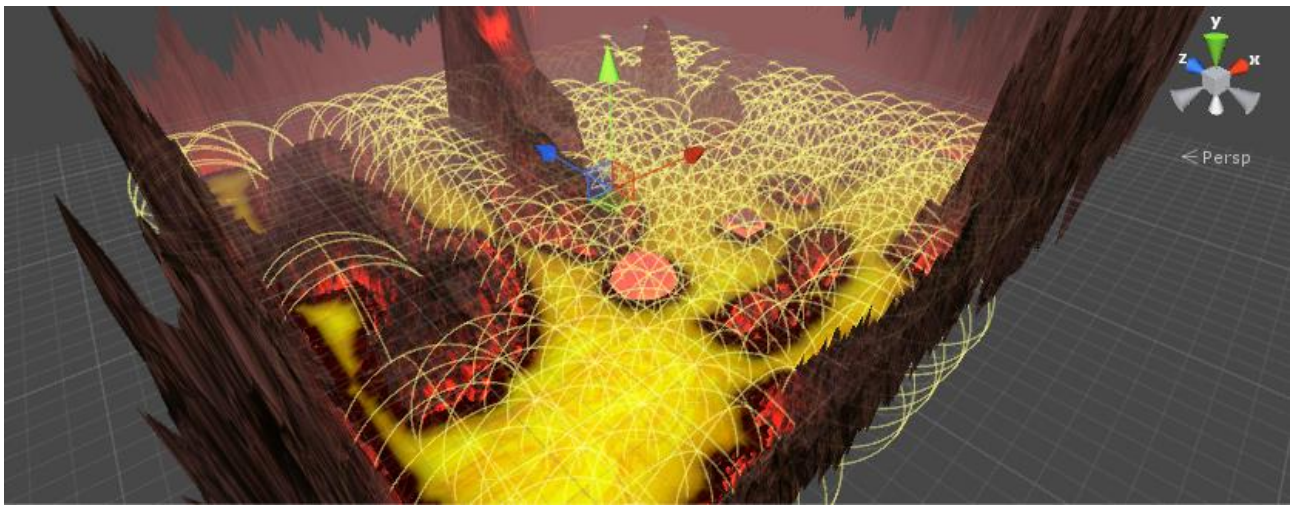
Lava lighting is submission of multiple point light. When **"Assign"** is pressed, lava creator instantiate those points from given prefab (**Light Prefab**) on the entire lava surface, maintaining a space of each other given by **Distance** variable.

Since lights are generated **only above the lava** (not under colliders or above) less space gives you better distribution on small fragments of lava, but on the other hand, it cumulates light exposure. You may adjust that with **Intensity** and **Range** in *Hell lava>Resources>Prefabs> pointLight.prefab* inspector**.**

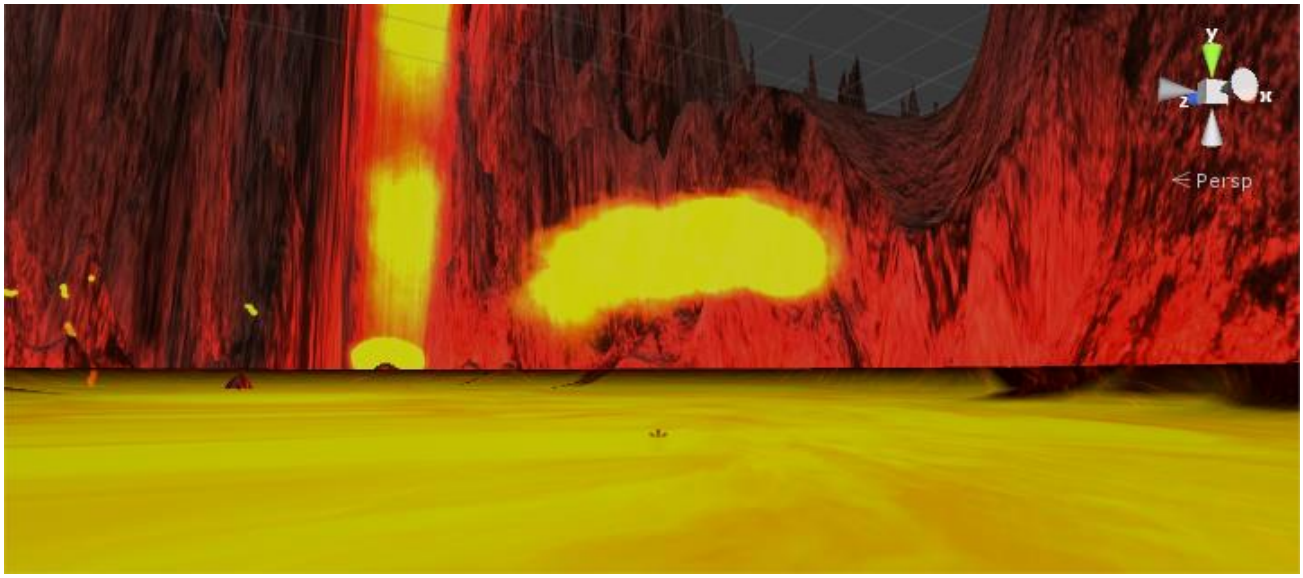
Generated light would be save as children to lava **Parent Object**, named **lighting**.

If you are planing to build big lava surface, loot of point lights can be very burdensome for player computer. It is highly recommended to set in **culling mask** (available in *Hell lava>Resources>pointLight.prefab* inspector) only necessary layers, that is layers with grounds and objects that NEED to be illuminated (note, that lava surface is not listed). Also **Render Mode** should stay set to *Not Important.*

IMPORTANT! You won't be able to generate correct shades, so lava point lights are set to not produce them.

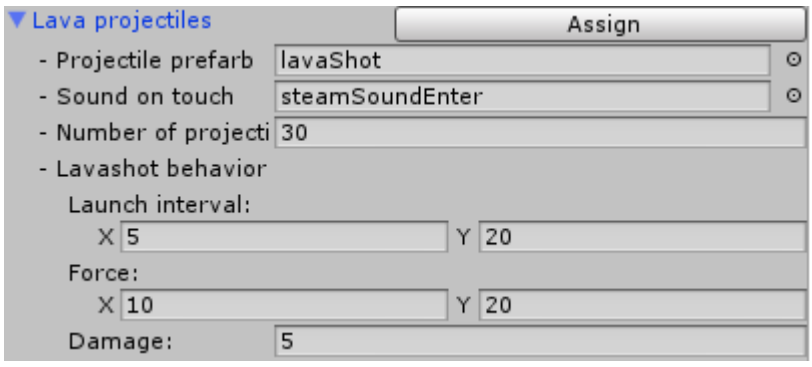


Example of lighting effect



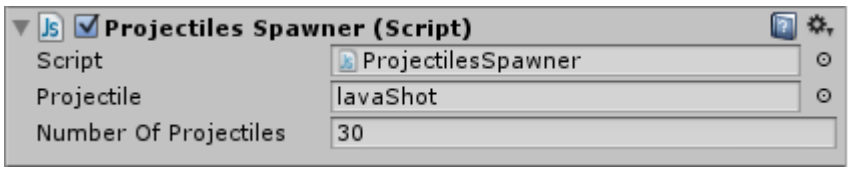
Lava projectiles

Lava projectiles are one of the lava environment features, which consists of spitting hot object from beneath the lava surface.

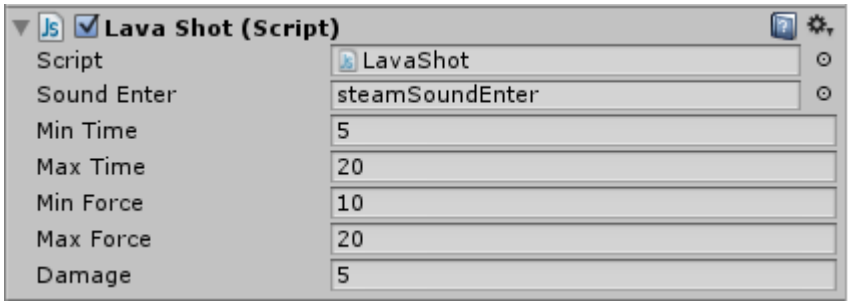


Lava projectile in mid-ari

Pressing **Assign** will effect in adding *ProjectilesSpawner.cs* to selected parent object with filled variables, and also will change variables in script *LavaShot.cs*, that is attached to prefab called **lavaShot** in Hell Lava>Resources> Prefabs.



The mechanic principle is simple. *ProjectilesSpawner.cs* will instantiate **Projectile prefab** in defined N**umber of projectiles.**

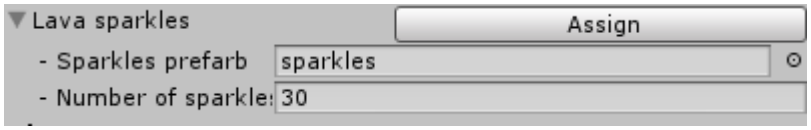


Every instantiated by *ProjectilesSpawner.cs* projectile game object, have attached own internal script – *LavaShot.cs.* To *LavaShot.cs* will be assigned variables:

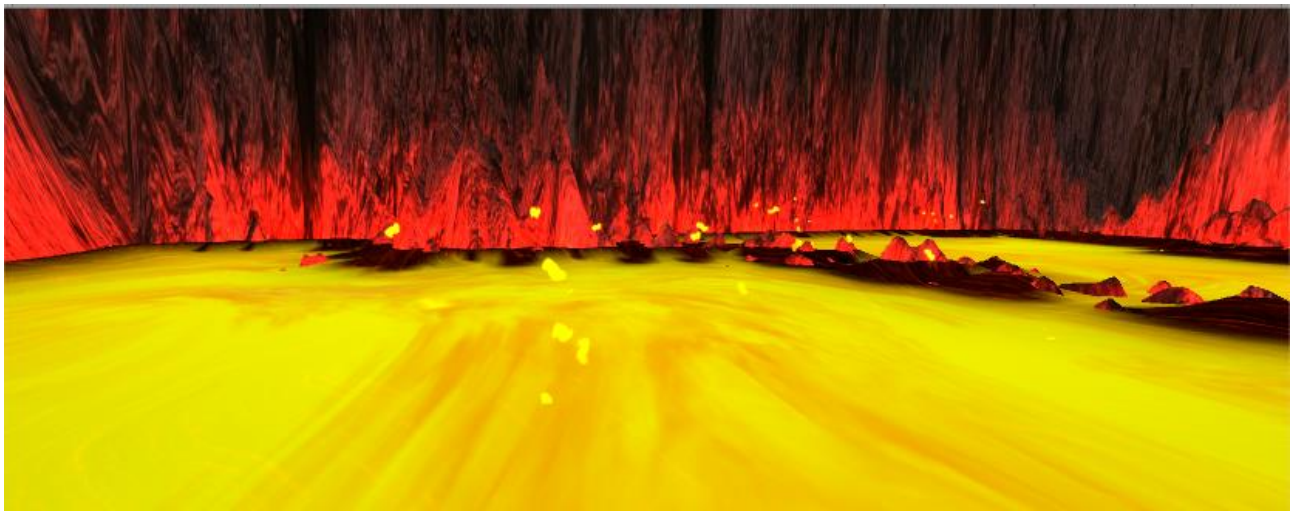
– Sound on touch → Sound enter
– Launch intervals X → Min Time,
– Launch intervals Y → Max Time,
– Force X → Min Force,
– Force Y → Max Force,
– Damage → Damage.

Projectile basically lay at the bottom of lava, and after randomized time (from **Min Time** to **Max Time**), will be shot in the air with (x,y,z) force, where:
x – randomize force in the range of –**Max Force/2** to **Max force/2** on X axis,
y – randomize force in the range of **Min Force** to **Max force** on Y axis,
z – randomize force in the range of –**Max Force/2** to **Max force/2** on Z axis.
Projectile can also damage every object with health and collider, by amount defined in **Damage** (as long as health variables are set in Lava creator's global configuration). During one launch, projectile can inflict damage to specific object only once, but when launched next time, his memory is reset and can inflict damage to objects again. When projectile hits the object, **Sound on collision** will be instantiate to play lava steam sound, and if object is the player, FireUI animation will be launched for one second.

Lava sparkles



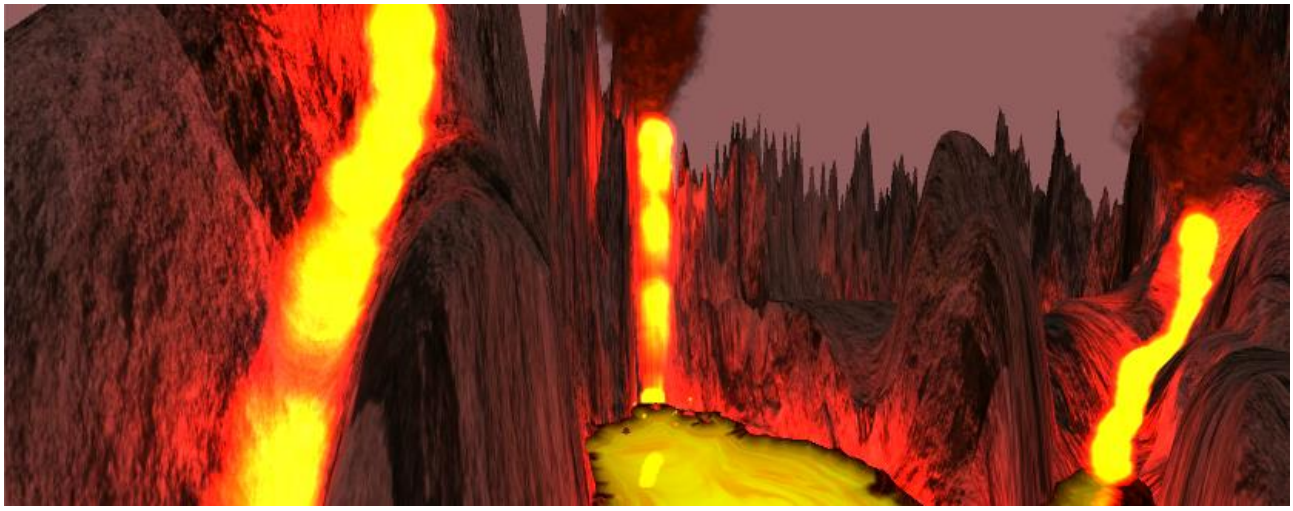
Lava sparkles is a fountain of yellow sparks, that are generated randomly on lava surface each time game is launched. This feature is handled by script called *SparkleSpawner.cs,* that is attached to **Parent Object** after pressed "Assign". **Sparkles prefab**s are instantiate by *SparkleSpawner.cs* at the beginning of the game, in number defined by **Number of sparkles.** All *SparkleSpawner.cs* variables are filled by Lava creator.



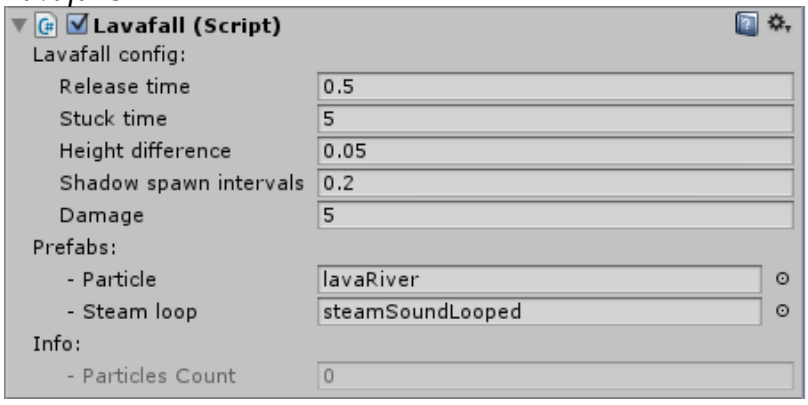
Lava sparks

Lavafall

Lavafall is a feature, that cannot be added from lava creator, because this is drag and drop object and it's up to user where he wants this feature. Lavafalls are objects with attached *Lavafall.cs* script engine and one game object's children: smoke particle system. *Lavafall.cs* instantiate special prepared prefabs, called *lavaRiver.prefab*, closed in physic spheres. When spheres fall by gravity force, particles looks like flowing liquid.



Lavafalls

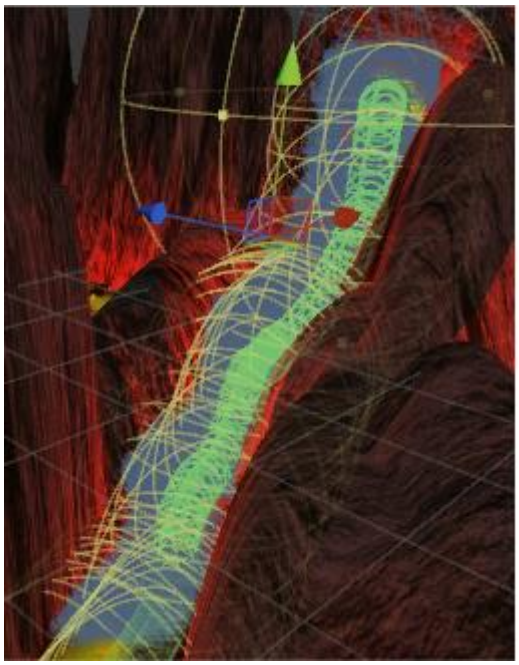


Lavafall.cs will instantiate **Particle** (*Hell Lava* > *References* > *Prefabs* > *lavaRiver.prefab*) with **Release Time** time intervals, what will incrase **Particles Count**. Lower Release Time mean better liquid effect and interaction with the world, but takes more CPU resources to handle lavafall. *Lavafall.cs* is cooperating with *LavafallRiver.cs*, the internal script of *lavaRiver.prefab*. If object height does not change more than **Height difference**, during **Stuck time**, *LavafallRiver.cs* will delete his game object. Then, *Lavafall.cs*'s **Particles Count** will be decremented. **Particles Count** cannot be changed. It's only for informational purpose.

Each *lavaRiver.prefab* instantiated by *Lavafall.cs,* is marking his path by creating kinematic sphere colliders. They are created every **Shadow spawn interval** second and are destroyed after **Release time** seconds. Thanks to that, entire lavafall can interact with environment on it's entire area. Script that handles collisions, remove health, instantiate sounds and launch FireUI animation – is called *LavafallShadow.cs.* This script is attached automatically by *LavafallRiver.cs*, so you don't have to worry about it.

Lavafall's can deal damage (as long as health variables are set in Lava creator's global configuration). **Damage** value will be subtracted from object's health script (if exist) each time, **Prefab** touch object.

IMPORTANT! Lavafall will ignore every object that is on the same layer as it is. Remember to assign unique layer for lavafall or use Layers tool from Lava creator.



Lavafall *scene view*

Installation

To enjoy you lavafall, just drag *lavafall.prefab* from *Hell Lava>Resources>Prefabs* to your scene. This will create ready to go game object. Just set position you like in **Transform** and make sure, that unique layer is assigned to lavafall game object. After game launch, lavafall will begin its run from created game object.

PlayerPack

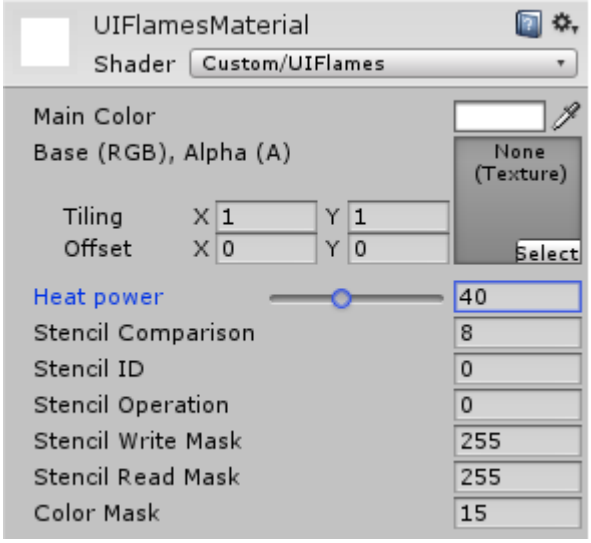
Player pack is a prefab (Hell Lava>Resources>Prefabs>PlayerPack.prefab), that after adding to game object as child, Hell Lava will start to recognize it as player and also some methods from *LavaTrigger.cs* will be able to return it as "player" object. Besides that, **PlayerPack** delivers features like flame screen effect, if player fell to lava, and lava ambient sound.

FireUI

PlayerPack have included script called *FireUI.cs*. It checks if object is actually owned by player (for multiplayer purpose) and launch on screen waving flames. If **Allow** is set to FALSE, FireUI will be never launched for this player.

Actual FireUI is handled by script *FireUIHandler.cs*. Whole mechanism is attached to screen Canvas as children object "**FireUI**". FireUI have assigned UI component *Image* and

FireUIHandler.cs script, which handles FireUI feature. Flames will show up with **Show Speed** [pixel/sec] when FireUI.cs order feature launch. To distort flames texture, UI image have assigned material (*UIFlamesMaterial materia*l) with specially designed shader – *UIFlames.shader*.

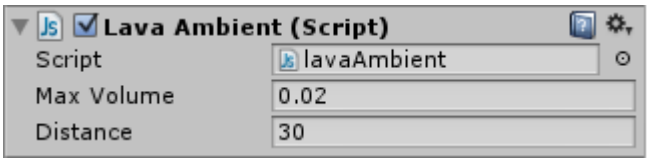


Using inspector of this material, user can change Heat power, that mean intensity of flames waving.

IMPORTANT NOTE: *FireUI.cs* is used by Hell Lava to determine if object is player or not.

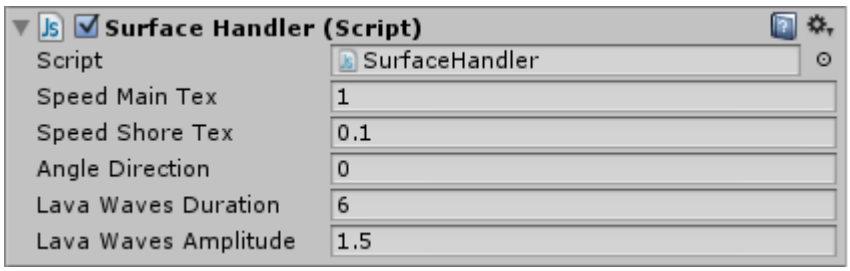
Lava ambient sound

Lava ambient sound is the feature, that allows to play ambient sound at the whole lava surface.



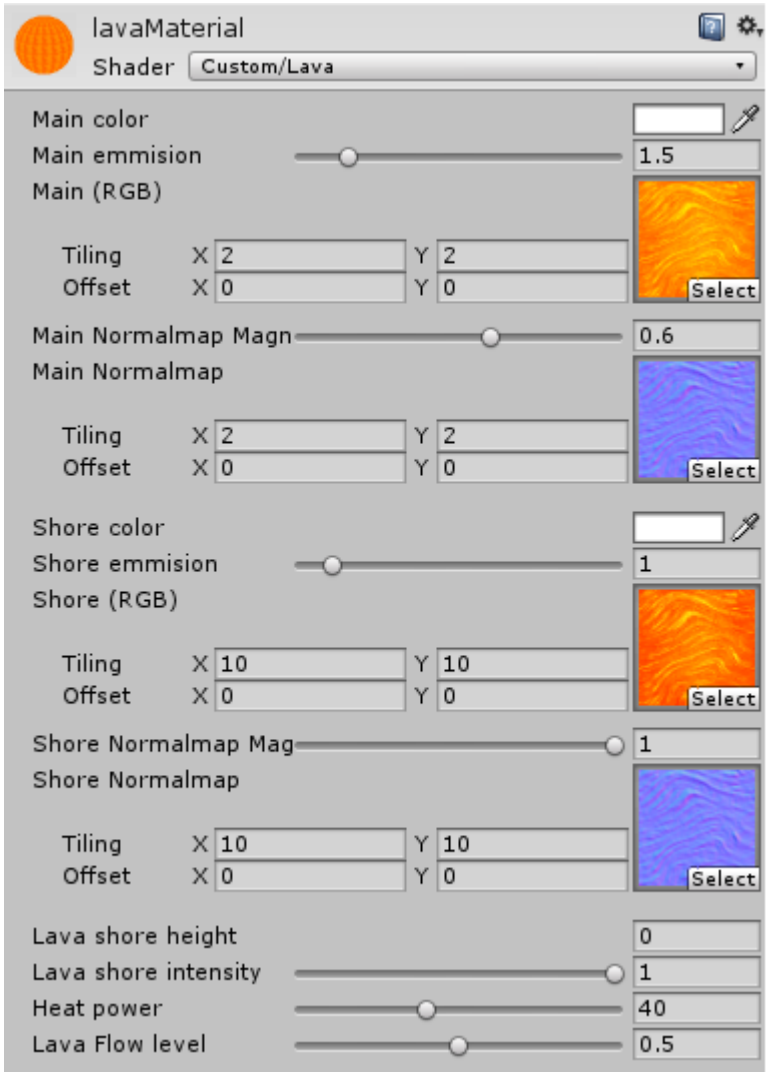
Idea is to add *Sound Source* component and script to object that suppose to "hear" lava ambient (for example player). If object is in range of **Distance** to lava, script will rise audio source volume from 0 to **Max Volume**. Audio source volume will achieve **Max Volume** value when distance between object and lava is equal 0. Lava Ambient is included in PlayerPack.

Lava surface



This feature is more mechanism than feature. *SurfaceHandler.cs* is added automatically during lava surface generation and when in play mode, passes calculated variables to *LavaShader.shader*.

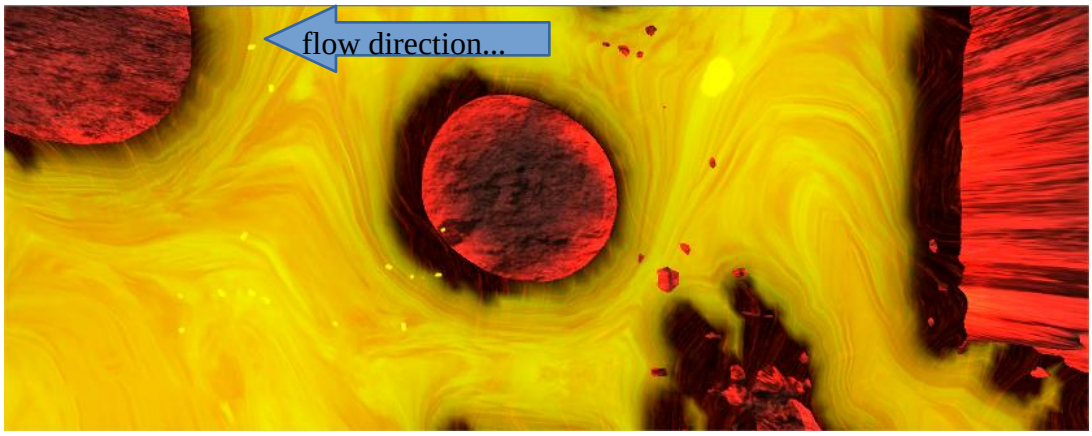
– **Speed Main Tex** is how fast should scroll main lava texture.
– **Speed Shore Tex** is how fast should scroll shore lava texture.
– **Angle Direction** is the direction of lava flow, given in degrees.
– **Lava Waves Duration** is the duration of one lava wave in seconds.
– **Lava Waves Amplitude** is the maximum swing of wave.



SurfaceHandler.cs calculate those variables and passes to lavaShader info like
– flow speed,
– direction,
– actual shore height.
Form shader you can also set **Lava shore intensity** to determine how shore should be visible, change **Heat power**, to make lava surface heat distortion and also configure **Lava Flow level** so the flow around colliders will look best to given lava surface resolution.

Lava surface will ignore any light, that is cast on it, and will emit it's own light (does NOT glow and does NOT cast light on other objects).

Lava surface will flow around terrain and objects. Also shore will be more intense behind ground in the direction of mainstream and less visible in the opposite direction.

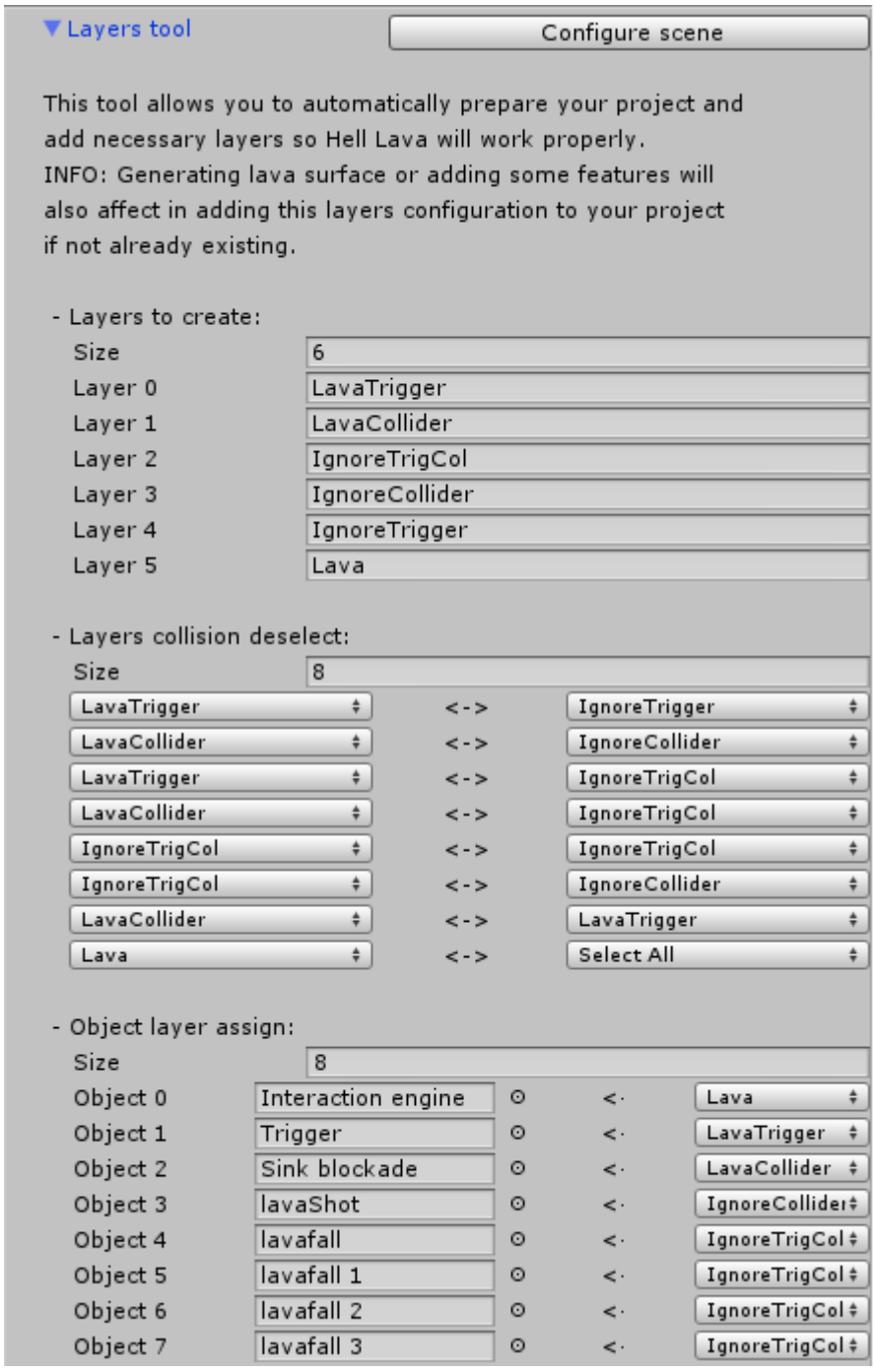


V – Tools

Hell Lava provide tools that will help you manage and configure your lava environment.

Layers tool

Layers tool is a function, from which user can add layers, set dependence between them and assign to selected objects. This is especial useful for Hell Lava, since it can automatically find object used in lava environment and assign required layers, saving loot of user's time.



In **Layers to create** we type layers that we want to create.

LavaTrigger, LavaCollider, IgnoreTrigCol, IgnoreCollider, IngoreTrigger and Lava are default layers, that will always appear in **Layers to create.**

The same applies to **Layers collision deselect**, where given dependencies will always be present. In this section we can choose from already existing layers and those we typed in **Layers to create**. Between selected layers, will be removed collision in Layer Collision Matrix (Edit>Project settings>Physics). Also you can choose "*Select all*", so selected layer will have no collision with any layer. The only restriction is that you can't type *Select All* ↔ *Select All*, what would be terrible in consequences.

In Last section, **Object layer assign**, we will assign prepared layers to objects. Layers tool will always try to find:

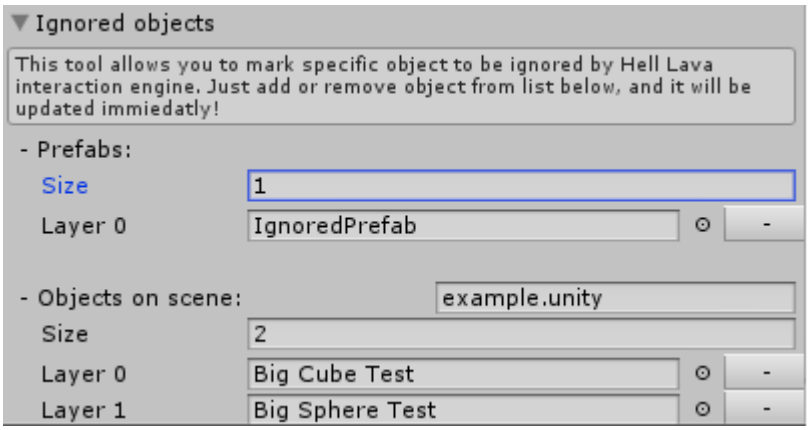
- Interaction engines,
- Triggers,
- Sink blockades,
- lavaShots,
- lavafalls.

FireUI tool

FireUI Tool is a method to automatically add necessary objects and components to UI Canvas so FireUI feature can work properly. When "**Add FireUI**" button is pressed, script will find reference to any canvas present on scene or add one, if none are added. Then shall be created children to this Canvas named **FireUI** with attached UI Image and FireUIHandler script. All variables will be set to defaults.

This tool is also used at the end of lava surface generate process.

Ignored objects



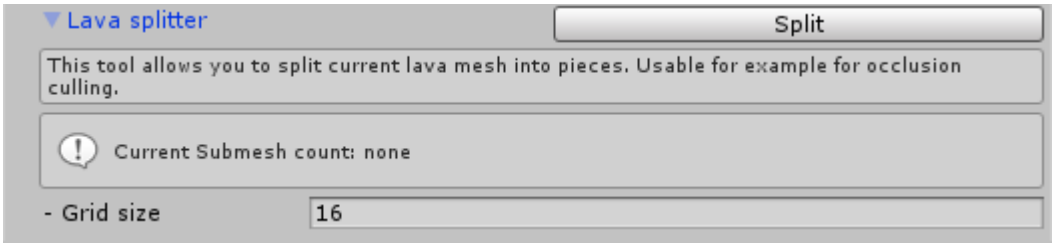
This tool allows you mark object, that shall be ignored by Lava Interaction Engine.

First section "**Prefabs**" is for asset prefabs and for every scene this array will always be the same. Second section is desired for objects present on currently opened scene. Name of that scene will be displayed right above the array.

Adding or removing object with this tool will make changes immediately, that mean add or remove from object script called *IgnoreObject.cs*. Hell Lava checks if object have this script attached and ignore him, if it does.

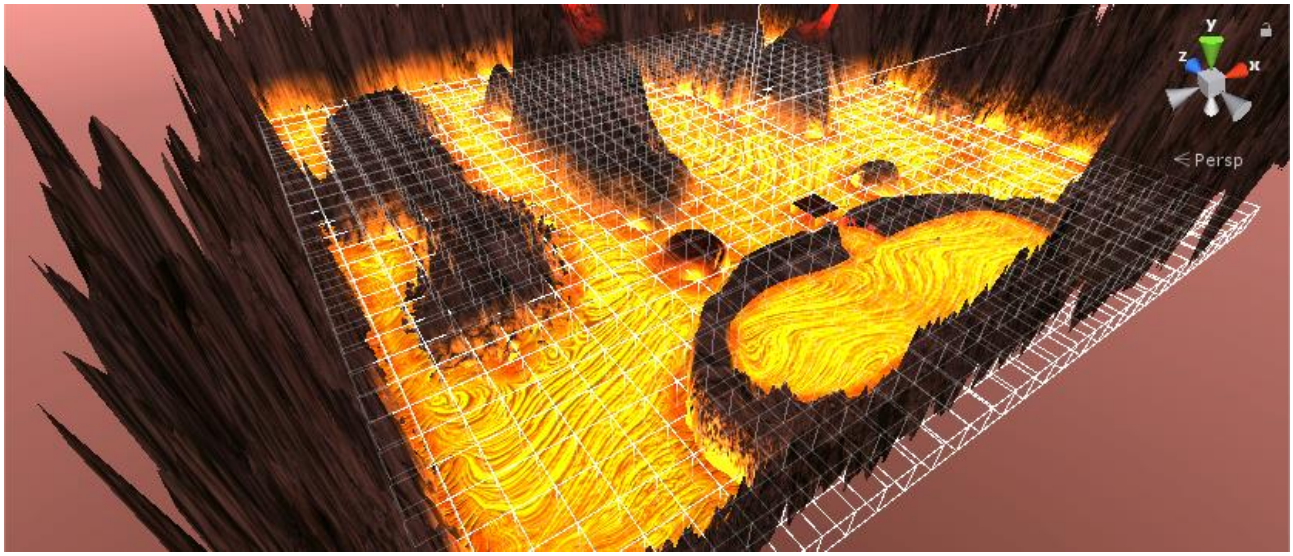
Pressing "-" button will remove object from array, and at the same unmark it from ignored objects. The same situation is when size of the array is changed to smaller than previous value. All object beyond the new size will be removed from ignored object's list.

Lava Splitter

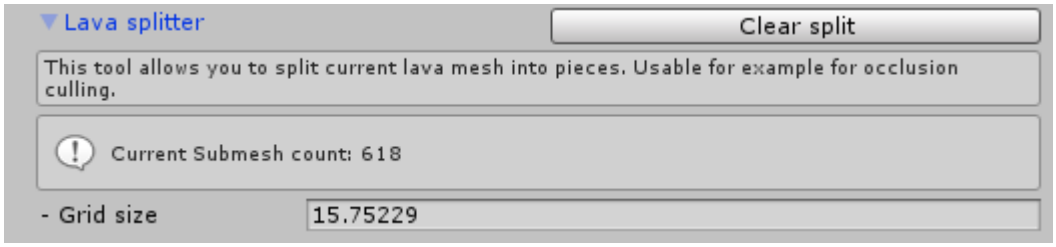


Lava splitter allows you to split lava mesh into pieces of **Grid size**. It requires **Parent Object** set in the Lava Creator. This tool is useful for example if you have really big lava mesh that cannot be occluded as it is.

If **Parent Object** is set and tool is expanded, grid will be drawn on lava surface to indicate how mesh would be split with current **Grid size.**



If lava is already split, "**Split**" button will be replaced with "**Clear split**". Pressing it will cause to remove all submeshes and restore parent lava object to original state.



Also, tool always display how many submeshes are created. If lava is not split or could not be split, "**none**" value will be displayed.

VI – Programmer Guide

The power of Hell Lava lies in it's flexibility. Hell Lava Interaction Engine provides many data about objects in lava and functions, that will help manage them. Most important is class *InLava.cs* (Hell Lava>Scripts>Class), that contains information about each object in lava.

InLava class

InLava – Contains information about objects in lava and can determine their behavior. Class is used by Interaction Engine to manage objects in lava, and also by lavafalls and lava projectiles to determine whole object and subtract damage.

Variables:

–	transform	: Transform	- single object's or whole object's transform
–	objectDrag	: float	- displacement variables – initial drag
–	objectAngularDrag	: float	- displacement variables – initial angular drags
–	wholeObject	: InLava	- whole object
–	healthScript	: Component	- whole object health script
–	lavaHeightAtObjects	: float	- lava height at transform position
–	howDeepInLava	: float	- object immersion depth in lava
–	inLava	: bool	- is transform in lava?
–	lavaDeath	: bool	- is whole object died in lava?
–	isPlayer	: bool	- is whole object a player?
–	fireUIScript	: FireUI	- whole object FireUI script
–	childCounter	: int	- counter of whole object child still being in lava
–	enter	: bool	- does it first transform contact with lava?

Constructors:

InLava(transform : Transform, isWholeObject : bool = false) – Constructor that will create InLava object and find WholeObject, basing on transform of SINGLE object (given as parameter) and recognition system (configured in Lava Creator) OR thread transform as already WholeObject, where constructor will try to find **health script** (basing on given parameters during InLava Initializon System), and **FireUI scrip**t. If FireUI script will be found, whole object will be treated as player and **isPlayer** will become true.

InLava(copy : InLava) – Copy constructor.

Methods:

int **GetHp**() **–** If health system was initialized and InLava have assigned **healthScript**, function will return object health. Otherwise, will return 0.

void **DealDamage**(damage : float) – If health system was initialized and InLava have assigned **healthScript**, function will subtract given amount of health. Otherwise nothing will happen. If object have **isPlayer** = true and **fireUIScript** != null, FireUI will be launched for 0.1 sec.

void **SetDisplacement**(lavaDrag : float, lavaAngularDrag : float) – if transform have attached rigidbody, sets drag and angular drag. Otherwise nothing will happen.

void **ResetDisplacement**() – if transform have attached rigidbody, reassign drag and angular drag initial values.

Static Functions:

void **InitInLavaSystem**() – Function will assign health system and recognition system basing on data

set in Lava Creator global configuration. Data are passed using PlayerPrefs.

MeshData class

MeshData is mainly a storage of lava surface data, used to save and load information by Lava Creator, however, class provides some functions used by Interaction Engine, to determine if object is in lava or not.

Variables:

– material : Material - lava material
– width : int - lava mesh width
– height : int - lava mesh height
– dist : float - distance between two vertices
– roundStep : float - step between degrees
– antialiasingLevel : int - antialiasing level
– steepness : float - lava lift steepness.
– lavaElevation : float - the value of which lava will be raised at the shore
– autoDist : bool - calculate distance automatically?
– isConfigured : bool - was the lava configured? Assigned by Layers tool

Constructor:
Since MeshData extends ScriptableObject, it cannot be constructed in standard way. To use object with type MeshData, you have to create instance of ScritableObject:
var meshData: MeshData = ScriptableObject.**CreateInstance**<MeshData>();

void **Assign**(toCopy : MeshData) – Similar to copy constructor

Functions:
void **DefaultInit**() – fills MeshData variables with default values.

void **CreateAsset**(path : String, name : String) – save MeshData to asset file at given path under given name. This function is available only in Unity Edit mode.

void **LoadAsset**(path : String, name : String) – load data from file at given path and assign loaded data to MeshData.

bool **SampleHeight**(pos : Vector3, collider : Collider, out (float meshHeight, float distance, bool objectInLava)) – this function samples lava surface height at given position and returns additional data about given collider status:
– meshHeight – mesh height at given position,
– distance – if objectInLava == false : collider distance to lava surface
if objectInLava == true : collider distance to complete immersion
– objectInLava – true: collider in lava; false: collider not in lava
If could not determine even mesh height at given position, function will return false. If could not determine collider distance to lava surface or complete immersion, function will return (height, 1, false).

Interaction Engine – in lava objects management

Interaction engine provides some functions and variables, so user can use it in its own scripts. All of them are available from *LavaTrigger.cs* script, after getting reference to *LavaTrigger* component attached to specific object.

Variables:
- lavaTransform : Transform - used to determine lava trigger transform, for other scripts
- inLavaObjects : List<InLava> - generic list with all objects present in lava box collider

Functions:
List<InLava> **ObjectsInLava**() – return genetic list with all object, that are in lava.

InLava **ObjectInLava**(object : GameObject) – return InLava object that whole object is equal to given object. If not found, return null.

bool **IsPlayerInLava**() – return true if found any player that is in lava. Otherwise or when no player was found, return false. Helpful in singleplayer games.

bool **IsPlayerInLava**(playerObject : GameObject) – return true if found given player in lava. Otherwise or when no player was found, return false. Helpful in multiplayer games.

bool **PlayerDiedInLava**() – return true if found any player that died in lava. Otherwise or when no player was found, return false. Helpful in singleplayer games.

bool **PlayerDiedInLava**(playerObject : GameObject) – return true if given player died in lava. Otherwise or when no player was found, return false. Helpful in multiplayer games,

float **LavaHeightAtPlayer**() – return lava height at first found player position. If no player was found, return 0. Helpful in singleplayer games.

float **LavaHeightAtPlayer**(playerObject : GameObject) – return lava height at given player position. If player was not found, return 0. Helpful in multiplayer games.

List<InLava> **PlayersInLava**() – return InLava generic list of all players in lava. Helpful in multiplayer games.

Other Scripts

Hell Lava functionality cover also triggering scripts, that handles some mechanisms. Below are given all scripts, that can be trigged to get some specific effect.

FireUI usage

FireUIHandler.cs (Hell Lava>Scripts) is a script that handle showing flames on player screen. LavaTrigger often uses it when players is in lava. *FireUIHandler.cs* can be triggered by using:
`FireUIHandler.Get().LaunchFlames()` or `FireUIHandler.Get().LaunchFlames(timeInSeconds : float)`

First way will launch UI flames for as long as method is called, for example in Update() instance. Second method will launch flames only for given amount of time (in second). When **time** become

positive, script will remove Time.**deltaTime** value until it's equal 0 or less. Then, flames will become invisible.

SoundGenerator usage

SoundGenerator.cs (Hell Lava>Scripts) is a script that will generate looped lava steam sound as long, as it lives. The only requirement is attached SoundSource to the same game object as *SoundGenerator.cs* is.

At the first moment of script life, SoundSource volume will be set to 0, but with speed of SoundGenerator.**damping** (volume unit per second), volume will increase to SoundSource volume initial value.

Script can destroy its own game object. It is used when steam loop sound is no longer needed. First way is to set SoundGenerator.**destroy** to true. Script will volume down sound with speed of SoundGenerator.**damping** until it reach 0 and destroy its own game object. Second way is by delayed destroy, using SoundGenerator.**timedDestroy**. After given amount of seconds, script will set SoundGenerator.**destroy** to true and whole process will begin look like in first way. You can also set SoundGenerator.**timedDestroy** to negative value, so script will set **destroy** to true immediately. Only way to stop destroy process is to first set **timedDestroy** to 0, and next assign false to **destroy**.

SplashHandler usage

SplashHandler.cs (Hell Lava>Scripts) is a script that handle lava splash particle functionality. Check if two particle systems are added to its game object (one of them as child) and keeps rotation always same as at the first moment of game object's life. Script also handle destroy process if needed.

First way to destroy splash prefab through script is to set SplashHandler.**destroy** to true. Script will turn of both particle systems and wait until their energy is gone. Then will destroy its own game object. Second way is by delayed destroy, using SplashHandler.**timedDestroy**. After given amount of seconds, script will set SplashHandler.**destroy** to true and whole process will begin look like in first way. You can also set SplashHandler.**timedDestroy** to negative value, so script will set **destroy** to true immediately. Only way to stop destroy process is to first set **timedDestroy** to 0, and next assign false to **destroy**.

Example

Below is an example how to use Hell Lava Interaction Engine functions and mechanisms. Scenario is that we will check if certain object is in lava, so we can launch Fire UI, only if its immersion is greater than 1. Also we have few lavas in scene, so we have to check them all.

```
using UnityEngine;

public class Example : MonoBehaviour
{
    public GameObject objectToCheck;
    private LavaTrigger[] lavaTriggers;
    void Start() {
        //we need to find all lavas available at scene
        lavaTriggers = FindObjectsOfType<LavaTrigger>();
    }
    void Update(){
        //try to find our objectToCheck in any available lava
        InLava inLavaWholeObject = null;
        for (int i = 0; i< lavaTriggers.Length; i++)
        {
```

```
            inLavaWholeObject = lavaTriggers[i].ObjectInLava(objectToCheck);
            if (inLavaWholeObject != null)
                break;
        }
        if (inLavaWholeObject != null){
            //object was found! But it does not mean it is in lava
            if (inLavaWholeObject.wholeObject.inLava){
                //it is in lava indeed! Check it's depth immersion.
                if (inLavaWholeObject.wholeObject.howDeepInLava <= 1){
                    //as we wanted, immersion depth is greater than 1
                    //we will launch FireUI script, as long our object have it
                    if (inLavaWholeObject.wholeObject.fireUIScript != null)
                        inLavaWholeObject.wholeObject.fireUIScript.timedLaunch = 0.1f;
                }
            }
        }
        else
            Debug.Log("Object was not found");
    }
}
```

VII – Troubleshooting

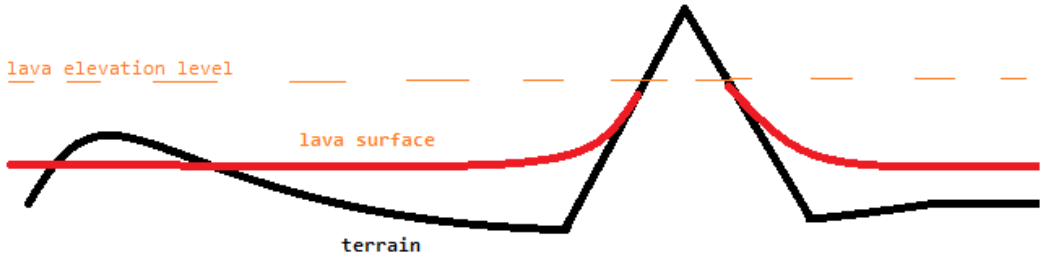
Here will be described all know problems, that you may encounter.

1. Bad mesh resolution

In Unity, mesh cannot have more than 65000 vertices. Bigger lava surface area mean more vertices, so it may be, that script will assign bigger distance value between vertices to fit in 65000 vertices. You may try to set lower **Round step** value in lava creator to avoid interference and noise in surface mesh. Lower **Round step** value takes more time to generate mesh. For example, to generate lava surface 500x500 with distance set to 2 and **Round step** to 1, it takes around 1.5 minutes on Intel Core i5-3210M CPU @ 2.50GHz. Changing **Round step** to 0.1, extended that time to 17 minutes.

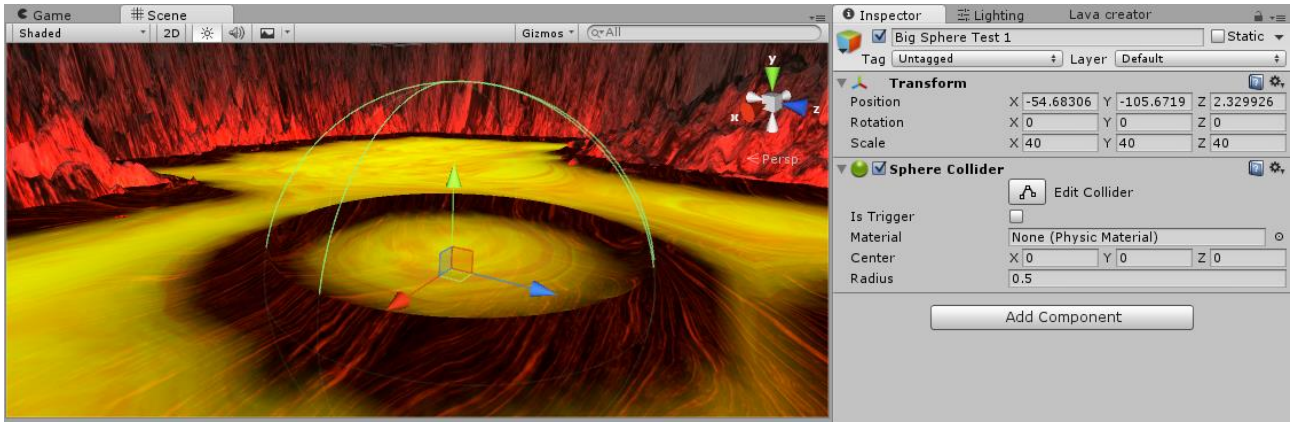
2. Lava do not lift on some colliders

The reason is because colliders vertices, that was ignored, are lower than **Lava Elevation** value was set in lava creator:



Notice that also mesh under ignored vertices won't be cut. You can raise ignored vertices or lower the **Lava Elevation** value.

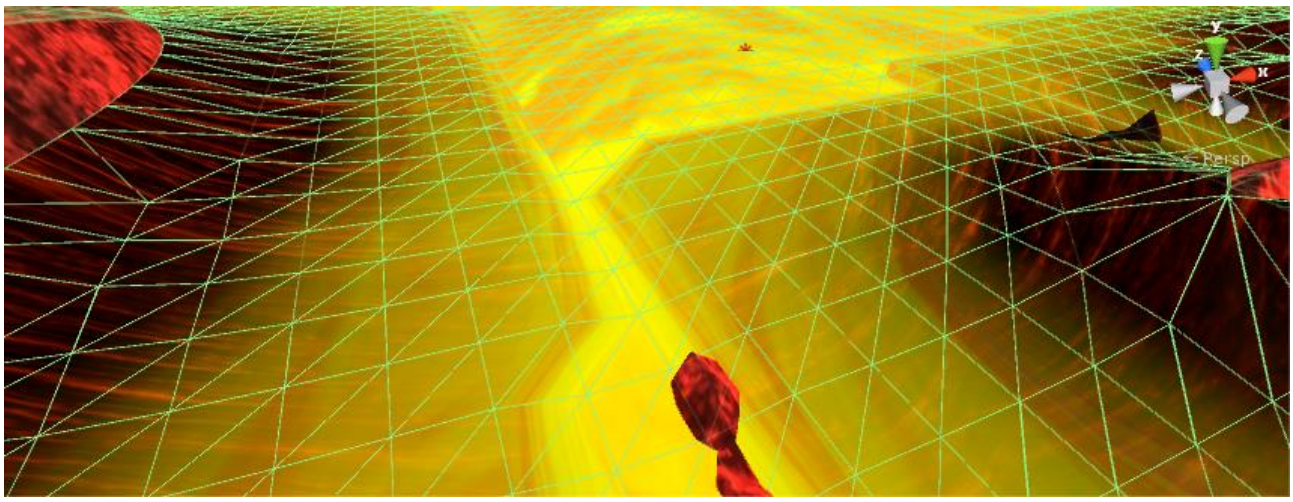
3. Weird disruption on lava surface



If you experience weird disruption on lava surface, like ditches and bumps, check if there is no collider, that was placed at the level of lava surface, during mesh creation.

The only way is to remove collider for the time of creating lava surface. When you remove collider that disrupt lava, press again *"Generate"* in lava creator.

You may also notice some kind of slope on lava surface.



It is caused by lack of lava antialiasing level. Bigger empty field on lava surface (without objects) require higher antialiasing level. Just increase value of **Antialiasing level** in lava creator and press *"Generate"* button. Try with different values until get expected effect. Just remember that higher **Antialiasing level** take more time to generate lava surface mesh.

4. I'm getting error *BCE0005: Unknown identifier: 'LavaTrigger'*, or similar.

Just move named in error scripts inside "Standard Assets" folder, or to folder named "Plugin", created directly in "Asset" folder.

https://docs.unity3d.com/Manual/ScriptCompileOrderFolders.html

For the most part, you can choose any names you like for the folders in your project but Unity reserves some names to indicate that the contents have a special purpose. Some of these folders have an effect on the order of script compilation. Essentially, there are four separate phases of script compilation and the phase where a script will be compiled is determined by its parent folder.

This is significant in cases where a script must refer to classes defined in other scripts. The basic rule is that anything that will be compiled in a phase after the current one cannot be referenced. Anything that is compiled in the current phase or an earlier phase is fully available.

Another situation occurs when a script written in one language must refer to a class defined in another language (say, a UnityScript file that declares variables of a class defined in a C# script). The rule here is that the class being referenced must have been compiled in an earlier phase.

The phases of compilation are as follows:

Phase 1: Runtime scripts in folders called Standard Assets, Pro Standard Assets and Plugins.

Phase 2: Editor scripts in folders called Standard Assets/Editor, Pro Standard Assets/Editor and Plugins/Editor.

Phase 3: All other scripts that are not inside a folder called Editor.

Phase 4: All remaining scripts (ie, the ones that are inside a folder called Editor).

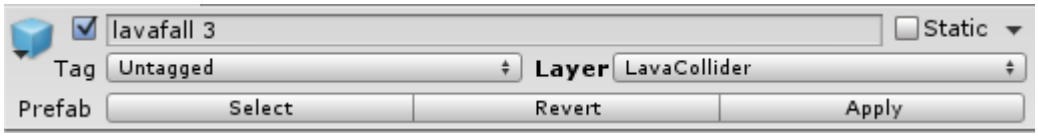
Additionally, any script inside a folder called WebPlayerTemplates at the top level of the Assets folder will not be compiled at all. This behavior is slightly different from the other special folder names which also work within sub-folders (eg, Scripts/Editor works as an editor script folder but Scripts/WebPlayerTemplates does not prevent compilation).

A common example is where a UnityScript file needs to reference a class defined in a C# file. You

can achieve this by placing the C# file inside a Plugins folder and the UnityScript file in a non-special folder. If you don't do this, you will get an error saying the C# class cannot be found.

5. After using Layers tool from Lava Creator, some objects are ignored during layer assigning, even if they are present on objects list.

If before you had changed object's layer through inspector, Layers tool won't be able to assign own layer until objects changes are accepted. You can notice that by bold font label next to changed field. Just press **Apply** and Layers tool will again be able to change layer for this game object. Just remember that pressing "Apply" will save changes to object's prefab.



6. Lavafalls are not interacting with my objects!

Lavafalls are using OnTriggerStay message to detect if any object is in their stream. Since Unity 5.1.0, where OnTriggerStay has been optimized, it just stopped working. The only workaround is to add to object (that suppose to interact with lava) script, with simple:

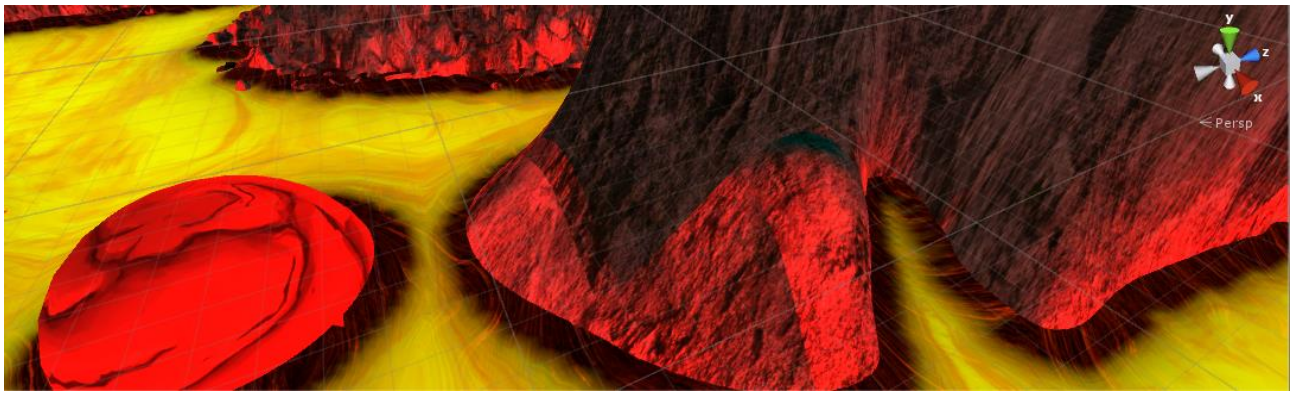
```
Private void OnTriggerStay(Collider coll){}
```

After that, Lavafall's OnTriggerStay starts to work.

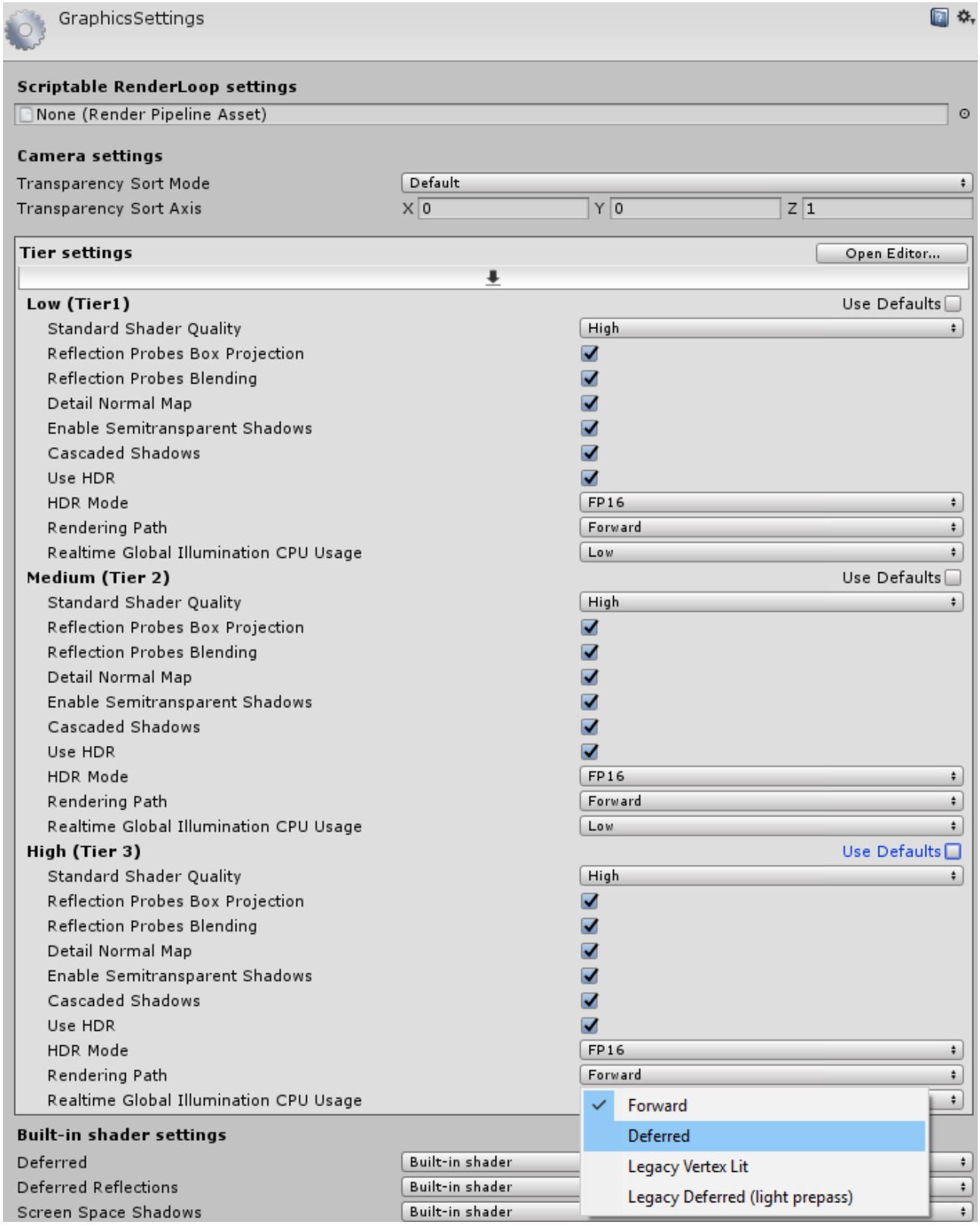
Note: On Unity 5.1.3 problem is gone.

7. Lava lighting fails in lighting

You may notice, that after adding lighting system, or just after launching example scene, ground is not is not illuminated properly.



This problem can be solved by switching **rendering path** method from *Forward* to *Deffered*, as is much more "light friendly". To switch that option, from top bar select *Edit>Project Settings>Graphic*. In the Inspector find the **Tier settings** and in the each tier, you will find Rendering Path option. Just switch to *Deffered* and problem is gone.


GraphicsSettings
Scriptable RenderLoop settings
None (Render Pipeline Asset)
Camera settings
Transparency Sort Mode
Default
Transparency Sort Axis
X 0
Y 0
Z 1
Tier settings
Open Editor...
Low (Tier1)
Use Defaults
Standard Shader Quality
High
Reflection Probes Box Projection
Reflection Probes Blending
Detail Normal Map
Enable Semitransparent Shadows
Cascaded Shadows
Use HDR
HDR Mode
FP16
Rendering Path
Forward
Realtime Global Illumination CPU Usage
Low
Medium (Tier 2)
Use Defaults
Standard Shader Quality
High
Reflection Probes Box Projection
Reflection Probes Blending
Detail Normal Map
Enable Semitransparent Shadows
Cascaded Shadows
Use HDR
HDR Mode
FP16
Rendering Path
Forward
Realtime Global Illumination CPU Usage
Low
High (Tier 3)
Use Defaults
Standard Shader Quality
High
Reflection Probes Box Projection
Reflection Probes Blending
Detail Normal Map
Enable Semitransparent Shadows
Cascaded Shadows
Use HDR
HDR Mode
FP16
Rendering Path
Forward
Realtime Global Illumination CPU Usage
Forward
Deferred
Legacy Vertex Lit
Legacy Deferred (light prepass)
Built-in shader settings
Deferred
Built-in shader
Deferred Reflections
Built-in shader
Screen Space Shadows
Built-in shader