Indi Games Engine

Release 0.0.1

admin@indigames.net

Sep 12, 2022

GETTING STARTED

Indigames Game Engine is a flexible, efficient, free to use game engine, supports developing high quality games with ease and speed.

The documents include detailed instructions, and step-by-step tutorials to help you quickly learn how to develop cross-platform games with Indigames' Engine.

Note: This project is under active development.

CHAPTER

ONE

CONTENTS

1.1 Installation

1.1.1 From sources

Compiling igeCreator from sources requires using Visual Studio 2019 and CMake. You will need to clone the repository and run the `scripts\genProject.bat`, the visual studio project will be generated in `project\igeCreator.sln`.

1.1.2 From a release build

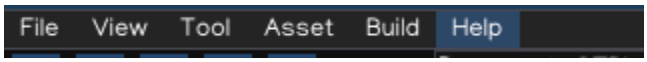
You can download the release build by checking a releases list.

1.2 Editor Layout

When launching igeCreator for the first time, you will see the Editor window similar to this:

1.2.1 Menu Bar

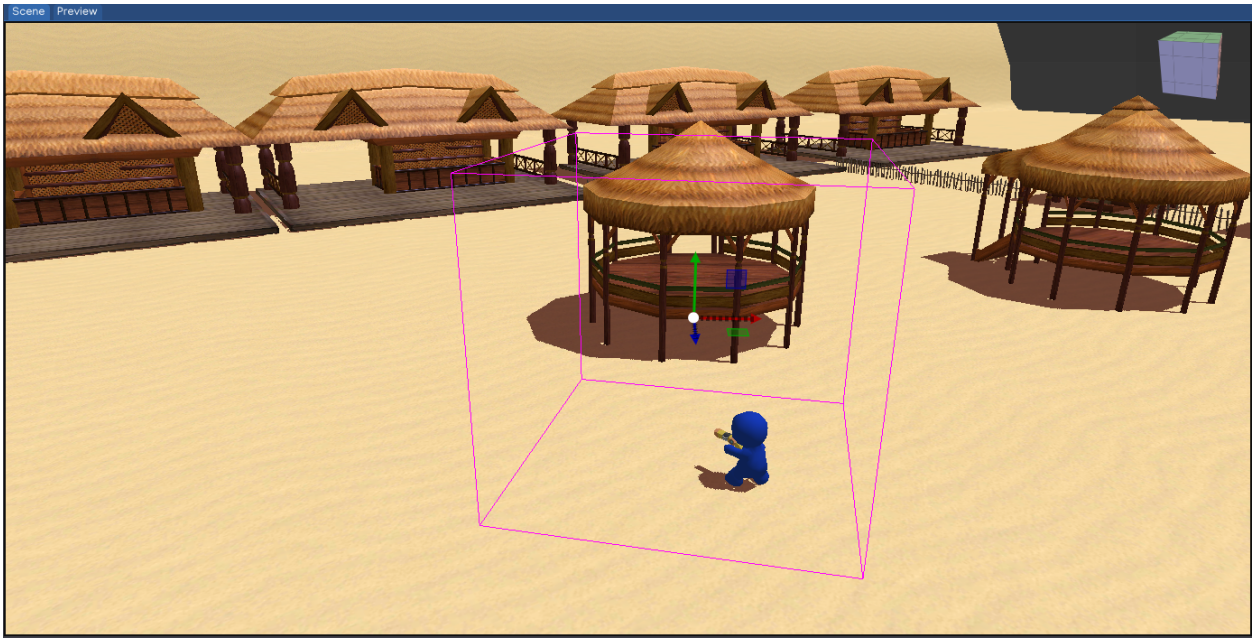
Menu Bar provides some functions to control the editor windows, as well as tools and other settings related to the scene.



1.2.2 Toolbar

Toolbar provides controls onto your scene. It allows you to play, pause, resume, stop the game preview. It also alows changing *Gizmo* and *Camera* modes.

1.2.3 Scene View



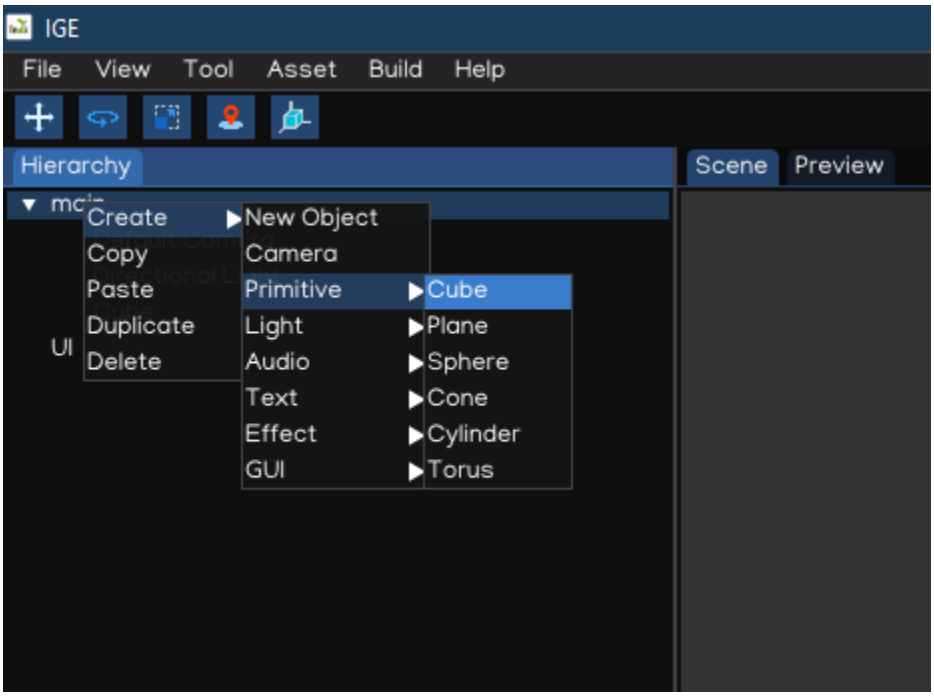
The Scene View is the main view of igeCreator editor. It will give you a real-time feedback of what is happening in your current scene while manipulate the objects and settings using the editor.

To asjust the editor camera, use controls below:

Action	Input
Rotate	[Mouse] Drag Right Button
Zoom	[Mouse] Scroll Middle Button
Move	[Mouse] Drag Middle Button
Focus	[Keyboard] Press *F* Key

To add game object to the scene, just drag and drop the asset files in the scene view, based on the file type the engine will create game object and attach relevant component(s) automatically.

The game object can also be added to the scene by seleting and right-clicking the parent object to show the `Create Menu` with various types of object to create.



Also, the object can be manipulated with actions below:

Action	Input
Select	[Mouse] Click Left Button
Multi Select	[Mouse] Drag Left Button
Copy	[Keyboard] Press *Ctrl* + *C* Key
Paste	[Keyboard] Press *Ctrl* + *V* Key
Duplicate	[Keyboard] Press *Ctrl* + *D* Key
Delete	[Keyboard] Press *Del* Key

1.2.4 Game Preview

The Preview, like the Scene View, reflects what is happening in your scene, from your game active camera. The editor will automatically focus the Preview when playing the scene.

Note: The GUI layer is hidden in editing mode, so that developer can focus on adjusting the 3D scene. In playing mode, the game will be played just like it will be on devices.

1.2.5 Hierarchy

The *Hierarchy* window shows the current scene hierarchy with relations between objects. Besides, you can also create/select/delete/move/copy/paste/drag objects in this view.

User can select object by clicking the item in the tree. Multiple selection can be done with with help of using `Ctrl` and `Shift` keys.

User can also drag and drop object to create parent-children relationship in the hierarchy tree. Assets drag and drop in hierarchy is also implemented.

To create prefab, just simply drag the item in hierarchy to `prefabs` folder in the Assets Browser.

Tip: To focus the camera on an object in complex scene, select it node in hierarchy and press `F` key.

1.2.6 Inspector

In the *Inspector* you'll be able to view and edit the currently selected object. Adding, tweaking and removing components, changing object settings (name, tag, transform...).

All the object has `Transform` component by default. The GUI element will have `RectTransform` which is a derivative of `Transform` component specilized for 2D and GUI.

Besides, there are various types of component which can be added into a game object, such as:

Component	Usage
Camera	Camera in game
Figure	Model (IGE Engine format)
Sprite	Sprite in game
Animator	Animation controller
Particle	Particle effect

continues on next page

Table 1 – continued from previous page

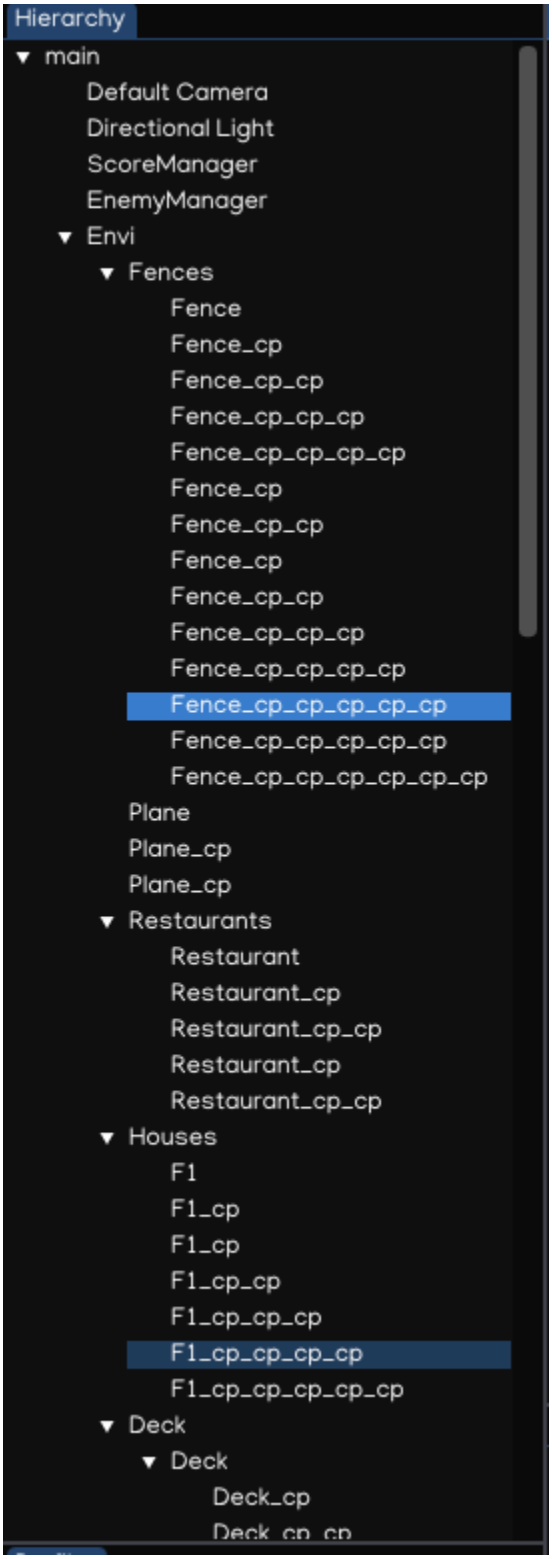
Component	Usage
Script	Scripting, to control object's behavior
Text	Text in game, using TTF or Bitmap
AmbientLight	Ambient Light
DirectionalLight	Directional Light
PointLight	Point Light
SpotLight	Spot Light
AudioSource	Audio source
AudioListener	Audio Listener
Canvas	Canvas for rendering GUI
UIImage	GUI Image
UIText	GUI Text
UITextField	GUI Text Field
UIButton	GUI Button
UISlider	GUI Slider
UIScrollView	GUI Scroll View
UIScrollBar	GUI Scroll Bar
UIMask	GUI Mask
PhysicBox	Physic Box collider
PhysicSphere	Physic Sphere collider
PhysicCapsule	Physic Capsule collider
PhysicMesh	Physic Mesh collider
PhysicSoftBody	Physic Soft-Body and cloth simulation
Navigable	Mark object/mesh as navigable
NavMesh	Navigation mesh
DynamicNavMesh	Dynamic navigation mesh
NavAgent	Navigation agent
NavObstacle	Navigation obstacle
NavArea	Mark the navigation area
OffMeshLink	Link between navigation areas

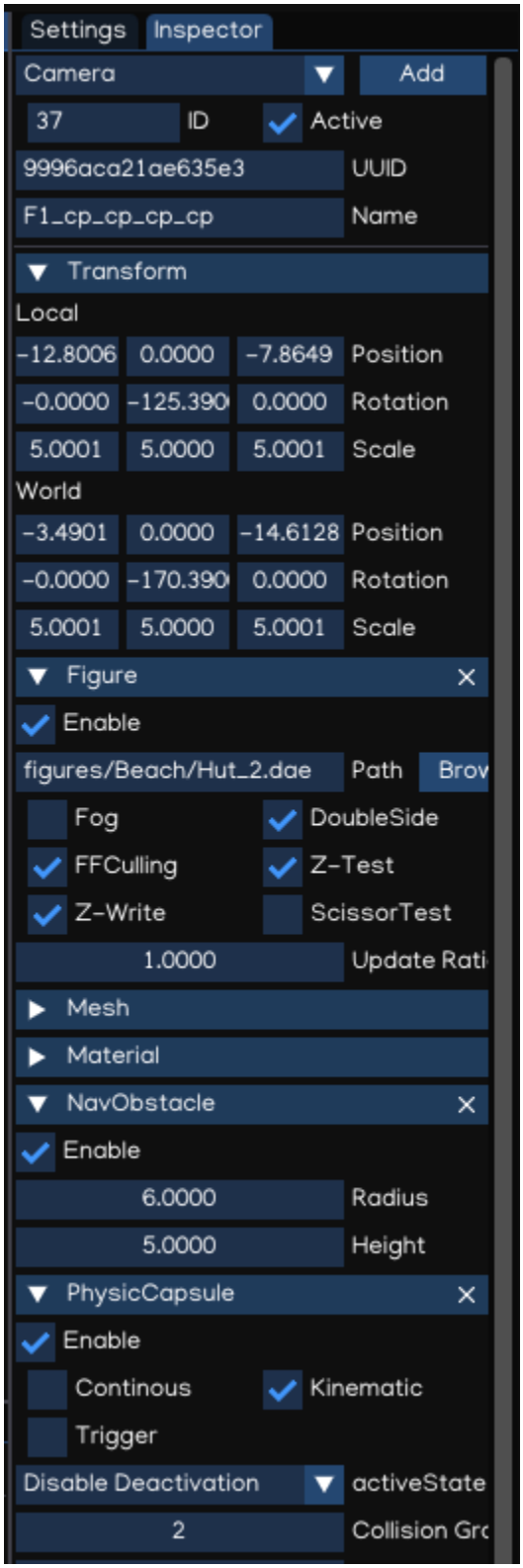
Note: Usage of each component will be discussed in `Tutorials` sections.

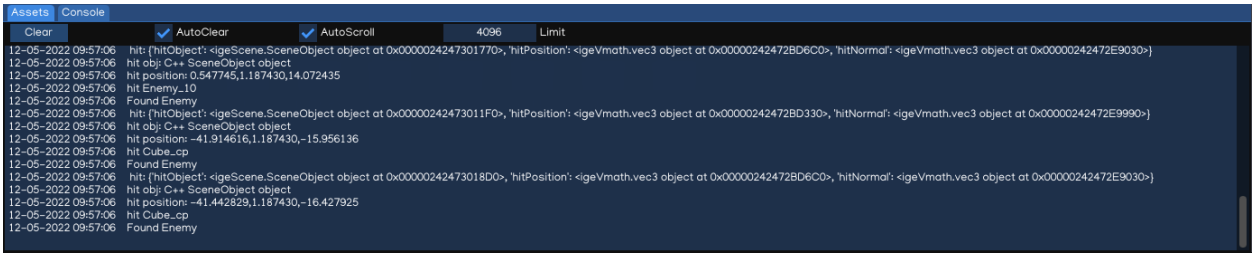
1.2.7 Console

Show log from the engine as well as the game so that it's easier for developer to debug.

Note: The console reflects the log from Python API, so to print the log user just need to use `print()` function from Python API.


Hierarchy
main
Default Camera
Directional Light
ScoreManager
EnemyManager
Envi
Fences
Fence
Fence_cp
Fence_cp_cp
Fence_cp_cp_cp
Fence_cp_cp_cp_cp
Fence_cp
Fence_cp_cp
Fence_cp
Fence_cp_cp
Fence_cp_cp_cp
Fence_cp_cp_cp_cp
Fence_cp_cp_cp_cp_cp
Fence_cp_cp_cp_cp_cp
Fence_cp_cp_cp_cp_cp_cp
Plane
Plane_cp
Plane_cp
Restaurants
Restaurant
Restaurant_cp
Restaurant_cp_cp
Restaurant_cp
Restaurant_cp_cp
Houses
F1
F1_cp
F1_cp
F1_cp_cp
F1_cp_cp_cp
F1_cp_cp_cp_cp
F1_cp_cp_cp_cp_cp
Deck
Deck
Deck_cp

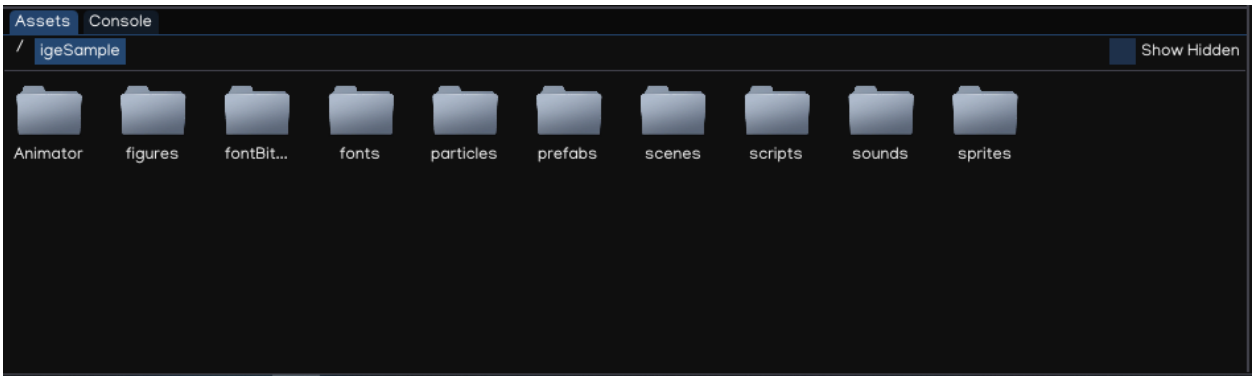

Settings
Inspector
Camera
Add
37
ID
Active
9996aca21ae635e3
UUID
F1_cp_cp_cp_cp
Name
Transform
Local
-12.8006
0.0000
-7.8649
Position
-0.0000
-125.390
0.0000
Rotation
5.0001
5.0000
5.0001
Scale
World
-3.4901
0.0000
-14.6128
Position
-0.0000
-170.390
0.0000
Rotation
5.0001
5.0000
5.0001
Scale
Figure
Enable
figures/Beach/Hut_2.dae
Path
Brov
Fog
DoubleSide
FFCulling
Z-Test
Z-Write
ScissorTest
1.0000
Update Rati
Mesh
Material
NavObstacle
Enable
6.0000
Radius
5.0000
Height
PhysicCapsule
Enable
Continous
Kinematic
Trigger
Disable Deactivation
activeState
2
Collision Gro



1.2.8 Asset Browser

Provides access to all assets of the project. User can create/move/delete files as well as using right-clicking context menu to perform various actions.

The Asset Browser allows you to drag and drop assets to places like *Scene View* to create object, or *Inspector* to configure object...



1.3 Your First Scene

1.3.1 Create Project

Go to the menu bar: `File -> New Project` to create new project. This action also create new empty scene for the newly created project.

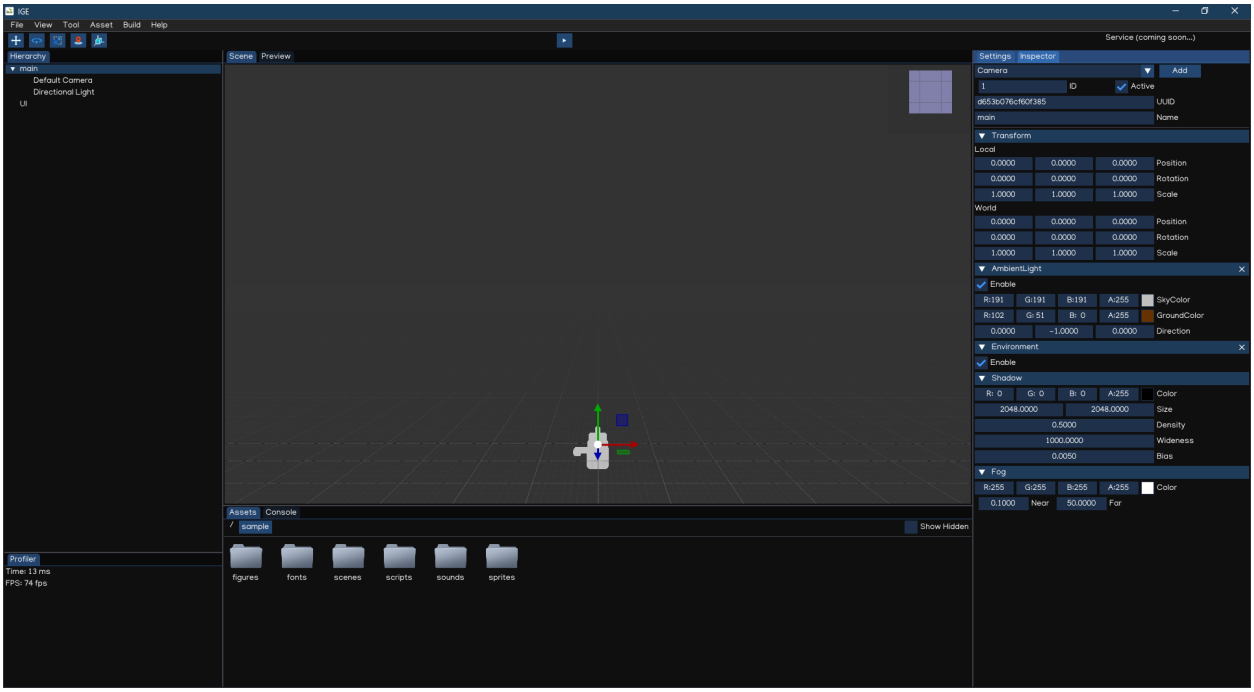
This scene is composed of two object: a directional light, and a camera.

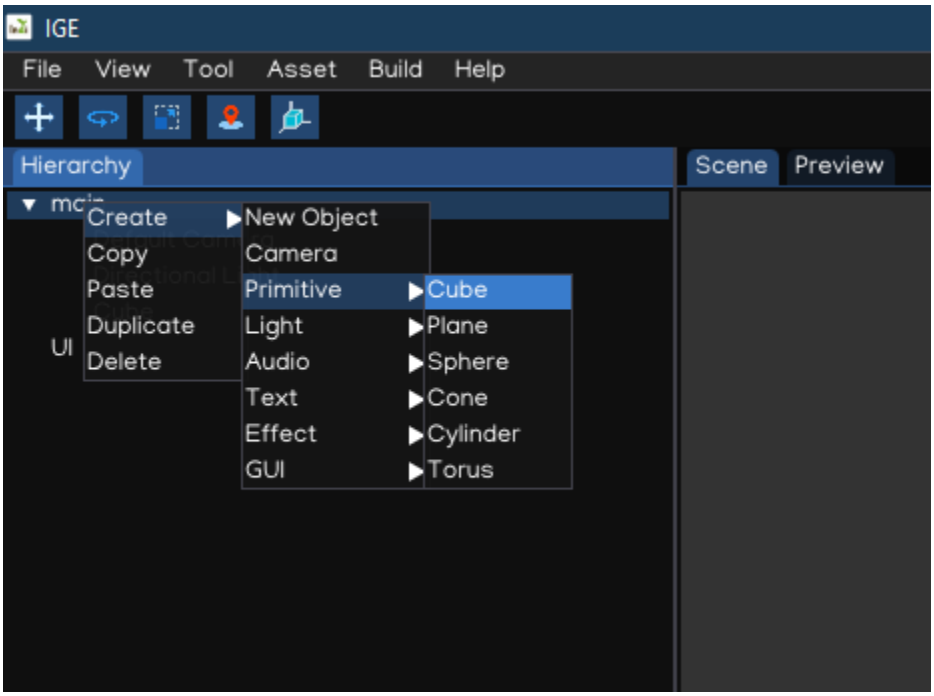
Having a camera in a scene is essential for the game to show something onto the screen.

You can go to the menu bar: `File -> Save Scene` to save the scene. Then you can click the *Play* button in the *Toolbar* to preview the scene.

A project can contain multiple scenes. To create a new scene, go to `File -> New Scene`. To load a scene, go to `File -> Load Scene` or just drag a file with *.scene* extension in the *Scene View*.

To change a scene at runtime, we need to use *Python API* which will be introduced later.


IGE
File View Tool Asset Build Help
Hierarchy
main
Default Camera
Directional Light
UI
Scene Preview
Service (coming soon...)
Settings Inspector
Camera
Add
ID
Active
UUID
Name
Transform
Local
Position
Rotation
Scale
World
AmbientLight
Enable
SkyColor
GroundColor
Direction
Environment
Shadow
Color
Size
Density
Wideness
Bias
Fog
Near
Far
Assets Console
sample
Show Hidden
figures
fonts
scenes
scripts
sounds
sprites
Profiler
Time: 13 ms
FPS: 74 fps

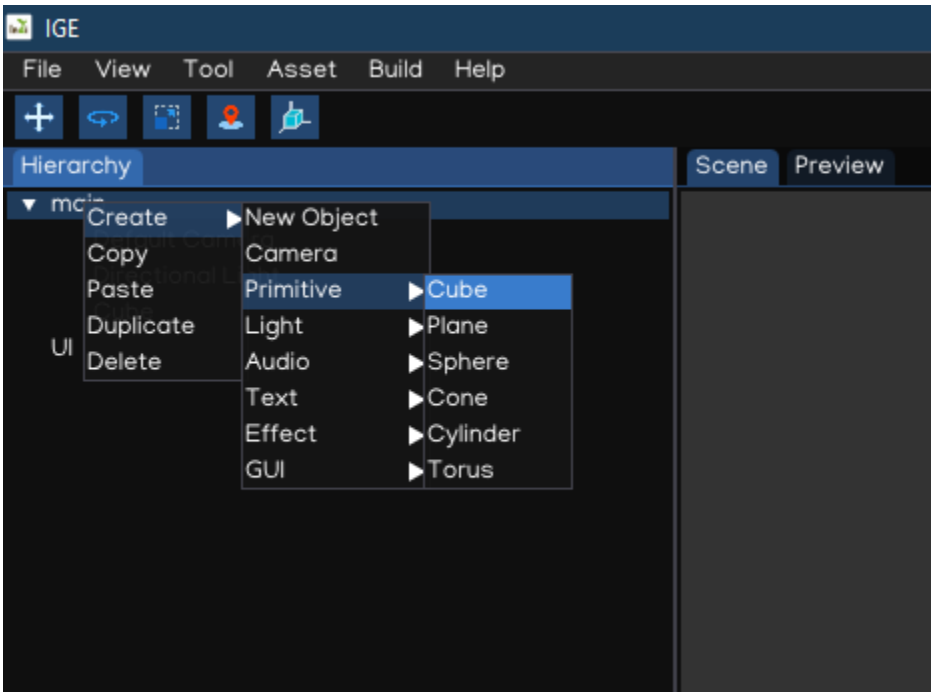

IGE
File View Tool Asset Build Help
Hierarchy
Scene Preview
Create
Copy
Paste
Duplicate
Delete
New Object
Camera
Primitive
Light
Audio
Text
Effect
GUI
Cube
Plane
Sphere
Cone
Cylinder
Torus
UI

1.3.2 Project Structure

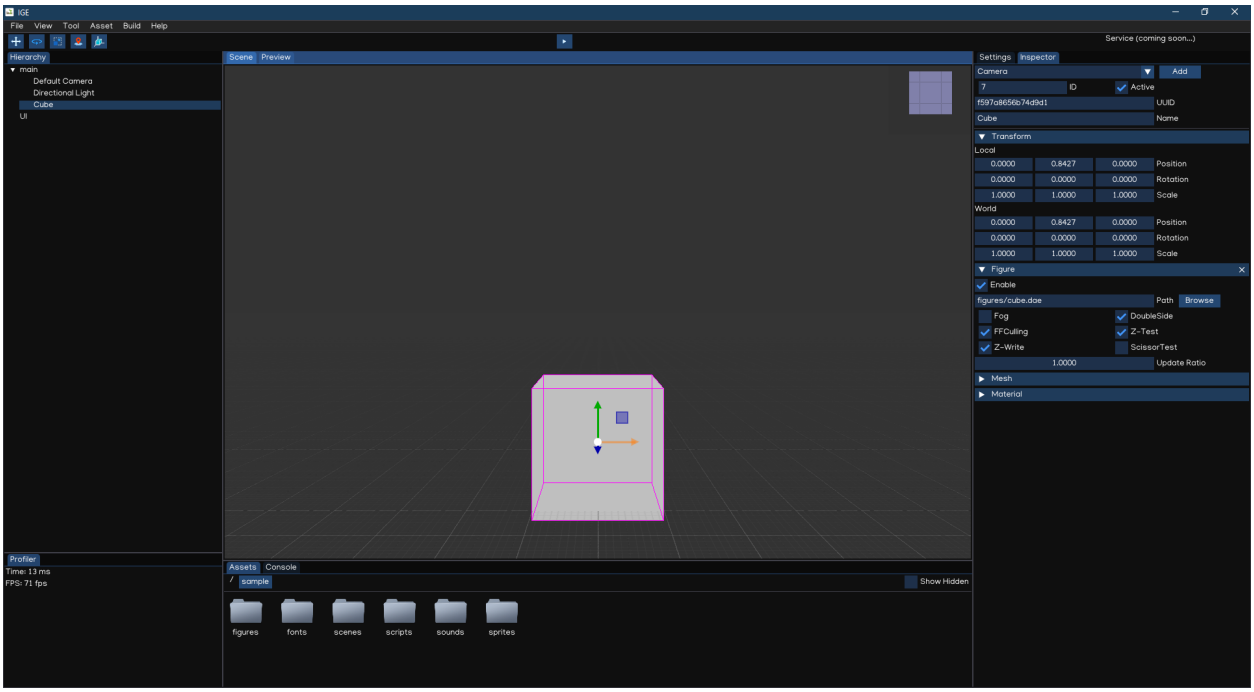
Item	Meaning
config	[Folder] Contains project's configuration.
figures	[Folder] Contains models and animations.
fonts	[Folder] Contains fonts used in the project.
scenes	[Folder] Contains scene files.
scripts	[Folder] Contains game logic source codes.
sounds	[Folder] Contains audio files.
sprites	[Folder] Contains UI and 2D images.
*.igeproj	[File] The project file

1.3.3 Create Object

In order to add an object to the scene, select and right-click an item in Hierarchy, select `Create -> Primitive -> Cube`.



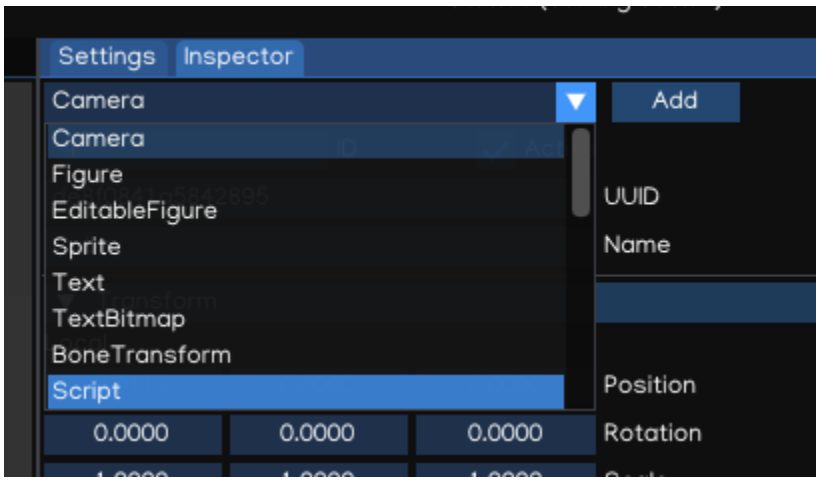
You should now see a cube in your scene.



1.3.4 Scripting

To control behavior of an object, we use Script Component.

In the Inspector, add new Script Component.

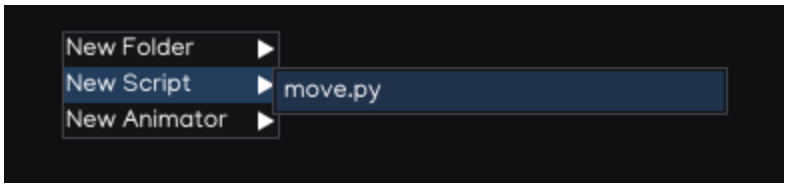


In the Asset Browser, go to `scripts`, right-click then select `New Script`, then name it `move.py`.

Open the newly created file, edit it with content below:

```
import math
import igeVmath as vmath
from igeScene import Script

class Move(Script):
    def __init__(self, owner):
```

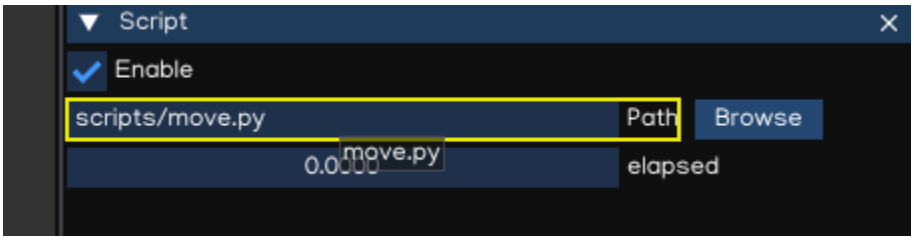
(continues on next page)



(continued from previous page)

```
        super().__init__(owner)
        self.elapsed = 0.0

    def onUpdate(self, dt):
        self.elapsed = self.elapsed + dt
        self.owner.transform.position = vmath.vec3(0, math.sin(self.elapsed), 0)
```

Then drag the file in Script component Inspector.



Save the scene, by pressing `Ctrl + S` or `File -> Save Scene`. Then you can press the `Play` button to test it, the cube should keep moving up and down follow `sin` pattern continuosly.

1.4 Input

Input allows the user to interact with the game using input devices.

IGE supports many types of inputs, including:

- Touch Screen
- Mouse
- Keyboard
- *(WIP)* Motion Sensors: Accelerometor, Gyroscope
- *(WIP)* Joystick
- *(WIP)* Controller

1.4.1 Using Touch Screen

The Input module is a Python module which provides functions to work with input devices.

To simplify the implementation, the Touch Screen and Mouse inputs are implemented in `igeCore.input.touch` module. We support multiple touch by default.

Mouse events are map to touch, with special finger Id for left, right and middle buttons.

Below is an example of how to use Touch to control UI behavior:

```
from igeScene import Script
from igeCore.input.touch import Touch

class TouchTest(Script):
   def __init__(self, owner):
      super().__init__(owner)

   def onUpdate(self, dt):
      for i in range(0, Touch.count()):
         x,y = Touch.getPosition(i)
         if Touch.isPressed(i):
            print(f"Pressed {Touch.getId(i)} at ({x}, {y})")
```

1.4.2 Using Keyboard

To get access to Keyboard, use the `igeCore.input.keyboard` API.

Below is an example of how to use keyboard:

```
from igeScene import Script
from igeCore.input.keyboard import KeyCode, Keyboard

class KeyboardTest(Script):
   def __init__(self, owner):
      super().__init__(owner)

   def onUpdate(self, dt):
      if Keyboard.isPressed(KeyCode.KEY_SPACE):
         print("SPACE pressed - FIRE")
```

1.4.3 Using Virtual Keyboard

Use the API below to show/hide virtual keyboard.

```
from igeScene import Script
import igeCore
from igeCore.input.keyboard import KeyCode, Keyboard

class VirtualKeyboardTest(Script):
   def __init__(self, owner):
      super().__init__(owner)
```

(continues on next page)

(continued from previous page)

```
def onUpdate(self, dt):
    if not igeCore.isVirtualKeyboardShown(): # check if VK is show
        igeCore.showVirtualKeyboard("Input default text here...") # request show VK

    if Keyboard.isPressed(KeyCode.KEY_RETURN):
        text = igeCore.getInputText() # get the text
        igeCore.hideVirtualKeyboard() # hide the keyboard
```

1.5 Graphics

IGE graphics features help to create beautiful, optimized graphics across a range of platforms, from mobile to desktop through an easy to use workflow.

1.5.1 Assets workflow

Graphics assets including model, animation, texture and shader can be loaded, converted and displayed using IGE.

Animation and model files such as Collada DAE and FBX are imported to IGE then converted to IGE optimized format in which:

- `*.pyxf`: Use for model
- `*.pyxa`: Use for animation

Texture files are imported and converted to:

- `*.pyxi`: Use for texture

1.5.2 Render Pipeline

The builtin render pipeline is implemented using forward rendering technique, which utilize OpenGL 3.x / OpenGLES 3.x API.

Forward rendering renders each object in one or more passes:

- OpaquePass
- TransparentPass
- ShadowPass

1.5.3 Camera

A game represents game objects in a 3D space. The device's screen is 2D space, thus using camera help to capture the scene to display it in the device screen.

Camera can be created by adding a Camera component to a game object, or using `Create Menu -> Camera`.

Using perspective camera, objects which are far away are smaller than those nearby which is similar to the real life. Orthographic camera is useful to display the scene where all objects appear at the same scale, like GUI or isometric view.

Camera inspector reference:

Property	Function
FOV	Field of view
Near	Near clipping
Far	Far clipping
Aspect	Aspect ratio
Up	Up vector: 0 = X, 1 = Y, 2 = Z
Ortho	Orthographic or perspective camera
OrtW	Ortho width
OrtH	Ortho height
LockTarget	Lock target, create follow camera
Target	Position of target to follow
wBase	Whether width based or heigh based scaled
ScrScale	Screen scale factor
ScrOffset	Screen offset factor
ScrRot	Screen rotation factor
ClearColor	Color set to when clear screen

Camera can be controlled by using Python API, with module `igeScene.Camera`. Check the Camera API Document for more info.

Multiple camera also supported, but only one active camera can be used at a time (in combination with builtin GUI Camera). To set current camera as active, use Python API as example below:

```
from igeScene import Script

class GameManager(Script):
    def __init__(self, owner):
        super().__init__(owner)
```

(continues on next page)

(continued from previous page)

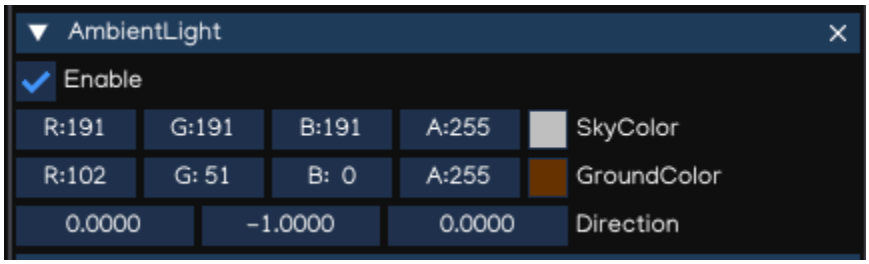
```
    def onUpdate(self, dt):
        # find a camera and set it active
        camera = self.owner.scene.findObjectByName("MyCamera")
        if camera is not None:
            self.owner.scene.activeCamera = camera
```

1.5.4 Lighting

Ambient Light

Ambient light is diffuse environmental light that is present all around the Scene and doesn't come from any specific source object. It can be an important contributor to the overall look and brightness of a scene.

Ambient light can be useful in a number of cases, depending upon your chosen art style. An example would be bright, cartoon-style rendering where dark shadows may be undesirable or where lighting is perhaps hand-painted into textures. It can also be useful if you need to increase the overall brightness of a scene without adjusting individual lights.

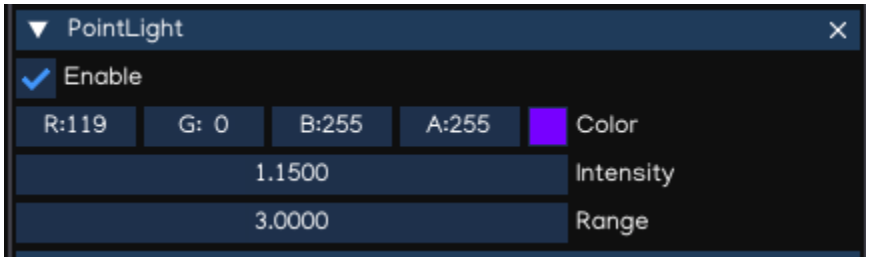


Property	Function
SkyColor	Ambient sky color
GroundColor	Ambient ground color
Direction	Ambient direction vector

Tip: AmbientLight component is usually attached to the root node of the object hierarchy tree, because one scene needs only one Ambient light settings.

Point Light

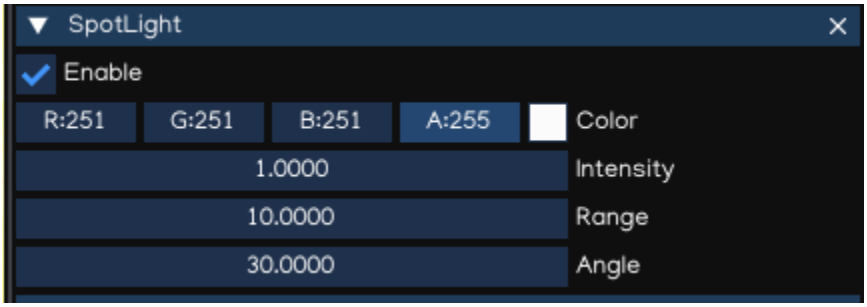
A Point Light is located at a point in space and sends light out in all directions equally. The direction of light hitting a surface is the line from the point of contact back to the center of the light object.



Property	Function
Color	Light color
Intensity	Light intensity value
Range	Range of effectiveness

Spot Light

Like a Point Light, a Spot Light has a specified location and range over which the light falls off. However, a Spot Light is constrained to an angle, resulting in a cone-shaped region of illumination.

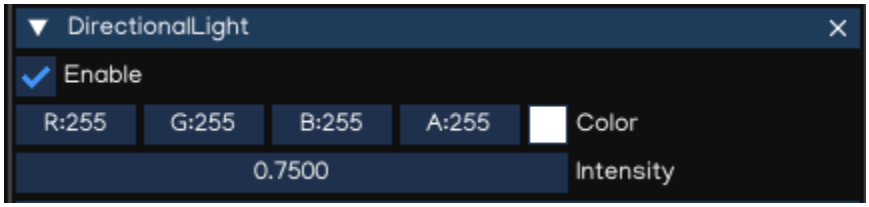


Property	Function
Color	Light color
Intensity	Light intensity value
Range	Range of effectiveness
Angle	Constrained angle

Directional Light

Directional Lights are useful for creating effects such as sunlight in your scenes. Behaving in many ways like the sun, directional lights can be thought of as distant light sources which exist infinitely far away. A Directional Light doesn't have any identifiable source position and so the light object can be placed anywhere in the scene. All objects in the scene are illuminated as if the light is always from the same direction.

By default, every new scene contains a Directional Light represents the sunlight/moonlight.



Property	Function
Color	Light color
Intensity	Light intensity value

Note: The direction of light is controlled by the rotation property of the object it attached to.

1.5.5 Shadows

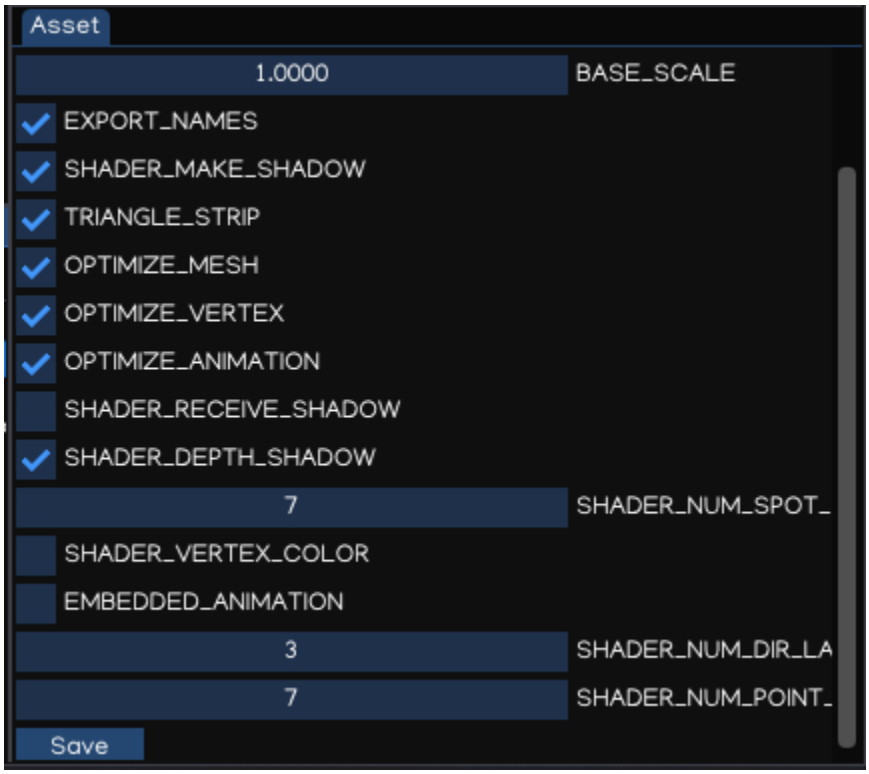
IGE uses a technique called shadow mapping to render real-time shadows.

Shadow mapping uses textures called shadow maps. Shadow map texture resolution is set to 2048x2048 by default, and can be as largest as 4096x4096. Using larger texture result in higher quality, but it costs more VRAM and may decrease game performance.

To display shadow, ensure to have:

- Shadow caster objects has enabled casting ability.
- Shadow receiver has been enabled receiving ability.
- Directional Light is ebabled and the light direction can cast shadow from shadow casters to shadow receiver.
- Shadow parameters setup correctly.

When importing models, the ability to cast/receive shadow is disabled by default, to preserve best performance. To enable these abilities, go to `Assets Browser`, select the file to modify, in `Assets` windows, enable it's flags accordingly then save it.



The shadow parameters can be adjusted with `Environment` component, attached to the root node of the hierarchy.

Property	Function
Color	Shadow color
Size	Shadow map texture size
Density	Shadow density
Wideness	Shadow wideness
Bias	Shadow Bias value

Note: With current implementation, only the first DirectionalLight can cast shadow because shadow transformation depends on the light direction.

Tip: Wideness and size are related, so wideness shoule be smaller as possible so it can improve shadow quality, or can use smaller size to improve performance.

1.5.6 Fogs

IGE provide basic fog setting to simulate fog.

Property	Function
Color	Fog color
Near	Fog near distance
Far	Fog far distance

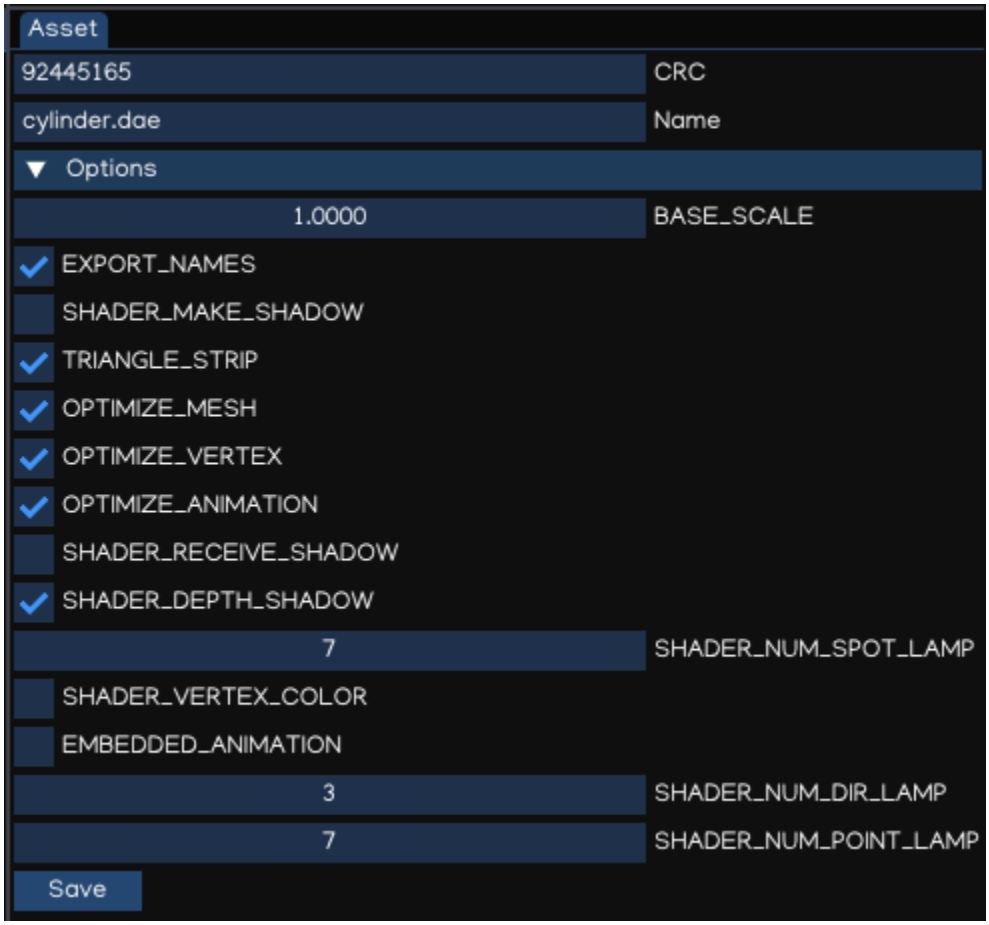
1.5.7 Model

Models are files that contain data about the shape and appearance of 3D objects, such as characters, terrain, or environment objects. Model files can contain a variety of data, including meshes, materials, and textures. They can also contain animation data, for animated objects. Usually, models are created using an 3D modeling software, such as Blender®, Autodesk® Maya®, Autodesk® 3ds Max® ..., and then import them into IGE.

IGE supports importing `.dae` and `.fbx` file formats. After importing to IGE, the files are converted to `.pyxf` format which is specially optimized for IGE. The game engine will automatically detect changes in the file system, and import model files accordingly.

Importing

In order to change importing options, go to `Assets Browser`, select the file to change settings, then look for `Assets` windows, then change the options when needed.

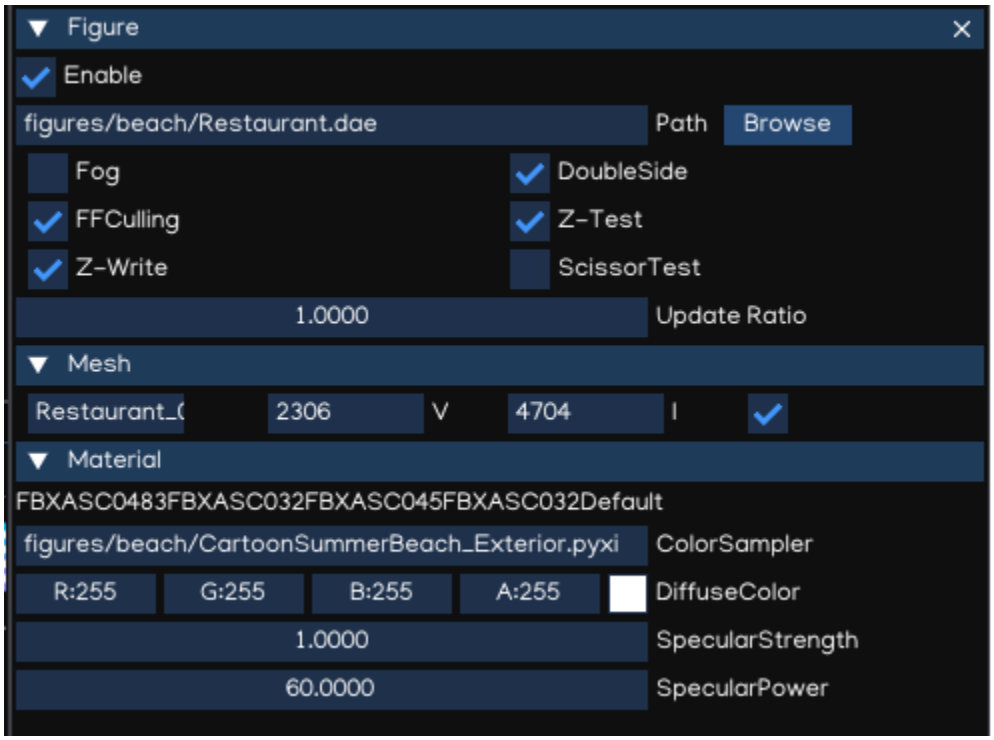


Property	Function
EXPORT_NAMES	Include meshes name in exported version
BASE_SCALE	Base scale factor (dae: 1.0, fbx: 100.0)
TRIANGLE_STRIP	[Optimize] Strip redundant trianges
OPTIMIZE_MESH	[Optimize] Optimize mesh
OPTIMIZE_VERTEX	[Optimize] Optimize vertex
OPTIMIZE_ANIMATION	[Optimize] Optimize animation
SHADER_MAKE_SHADOW	Enable shadow casting
SHADER_RECEIVE_SHADOW	Enable shadow receiving
SHADER_VERTEX_COLOR	Enable vertex color
SHADER_NUM_DIR_LAMP	Number of directional light
SHADER_NUM_POINT_LAMP	Number of point light
SHADER_NUM_SPOT_LAMP	Number of spot light
EMBEDDED_ANIMATION	Embbed animation, or build saparate anim file

Using Model

Model can be dragged to the `Scene View` to create scene object. Also, it can be attached to `Figure` or `EditableFigure` components of an empty object.

`Figure` component is used to render 'fixed' model, wothout ability of modifying mesh structures. It is the fasted way to render model using IGE. `EditableFigure` is used in case model's mesh need to be changed at run time.



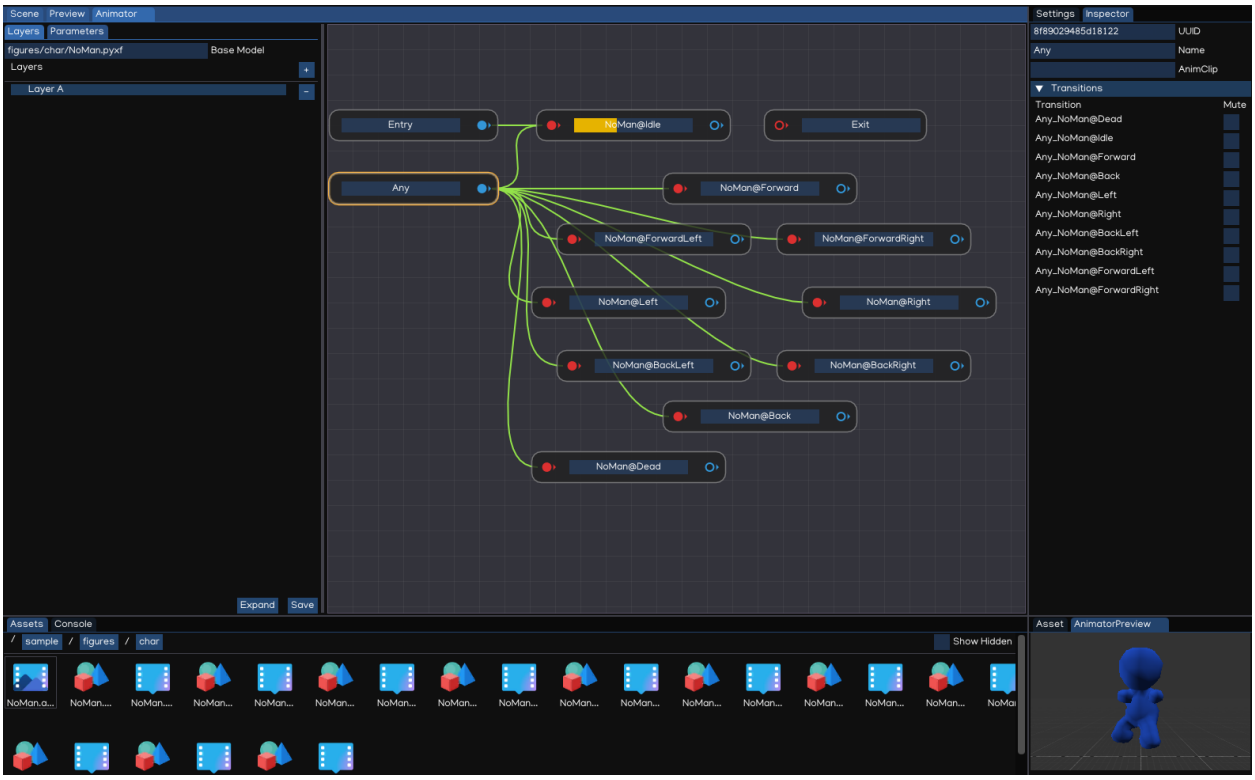
Property	Function
Path	Path to the model file
Fog	Enable/disable fog
DoubleSide	Enable/disable double side rendering
FFCulling	Enable/disable front-face culling
Z-Test	Enable/disable depth testing
Z-Write	Enable/disable depth writing
ScissorTest	Enable/disable scissor test
Update Ratio	Updating ratio, used to control animation speed
Mesh	List of meshes included in the model file
Material	List of materials included in the model file

For more details of scripting API, please refer to Python API Document.

1.6 Animation

IGE animation system provides:

- Easy workflow and setup of animations.
- Preview of animation clips, transitions and interactions between them.
- Management of complex interactions between animations with a visual programming tool.
- Layering and masking features.



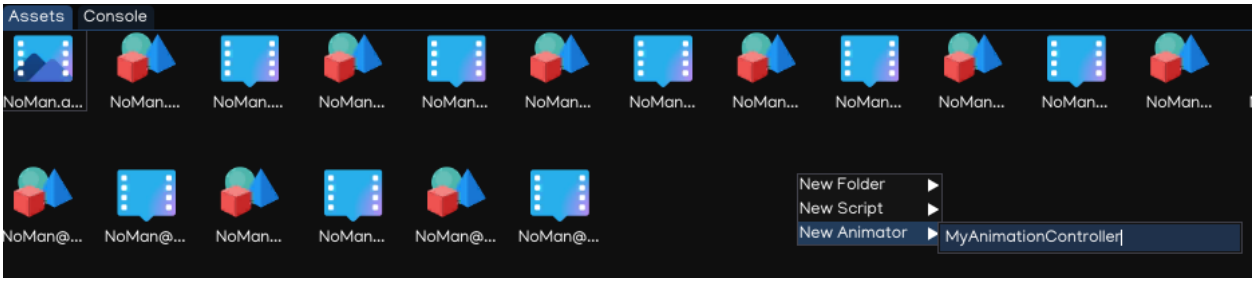
1.6.1 Animation Clips

Animation Clips are one of the core elements to IGE animation system, which are imported from external sources such as animation from Blender®, Autodesk® Maya®, Autodesk® 3ds Max® ... softwares. In `Assets Browser`, animation clip files have `.pyxa` extension.

1.6.2 Animator Controllers

An **Animator Controller** allows you to arrange and maintain a set of animations for a character or other animated scene objects. The controller has references to the animation clips used within it, and manages the various animation states and the transitions between them using a `Animation State Machine`.

To create an Animator Controller, right-click on the `Assets Browser`, select `New Animator`, like below:



Double-clicking the new created file will open `Animator Window` which can be used to create, view and modify the animator controller.

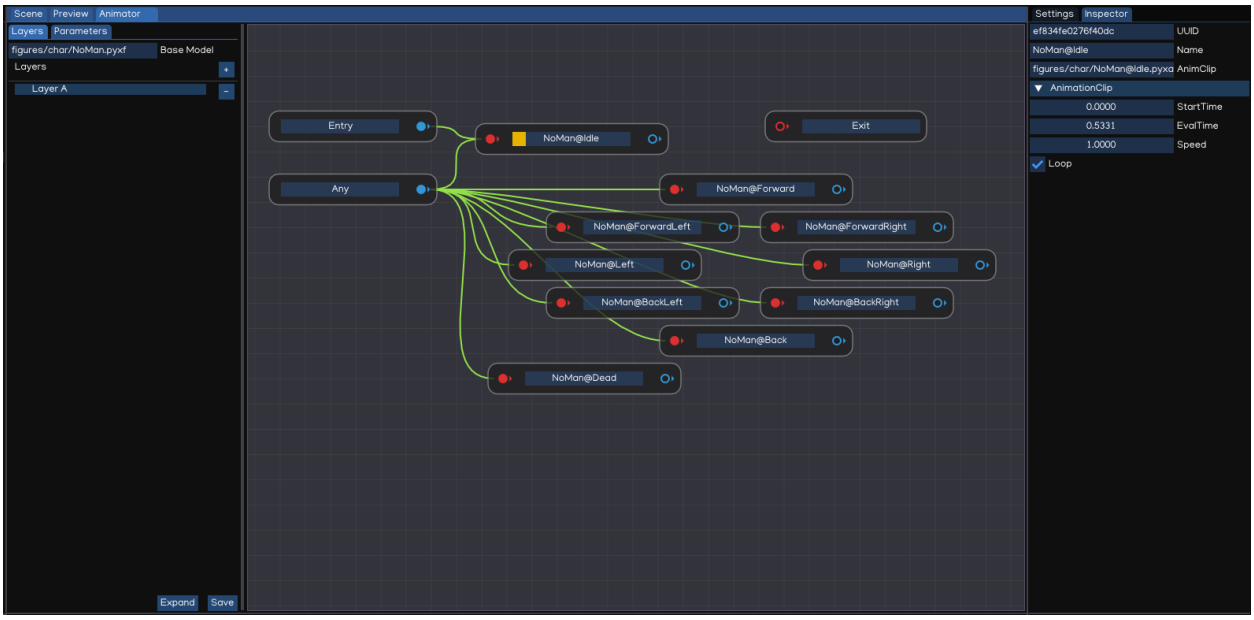
The animator controller is then finally applied to an object by attaching an `Animator` component that references them. See the Python API Document for further details about their usage.

1.6.3 The Animator Window

The **Animator Window** allows you to create, view and modify `Animator Controller` assets.

The window contains:

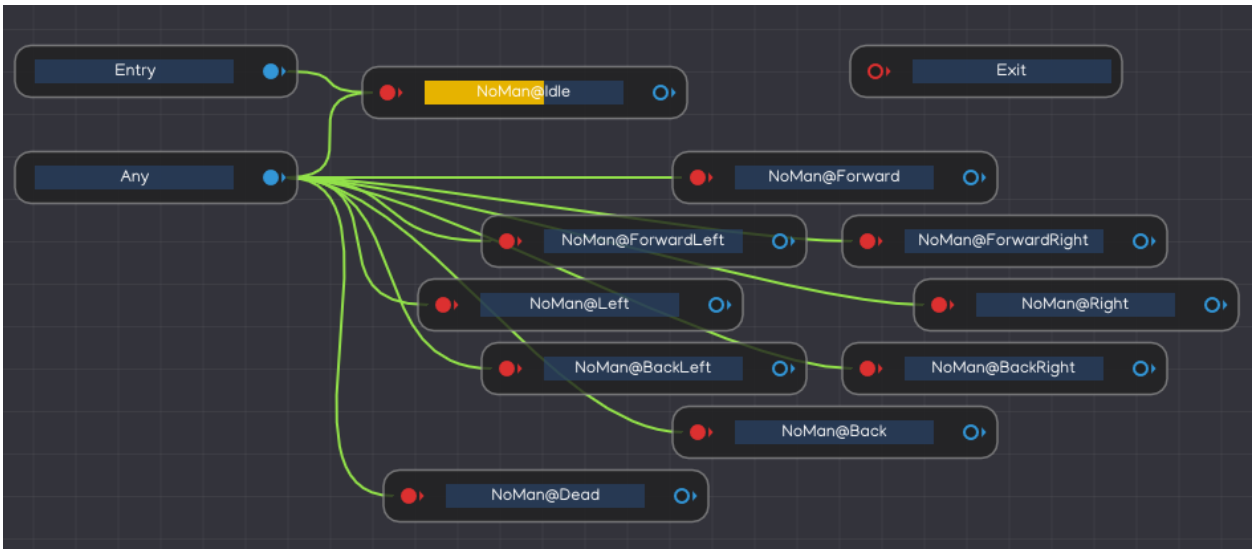
- *Layout Area*: use to create, arrange and connect states in your Animator Controller.
- *Layers Area*: use to view and edit layers within Animator Controller. IGE allows to have multiple layers within a single animator controller, to control different parts of the object using separate state machine.



- *Parameters Area*: allow to create, view and edit the parameters using in Animator Controller. Those parameters are variables which act as input for the state machine, to control the transitioning condition between states.
- *Inspector*: to edit state, or transition settings.

1.6.4 Animation State Machines

Animation State Machines represent an overview of all of the animation clips related to a particular animation object, and allow various events in the game to trigger different animations.

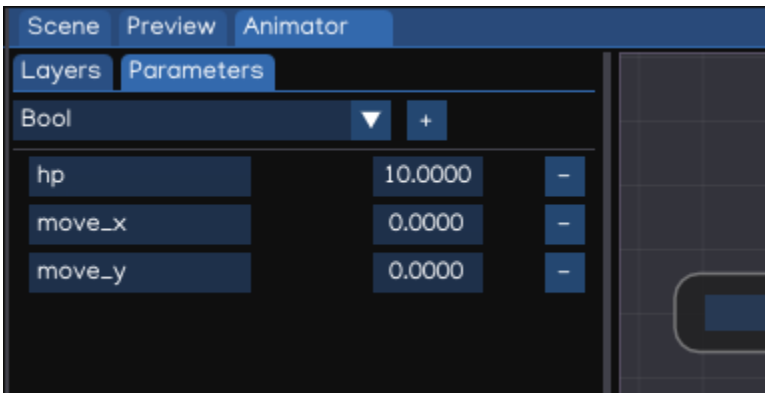


State Machines consist of States, Transitions and Events which together provide control overall animations behavior of a single object using `Animator Controller`.

1.6.5 Animation Parameters

Animation Parameters are variables that are defined within an `Animator Controller` that can be accessed and assigned values from scripts. This allow developer to control the behavior of animation system using IGE.

Parameter values can be set up using the **Parameters Area** of the **Animator Window**.



The parameters can be of four basic types:

- *Integer*: a integer number
- *Float*: a float number
- *Bool*: a true / false value
- *Trigger*: a true/false value that is reset by the controller when consumed by a transition

Parameters can be assigned values from a script using functions in the Animator class, using Python API below:

```
from igeScene import Script, Animator
from igeCore.input.touch import Touch
from igeCore.input.keyboard import KeyCode, Keyboard

class SimpleCharacter(Script):
   def __init__(self, owner):
      super().__init__(owner)

   def onStart(self):
      self.animator = self.owner.getComponent("Animator")
      self.animator.resetTrigger("fire")

   def onUpdate(self, dt):
      x,y = Touch.getPosition(0)
      fire = Keyboard.isPressed(KeyCode.KEY_SPACE)
      self.animator.setFloat("move_x", x)
      self.animator.setFloat("moveZ_y", y)
      self.animator.setTrigger("fire", fire)
```

More details about `Animator` API, please check Python API Document.

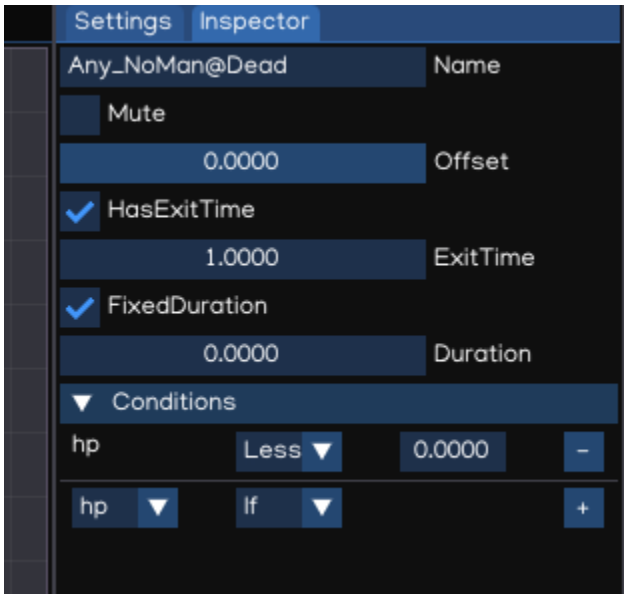
1.6.6 Animation transitions

Animation transitions allow the state machine
to switch or blend from one animation state to another. Transitions define not only how long the blend between states should take, but also under what conditions they should activate.

Each view in the animator window has:

- *Entry*: The entry node will be evaluated first to select which state the state machine begins with, by evaluating the state of your parameters when the state machine begins.
- *Exit*: used to indicate that a state machine should exit.
- *Any*: specify a situation where you want to go to a specific state regardless of which state you are currently in.
- *Other states*: animation states in the Animator Controller.

You can set up a transition to occur only when certain conditions are true. To set up these conditions, specify values of parameters in the Animator Controller, then setting up the transition condition in Inspector view.



Property	Function
Mute	Whether this transition is considered
Offset	The offset to begin in the destination state
HasExitTime	Make transition at the specific time specified in ExitTime
ExitTime	Represents the exact time at which the transition can take effect
FixedDuration	If checked, the transition time is interpreted in seconds.
Duration	Transition duration (normalized time or seconds, depends on FixedDuration flag).
Conditions	Transition conditions

Transition Conditions

A transition can have a single condition, multiple conditions, or no conditions at all. A condition consists of:

- An event parameter, the value of which is considered in the condition.
- A conditional predicate, if needed (for example, less or greater for floats).
- A parameter value, if needed.

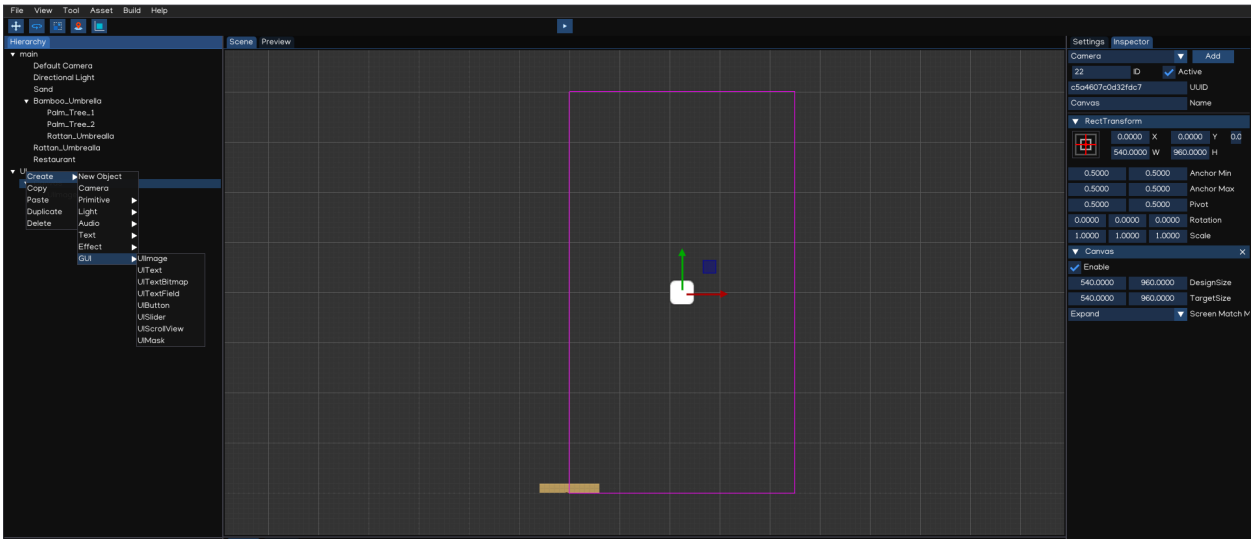
If HasExitTime is enabled for the transition and has one or more conditions, these conditions are only checked after the exit time of the state. This allows you to ensure that your transition only occurs during a certain portion of the animation.

1.7 Graphical User Interface

IGE includes is a set of tools for developing user interfaces for games and applications.

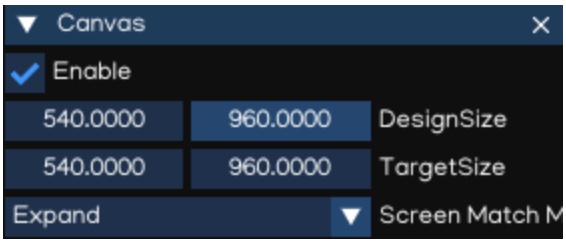
1.7.1 Canvas

The Canvas is a game object with a `Canvas` component on it. All UI elements must be children of a Canvas. Creating a new UI element, such as an `UIImage` using the menu `Create > GUI > UIImage`, automatically creates a Canvas, if there isn't already a Canvas in the scene.



Tip: To work with GUI, switch the Scene Camera to 2D mode. The Canvas will be displayed as a rectangle in the view, it help to easier posioning the UI elements on the scene.

The Canvas component can be setting up using Inspector.



Property	Function
DesignSize	Canvas design screen size
TargetSize	Target screen size (Editor only)
ScreenMatchMode	• *MatchWidthOrHeight*: match with width/height following a ratio • *Extend*: match the maximal screen scale ratios • *Shrink*: math the minimal screen scale ratios

1.7.2 RectTransform

The `RectTransform` is a new transform component that is used for all UI elements. It has position, rotation, and scale just like regular Transforms, but it also has a width and height, used to specify the dimensions of the rectangle.

Property	Function
X, Y, Z	Position X, Y, Z
W, H	Width and Height
AnchorMin	Lower left anchor handle
AnchorMax	Upper right anchor handle
Pivot	Pivot position
Rotation	Rotation value
Scale	Scale value

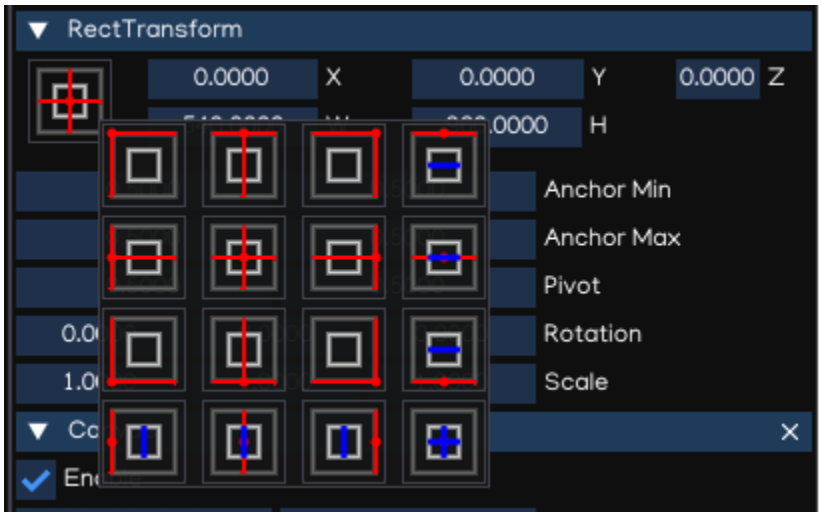
Tip: Use Z position to adjust the drawing order of elements, and may also help to resolve Z-fighting issues.

Pivot

Rotations, size, and scale modifications occur around the pivot so the position of the pivot affects the outcome of a rotation, resizing, or scaling.

Anchors

A child `RectTransform` can be anchored to the parent `RectTransform` in various ways:



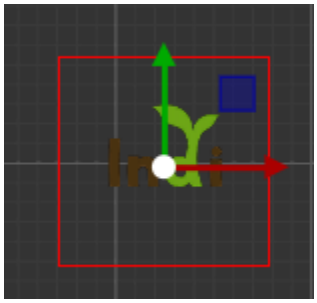
Tip: The blue arrow indicates that the child will stretch together with parent size, in horizontal, vertical or both accordingly.

1.7.3 UI Components

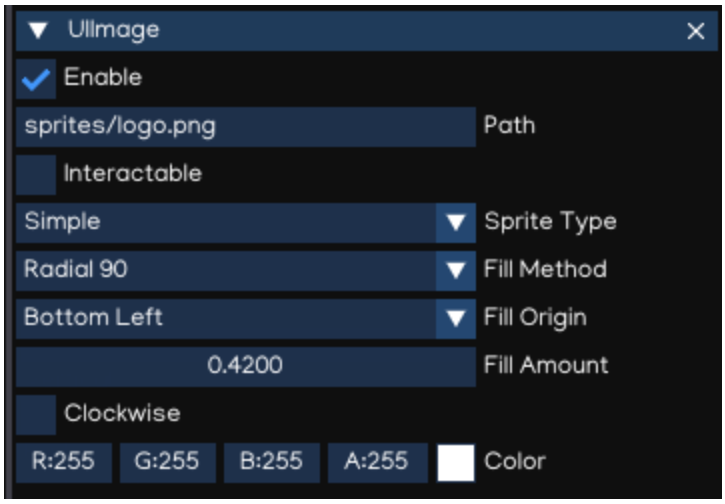
With the introduction of the UI system, new Components have been added that will help you create GUI specific functionality.

UIImage

The `UIImage` component is used to display an image on screen.



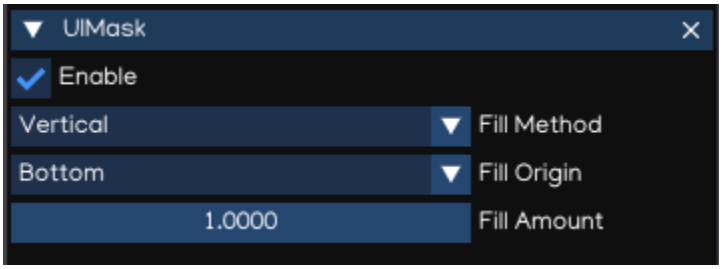
The `Inspector` window allows to change the image settings:



Property	Function
Path	The path to the image file
Inteactable	Ability to receive events using Script
Sprite Type	The Sprite type, can be: • *Simple*: simple sprite • *Sliced*: 9-slices sprite
Fill Method	Allow to fill just part of an image by: • *Horizontal* • *Vertical* • *Radial 90* • *Radial 180* • *Radial 360*
Fill Origin	Fill origin, can be: • *Left* • *Right* • *Bottom Left* • *Bottom Right* • *Top Left* • *Top Right*
Fill Amount	Amount of filling, from 0.0 to 1.0.
Clockwise	Fill direction, clockwise or counter-clockwise
Color	Diffuse color

UIMask

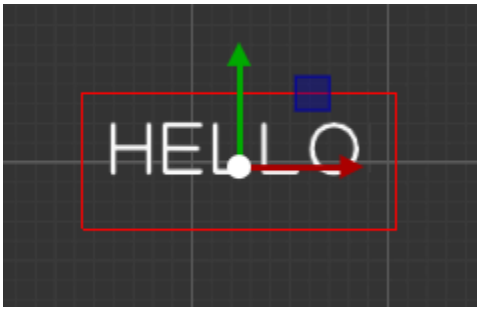
An `UIMask` is not a visible UI control but rather a way to modify the appearance of a control's child elements. The mask restricts the child elements to the shape of the parent. So, if the child is larger than the parent then only the part of the child that fits within the parent will be visible.



Property	Function
Enable	Enable/disable mask
Fill Method	Allow to fill just part of an image by: • *Horizontal* • *Vertical* • *Radial 90* • *Radial 180* • *Radial 360*
Fill Origin	Fill origin, can be: • *Left* • *Right* • *Bottom Left* • *Bottom Right* • *Top Left* • *Top Right*
Fill Amount	Amount of filling, from 0.0 to 1.0.
Clockwise	Fill direction, clockwise or counter-clockwise

UIText

The `UIText` component has a Text area for entering the text that will be displayed.



It is possible to set the font, font style and font size, and alignment of the text using Inspector.

Property	Function
RectAutoScale	Auto resize the Rect Transform with text size
Text	The text to display
Font	The font to display (.ttf, .otf, .pybm)
Size	The font size
Color	Text color
AlignHorizontal	Horizontal alignment
AlignVertical	Vertical alignment

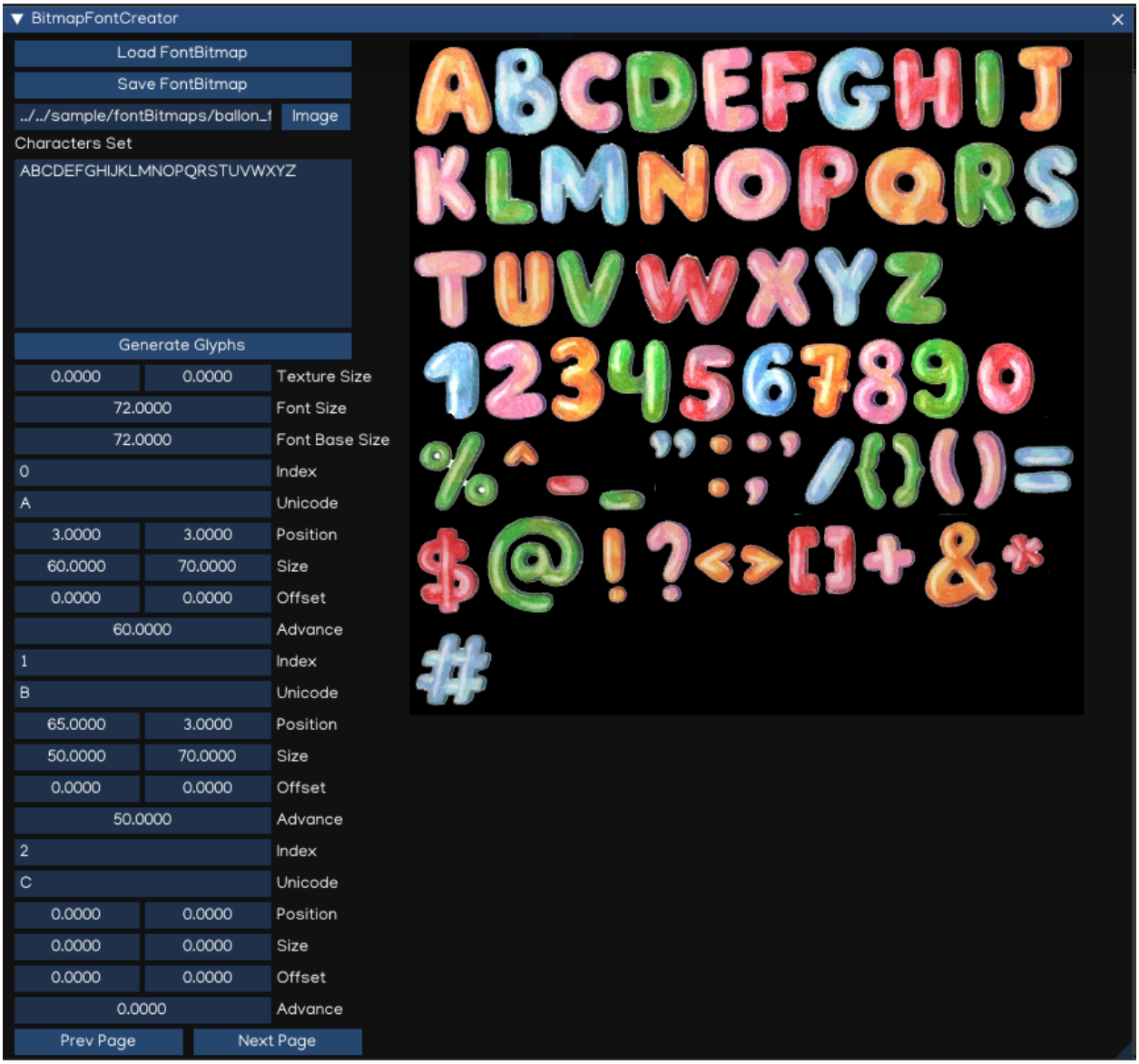
The `UIText` support drawing text using true-type font (.ttf, .otf) and bitmap font (.pybm) formats.

`Bitmap Font Creator` can be used to create bitmap font, which can be found at `Menu -> Tool -> Bitmap Font Creator`.

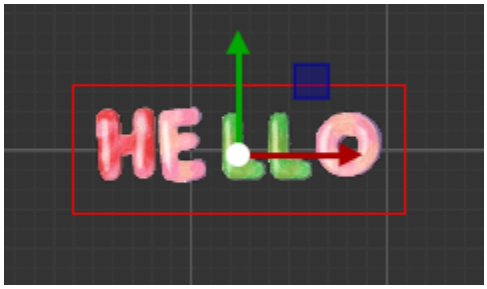
Property	Function
Load FontBitmap	Load the saved bitmap font
Save FontBitmap	Save the bitmap font
Image	Path to the image file (.pyxi)
Characters Set	Characters set to be generated
Generate Glyphs	Generate/reset glyphs for input characters set
Texture Size	The image size
Font Size	The font size
Font Base Size	The font base size
Index	Glyph index
Unicode	Character in Unicode format
Position	Top-left position of the character in the image
Size	Size of the character
Offset	Character offset
Advance	Character advance width

To create new bitmap font, flows steps below:

- Accquire bitmap texture file which contains all the characters, copy it to `fonts` folder.
- Open `Bitmap Font Creator`, select the image file.
- Input all the characters that is supported in `Characters Set` textbox.
- Generate glyphs by pressing `Generate Glyphs` button.


BitmapFontCreator
Load FontBitmap
Save FontBitmap
../../sample/fontBitmaps/ballon_f
Image
Characters Set
ABCDEFGHIJKLMNOPQRSTUVWXYZ
Generate Glyphs
0.0000
0.0000
Texture Size
72.0000
Font Size
72.0000
Font Base Size
0
Index
A
Unicode
3.0000
3.0000
Position
60.0000
70.0000
Size
0.0000
0.0000
Offset
60.0000
Advance
1
Index
B
Unicode
65.0000
3.0000
Position
50.0000
70.0000
Size
0.0000
0.0000
Offset
50.0000
Advance
2
Index
C
Unicode
0.0000
0.0000
Position
0.0000
0.0000
Size
0.0000
0.0000
Offset
0.0000
Advance
Prev Page
Next Page

- For each glyphs, input the position, size, offset and advance value.
- Save the font by pressing `Save FontBitmap` button.
- Test the font by create `UIText` component, then drag and drop the newly created font in the `Inspector` window.



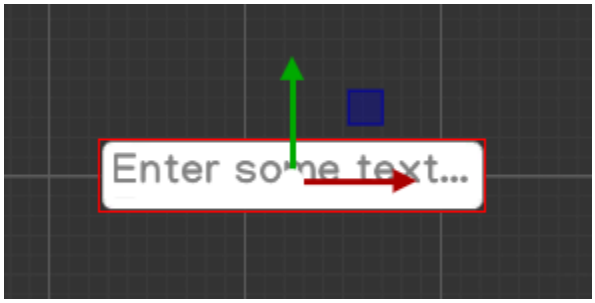
Note: Bitmap font only displayed as RGB texture if background use alpha channel. Otherwise, it will render as *grayscale* color to resolve alpha issue.

Tip: Saved Bitmap fonts can be modified with new characters set. Just need to add more character in the `Characters Set` textbox, then press `Generate Glyphs`, it will create new glyphs without affects existing glyphs.

Tip: Better to use an image editor (such as Paint.NET(R), MS Paint(R), Adobe(R) Photoshop(R)) to mesure the character attributes to put in the glyphs parameters.

UITextField

`UITextField` is used to display an editable text box to the user.



The usage of this component is similar to `UIText`, except it allows text to be input by user.

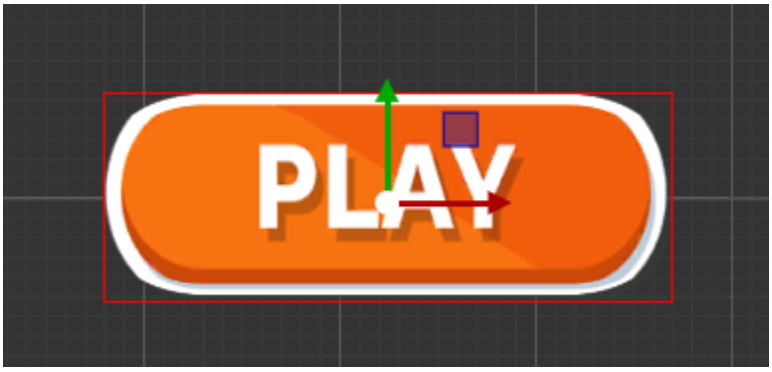
Property	Function
RectAutoScale	Auto resize the Rect Transform with text size
Text	The text to display
Font	The font to display (.ttf, .otf, .pybm)
Size	The font size
Color	Text color
Background	Text background color
AlignHorizontal	Horizontal alignment
AlignVertical	Vertical alignment

To handle the input ended event, add this code to `Script`:

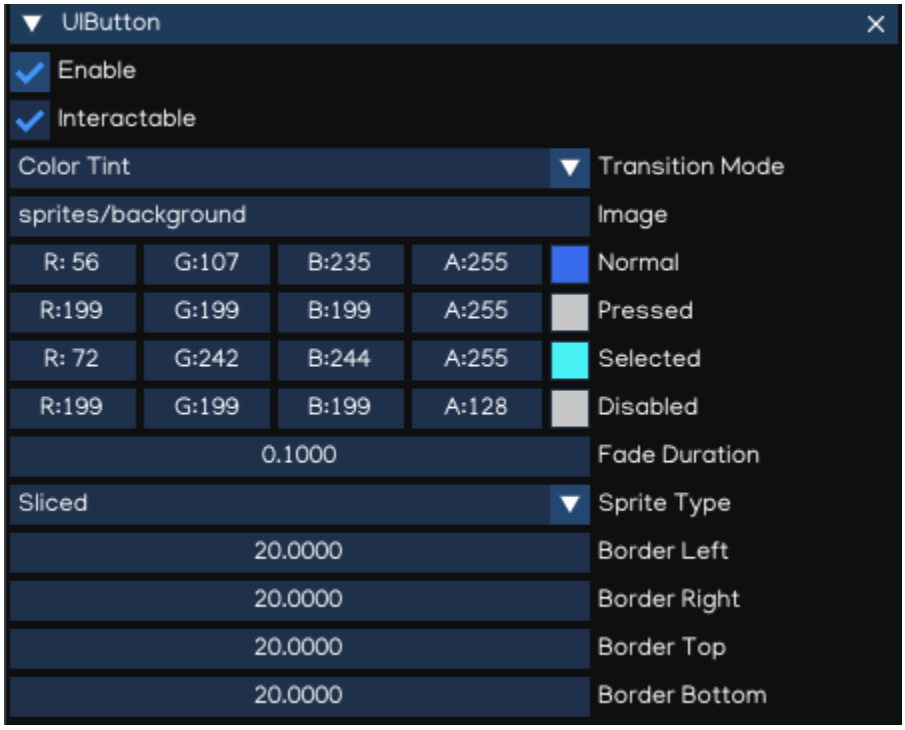
```
from igeScene import Script

class TxtUserName(Script):
   def __init__(self, owner):
      super().__init__(owner)
      # Read the value from UITextField
      self.username = owner.getComponent("UITextField").text
      print(f"Welcome {self.username}!")

   # Invoked at input ended
   def onValueChanged(self, val):
      self.username = val
      print(f"Welcome back {self.username}!")
```

UIButton

The `UIButton` component implement a button in GUI, which responds to a click from the user and is used to initiate or confirm an action.



The Inspector properties are as below:



Property	Function
Inteactable	Ability to receive events using Script
Transition Mode	The transition between button states: • *Color Tint* • *Sprite Swap*
Image	Background image
Normal	Color/sprite of the Normal state
Pressed	Color/sprite of the Pressed state
Selected	Color/sprite of the Selected state
Disabled	Color/sprite of the Disabled state
Fade Duration	Transition Duration
Color	Diffuse color
Sprite Type	The Sprite type, can be: • *Simple*: simple Sprite • *Sliced*: 9-slices sprite
Border Left	Border left percentage
Border Right	Border right percentage
Border Top	Border top percentage
Border Bottom	Border bottom percentage

The action can be controlled using `Script`, which `onClick` callback like below:

```
from igeScene import Script

class BtnNoAds(Script):
   def __init__(self, owner):
```

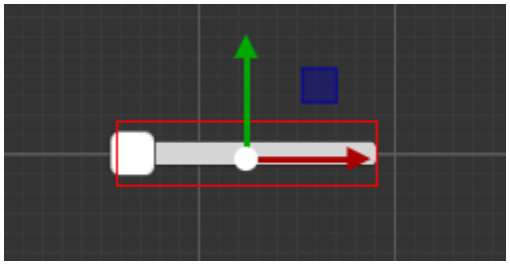
(continues on next page)

(continued from previous page)

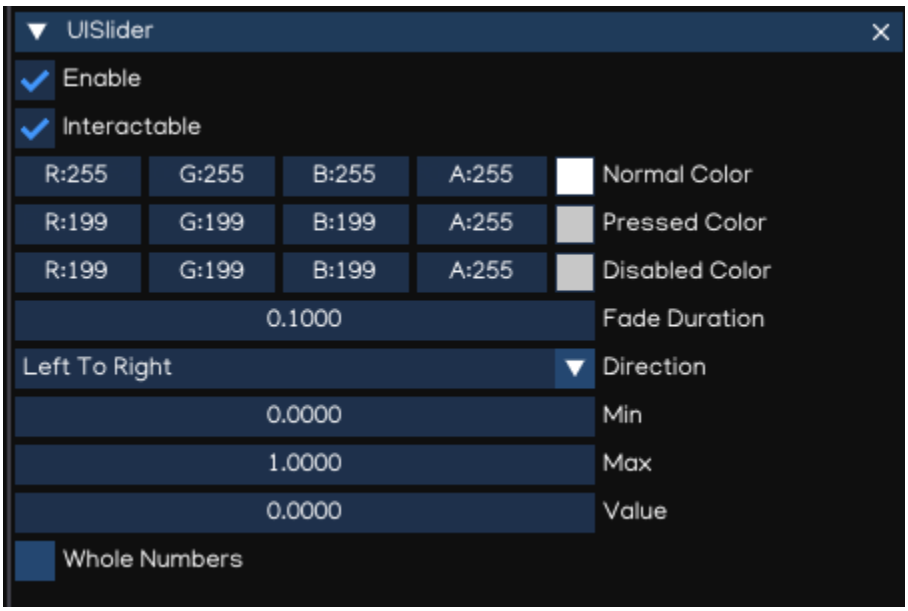
```
        super().__init__(owner)

    def onClick(self):
        print("NoAds Button Clicked, process purchasing...")
```

UISlider

The `UISlider` allows user to select a numeric value from a range by dragging the mouse.



The Inspector properties are as below:



Property	Function
Inteactable	Ability to receive events using Script
Normal	Color of the Normal state
Pressed	Color of the Pressed state
Disabled	Color of the Disabled state
Fade Duration	Transition Duration
Direction	Slider direction • *Left To Right* • *Right To Left* • *Bottom To Top* • *Top To Bottom*
Min	Min value
Max	Max value
Value	Current value
Whole Numbers	Constrained value to integer number when checked

To handle value changed event, add this code to `Script`:

```
from igeScene import Script

class VolumeSlider(Script):
   def __init__(self, owner):
      super().__init__(owner)

   def onValueChanged(self, val):
      self.volume = val
```

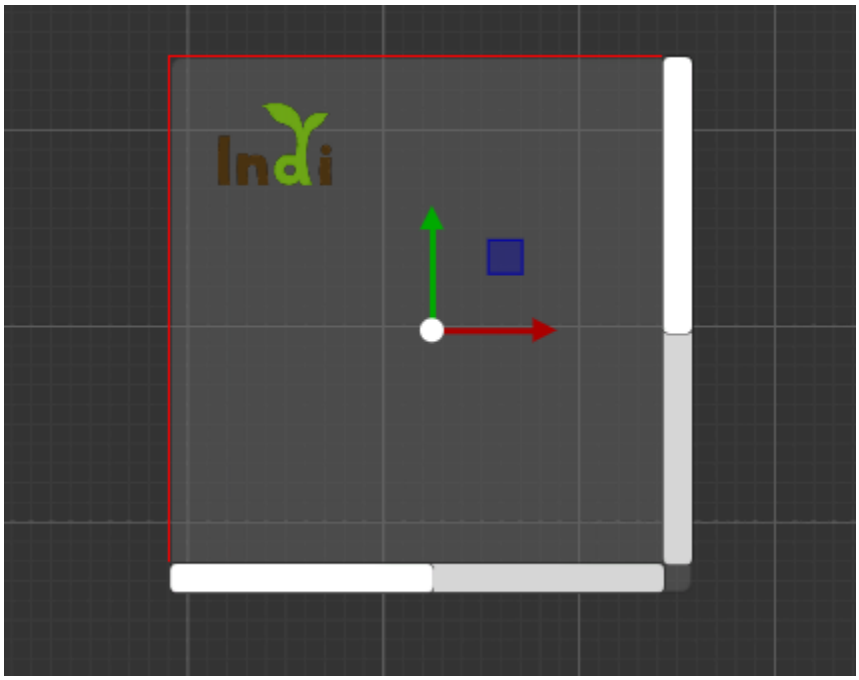
UIScrollView

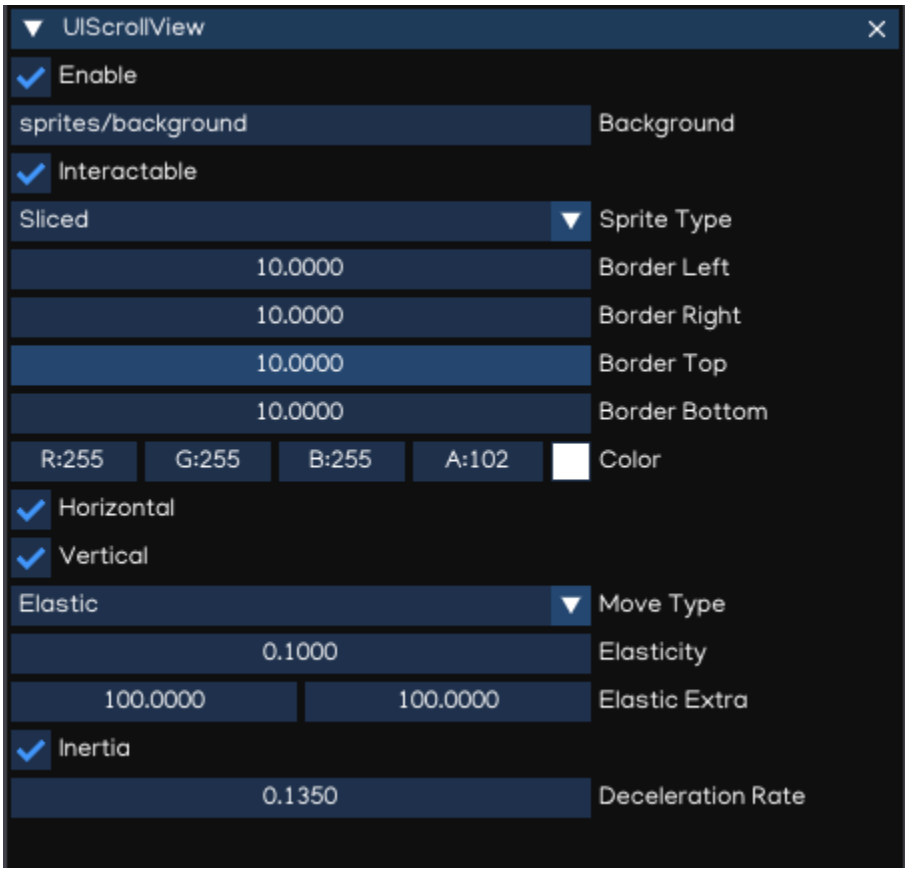
An `UIScrollView` can be used to scroll the content that takes up a lot of space and needs to be displayed in a small area. It is usually combined with an `UIMask` in order to create a scroll view, and with one or two `UIScrollBar` that can be dragged to scroll horizontally or vertically.

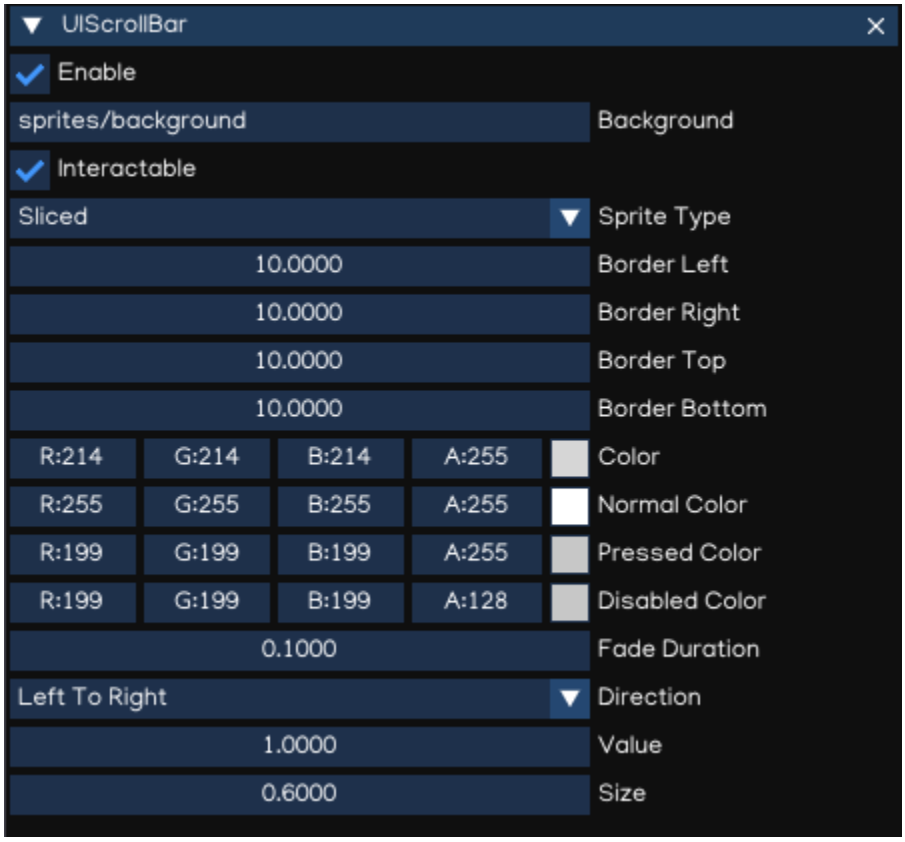
The Inspector properties are as below:

Property	Function
Inteactable	Ability to receive events using Script
Background	Background image
Sprite Type	Sprite type, either *Simple* or *Sliced*
Color	Diffuse color
Horizontal	Enable/disable horizontal scrollbar reference
Vertical	Enable/disable vertical scrollbar reference
Move Type	Movement type, either *Claimed* or *Elastic*
Elasticity	The amount of bounce used in the elasticity mode
Elastic Extra	The extra boundary allowed in Elastic mode.
Inertia	Allow content to move after pointer releasing
Deceleration Rate	Determines how quickly the contents stop moving

To support UIScrollView implement, the `UIScrollBar` is introduced to allow the user to scroll the view using drag handler.


Indi


UIScrollView
Enable
sprites/background
Background
Interactable
Sliced
Sprite Type
10.0000
Border Left
10.0000
Border Right
10.0000
Border Top
10.0000
Border Bottom
R:255
G:255
B:255
A:102
Color
Horizontal
Vertical
Elastic
Move Type
0.1000
Elasticity
100.0000
100.0000
Elastic Extra
Inertia
0.1350
Deceleration Rate



Property	Function
Inteactable	Ability to receive events using Script
Background	Background image
Sprite Type	Sprite type, either *Simple* or *Sliced*
Color	Diffuse color
Normal Color	Color of the handler in normal state
Pressed Color	Color of the handler in dragging state
Disabled Color	Color of the handler in disabled state
Fade Duration	Fading duration, in second
Direction	Dragging direction • *Left To Right* • *Right To Left* • *Bottom To Top* • *Top To Bottom*
Value	Current value
Size	Handler size

To handle value changed event, add this code to `Script`:

```
from igeScene import Script

class HScrollBar(Script):
   def __init__(self, owner):
```

(continues on next page)

(continued from previous page)

```
        super().__init__(owner)

    def onValueChanged(self, val):
        self.position = val
```

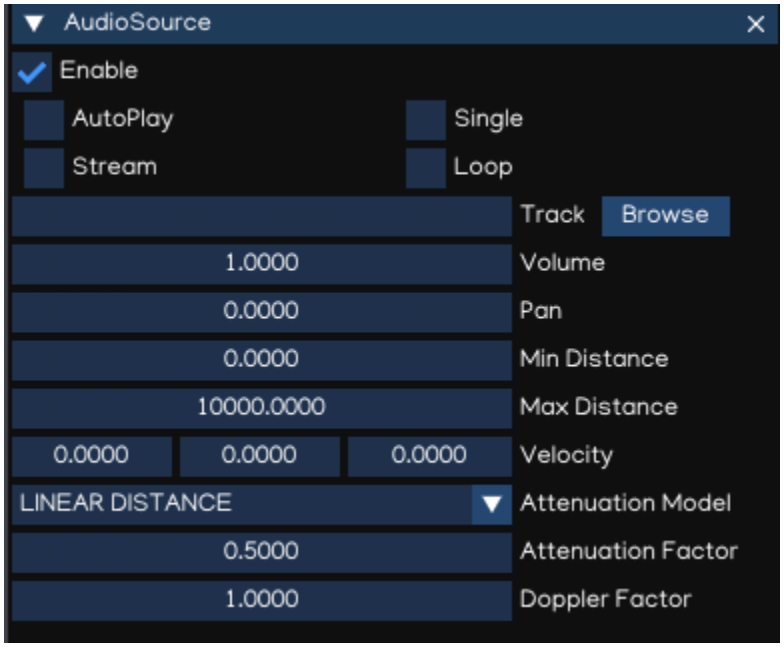
1.8 Audio

Indigames engine supports playing sounds in 3D space. Sounds are emitted by objects (sources) and heard by receivers (listeners).

1.8.1 AudioSource

The `AudioSource` is used to play an audio track, at the position of the object it is attached to, in 3D space.

Indigames engine supports playing *.ogg, .wav, .mp3, .mp4* formats.



Property	Function
AutoPlay	Whether auto play when loaded
Stream	Should stream audio or preload to memory
Single	Only one instance of this should play at the same time
Loop	Enable this to make the Audio track loop
Track	Audio track
Volume	Volume at a distance of one meter from the AudioListener
Pan	Panning value: -1 is Left, 0 is Center, 1 is Right
Min Distance	Audio source min distance: distance < min means max volume
Max Distance	Audio source max distance: distance > max means zero volume
Velocity	Audio source velocity
Attenuation Model	Attenuation model: • *NO ATTENUATION* • *INVERSE DISTANCE* • *LINEAR DISTANCE* • *EXPONENTIAL DISTANCE*
Attenuation Factor	Attenuation rolloff factor
Doppler Factor	Factor to reduce or enhance doppler effect

Refer to AudioSource API for usage within Python Script.

1.8.2 AudioListener

The AudioListener receives input from AudioSource in the scene and plays sounds through the computer speakers. It's usually attached to the main camera.

The audio system will play through only one listener at the same time, which is fisrt enabled AudioListener available.

Property	Function
Enable	Enable/disable the audio listener

Refer to AudioListener API for usage of AudioListener component within Python Script.

1.8.3 AudioManager

The `AudioManager` is automatically created and attached to the root object, to have the global setting of the Audio system.

Property	Function
Global Volume	Global volume of audio system

The AudioManager properties also can be controlled using Python Script. Refer to AudioManager Document for more details.

1.9 Physic

IGE built-in 3D physics engine is an integration of the Bullet Physic, which is a 3D physic engine.

1.9.1 Rigidbody

In physics simulation, rigid bodies enable physics-based behaviour such as movement, gravity, and collision. A `Rigidbody` is the main component that enables physical behaviour for a game object. With a Rigidbody attached, the object will immediately respond to gravity. If one or more `Collider` components are also added, the game object is moved by incoming collisions.

Property	Function
CCD	Enable/disable Continous Collision Detection mode
Kinematic	Set Rigidbody to Kinematic or Dynamic mode
Trigger	Enable trigger collision events
ActiveState	Set activation state
CollisionGroup	Collision group value
CollisionMask	Collision mask value
Mass	The mass of the object (in kilograms by default).
Friction	Friction value
Restitution	Restitution value (aka bounciness value)
LinearVelocity	Linear velocity
LinearFactor	Linear factor
LinearSleepThreshold	Linear sleeping threshold
AngularVelocity	Angular velocity
AngularFactor	Angular factor
AngularSleepThreshold	Angular sleeping threshold
PositionOffset	Position offset (adjust the center of the physic object)
Constraints	List of constraints applied in Rigidbody

Note: If the game object contains Rigidbody component, it's Transform will be controlled by the Rigidbody. Thus, to


Rigidbody
Enable
CCD
Kinematic
Trigger
Active
ActiveState
1
CollisionGroup
-1
CollisionMask
1.0000
Mass
0.5000
Friction
1.0000
Restitution
0.0000
0.0000
0.0000
LinearVelocity
1.0000
1.0000
1.0000
LinearFactor
0.8000
LinearSleepThreshold
0.0000
0.0000
0.0000
AngularVelocity
1.0000
1.0000
1.0000
AngularFactor
1.0000
AngularSleepThreshold
0.0000
0.0000
0.0000
PositionOffset
Constraints
Fixed Constraint
Add

change the transform just apply force or torque to the Rigidbody by using Python Script.

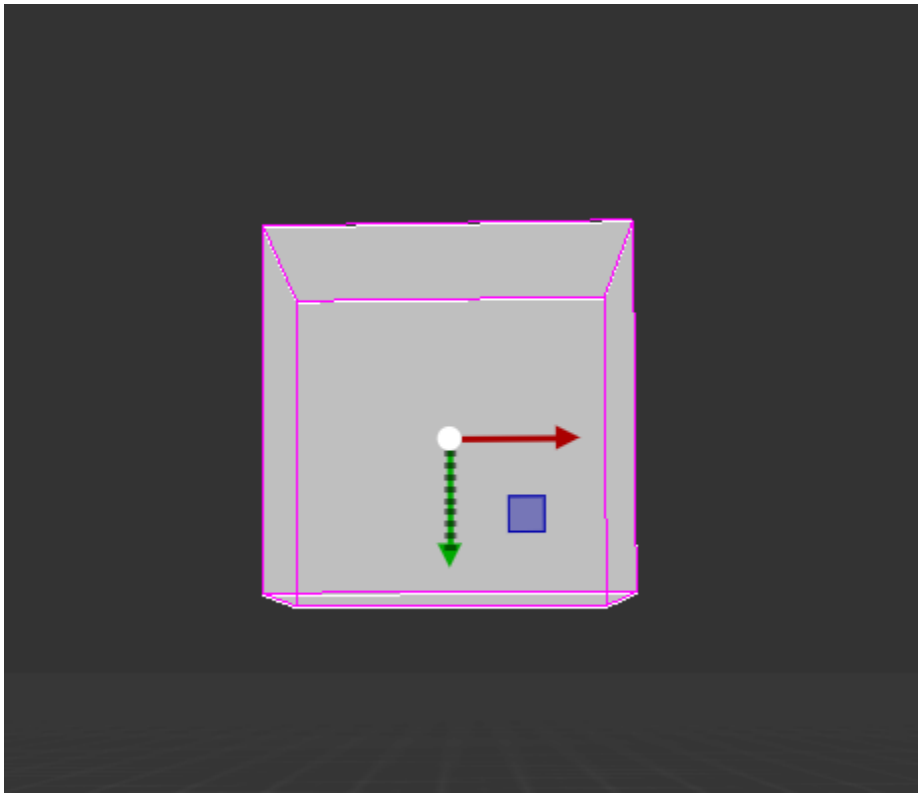
Note: When Trigger is enabled, use Python Script to receive triggered events. Refer to Rigidbody API for more details.

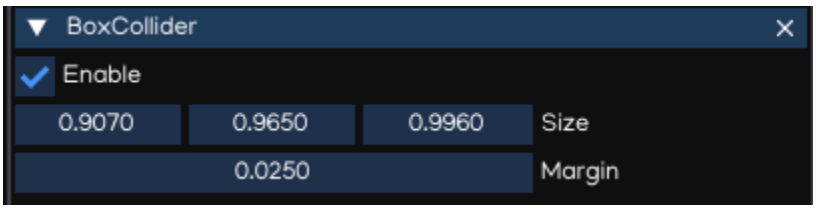
1.9.2 Collision

To configure collision between game objects, you need to use Colliders. Colliders define the shape of the game object for the purposes of physical collisions.

BoxCollider

The `BoxCollider` is a basic cuboid-shaped collision primitive, which are useful for items such as crates, chests, or floors using thin boxes. It can also be used to create complex collision shape using `CompoundCollider` component.



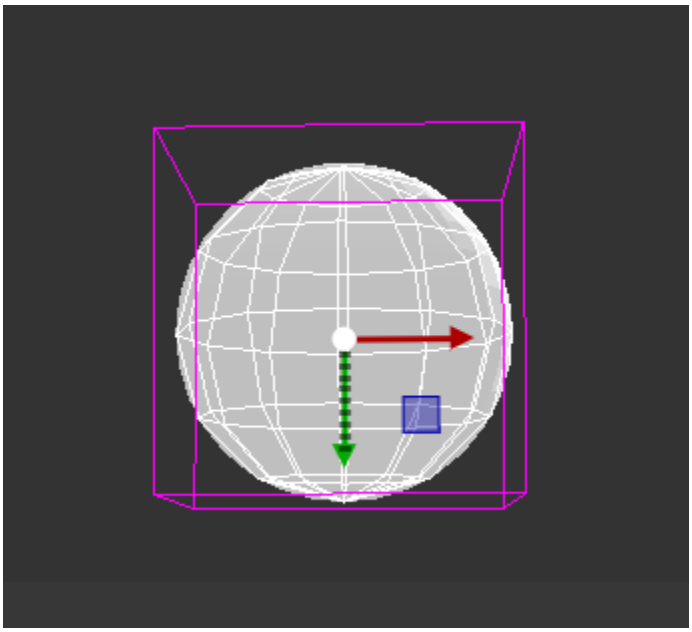


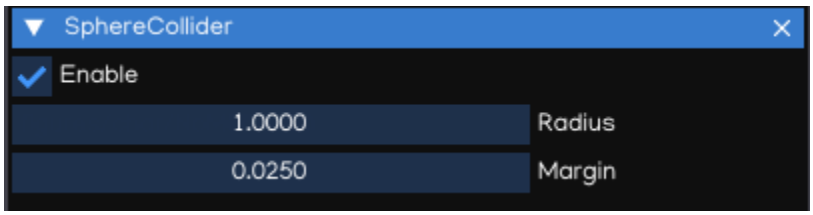
Property	Function
Size	Size of the collider in X, Y, Z direction
Margin	Collision margin

Note: Collision margin is used to optimize physic calculation, should keep it larger than 0.

SphereCollider

The `SphereCollider` is a basic sphere-shaped collision primitive.

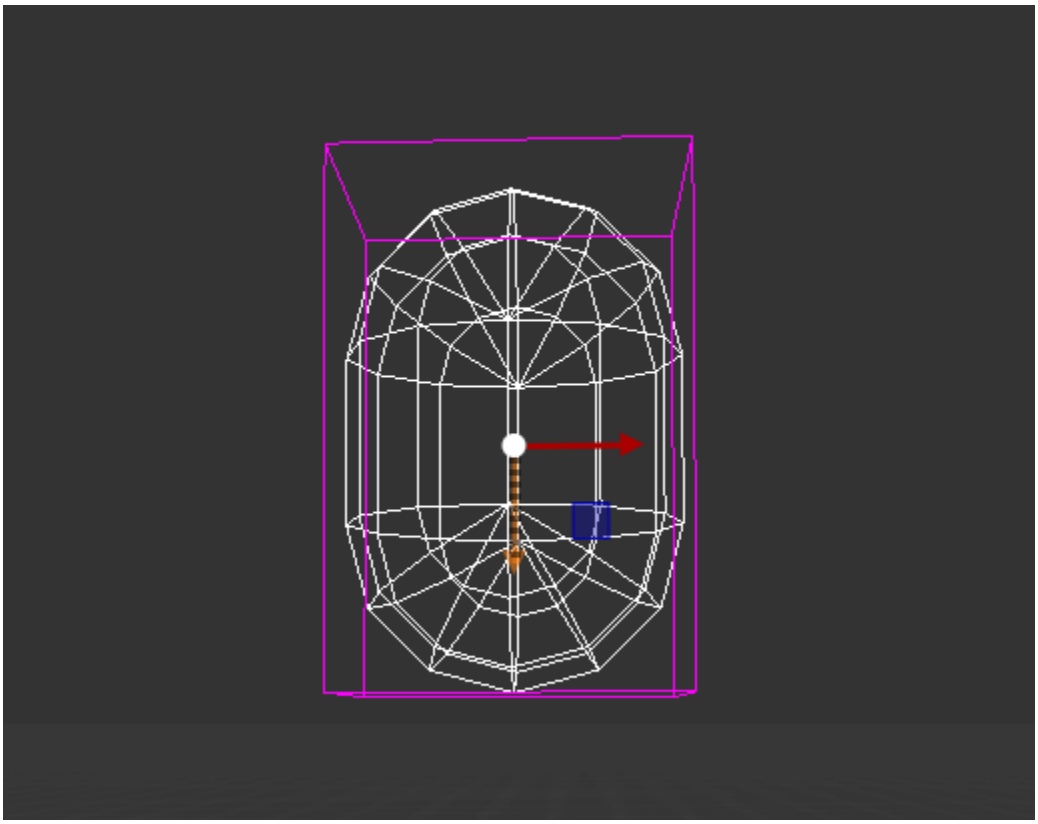




Property	Function
Radius	The radius of the sphere shape
Margin	Collision margin

CapsuleColider

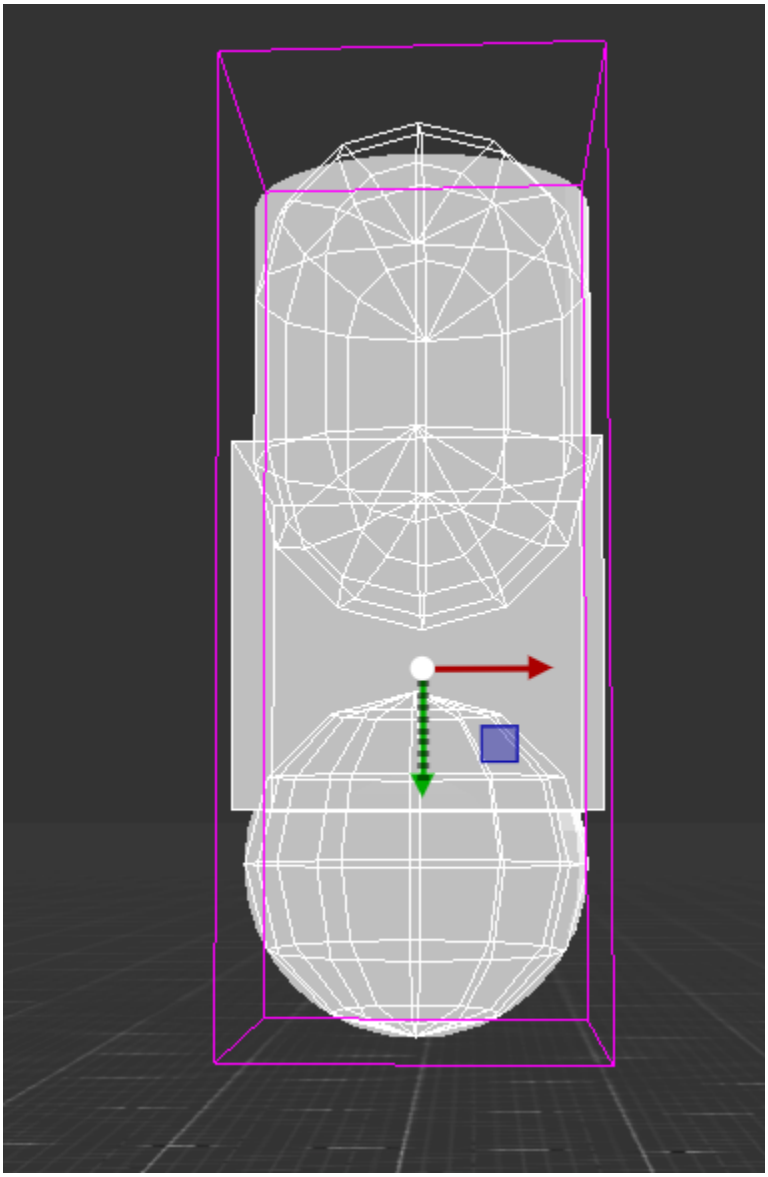
The `CapsuleCollider` is made of two half-spheres joined together by a cylinder, to create a capsule primitive shape.



Property	Function
Height	The total height of the collider
Radius	The radius of the collider width
Margin	Collision margin

CompoundCollider

Compound colliders approximate the shape of an object while keeping a low processor overhead, by combining primitive colliders of the child objects. When you create a compound collider like this, you should only use one Rigidbody component, placed on the owner object in the hierarchy.

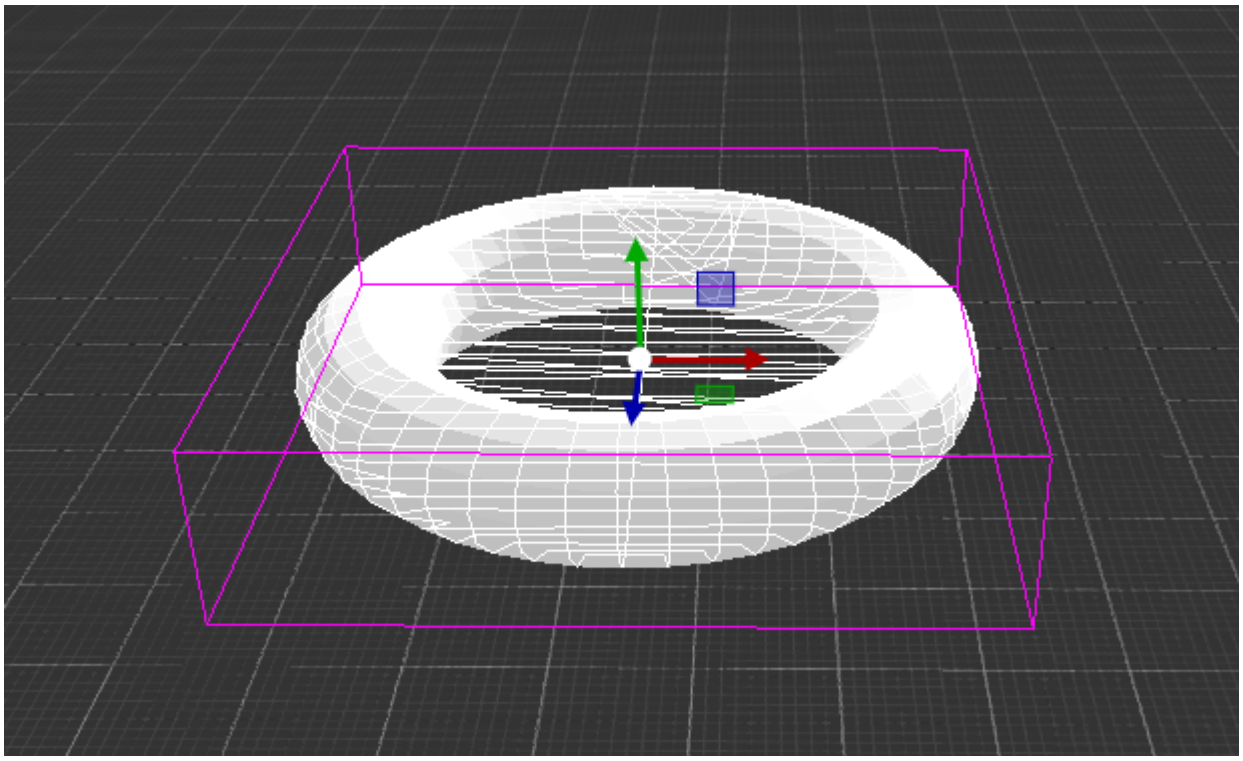


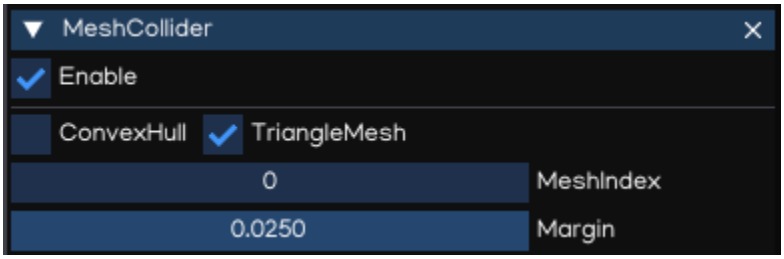
Note: CompoundCollider do not work with child objects which contains other CompoundCollider or MeshCollider.

Note: Should have only one Rigidbody attached to the whole hierarchy which the root object contains both CompoundCollider and Rigidbody. Otherwise, the simulation may not work as designed.

MeshCollider

The `MeshCollider` create Collider from meshes in `FigureComponent`. It is more accurate for collision detection than using primitives colliders.





Property	Function
ConvexHull	Create and convex hull from mesh
TriangleMesh	Use the triangle mesh
Margin	Collision margin

Note: Using MeshCollider results in higher processing overhead than primitive colliders, so it is best to use MeshColliders sparingly.

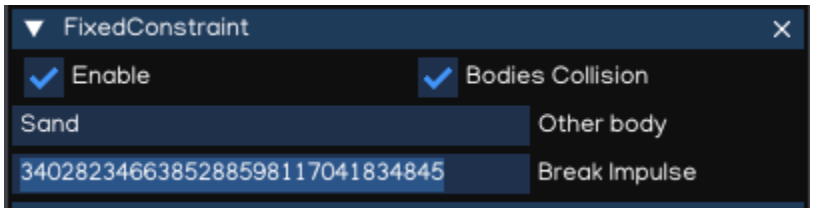
Note: Using TriangleMesh is only allowed if the Rigidbody is Kinematic.

1.9.3 Constraints

A constraint is used to connect a Rigidbody to another Rigidbody or a fixed point in space. Constraints apply forces that move rigid bodies, and limits restrict that movement.

FixedConstraint

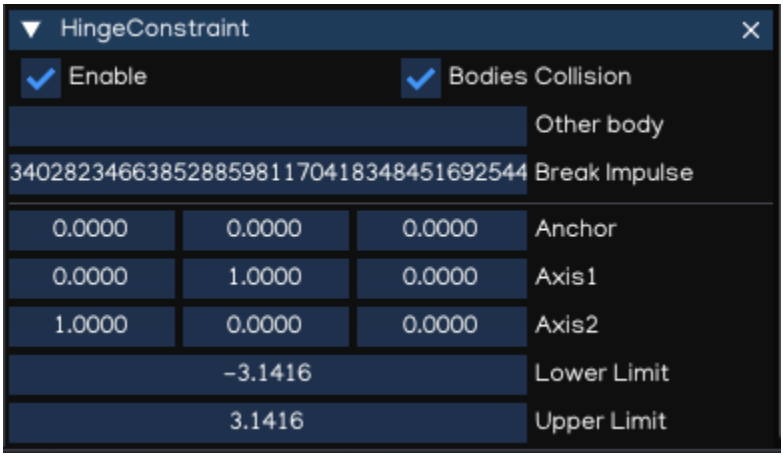
`FixedConstraint` restricts an object's movement to be dependent upon another object. The best scenarios for using them are when you have objects that you want to easily break apart from each other, or connect two object's movement without parenting.



Property	Function
Bodies Collision	Enable/disable collision between linked bodies
Other body	Other Rigidbody or Softbody component
Break Impulse	The force that needs to be applied for this constraint to break.

HingeConstraint

The `HingeConstraint` groups together two Rigidbodies, constraining them to move like they are connected by a hinge. It is perfect for doors, but can also be used to model chains, pendulums, etc…



Property	Function
Bodies Collision	Enable/disable collision between linked bodies
Other body	Other Rigidbody or Softbody component
Break Impulse	The force that needs to be applied for this constraint to break
Anchor	The position of the axis around which the body swings, in local space
Axis1	Rotation around Z
Axis2	Rotation around X
Lower Limit	The lowest angle the rotation can go
Upper Limit	The highest angle the rotation can go

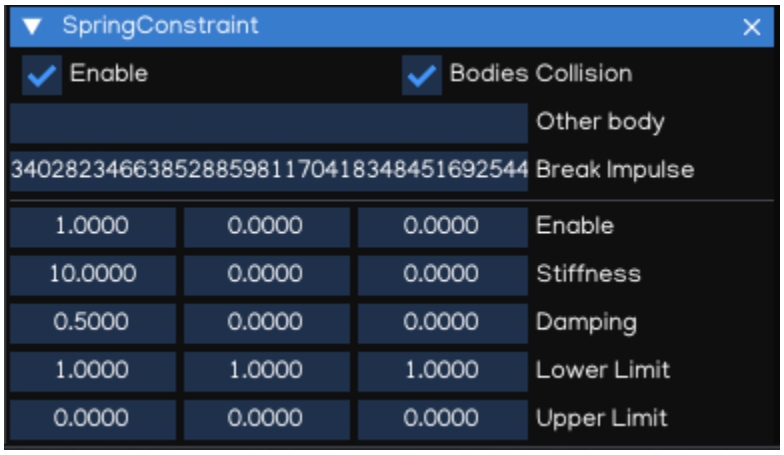
SliderConstraint

A `SliderConstraint` allows a object controlled by Rigidbody to slide along a line in space, like sliding doors, for example.

Property	Function
Bodies Collision	Enable/disable collision between linked bodies
Other body	Other Rigidbody or Softbody component
Lower Limit	Lower limit of the slider
Upper Limit	Upper limit of the slider

SpringConstraint

The `SpringConstraint` joins two Rigidbodies together but allows the distance between them to change as though they were connected by a spring.



Property	Function
Bodies Collision	Enable/disable collision between linked bodies
Other body	Other Rigidbody or Softbody component
Enable	Enable/disable spring on X, Y, Z axis
Stiffness	Spring stiffness in X, Y, Z axis
Damping	Amount that the spring is reduced when active
Lower Limit	Lower limit of the distance range over which the spring will not apply any force
Upper Limit	Upper limit of the distance range over which the spring will not apply any force

Dof6SpringConstraint

`Dof6SpringConstraint` incorporate all the functionality of the other constraint types and provide greater customization.

Property	Function
Bodies Collision	Enable/disable collision between linked bodies
Other body	Other Rigidbody or Softbody component
Lower Limit	Lower limit of the axis
Upper Limit	Upper limit of the axis
Target velocity	Target velocity
Bounce	Bounciness
Enable Spring	Enable/disable spring
Stiffness	Spring stiffness value
Damping	Spring damping value
Enable Motor	Enable/disable motor
Max Motor Force	Max motor force
Enable Servo	Enable/disable Servo
Servo Target	Servo target

The first 3 dof axis are linear axis, which represent translation of rigidbodies, and the latter 3 dof axis represent the angular motion. Each axis can be either locked, free or limited.

For each axis:

- Lowerlimit == Upperlimit -> axis is locked.
- Lowerlimit > Upperlimit -> axis is free.
- Lowerlimit < Upperlimit -> axis is limted in this range.

Check Bullet Physic manual document for more information.

1.9.4 Softbody

The soft body dynamics provides rope, cloth simulation and volumetric soft bodies, on top of the existing rigid body dynamics. The Softbody component works with FigureComponent, it manipulates Figure meshes to simulate deformable objects like cloth, fluid, jelly,...

Property	Function
CCD	Enable/disable Continous Collision Detection mode
Kinematic	**[Ignored]** Softbody is Dynamic object as alway.
Trigger	Enable trigger collision events
ActiveState	Set activation state
CollisionGroup	Collision group value
CollisionMask	Collision mask value
Mass	The mass of the object (in kilograms by default).
Friction	Friction value
Restitution	Restitution value (aka bounciness value)
LinearVelocity	Linear velocity
LinearFactor	Linear factor
LinearSleepThreshold	Linear sleeping threshold
AngularVelocity	Angular velocity

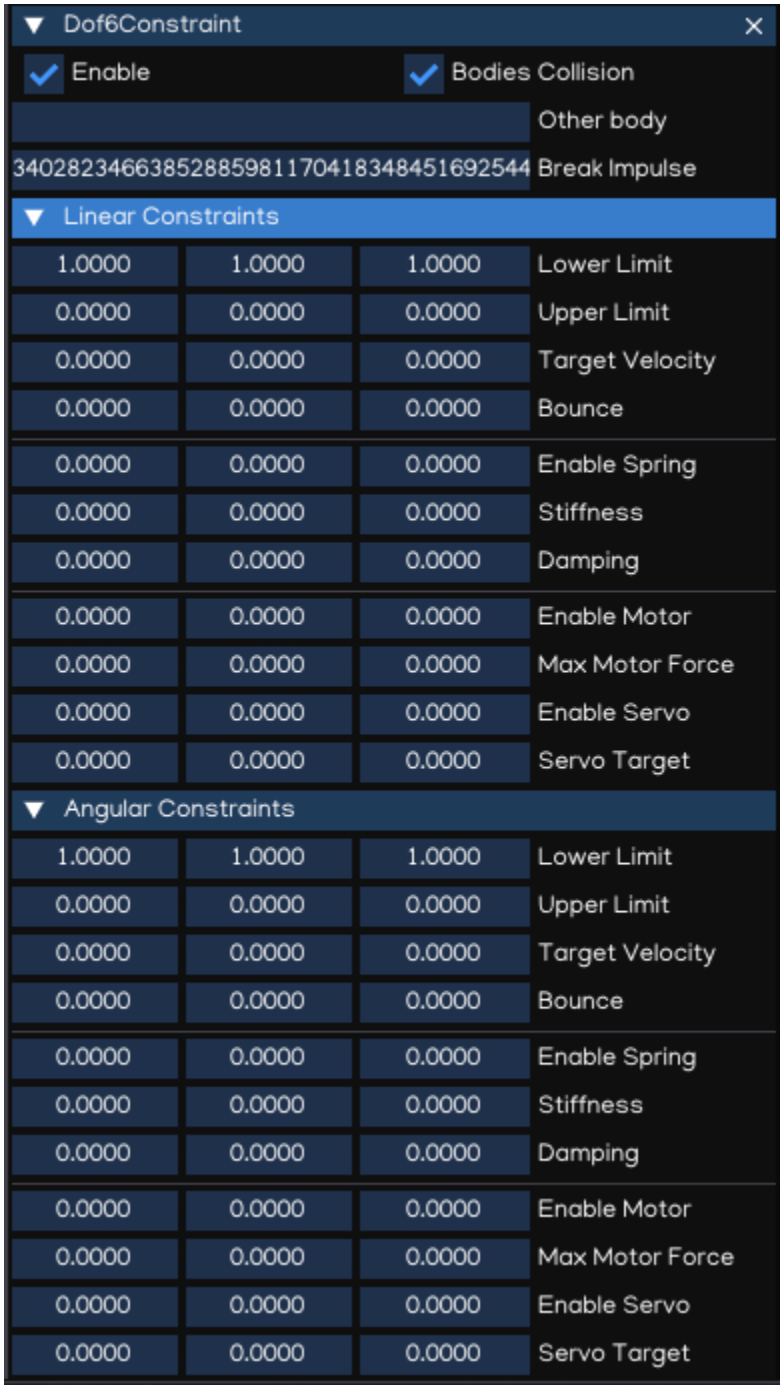
continues on next page

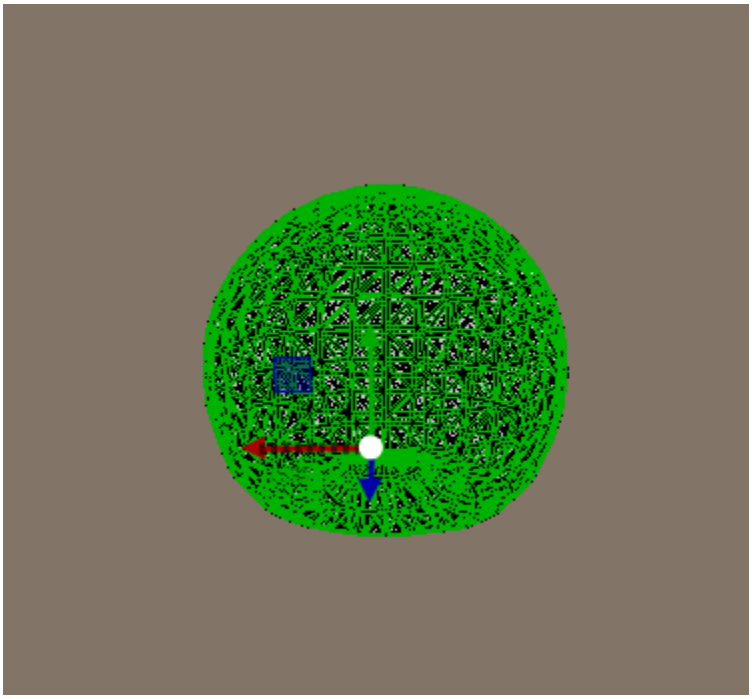
Table 2 – continued from previous page

Property	Function
AngularFactor	Angular factor
AngularSleepThreshold	Angular sleeping threshold
PositionOffset	**[Ignored]** Use mesh data without offset
SelfCollision	Enable/disable collision between parts of the shape
SoftCollision	Enable/disable soft collision
SpringStiffness	Spring stiffness value
RestLengthScale	Scale resting length of all springs
NumIterations	Positions solver iterations (pIterations)
SleepThreshold	Sleeping threshold
GravityFactor	Gravity factor
VelocityFactor	Velocities correction factor (kVCF)
DampingCoeff	Damping coefficient value (kDP)
PressureCoeff	Pressure coefficient value (kPR)
VolumeConvCoeff	Volume conversation coefficient [kVC]
FrictionCoeff	Dynamic friction coefficient (kDF)
PoseMatchCoeff	Pose matching coefficient (kMT)
RigidHardness	Rigid contacts hardness (kCHR)
KineticHardness	Kinetic contacts hardness (kKHR)
SoftHardness	Soft contacts hardness (kSHR)
AnchorHardness	Anchors hardness (kAHR)
AeroModel	Aerodynamic model (default: V_Point) • *V_Point*: Vertex normals are oriented toward velocity • *V_TwoSided*: Vertex normals are flipped to match velocity • *V_TwoSidedLiftDrag*: Vertex normals are flipped to match velocity and lift and drag forces are applied. • *V_OneSided*: Vertex normals are taken as it is • *F_TwoSided*: Face normals are flipped to match velocity • *F_TwoSidedLiftDrag*: Face normals are flipped to match velocity and lift and drag forces are applied • *F_OneSided*: Face normals are taken as it is
WindVelocity	Wind velocity for interaction with the air
Constraints	List of constraints applied

Softbody also works with all type of Constraints, together with Rigidbodies or other Softbodies.

Check Bullet Physic manual document for more information.


Dof6Constraint
Enable
Bodies Collision
Other body
340282346638528859811704183484516925440
Break Impulse
Linear Constraints
1.0000 1.0000 1.0000 Lower Limit
0.0000 0.0000 0.0000 Upper Limit
0.0000 0.0000 0.0000 Target Velocity
0.0000 0.0000 0.0000 Bounce
0.0000 0.0000 0.0000 Enable Spring
0.0000 0.0000 0.0000 Stiffness
0.0000 0.0000 0.0000 Damping
0.0000 0.0000 0.0000 Enable Motor
0.0000 0.0000 0.0000 Max Motor Force
0.0000 0.0000 0.0000 Enable Servo
0.0000 0.0000 0.0000 Servo Target
Angular Constraints
1.0000 1.0000 1.0000 Lower Limit
0.0000 0.0000 0.0000 Upper Limit
0.0000 0.0000 0.0000 Target Velocity
0.0000 0.0000 0.0000 Bounce
0.0000 0.0000 0.0000 Enable Spring
0.0000 0.0000 0.0000 Stiffness
0.0000 0.0000 0.0000 Damping
0.0000 0.0000 0.0000 Enable Motor
0.0000 0.0000 0.0000 Max Motor Force
0.0000 0.0000 0.0000 Enable Servo
0.0000 0.0000 0.0000 Servo Target



1.9.5 PhysicManager

The `PhysicManager` is automatically created and attached to the root object, to have the global setting of the Physic system.

Property	Function
Deformable	Enable/disable physic with Softbody simulation
Debug	Show Physic debug
NumIterations	Number of iterations per frame
NumSubsteps	Number of substeps. If NumSubSteps > 0, interpolate motion between fixedTimeStep
TimeStep	Fixed time step value (default: 1/60)
UpdateRatio	Update ratio, useful to do slow motion effect
Gravity	Global gravity value

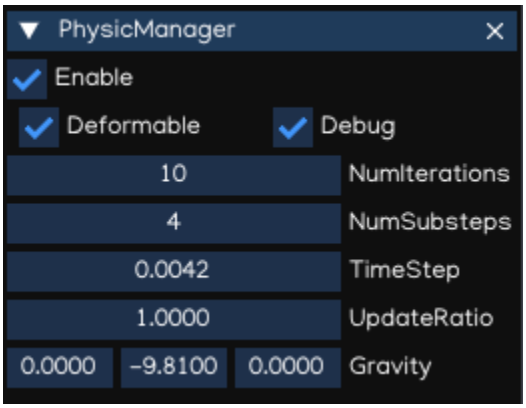
Please refer to Bullet Physic Manual and Python API Document document for more details of Physic usage using IGE.

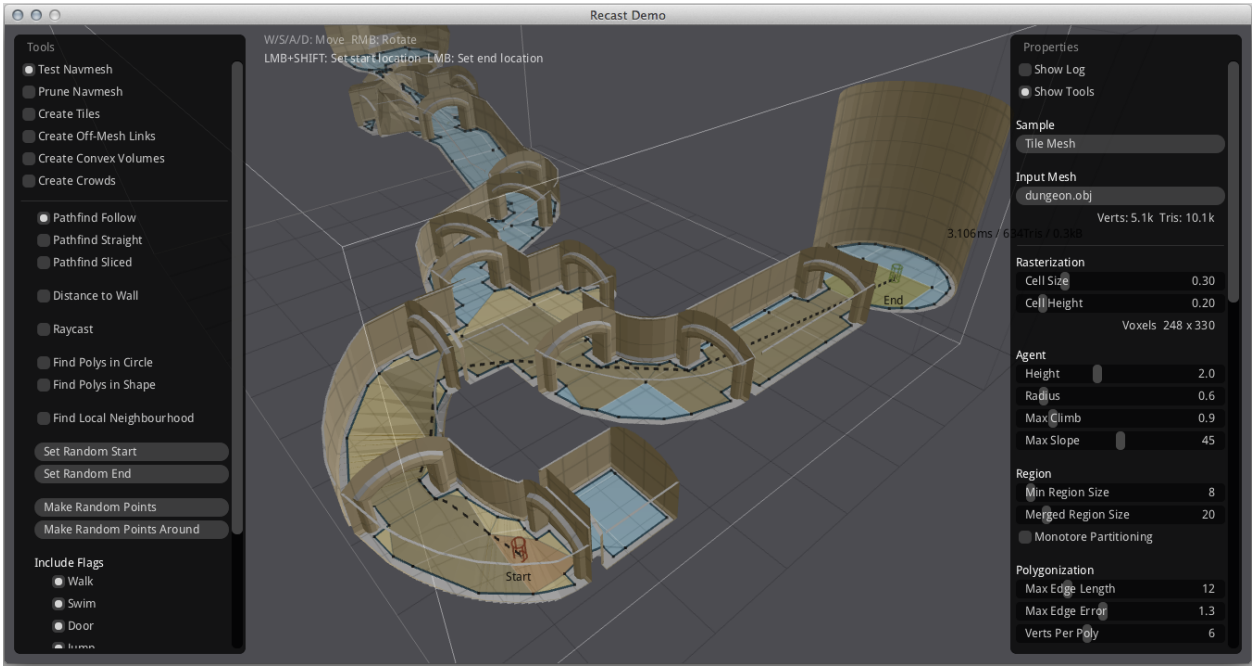
1.10 Navigation

The navigation system allows you to create characters that can intelligently move around the game world, using navigation meshes that are created automatically from your Scene geometry. Dynamic obstacles allow you to alter the navigation of the characters at runtime, while off-mesh links let you build specific actions like opening doors or jumping down from a ledge.

IGE Navigation system implement Recast & Detour libraries which provide both navigation mesh contruction toolset and path-finding toolkit.


Softbody
Enable
CCD
Kinematic
Trigger
Active
ActiveState
1
CollisionGroup
-1
CollisionMask
0.0000
Mass
0.5000
Friction
1.0000
Restitution
0.0000 0.0000 0.0000
LinearVelocity
1.0000 1.0000 1.0000
LinearFactor
0.8000
LinearSleepThreshold
0.0000 0.0000 0.0000
AngularVelocity
1.0000 1.0000 1.0000
AngularFactor
1.0000
AngularSleepThreshold
0.0000 0.0000 0.0000
PositionOffset
SelfCollision
SoftCollision
0.5000
SpringStiffness
1.0000
RestLengthScale
1
NumIterations
0.0400
SleepThreshold
1.0000
GravityFactor
1.0000
VelocityFactor
1.0000
DampingCoeff
0.0000
PressureCoeff
0.0000
VolumeConvCoeff
0.2000
FrictionCoeff
0.0000
PoseMatchCoeff
1.0000
RigidHardness
0.1000
KineticHardness
1.0000
SoftHardness
1.0000
AnchorHardness
V_TwoSided
AeroModel
0.0000 0.0000 0.0000
WindVelocity
0
Mesh Index
Constraints
Fixed Constraint
Add


PhysicManager
Enable
Deformable
Debug
10
NumIterations
4
NumSubsteps
0.0042
TimeStep
1.0000
UpdateRatio
0.0000
-9.8100
0.0000
Gravity


Recast Demo
Tools
Test Navmesh
Prune Navmesh
Create Tiles
Create Off-Mesh Links
Create Convex Volumes
Create Crowds
Pathfind Follow
Pathfind Straight
Pathfind Sliced
Distance to Wall
Raycast
Find Polys in Circle
Find Polys in Shape
Find Local Neighbourhood
Set Random Start
Set Random End
Make Random Points
Make Random Points Around
Include Flags
Walk
Swim
Door
W/S/A/D: Move RMB: Rotate
LMB+SHIFT: Set start location LMB: Set end location
Start
End
Properties
Show Log
Show Tools
Sample
Tile Mesh
Input Mesh
dungeon.obj
Verts: 5.1k Tris: 10.1k
Rasterization
Cell Size 0.30
Cell Height 0.20
Voxels 248 x 330
Agent
Height 2.0
Radius 0.6
Max Climb 0.9
Max Slope 45
Region
Min Region Size 8
Merged Region Size 20
Monotore Partitioning
Polygonization
Max Edge Length 12
Max Edge Error 1.3
Verts Per Poly 6

1.10.1 NavMesh

NavMesh is a data structure which describes the walkable surfaces of the game world and allows to find path from one walkable location to another in the game world. The data structure is built automatically from your level geometry.

NavMesh collects geometry from its child nodes that have been tagged with the Navigable component. By default the Navigable component behaves recursively, unless the recursion is disabled.

The easiest way to make the whole scene participate in navigation mesh generation is to create the NavMesh component to the scene root node, and Navigable to the game object that act as navigating routes.

The navigation mesh generation must be triggered manually by pressing “Build” button which canbe found in NavMesh inspector window.

Property	Function
Debug	Draw debug
Build	Build NavMesh data
TileSize	The width/height size of tile's on the xz-plane
CellSize	The xz-plane cell size to use for fields
CellHeight	The y-axis cell size to use for fields
AgentHeight	Agent height
AgentRadius	Agent radius
AgenMaxClimb	Maximum ledge height that is considered to still be traversable
AgentMaxSlope	The maximum slope that is considered walkable
RegionMinSize	The minimum number of cells allowed to form isolated island areas
RegionMergeSize	Regions with span count smaller than this will be merged with larger regions
EdgeMaxLength	The maximum allowed length for contour edges along the border of the mesh
EdgeMaxError	The maximum distance a contour's border edges should deviate original contour
SampleDistance	The sampling distance to use when generating the detail mesh
SampleMaxError	The maximum distance the detail mesh surface should deviate from heightfield
Padding	The bounding box padding to generate navigation data
PartitionType	Partitioning type: • Watershed: build distance fields and regions data • Monotone: build monotone regions (faster but less accurate)

Note: NavMesh does not support NavObstacle to be added dynamictically at runtime. So, it's better to be used with static geometry only.

1.10.2 DynamicNavMesh

DynamicNavMesh supports the addition and removal of dynamic obstacles. Using DynamicNavMesh has the trade-off over traditional NavMesh is that it will cost almost twice the memory consumption. However, the addition and removal of obstacles is significantly faster than partially rebuilding a NavMesh.


DynamicNavMesh
Enable
Debug
Build
64
TileSize
0.3000
CellSize
0.2000
CellHeight
2.0000
AgentHeight
0.6000
AgentRadius
0.9000
AgentMaxClimb
45.0000
AgentMaxSlope
8.0000
RegionMinSize
20.0000
RegionMergeSize
12.0000
EdgeMaxLength
1.3000
EdgeMaxError
6.0000
SampleDistance
1.0000
SampleMaxError
1.0000
1.0000
1.0000
Padding
Watershed
PartitionType
1024
MaxObstacle
16
MaxLayer

Property	Function
Debug	Draw debug
Build	Build NavMesh data
TileSize	The width/height size of tile's on the xz-plane
CellSize	The xz-plane cell size to use for fields
CellHeight	The y-axis cell size to use for fields
AgentHeight	Agent height
AgentRadius	Agent radius
AgenMaxClimb	Maximum ledge height that is considered to still be traversable
AgentMaxSlope	The maximum slope that is considered walkable
RegionMinSize	The minimum number of cells allowed to form isolated island areas
RegionMergeSize	Regions with span count smaller than this will be merged with larger regions
EdgeMaxLength	The maximum allowed length for contour edges along the border of the mesh
EdgeMaxError	The maximum distance a contour's border edges should deviate original contour
SampleDistance	The sampling distance to use when generating the detail mesh
SampleMaxError	The maximum distance the detail mesh surface should deviate from heightfield
Padding	The bounding box padding to generate navigation data
PartitionType	Partitioning type: • Watershed: build distance fields and regions data • Monotone: build monotone regions (faster but less accurate)
MaxObstacle	Max number of obstacles allowed (lower is better)
MaxLayer	Maximum number of layers that are allowed to be constructed

1.10.3 Navigable

Navigable is a Component which tags geometry for inclusion in the navigation mesh. Optionally auto-includes geometry from child nodes.

Property	Function
Recursive	Whether geometry is collected from child nodes

1.10.4 NavArea

NavArea is a utility to mark a region differentiate with others, and potential have different navigation cost to travel through. It's useful to predefine all type of areas, such as Ground, Water, Sand, Snow ... as areaId, up to 64 different area types. The areaId then assigned to NavArea component, to configure traversal cost for the agent to go through.

Property	Function
ID	Area Id, from 0 - 62

Navigation System supports different filters for each type of NavAgent, up to 16 types. For each agent type, the area cost canbe configured separately, providing abilities to customize agent behaviors.

To configure area cost for each area, for each type of agent, use Python API Document, as below:

```
from igeScene import Script, NavAgentManager
from enum import Enum

class AgentType(Enum):
   MC = 0
   NPC = 1

class AreaType(Enum):
   GROUND = 63
   WATER = 0
   SNOW = 1

class AgentManager(Script):
   def __init__(self, owner):
      super().__init__(owner)
      self.navAgentManager = None

   def onStart(self):
      self.navAgentManager = owner.getComponent("NavAgentManager")
      self.navAgentManager.setAreaCost(AgentType.MC, AreaType.GROUND, 1.0)
      self.navAgentManager.setAreaCost(AgentType.MC, AreaType.WATER, 5.0)
      self.navAgentManager.setAreaCost(AgentType.MC, AreaType.SNOW, 2.0)
      self.navAgentManager.setAreaCost(AgentType.NPC, AreaType.GROUND, 1.0)
      self.navAgentManager.setAreaCost(AgentType.NPC, AreaType.WATER, 100.0)
      self.navAgentManager.setAreaCost(AgentType.NPC, AreaType.SNOW, 2.0)
```

Note: For regions which are not marked using NavArea, it will have areaId set to 63, and areaCost set to 1, by default.

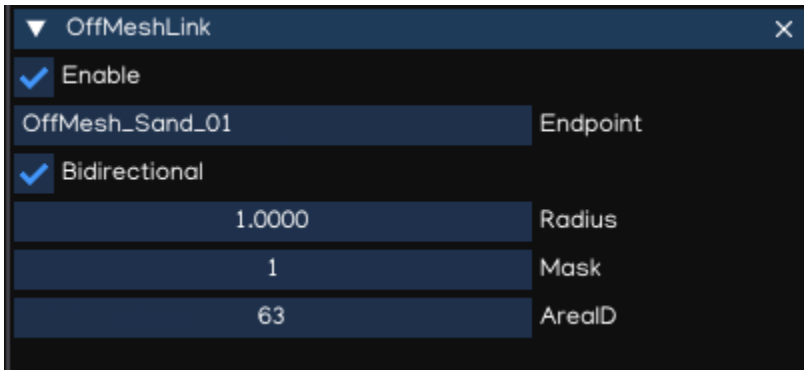
1.10.5 OffMeshLink

Off-Mesh Links are used to create paths crossing outside the walkable navigation mesh surface. For example, jumping over a ditch or a fence, or opening a door before walking through it, can be all described as Off-mesh links.

To use OffMeshLink optimally, follow steps below:

1. First create two cylinders, scale to (0.1, 0.2, 0.1) to make it easier to work with them.
2. Move the first cylinder inside the first NavMesh surface.
3. Move the second cylinder inside the other NavMesh surface, at the location where the link should land.
4. Select the first cylinder and add an OffMeshLink component to it.
5. Drag the second cylinder from Hierarchy to the Endpoint in the Inspector.

If the path via the off-mesh link is shorter than via walking along the Navmesh, the off-mesh link will be used.

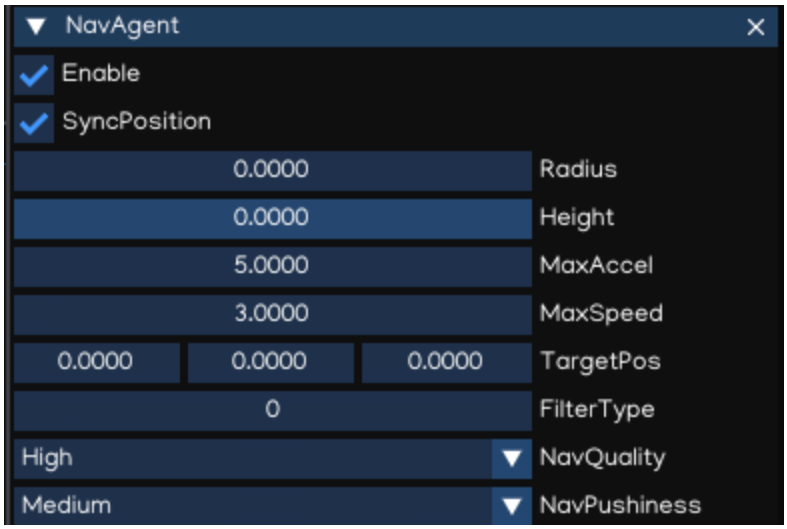


Property	Function
Endpoint	The endpoint object, which position is the landing position.
Bidirectional	If enabled, the link can be traversed in either direction.
Radius	Radius of the link, where the center point is object position.
Mask	Off-Mesh link mask
AreaId	Area Id, which pre-setup for traversal cost.

1.10.6 NavAgent

NavAgent components help you to create characters which avoid each other and obstacles while moving towards their goal.

Property	Function
SyncPosition	Update position by NavAgentManager, or not
Radius	The agent's radius
Height	The agent's height
MaxAccel	The agent's max acceleration
MaxSpeed	The agent's max velocity
TargetPos	Target position to travel to
FilterType	The agent's filter type
NavQuality	The agent's navigation quality
NavPushiness	The agent's navigation pushiness



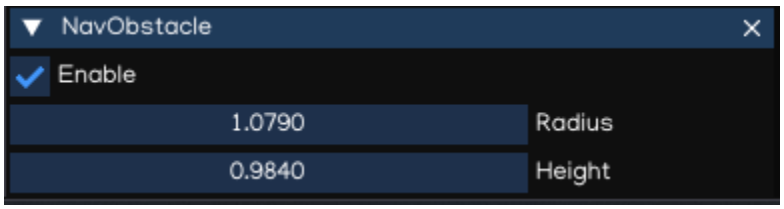
The NavAgent handles both the pathfinding and the movement control of a character. In your scripts, navigation can be as simple as setting the desired destination point:

```
from igeScene import Script, NavAgent
import igeVmath as vmath

class MCAgent(Script):
   def __init__(self, owner):
      super().__init__(owner)
      self.navAgent = None

   def onStart(self):
      self.navAgent = owner.getComponent("NavAgent")
      self.navAgent.targetPosition = vmath.vec3(10, 10, 10)
```

1.10.7 NavObstacle

NavObstacle components can be used to describe obstacles the agents should avoid while navigating. For example the agents should avoid physics controlled objects, such as crates and barrels while moving.

To do this, add NavObstacle component to the object, then configure it's properties:



Property	Function
Radius	The obstacle's radius
Height	The obstacle's height

Then the NavAgent will avoid the obstacle object while navigating, even if the object is moving around.

Note: NavObstacle only works with DynamicNavMesh. It's ignored if the scene use NavMesh instead.

1.10.8 NavAgentManager

NavAgentManager is used to control the navigating of all NavAgents in the Scene. It's automatically created when creating NavMesh or DynamicNavMesh, and usually added to the root object of the Scene.

Property	Function
Max Agents	Max number of agents
Max Agent Radius	The agent's max radius

NavAgentManager also provides useful functions to control the agents by using Python Script. Refer to Python API Document for more information.

1.11 Particle System

IGE Particle system implements Effekseer, allows playing effects created with Effekseer on IGE Engine.

1.11.1 Effekseer Editor

Effekseer is a tool that allows easy creation of beautiful particle effects for games and movies.

Check the Effekseer Tutorial to learn how to work with Effekseer Editor.

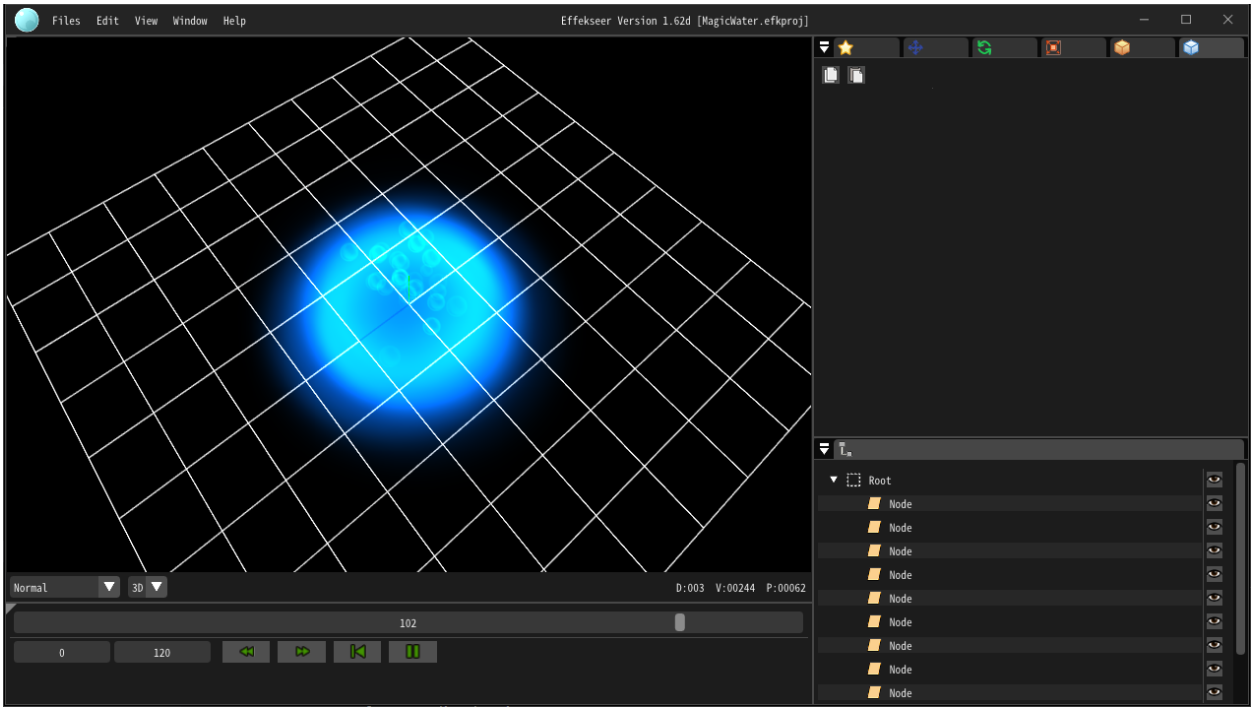
Note: IGE Engine implements Effekseer 1.60c runtime, which supports loading effects produced by the Effekseer version 1.6x.

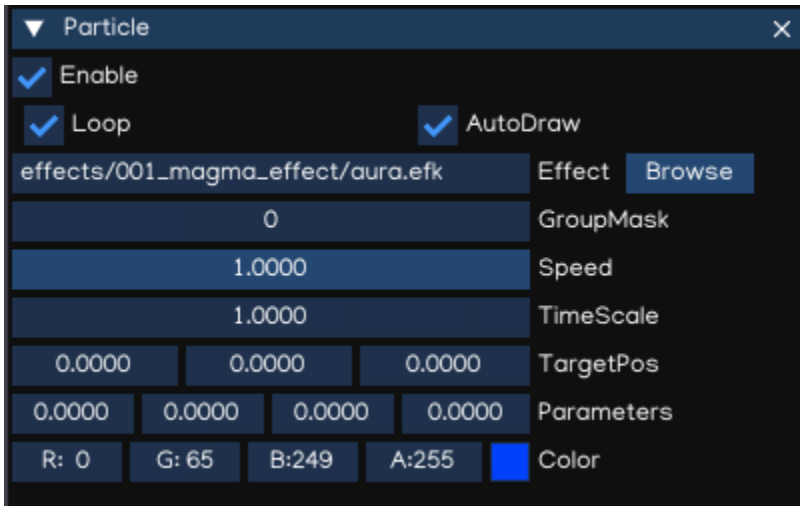
1.11.2 Particle

Particle component is used to load and display Effekseer effect in IGE Engine. It canbe used both in 3D and UI objects.

To add particle effects to your project, follow steps below:

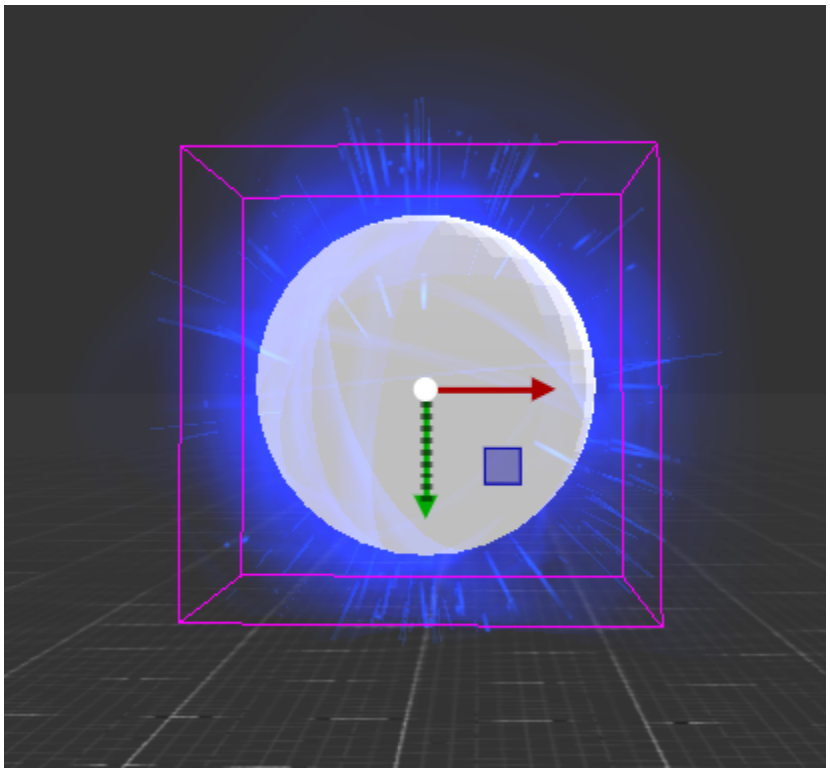
1. Create effect using Effekseer Editor, or download effect from sample repo.
2. Copy your effect files (.efk), textures, sounds, materials, etc. into `<project>/effects` folder.
3. Add `Particle` component to the game object.
4. Drag & drop the .efk file to the Inspector
5. Configure the effect parameters


Files
Edit
View
Window
Help
Effekseer Version 1.62d [MagicWater.efkproj]
Root
Node
Normal
3D
D:003 V:00244 P:00062
102
0
120


Particle
Enable
Loop
AutoDraw
effects/001_magma_effect/aura.efk
Effect
Browse
0
GroupMask
1.0000
Speed
1.0000
TimeScale
0.0000
0.0000
0.0000
TargetPos
0.0000
0.0000
0.0000
0.0000
Parameters
R: 0
G: 65
B:249
A:255
Color

Property	Function
Effect	Path to .efk file, inside `effects` folder
Loop	Enable/disable loop
AutoDraw	Auto play and draw particle when loaded
GroupMask	Particle group mask, useful to control particles using Python Script.
Speed	Playing speed
TimeScale	Playing time scale, also affect displaying speed
TargetPos	Target position (used by particle effect)
Parameters	Particle parameters
Color	Particle diffuse color

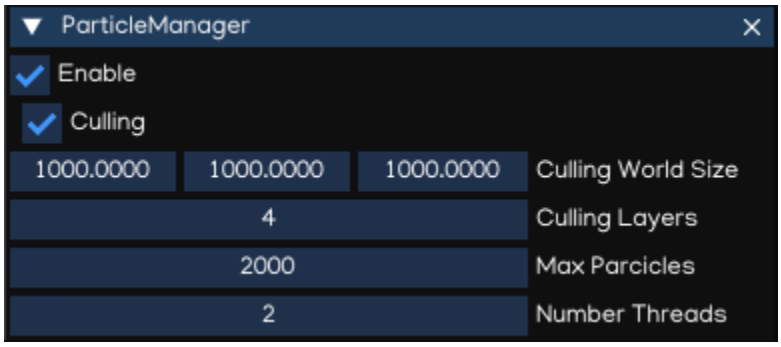
An example of using particle:



Note: In UI node, the effect may appear bigger because of scaling, just need to set the scale parameter to make it reasonable.

1.11.3 ParticleManager

ParticleManager is used to manage Particle instance and global configuration. It is automatically added to the root object when a Particle is used.



Property	Function
Culling	Enable/disable particle culling
Culling World Size	Culling world size
Culling Layers	Number of culling layers
Max Particles	Max number of particle intances
Number Threads	Number of running threads

For more information about Particle System, refer to Effekseer Document, and Python API Document.

1.12 Platform Configuration

IGE Creator works on Windows and MacOS workstation. The engine supports building games for Windows, MacOS, iOS, Android and WebGL platforms.

1.12.1 Dependencies

Windows Workstation

In order to work with IGE Engine on Windows machine, please make sure to install softwares below:

- Chocolatey installed from Chocolatey
- Python 3.9.x, 64 bit installed
- igeCore installed with 'python -m pip install igeCore'
- Git installed
- CMake 3.18.x installed ('choco install cmake –version=3.18.1')
- Visual Studio 19 with C++ Desktop components is required for Windows runtime.
- Java SDK 11, Android Studio and Android SDK are required for Android runntime.
- MinGW ('choco install mingw') and Emscripten ('choco install emscripten') are required for WebGL runtime.

Note: On Windows, igeCreator supports build for Windows, Android and WebGL platforms.

Note: Please remove Python 3.10 after installing emscripten, as support Python 3.10 is not yet ready with IGE.

MacOS Workstation

In order to work with IGE Engine on Windows machine, please make sure to install softwares below:

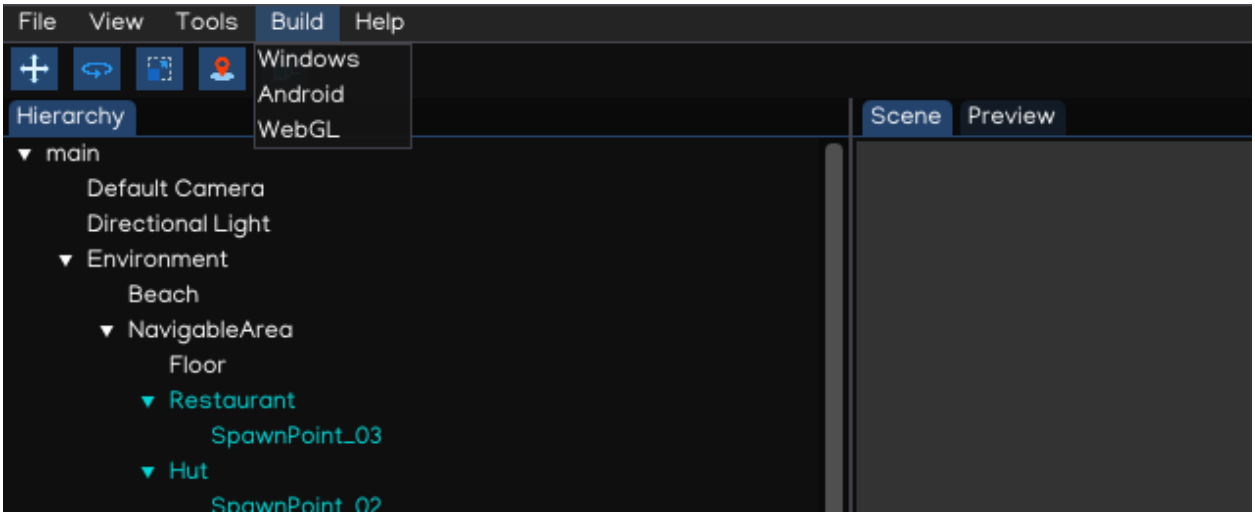
- Homebrew installed
- Python 3.9.x, 64 bit installed with 'brew install python3.9'
- igeCore installed with 'python3.9 -m pip install igeCore'
- Cocoa Pod installed ('sudo gem install cocoapods')
- XCode installed
- Git client installed
- Oracle Java SDK 11, Android Studio and Android SDK are required for Android runntime.
- Emscripten ('brew install emscripten') are required for WebGL runtime.

Note: The igeCreator runs on Intel-based MacOS computer only, Apple Silicon support is WIP.

Note: On macOS, igeCreator supports build for macOS, iOS, Android and WebGL platforms.

1.12.2 Build Menu

To start building for a specific platform, access the *Menu -> Build* as below:



1.12.3 Project Setting Panel

Generic Configuration

Property	Function
Name	Executable name
Label	Icon label
VersionName	Version came
VersionCode	Version code
BuldleID	iOS bundle ID, android package name
Orientation	Orientation: portrait / landscape
StartScene	Scene to start the game with.
Dependencies	List of modules used by the game.

Android Platform Settings

Property	Function
RomDir	Rom directory, default to 'mobile'
ConfigDir	Config directory, default to 'config/android'
Archs	Architecture, default to 'armeabi-v7a;arm64-v8a'
MinSdkVersion	Min Sdk Version
TargetSdkVersion	Target SDK Version
Permissions	List of required permissions
Features	List of using features

iOS Platform Settings

Property	Function
RomDir	Rom directory, default to 'mobile'
ConfigDir	Config directory, default to 'config/ios'
Archs	Architecture, default to 'arm64'
DeploymentTarget	Deployment target, default to '11.0'
DeviceFamily	Device family, default to '1,2' which mean iPhone and iPad
DevelpomentTeamId	Development team ID
CodeSignIdentity	Code sigining type: iPhone Distribution / iPhone Development
ProvisioningProfile	Provisioning profile, set to 'Automatic' for development build

1.13 Third-Person Shooter

Welcome to Indigames Game Engine tutorial series!

This tutorial will introduce how to work with IGE Engine to create a third-person shooter game.

Before starting, let make sure you have:

- **IGE Engine**: check *Installation* document if you haven't have it installed.
- **Tutorial Source Code**: checkout ige-tutorials, branch 01-basic-scene github repo.

1.13.1 1. About Scene

A scene is an abstract collection of game objects, representing a part of the game's world created by using the scene editor.

IGE implements a scene structure using a Scene Object and Component system.

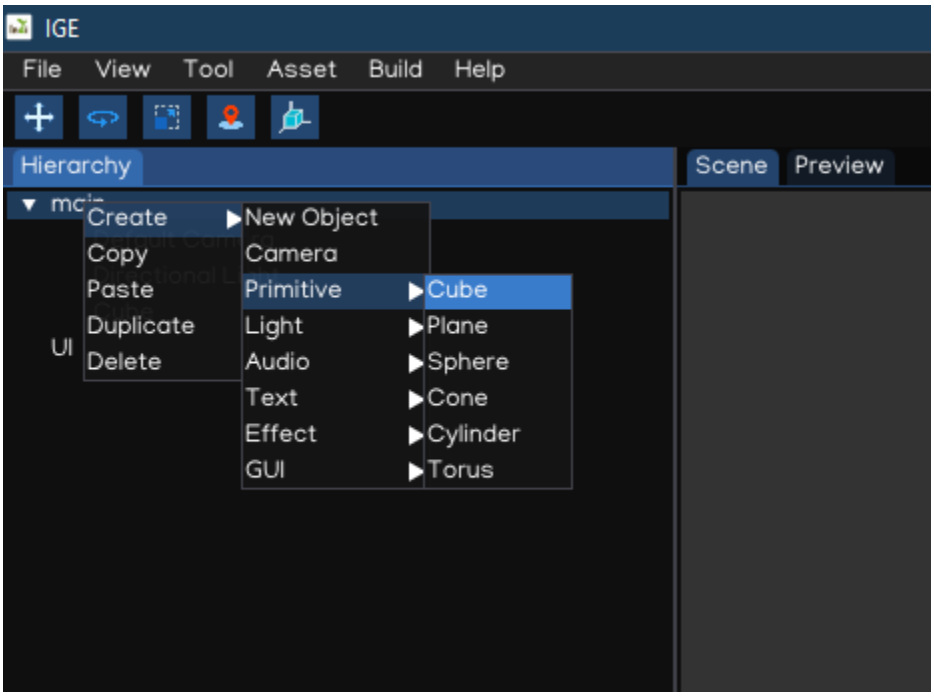
- The Scene Object manages the parent-child relationship of the Scene, and the spatial matrix transformation, so that all objects canbe managed and placed in the scene.
- The Component system allows Scene Object to have a variety of advanced features, such as Graphic components, Animation components, Light components, Audio components, and more.

The typical workflow of using Scene Object is to:

- Create a Scene Object
- Add Components
- Write Scripts that change the properties and behaviors of these Components

Create Object

To create a game object, right click on an item in the Hierarchy , select Create, then it will show `Object Creation Menu` with many types of object.



Alternative, drag the assets to the Scene View, it will also create object with the type based on the file extension.

Add Components

To add a component to a scene object, select it in the Scene view or Hierarchy, then in the Inspector select `Add Component`, it will show the `Add Component Menu`.

Creating scene object with `Object Creation Menu` or by dragging assets will add component related to the object types.

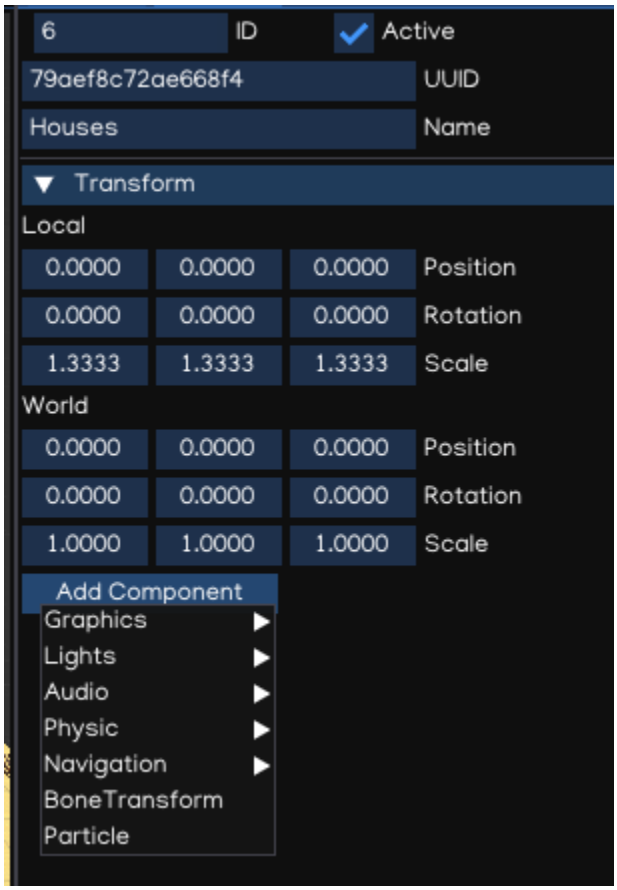
Scripting

Indigames Game Engine allow writing Python Script to control the scene object behavior. The Script canbe attached to an object using `Script` component, and canbe accesses using `getComponent(<class_name>)` from other scripts.

1.13.2 2. Scene Setup

Open The Scene

Open the project using igeCreator, you will see a screen similar to this:


6
ID
Active
79aef8c72ae668f4
UUID
Houses
Name
Transform
Local
0.0000
0.0000
0.0000
Position
0.0000
0.0000
0.0000
Rotation
1.3333
1.3333
1.3333
Scale
World
0.0000
0.0000
0.0000
Position
0.0000
0.0000
0.0000
Rotation
1.0000
1.0000
1.0000
Scale
Add Component
Graphics
Lights
Audio
Physic
Navigation
BoneTransform
Particle

Scene Navigation

Try to navigate the Scene using Scene View controls:

Action	Input
Rotate	[Mouse] Drag Right Button
Zoom	[Mouse] Scroll Middle Button
Move	[Mouse] Drag Middle Button
Focus	[Keyboard] Press *F* Key

Scene Management

Try adding new game object to make the environment more beautiful, by using `Object Creation Menu` and dragging assets from `figures` folder.

Also, try to modify the environment by adjust objects' position, rotation and scale values to change the environment layout as per your preferences.

Save the Scene using `Ctrl + S`, or `File -> Save Scene`.

1.13.3 3. Background Music

To play an audio clip, we need to use `AudioSource` component, either by dragging the audio file to scene to create new object with `AudioSource` attached, or just to add `AudioSource` component to an existing object. To make it simple, select *root* object, add `AudioSource` component, then drag the `audio/bgm.mp3` file to the inspector. The background music should be play once loaded, and should be looped as well. To save memory, it can also be streamed.

Let's add the background music to the `Environment` object, like as below:

Also, `AudioListener` is required to act as a listener in 3D space, it's usually added to the active camera. So, let's add `AudioListener` to the `Default Camera` object:

Save the Scene, then press *Play* button, the background music should be played and looped during the playing session.

1.13.4 4. Character Movement

Checkout ige-tutorials, branch 02-character-movement github repo.

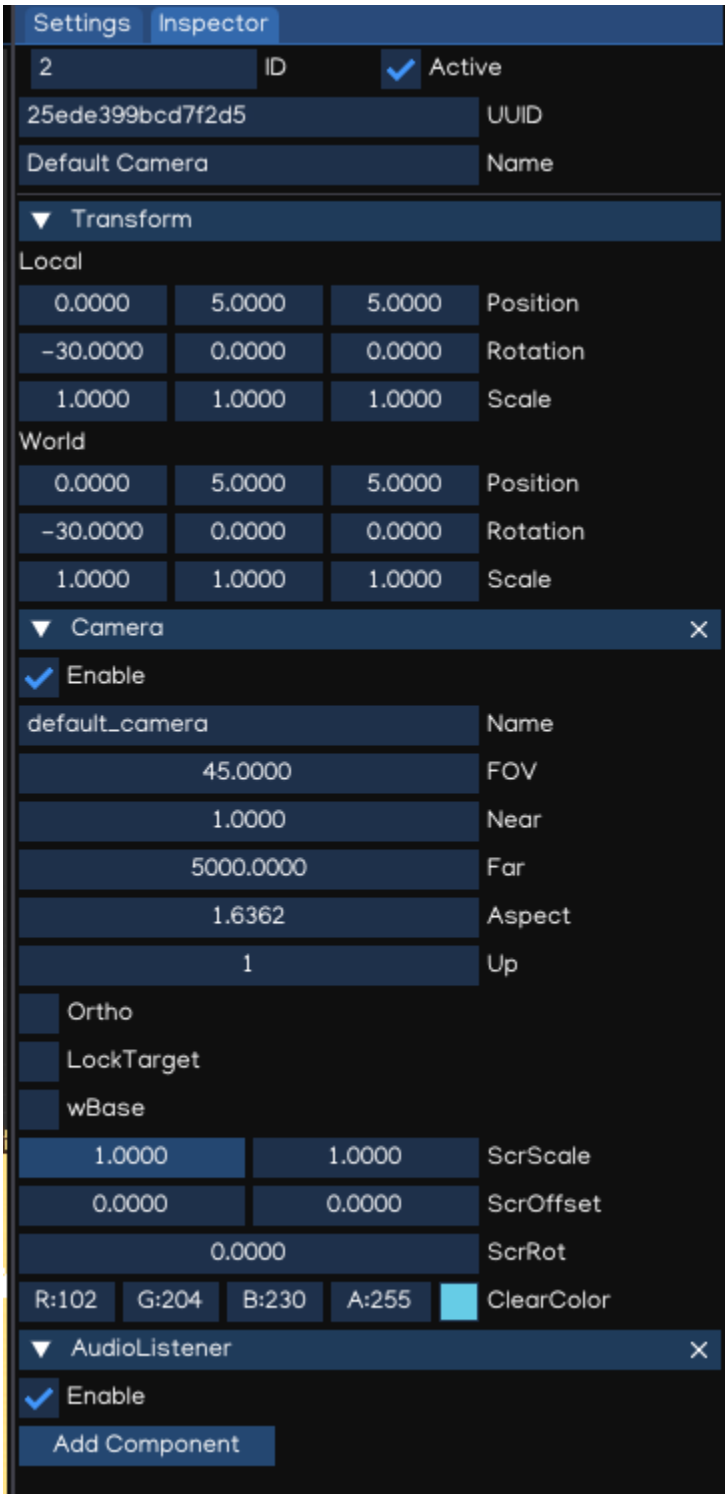
Add MC

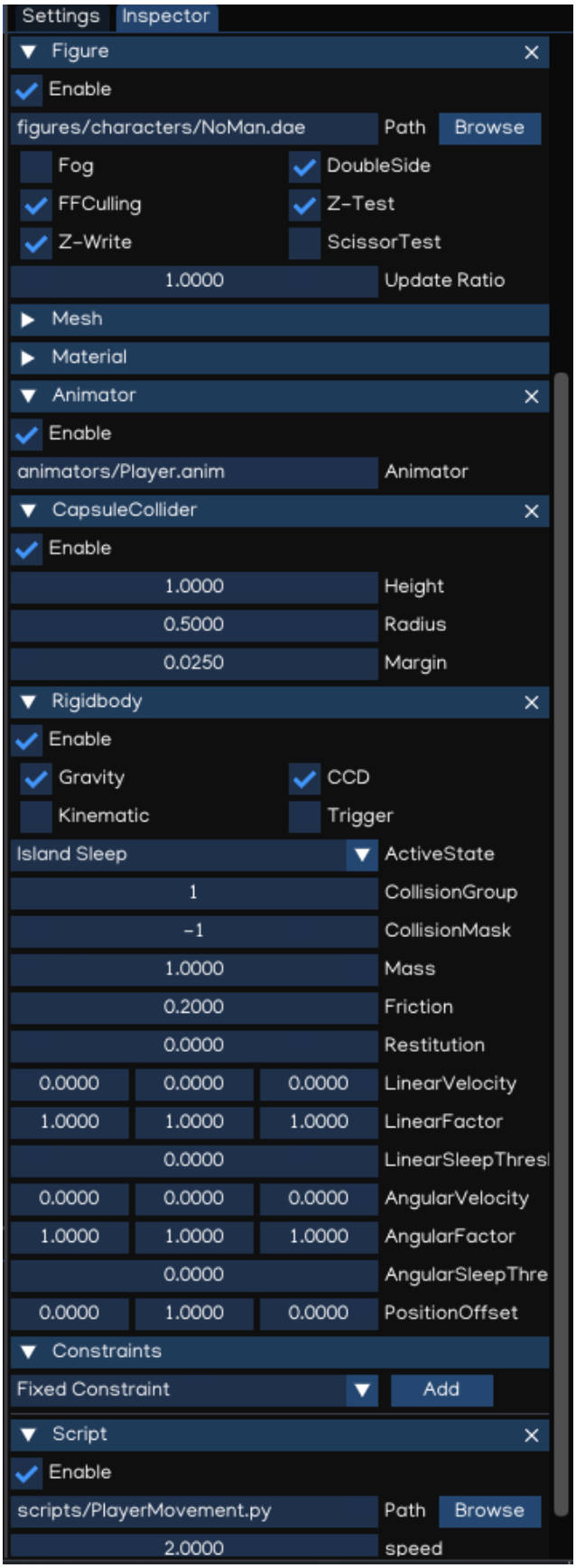
The MC prefab is located in `prefabs/MC.prefab` folder. Add the MC to the scene by dragging the prefab file in the Scene View.

In the Inspector, you can see the MC already have:

- **Figure**: using model from `figures/characters/NoMan.dae`
- **Animator**: using animator controller from `animators/Player.anim`
- **CapsuleCollider** and **Rigidbody**: Physic simulation
- **Script**: movement script located at `scripts/PlayerMovement`


Settings
Inspector
4
ID
Active
43a3d973cd808678
UUID
Environment
Name
Transform
Local
0.0000
0.0000
0.0000
Position
0.0000
0.0000
0.0000
Rotation
1.0000
1.0000
1.0000
Scale
World
0.0000
0.0000
0.0000
Position
0.0000
0.0000
0.0000
Rotation
1.0000
1.0000
1.0000
Scale
AudioSource
Enable
AutoPlay
Single
Stream
Loop
audio/bgm.mp3
Track
Browse
1.0000
Volume
0.0000
Pan
0.0000
Min Distance
10000.0000
Max Distance
0.0000
0.0000
0.0000
Velocity
LINEAR DISTANCE
Attenuation Model
0.5000
Attenuation Factor
1.0000
Doppler Factor
Add Component

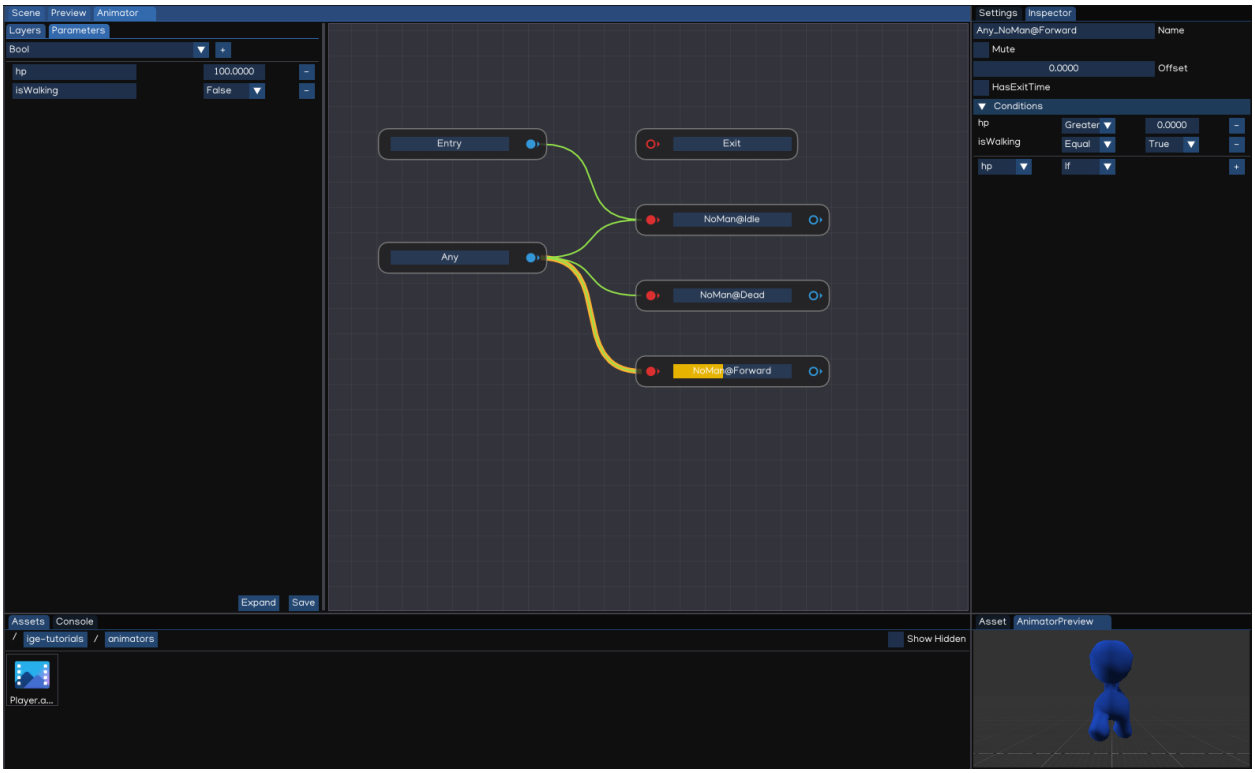

Settings
Inspector
2
ID
Active
25ede399bcd7f2d5
UUID
Default Camera
Name
Transform
Local
0.0000
5.0000
5.0000
Position
-30.0000
0.0000
0.0000
Rotation
1.0000
1.0000
1.0000
Scale
World
0.0000
5.0000
5.0000
Position
-30.0000
0.0000
0.0000
Rotation
1.0000
1.0000
1.0000
Scale
Camera
Enable
default_camera
Name
45.0000
FOV
1.0000
Near
5000.0000
Far
1.6362
Aspect
1
Up
Ortho
LockTarget
wBase
1.0000
1.0000
ScrScale
0.0000
0.0000
ScrOffset
0.0000
ScrRot
R:102
G:204
B:230
A:255
ClearColor
AudioListener
Enable
Add Component


Settings
Inspector
Figure
Enable
figures/characters/NoMan.dae
Path
Browse
Fog
DoubleSide
FFCulling
Z-Test
Z-Write
ScissorTest
1.0000
Update Ratio
Mesh
Material
Animator
Enable
animators/Player.anim
Animator
CapsuleCollider
Enable
1.0000
Height
0.5000
Radius
0.0250
Margin
Rigidbody
Enable
Gravity
CCD
Kinematic
Trigger
Island Sleep
ActiveState
1
CollisionGroup
-1
CollisionMask
1.0000
Mass
0.2000
Friction
0.0000
Restitution
0.0000 0.0000 0.0000
LinearVelocity
1.0000 1.0000 1.0000
LinearFactor
0.0000
LinearSleepThres
0.0000 0.0000 0.0000
AngularVelocity
1.0000 1.0000 1.0000
AngularFactor
0.0000
AngularSleepThre
0.0000 1.0000 0.0000
PositionOffset
Constraints
Fixed Constraint
Add
Script
Enable
scripts/PlayerMovement.py
Path
Browse
2.0000
speed

Character Animation

IGE Animation makes use of Animator Controller, which control the animation using State Machine defined in `.anim` file.

Open `animators/Player.anim` by double clicking the file icon in `AssetBrowser`, the Animator Editor appears like below:



Every animator controller implements internal state machine system, which consists at least `Entry`, `Exit` and `Any` states. The `Entry` state help to configure the initial state of the animation. The `Exit` state is to end animation. And the `Any` state is a helper state to simplify the state diagram.

The player has other three states: `Idle`, `Move`, `Dead`.

To decide what state to play next, the `Parameters` and `Conditions` can be used.

- `Parameters`: define global parameters and their values.
- `Conditions`: attached to each transition, with compare the parameters' values which predefined threshold.

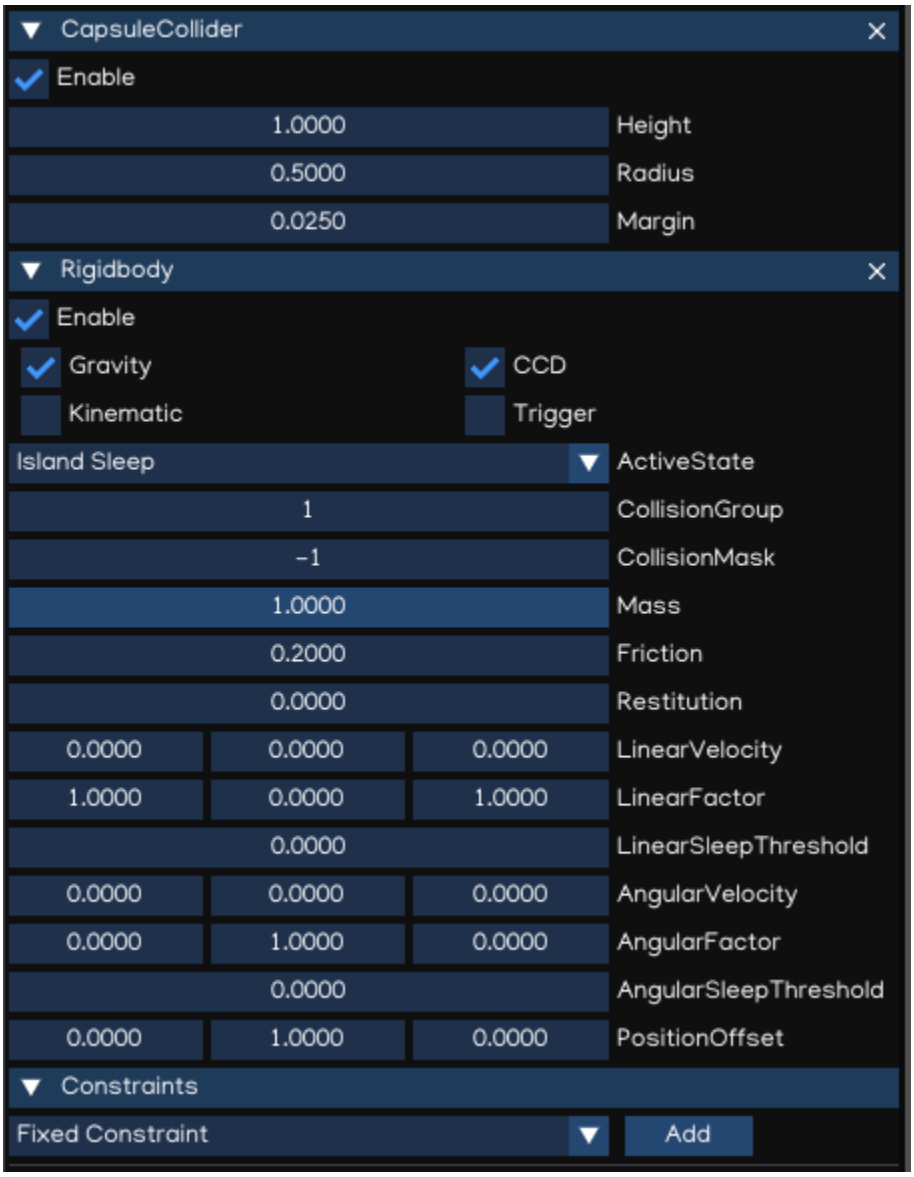
The animation transition happens when all conditions are meet, or `HasExitTime` checked and the `ExitTime` value reached.

The animation is controllable using Python Script, by setting the parameters' values at runtime.

Character Physic

In the Inspector, the character object includes a `Capsule` collider and a `Ridgidbody`. This is a dynamic object, thus `IsKinematic` is set to *false*.

Note: Notice that, the movement along *Y-Axis* is fixed, by setting the second parameter of `LinearFactor` to zero. Also, the rotation along *X-Axis* and *Z-Axis* is locked, by setting the first and the third parameters of `AngularFactor` to zero.



Character Movement Script

The `PlayerMovement.py` script is as below:

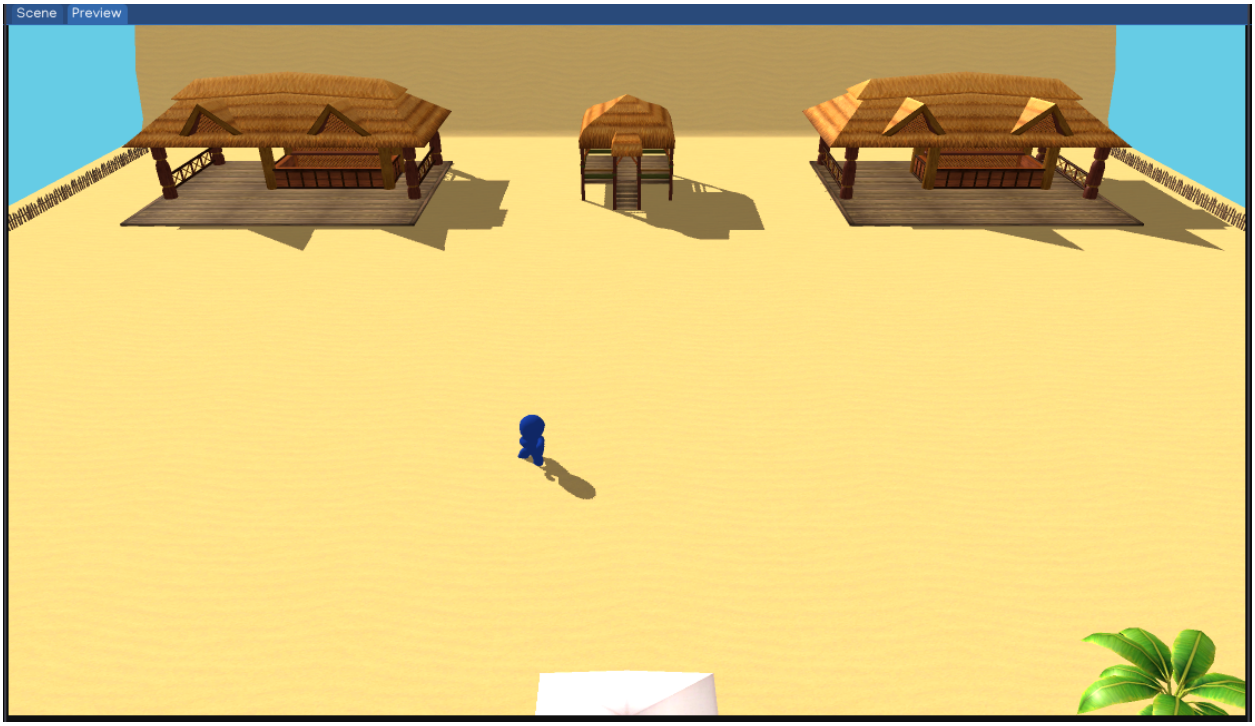
```
import igeVmath as vmath
from igeCore.input.keyboard import Keyboard, KeyCode
from igeScene import Script

class PlayerMovement(Script):
   def __init__(self, owner):
      super().__init__(owner)
      self.speed = 2.0
      self._transform = None
      self._rigidbody = None
      self._animator = None
      self._movement = vmath.vec3(0, 0, 0)
      self._isWalking = False

   def onStart(self):
      self._transform = self.owner.getComponent("Transform")
      self._rigidbody = self.owner.getComponent("Rigidbody")
      self._animator = self.owner.getComponent("Animator")
      self._movement = vmath.vec3(0, 0, 0)
      self._isWalking = False

   def onUpdate(self, dt):
      h, v = [0, 0]
      if Keyboard.isPressed(KeyCode.KEY_W) or Keyboard.isPressed(KeyCode.KEY_UP):
            v = -1.0
      if Keyboard.isPressed(KeyCode.KEY_S) or Keyboard.isPressed(KeyCode.KEY_DOWN):
            v = 1.0
      if Keyboard.isPressed(KeyCode.KEY_A) or Keyboard.isPressed(KeyCode.KEY_LEFT):
            h = -1.0
      if Keyboard.isPressed(KeyCode.KEY_D) or Keyboard.isPressed(KeyCode.KEY_RIGHT):
            h = 1.0
      if h != 0 or v != 0:
            self._movement = vmath.vec3(h, 0, v)
            self._movement.normalize()
            self._movement = self._movement * self.speed * dt
            newRotation = vmath.quat_look_rotation(self._movement, vmath.vec3(0.0, 1.0,␣
↪0.0))
            self._rigidbody.moveRotation(newRotation)
            self._rigidbody.movePosition(self._transform.position + self._movement)
            if not self._isWalking:
               self._isWalking = True
               self._animator.setValue("isWalking", self._isWalking)
      elif self._isWalking:
         self._isWalking = False
         self._animator.setValue("isWalking", self._isWalking)

   def onDestroy(self):
      self._transform = None
      self._rigidbody = None
```

(continues on next page)

(continued from previous page)

```
        self._animator = None
        self._playerHealth = None
        self._movement = None
```

Click `Play` button, then in the playing mode, the main character can be controlled by pressing arrow keys or WASD keys. The character also has collision with the houses and other objects in the scene.



1.13.5 5. Camera Setup

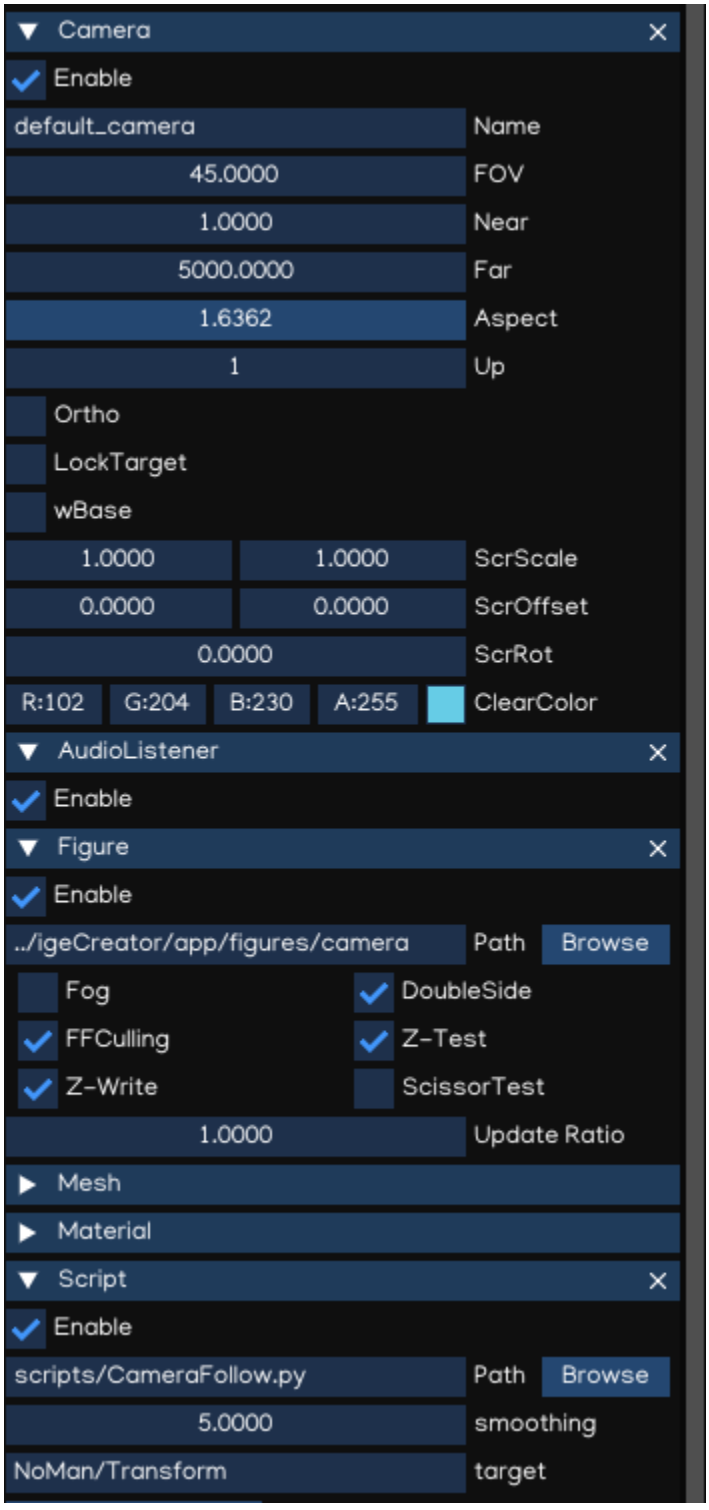
Checkout ige-tutorials, branch 03-camera-setup github repo.

Navigate to `Default Camera` object, add a `Script` component. Drag and drop `scripts/CameraFollow.py` from `AssetBrowser` to the newly created `Script`. Lastly, drag and drop the `NoMan` from `Hierarchy` to `target` property, then select `Transform`.

The `CameraFollow.py` script is as below:

```
from igeScene import Script
import igeVmath as vmath

class CameraFollow(Script):
    def __init__(self, owner):
        super().__init__(owner)
        self.target = None
        self.smoothing = 5.0
        self._offset = vmath.vec3()

    def onStart(self):
```

(continues on next page)


Camera
Enable
default_camera
Name
45.0000
FOV
1.0000
Near
5000.0000
Far
1.6362
Aspect
1
Up
Ortho
LockTarget
wBase
1.0000
1.0000
ScrScale
0.0000
0.0000
ScrOffset
0.0000
ScrRot
R:102
G:204
B:230
A:255
ClearColor
AudioListener
Enable
Figure
Enable
../igeCreator/app/figures/camera
Path
Browse
Fog
DoubleSide
FFCulling
Z-Test
Z-Write
ScissorTest
1.0000
Update Ratio
Mesh
Material
Script
Enable
scripts/CameraFollow.py
Path
Browse
5.0000
smoothing
NoMan/Transform
target

(continued from previous page)

```
        if self.target is None:
            self.target = self.owner.scene.findObjectByName("MC").getComponent("Transform
↪")
        if self.target is None:
            return
        self._offset = self.owner.transform.position - self.target.position

    def onUpdate(self, dt):
        targetCamPos = self.target.position + self._offset
        self.owner.transform.position = vmath.lerp(self.smoothing * dt, self.owner.
↪transform.position, targetCamPos)

    def onDestroy(self):
        self.target = None
        self._offset = None
```

Save the scene, and after press `Play`, the camera will follow the main character while moving around.

1.13.6 6. Add Enemy

Checkout ige-tutorials, branch 04-enemy-setup github repo.

Like the MC, the Enemy prefab is added at `prefabs/Enemy.prefab`. Create an enemy by drag and drop the prefab to the root node in the `Hierarchy`.

In the Inspector, the `Enemy` object contains:

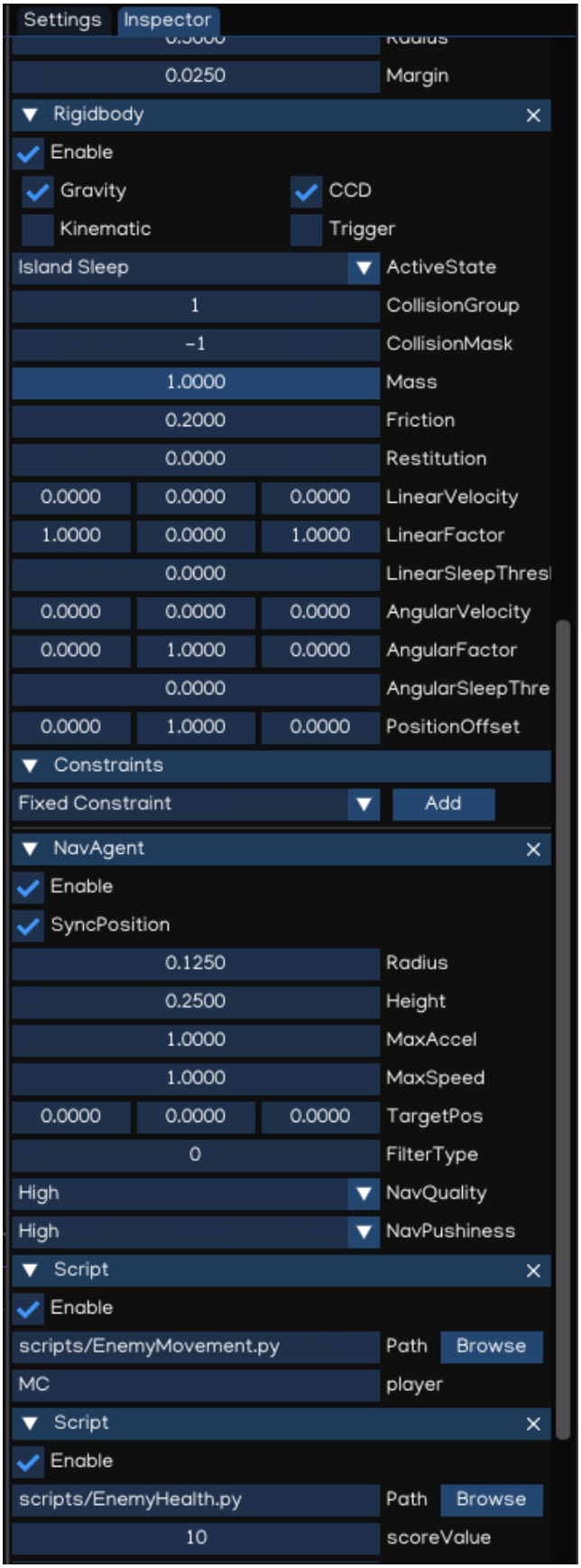
- **Figure**: similar to MC, but the `Diffuse Collor` changed to Red instead of Blue.
- **Animator**: same as MC
- **Rigidbody** and **Collider**: same as MC
- **NavAgent**: use NavAgent to find and navigate the object in the map
- **Script**: `EnemyMovement.py` and `EnemyHealth.py` control the movement and heal of the enemy.

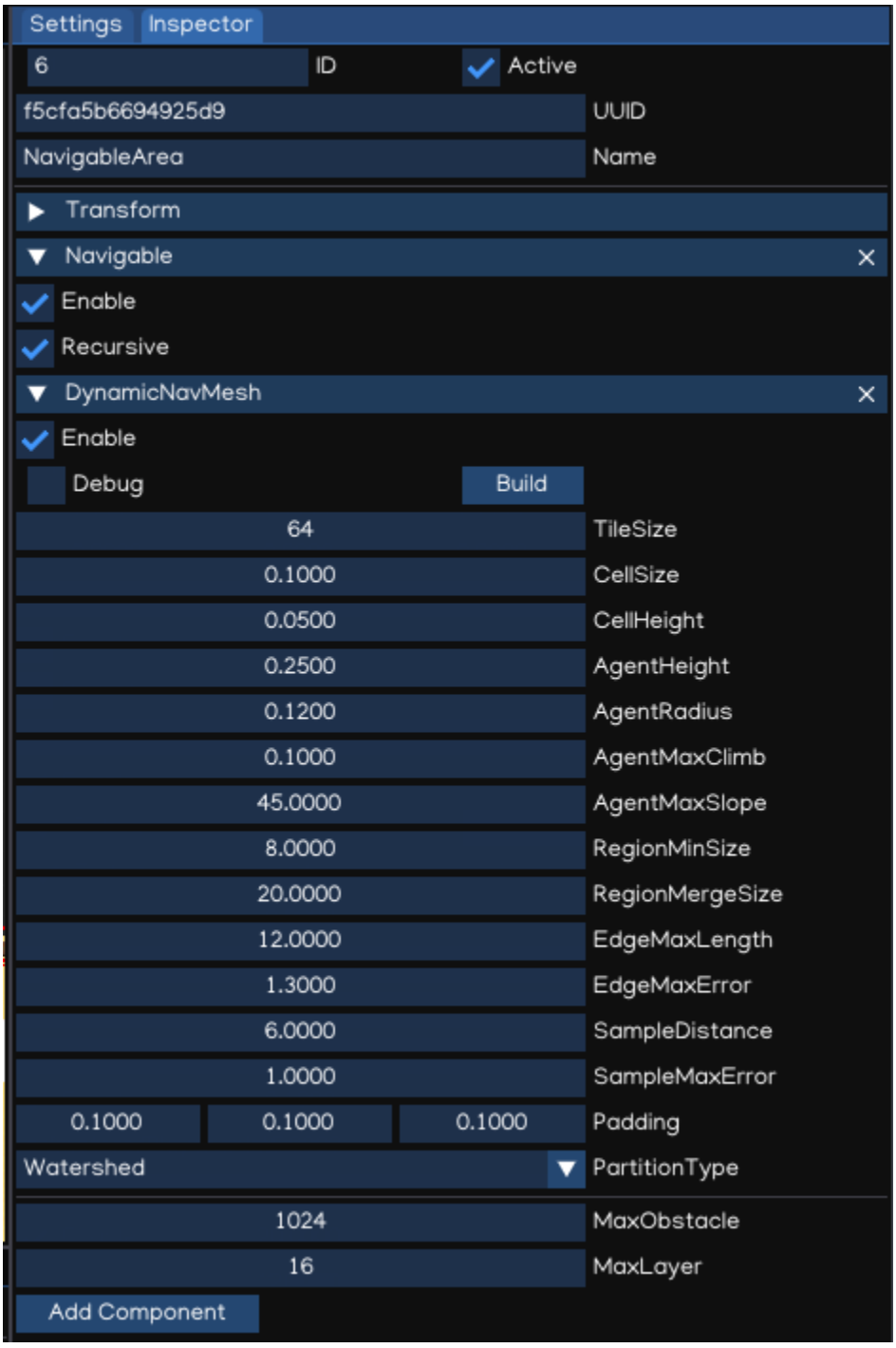
To enable NavAgent auto targeting, we also need to setup the NavMesh. The `DynamicNavMesh` component is added to `NavigableArea` object, along with `Navigable` component.

The `EnemyMovement.py` script is as below:

```
from igeScene import Script
import igeVmath as vmath

class EnemyMovement(Script):
   def __init__(self, owner):
      super().__init__(owner)
      self.player = None
      self._transform = None
      self._navAgent = None
      self._rigidbody = None
      self._playerTransform = None
      self._playerHealth = None
      self._enemyHealth = None
      self._animator = None
      self._isWalking = False

   def onStart(self):
      self._transform = self.owner.getComponent("Transform")
      self._rigidbody = self.owner.getComponent("Rigidbody")
      self._navAgent = self.owner.getComponent("NavAgent")
      self._enemyHealth = self.owner.getComponent("EnemyHealth")
      self._animator = self.owner.getComponent("Animator")
      if self.player is None:
            self.player = self.owner.scene.findObjectByName("MC")
      if self.player is not None:
            self._playerTransform = self.player.getComponent("Transform")
            self._playerHealth = self.player.getComponent("PlayerHealth")

   def onUpdate(self, dt):
      if self._enemyHealth.hp > 0.0 and self._playerHealth.hp > 0.0:
            self._navAgent.targetPosition = self._playerTransform.position
            movement = self._playerTransform.position - self._transform.position
            movement.normalize()
            newRotation = vmath.quat_look_rotation(movement, vmath.vec3(0.0, 1.0, 0.0))
```

(continues on next page)


Settings
Inspector
Radius
0.0250
Margin
Rigidbody
Enable
Gravity
CCD
Kinematic
Trigger
Island Sleep
ActiveState
1
CollisionGroup
-1
CollisionMask
1.0000
Mass
0.2000
Friction
0.0000
Restitution
0.0000 0.0000 0.0000
LinearVelocity
1.0000 0.0000 1.0000
LinearFactor
0.0000
LinearSleepThres
0.0000 0.0000 0.0000
AngularVelocity
0.0000 1.0000 0.0000
AngularFactor
0.0000
AngularSleepThre
0.0000 1.0000 0.0000
PositionOffset
Constraints
Fixed Constraint
Add
NavAgent
Enable
SyncPosition
0.1250
Radius
0.2500
Height
1.0000
MaxAccel
1.0000
MaxSpeed
0.0000 0.0000 0.0000
TargetPos
0
FilterType
High
NavQuality
High
NavPushiness
Script
Enable
scripts/EnemyMovement.py
Path
Browse
MC
player
Script
Enable
scripts/EnemyHealth.py
Path
Browse
10
scoreValue


Settings
Inspector
6
ID
Active
f5cfa5b6694925d9
UUID
NavigableArea
Name
Transform
Navigable
Enable
Recursive
DynamicNavMesh
Enable
Debug
Build
64
TileSize
0.1000
CellSize
0.0500
CellHeight
0.2500
AgentHeight
0.1200
AgentRadius
0.1000
AgentMaxClimb
45.0000
AgentMaxSlope
8.0000
RegionMinSize
20.0000
RegionMergeSize
12.0000
EdgeMaxLength
1.3000
EdgeMaxError
6.0000
SampleDistance
1.0000
SampleMaxError
0.1000
0.1000
0.1000
Padding
Watershed
PartitionType
1024
MaxObstacle
16
MaxLayer
Add Component

(continued from previous page)

```
            self._rigidbody.moveRotation(newRotation)
            if not self._isWalking:
               self._isWalking = True
               self._animator.setValue("isWalking", self._isWalking)
      elif self._navAgent.hasTarget():
            self._navAgent.resetTarget()
            self._isWalking = False
            self._animator.setValue("isWalking", self._isWalking)

   def onDestroy(self):
      self.player = None
      self._transform = None
      self._navAgent = None
      self._rigidbody = None
      self._playerTransform = None
      self._playerHealth = None
      self._enemyHealth = None
      self._animator = None
```

The EnemyHealth.py script is as below:

```
from igeScene import Script

class EnemyHealth(Script):
   def __init__(self, owner):
      super().__init__(owner)
      self.maxHp = 20.0
      self.hp = 20.0
      self.scoreValue = 10
      self.sinkSpeed = 0.5
      self.hurtSfx = None
      self.deadSfx = None
      self._transform = None
      self._animator = None
      self._navAgent = None
      self._audio = None
      self._rigidbody = None
      self._isDead = False
      self._timer = 0

   def onStart(self):
      self.hp = self.maxHp
      self._isDead = False
      self._transform = self.owner.getComponent("Transform")
      self._animator = self.owner.getComponent("Animator")
      self._navAgent = self.owner.getComponent("NavAgent")
      self._audio = self.owner.getComponent("AudioSource")
      self._rigidbody = self.owner.getComponent("Rigidbody")

   def onUpdate(self, dt):
      if self._isDead:
            self._timer += dt
```

(continues on next page)

(continued from previous page)

```
            if self._timer >= 1.0:
                self._transform.position += vmath.vec3(0, -1, 0) * self.sinkSpeed * dt
                if (self._transform.position.y < -5.0):
                    self.owner.scene.removeObject(self.owner)

    def takeDamage(self, amount):
        self.hp -= amount
        self._animator.setValue("hp", self.hp)
        if self.hp <= 0.0:
            self.dead()
        else:
            self._audio.path = self.hurtSfx
            self._audio.play()

    def dead(self):
        if not self._isDead:
            self._isDead = True
            self._timer = 0.0
            self._navAgent.enable = False
            self._rigidbody.isKinematic = True
            self._audio.path = self.deadSfx
            self._audio.play()

    def onDestroy(self):
        self.hurtSfx = None
        self.deadSfx = None
        self._transform = None
        self._animator = None
        self._navAgent = None
        self._audio = None
        self._rigidbody = None
```

Click `Play` button, the Enemy will keep running toward the MC while he is moving around the map.

1.13.7 7. GUI & HUD

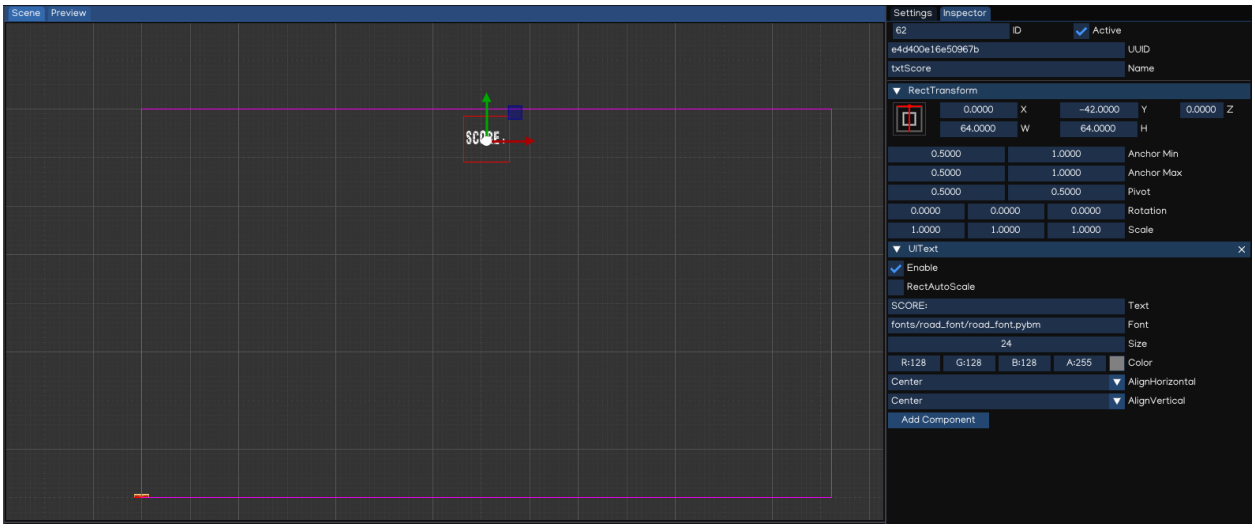
In this section, we will add a health indicator and display score in the screen.

Add Score

Add `SCORE:` label:

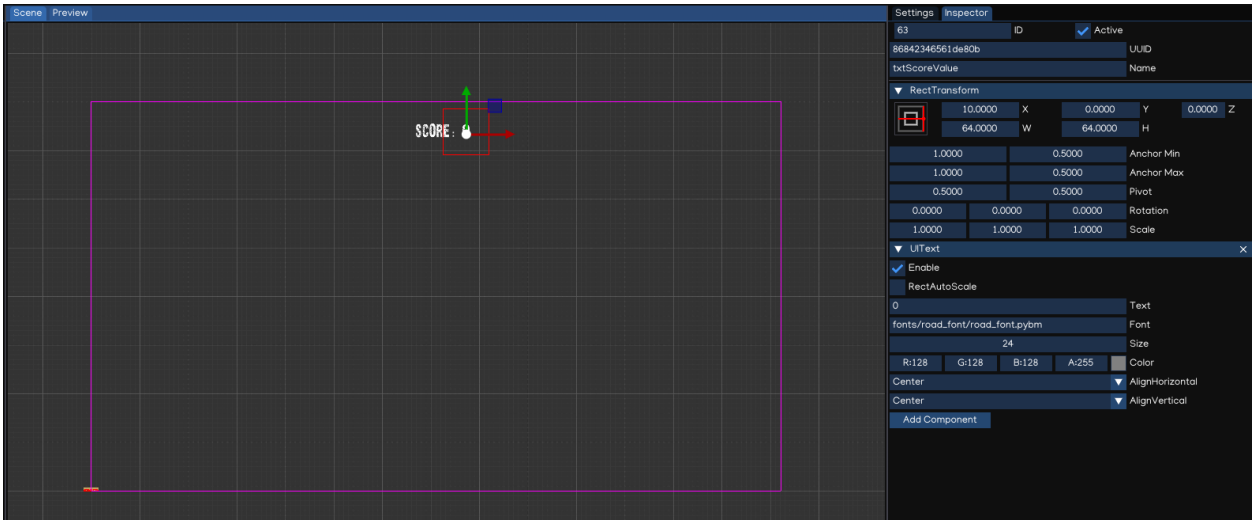
- Right-click the `UI` node in Hierarchy, select `Create -> GUI -> UIText`, it will create new object with UIText component
- Select the new object, rename it as `txtScore`.
- In the Inspector, change `Text` to `SCORE:`.
- Go to AssetBrowser, open `fonts/road_font`, then drag the `road_font.pybm` to the `Font` section in Inspector.
- Change the `Size` to `24`.
- Adjust the Anchor and Position like below:


Scene
Preview


Scene
Preview
SCORE:
Settings
Inspector
62
ID
Active
e4d400e16e50967b
UUID
txtScore
Name
RectTransform
0.0000
X
-42.0000
Y
0.0000
Z
64.0000
W
64.0000
H
0.5000
1.0000
Anchor Min
0.5000
1.0000
Anchor Max
0.5000
0.5000
Pivot
0.0000
0.0000
0.0000
Rotation
1.0000
1.0000
1.0000
Scale
UIText
Enable
RectAutoScale
SCORE:
Text
fonts/road_font/road_font.pybm
Font
24
Size
R:128
G:128
B:128
A:255
Color
Center
AlignHorizontal
Center
AlignVertical
Add Component

Add score value textfield:

- Select `txtScore`, right-click and select `Create -> GUI -> UIText` to create new textfield for score value.
- Rename the new object as `txtScoreValue`
- Adjust the Inspector elements like image below:

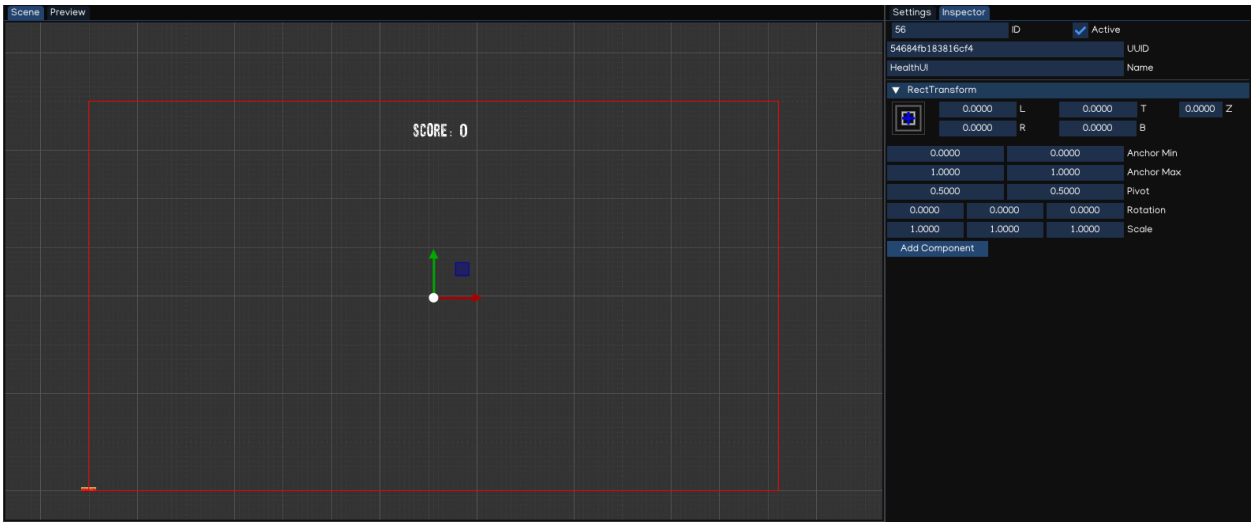


Now the screen should show `SCORE: 0` at the middle-top of the screen. We will show the real score in the next tutorial.

Add Health Bar

We can add `HealthUI` object to group the UI elements related to player health:

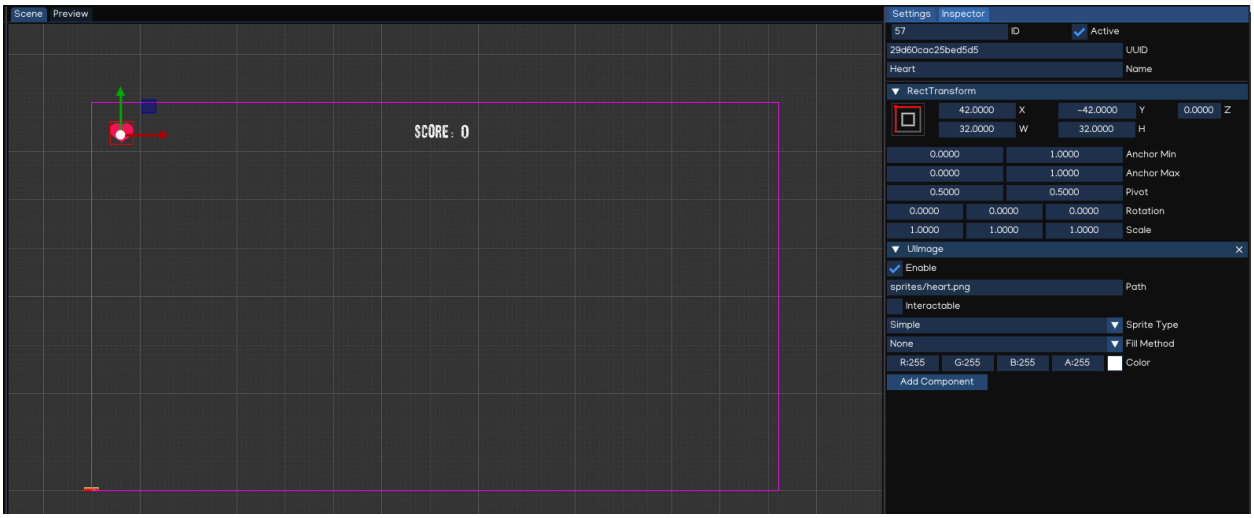
- Right-click the `Canvas` object, select `Create -> New Object`
- Name the new object as `HealthUI`.
- Adjust the RectTransform so that it will span the whole screen.



We add heart icon to indicate the player health:

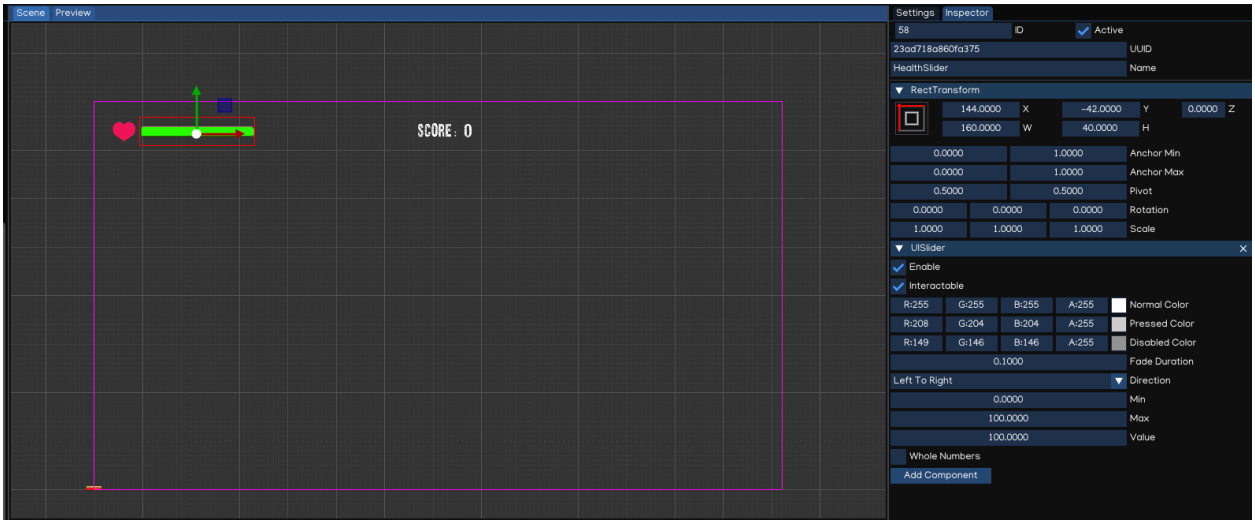
- Right-click the `HealthUI` object, select `Create -> GUI -> UIImage`

- Name the new object as `Heart`
- Drag `sprites/heart.png` from AssetBrowser to the Inspector
- Adjust the RectTransform to pin the icon to the top-left of the screen



We also add a Health Bar, by using UISlider component:

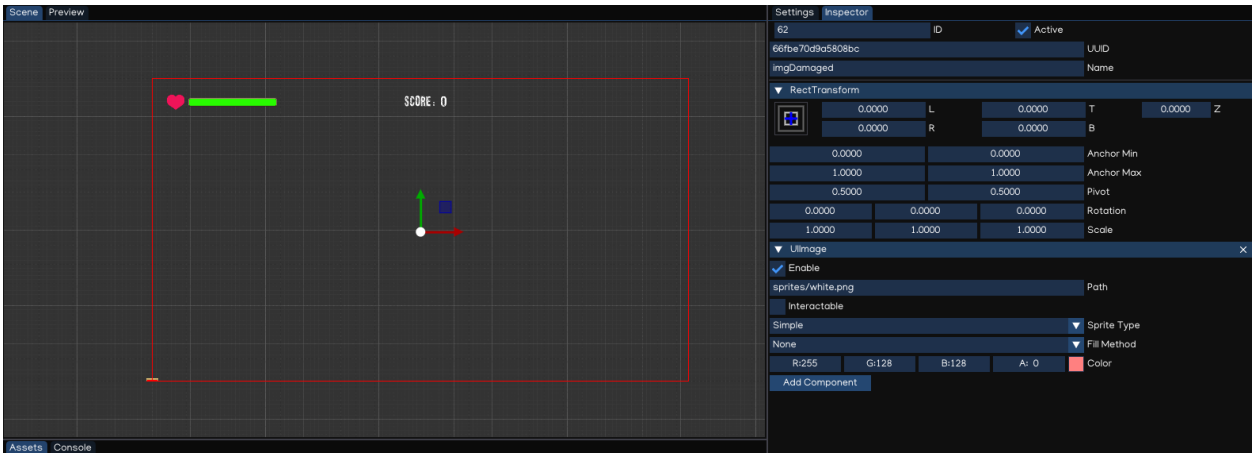
- Right-click the `HealthUI` object, select `Create -> GUI -> UISlider`
- Name the new object as `HealthSlider`
- The health slider is changed automatically, so we need to remove the handle, by delete `handleArea` child object.
- Change the background color to light-red color, by selecting `background`, then adjust color accordingly.
- Change the fill color to light-green, by selecting `fillArea -> fill` object, then adjust the color to light-green
- Select the `HealthSlider`, then adjust the RectTransform like below:



To provide graphical feedback when player is being attacked, we add a splash effect, by using UIImage component.

- Right-click the `HealthUI` object, select `Create -> GUI -> UIImage`
- Name the new object as `imgDamaged`

- Drag `sprites/white.png` from AssetBrowser to the Inspector
- Adjust color alpha to `0`
- Adjust the RectTransform to span the image full screen



This should be enough to display player health and score to the screen.

Checkout ige-tutorials, branch 05-gui-hud github repo.

1.13.8 8. MC Health

In this section, we will make the enemy attack, and adjust the player health on the UI accordingly.

Player Health

- In AssetBrowser, open `prefabs/MC.prefab` by double-clicking it.
- In AssetBrowser, create new script by navigating to `scripts`, then right-click, select `New Script`, enter `PlayerHealth` in the textfield.

The `PlayerHealth.py` is as below:

```
import igeVmath as vmath
from igeScene import Script

class PlayerHealth(Script):
   def __init__(self, owner):
      super().__init__(owner)
      self.maxHp = 100.0
      self.hp = 100.0
      self.healthSlider = None
      self.damageImage = None
      self.flashSpeed = 5.0
      self.deadSfx = None
      self.hurtSfx = None
      self._animator = None
      self._audio = None
      self._damaged = False

   def onStart(self):
      self._animator = self.owner.getComponent("Animator")
      self._audio = self.owner.getComponent("AudioSource")
      self.hp = self.maxHp

   def onUpdate(self, dt):
      if self._damaged:
            self.damageImage.color = vmath.vec4(1.0, 0.0, 0.0, 0.3)
      else:
            self.damageImage.color = vmath.lerp(self.flashSpeed * dt, self.damageImage.
↪color, vmath.vec4(1.0, 0.0, 0.0, 0.0))
      self._damaged = False

   def takeDamage(self, amount):
      self._damaged = True
      self.hp -= amount
      self._animator.setValue("hp", self.hp)
      self.healthSlider.value = self.hp
      if self.hp <= 0:
            self._audio.path = self.deadSfx
            self._audio.play()
            self.owner.getComponent("PlayerMovement").enable = False
            self.owner.getComponent("PlayerShoot").enable = False
            self.owner.getComponent("PlayerHealth").enable = False
```

(continues on next page)

(continued from previous page)

```
        else:
            self._audio.path = self.hurtSfx
            self._audio.play()

    def onDestroy(self):
        self.healthSlider = None
        self.damageImage = None
        self.deadSfx = None
        self.hurtSfx = None
        self._animator = None
        self._audio = None
```

- Select `MC` object, create new `Script` component, drag `scripts/PlayerHealth.py` to the path.
- Drag `HealthSlider` to the Inspector, in `healthSlider` textfield, select `UISlider`
- Drag `imgDamaged` to the Inspector, in `damageImage` textfield, select `UIImage`
- Drag `audio/player_hurt.wav` and `audio/player_death.wav` audio to the inspector in `hurtSfx` and `deadSfx` textfields.
- Save the prefab, select reload prefab when asked.

Enemy Attack

- In AssetBrowser, open `prefabs/Enemy.prefab` by double-clicking it.
- In AssetBrowser, create new script by navigating to `scripts`, then right-click, select `New Script`, enter `EnemyAttack` in the textfield.
- Select `Enemy` object, create new `Script` component, drag `scripts/EnemyAttack.py` to the path.
- Save the prefab, select reload prefab when asked.

The `EnemyAttack.py` is as below:

```
from igeScene import Script

class EnemyAttack(Script):
    def __init__(self, owner):
        super().__init__(owner)
        self.timeBetweenAttack = 1.0
        self.attackDamage = 10
        self._animator = None
        self._player = None
        self._playerHealth = None
        self._enemyHealth = None
        self._playerInRange = False
        self._timer = 0.0

    def onStart(self):
        self._player = self.owner.scene.findObjectByName("MC")
        if self._player is not None:
            self._playerHealth = self._player.getComponent("PlayerHealth")
        self._enemyHealth = self.owner.getComponent("EnemyHealth")
```

(continues on next page)

(continued from previous page)

```
        self._animator = self.owner.getComponent("Animator")

    def onTriggerStart(self, other):
        if other == self._player:
            self._playerInRange = True

    def onTriggerStop(self, other):
        if other == self._player:
            self._playerInRange = False

    def onUpdate(self, dt):
        self._timer += dt
        if self._timer >= self.timeBetweenAttack and self._playerInRange and self._
↪enemyHealth.hp > 0:
            self.attack()

    def attack(self):
        self._timer = 0.0
        if self._playerHealth.hp > 0:
            self._playerHealth.takeDamage(self.attackDamage)

    def onDestroy(self):
        self._animator = None
        self._player = None
        self._playerHealth = None
        self._enemyHealth = None
```

Save the scene, press Play button, now if player is near to the enemy, he will be attacked and his health will be updated in HUD.

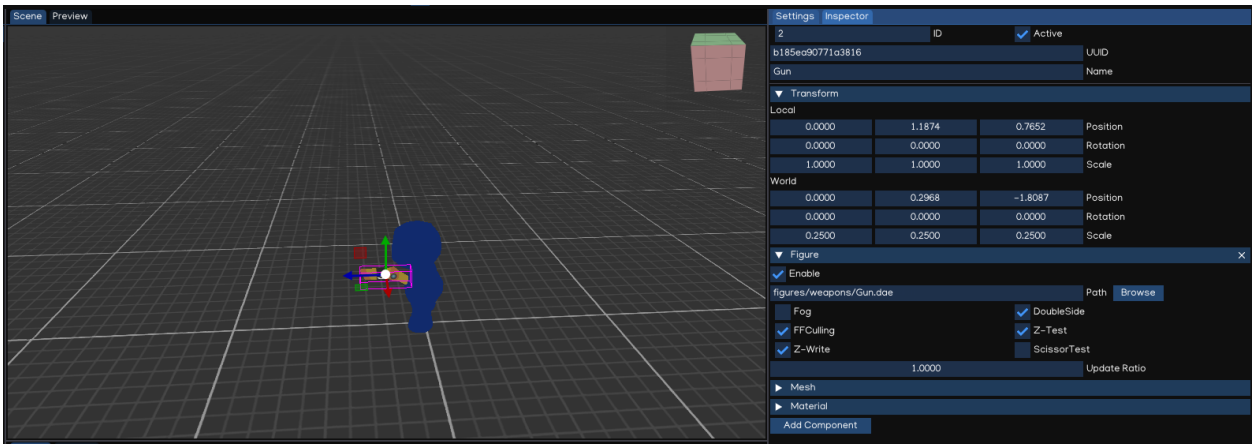
Checkout ige-tutorials, branch 06-player-health github repo.

1.13.9 9. MC Shooting

In this section, we will equip the MC with a gun and allow him to shoot enemy.

Add Gun to MC

- In AssetBrowser, open `prefabs/MC.prefab` by double-clicking it.
- Select `MC` object, right-click, select `New Object`, rename it to `Gun`.
- Select `Gun`, add `Figure` component, drag `figures/weapons/Gun.dae` to `Path`.
- Adjust `Transform` component as below:



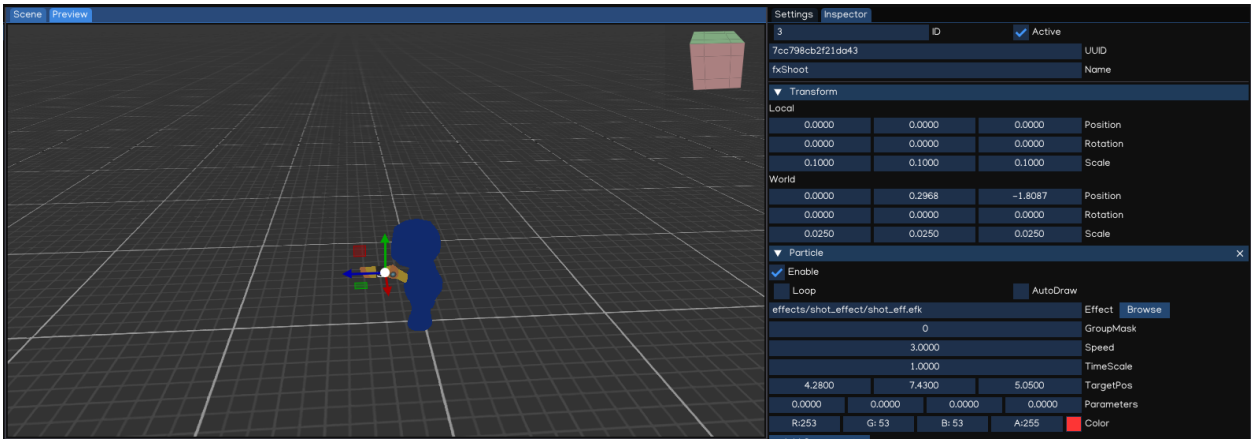
- Save the prefab.

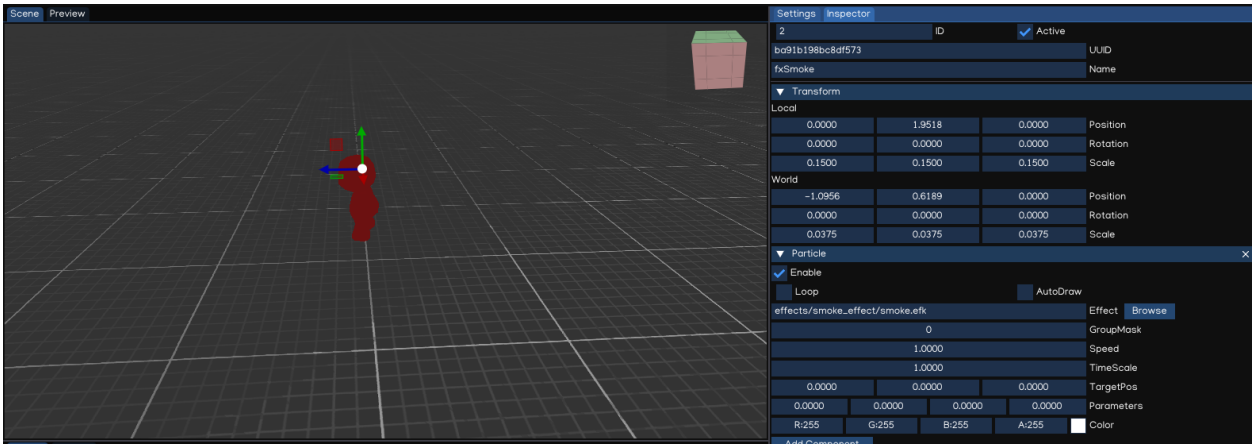
Add Fire Particle

- In AssetBrowser, open `prefabs/MC.prefab` by double-clicking it.
- Select `Gun`, right-click, select `New Object`, rename it to `fxShoot`.
- Select `fxShoot`, create `Particle` component, drag `effects/shot_effect/shot_eff.efk` to `Effect`.
- Adjust `Transform` component as below:
- Save the prefab.

Add Smoke Particle

- In AssetBrowser, open `prefabs/Enemy.prefab` by double-clicking it.
- Select `Enemy`, right-click, select `New Object`, rename it to `fxSmoke`.
- Select `fxShoot`, create `Particle` component, drag `effects/smoke_effect/smoke.efk` to `Effect`.
- Adjust `Transform` component as below:
- Save the prefab.


Scene
Preview
Settings
Inspector
3
ID
Active
7cc798cb2f21da43
UUID
fxShoot
Name
Transform
Local
0.0000
0.0000
0.0000
Position
0.0000
0.0000
0.0000
Rotation
0.1000
0.1000
0.1000
Scale
World
0.0000
0.2968
-1.8087
Position
0.0000
0.0000
0.0000
Rotation
0.0250
0.0250
0.0250
Scale
Particle
Enable
Loop
AutoDraw
effects/shot_effect/shot_eff.efk
Effect
Browse
0
GroupMask
3.0000
Speed
1.0000
TimeScale
4.2800
7.4300
5.0500
TargetPos
0.0000
0.0000
0.0000
0.0000
Parameters
R:253
G: 53
B: 53
A:255
Color


Scene
Preview
Settings
Inspector
2
ID
Active
ba91b198bc8df573
UUID
fxSmoke
Name
Transform
Local
0.0000
1.9518
0.0000
Position
0.0000
0.0000
0.0000
Rotation
0.1500
0.1500
0.1500
Scale
World
-1.0956
0.6189
0.0000
Position
0.0000
0.0000
0.0000
Rotation
0.0375
0.0375
0.0375
Scale
Particle
Enable
Loop
AutoDraw
effects/smoke_effect/smoke.efk
Effect
Browse
0
GroupMask
1.0000
Speed
1.0000
TimeScale
0.0000
0.0000
0.0000
TargetPos
0.0000
0.0000
0.0000
0.0000
Parameters
R:255
G:255
B:255
A:255
Color

Player Shooting

- In AssetBrowser, open `prefabs/MC.prefab` by double-clicking it.
- In AssetBrowser, create new script by navigating to `scripts`, then right-click, select `New Script`, enter `PlayerShoot` in the textfield.

The `PlayerHealth.py` is as below:

```
import igeVmath as vmath
from igeCore.input.keyboard import Keyboard, KeyCode
from igeScene import Script

class PlayerShoot(Script):
   def __init__(self, owner):
      super().__init__(owner)
      self.attackDamage = 20.0
      self.attackRange = 100.0
      self.timeBetweenAttack = 0.15
      self.shootSfx = None
      self.shootFx = None
      self._transform = None
      self._audio = None
      self._physic = None
      self._playerHealth = None
      self._timer = 0.0

   def onStart(self):
      self._transform = self.owner.getComponent("Transform")
      self._audio = self.owner.getComponent("AudioSource")
      self._physic = self.owner.scene.root.getComponent("PhysicManager")
      self._playerHealth = self.owner.getComponent("PlayerHealth")

   def onUpdate(self, dt):
      self._timer += dt
      if self._playerHealth.hp > 0 and Keyboard.isPressed(KeyCode.KEY_SPACE):
            self.shoot()

   def shoot(self):
      if self._timer < self.timeBetweenAttack:
            return
      self._timer = 0.0
      self._audio.path = self.shootSfx
      self._audio.play()
      self.shootFx.play()

      hit = self._physic.rayTestClosest(self._transform.position, self._transform.
↪forward * self.attackRange)
      if hit is not None:
            hitObject = hit["hitObject"]
            hitPosition = hit["hitPosition"]
            hitPosition.y += 0.3
            enemyHealth = hitObject.getComponent("EnemyHealth")
            if enemyHealth is not None and enemyHealth.hp > 0.0:
```

(continues on next page)

(continued from previous page)

```
                enemyHealth.takeDamage(self.attackDamage)
                smokeFx = hitObject.findChildByName("fxSmoke")
                if smokeFx is not None:
                    smokeFx.getComponent("Transform").position = hitPosition
                    smokeFx.getComponent("Particle").play()

    def onDestroy(self):
        self.shootSfx = None
        self.shootFx = None
        self._transform = None
        self._audio = None
        self._physic = None
        self._playerHealth = None
```

- Select `MC` object, add `Script` component, drag `scripts/PlayerShoot.py` to `Path`.
- Drag `fxShoot` to the Inspector, in the `shootFx` textfield
- Drag `audio/player_shoot.wav` to the `shootSfx` in the inspector.
- Save the prefab.

Update Score

We need to add `ScoreManager` script to the root object to manage game score:

- In AssetBrowser, navigate to `scripts`, create new script called `ScoreManager.py`.

The `ScoreManager.py` is as simple as below:

```
from igeScene import Script

class ScoreManager(Script):
    def __init__(self, owner):
        super().__init__(owner)
        self.scoreTxt = None
        self._score = 0

    def onStart(self):
        self._score = 0

    def score(self, value):
        self._score += value
        if self.scoreTxt is not None:
            self.scoreTxt.text = str(self._score)

    def onDestroy(self):
        self.scoreTxt = None
```

- Select `main` object, attach `ScoreManager.py` to it.
- Drag `txtScoreValue` from the UI to `scoreTxt` in the Inspector.
- Save the scene.

To add score, update `EnemyHealth.py` as below:

```
from igeScene import Script
import igeVmath as vmath

class EnemyHealth(Script):
   def __init__(self, owner):
      super().__init__(owner)
      self.maxHp = 20.0
      self.hp = 20.0
      self.scoreValue = 10
      self.sinkSpeed = 0.5
      self.hurtSfx = None
      self.deadSfx = None
      self._transform = None
      self._animator = None
      self._navAgent = None
      self._audio = None
      self._rigidbody = None
      self._scoreManager = None
      self._isDead = False
      self._timer = 0

   def onStart(self):
      self.hp = self.maxHp
      self._isDead = False
      self._transform = self.owner.getComponent("Transform")
      self._animator = self.owner.getComponent("Animator")
      self._navAgent = self.owner.getComponent("NavAgent")
      self._audio = self.owner.getComponent("AudioSource")
      self._rigidbody = self.owner.getComponent("Rigidbody")
      self._scoreManager = self.owner.scene.root.getComponent("ScoreManager")

   def onUpdate(self, dt):
      if self._isDead:
            self._timer += dt
            if self._timer >= 1.0:
               self._transform.position += vmath.vec3(0, -1, 0) * self.sinkSpeed * dt
               if (self._transform.position.y < -5.0):
                  self.owner.scene.removeObject(self.owner)

   def takeDamage(self, amount):
      self.hp -= amount
      self._animator.setValue("hp", self.hp)
      if self.hp <= 0.0:
            self.dead()
      else:
            self._audio.path = self.hurtSfx
            self._audio.play()

   def dead(self):
      if not self._isDead:
            self._isDead = True
            self._timer = 0.0
            self._navAgent.enable = False
```

(continues on next page)

(continued from previous page)

```
            self._rigidbody.isKinematic = True
            self._audio.path = self.deadSfx
            self._audio.play()
            self._scoreManager.score(self.scoreValue)

    def onDestroy(self):
        self.hurtSfx = None
        self.deadSfx = None
        self._transform = None
        self._animator = None
        self._navAgent = None
        self._audio = None
        self._rigidbody = None
        self._scoreManager = None
        self._timer = None
```

Press `Play` button, the MC now can shoot enemy by pressing SPACE. Once enemy dead, the score will be added and updated in the UI.

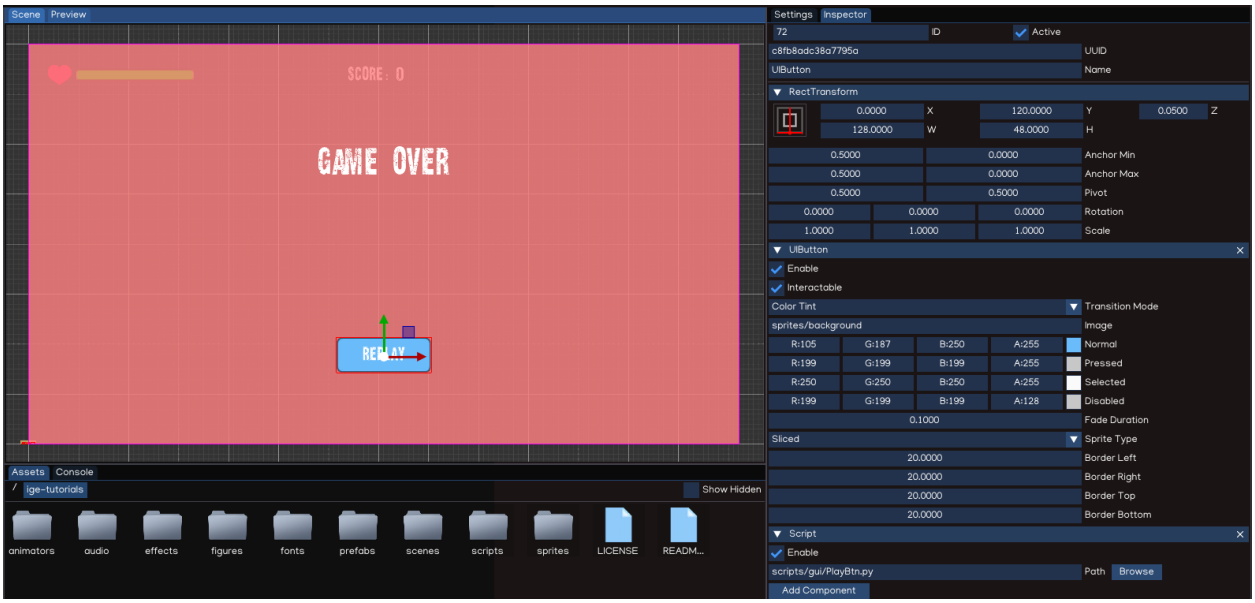
Checkout ige-tutorials, branch 07-player-shooting github repo.

1.13.10 10. Game Over

In this section, we will spawn enemy around the map, and calculate condition to make the game over, as well as provide ability to replay the game.

Game Over UI

The Game Over UI is as simple as below:



We display a layer with transparent red color, on top of that is `Game Over` text, and a *Replay* button to allow player to replay. In the AssetBrowser, add new script called `ReplayBtn.py` in `scripts/gui` folder, then attach the script to the `Replay` button.

Spawning Enemy

We add some spawning point in the map, for examples at the `Restaurant` and in the `Hut` object. We mark the point by adding dummy objects named `SpawnPoint_xx`.

Next, we create `EnemyManager` script, and attach it to the root node of the scene.

The `EnemyManager.py` is as below:

```
from igeScene import Script
import random

class EnemyManager(Script):
    def __init__(self, owner):
        super().__init__(owner)
        self.player = None
        self.enemyPrefab = None
        self.spawnTime = 3.0
        self.spawnPoint = None
        self.spawnPoint2 = None
```

(continues on next page)

(continued from previous page)

```
        self.spawnPoint3 = None
        self._playerHealth = None
        self._spawnTimer = 0.0
        self._spawnPoints = None
        self._enemyId = 0

    def onStart(self):
        self._enemyId = 0
        if self.player is None:
            self.player = self.owner.scene.findObjectByName("MC")
            if self.player is None:
                return
        self._playerHealth = self.player.getComponent("PlayerHealth")
        self._spawnPoints = []
        if self.spawnPoint is not None:
            self._spawnPoints.append(self.spawnPoint)
        if self.spawnPoint2 is not None:
            self._spawnPoints.append(self.spawnPoint2)
        if self.spawnPoint3 is not None:
            self._spawnPoints.append(self.spawnPoint3)

    def onUpdate(self, dt):
        self._spawnTimer += dt
        if self._spawnTimer >= self.spawnTime:
            self.spawn()

    def spawn(self):
        if self._playerHealth.hp <= 0:
            return
        spawnIndex = random.randrange(0, len(self._spawnPoints))
        self.owner.scene.loadPrefab(self.enemyPrefab, f"Enemy_{self._enemyId}", self.owner.
↪scene.root, self._spawnPoints[spawnIndex].position)
        self._enemyId += 1
        self._spawnTimer = 0.0

    def onDestroy(self):
        self.player = None
        self.enemyPrefab = None
        self.spawnPoint = None
        self.spawnPoint2 = None
        self.spawnPoint3 = None
        self._playerHealth = None
        self._spawnPoints = None
```

After attaching the script:

- Drag `MC` to `player` textbox
- Drag `prefabs/Enemy.prefab` from AssetBrowser to `enemyPrefab` textbox
- Drag `SpawnPoint_xx` to the `spawnPointxx` textbox
- Save the scene.

Game Over Script

Create new script named `GameManager.py` and attach to the root object.

The content of `GameManager.py` is as below:

```
from igeScene import Script, SceneManager

class GameManager(Script):
   def __init__(self, owner):
      super().__init__(owner)
      self._gameOverUI = None

   def onStart(self):
      self._gameOverUI = self.owner.scene.findObjectByName("GameOverUI")
      self._gameOverUI.active = False

   def play(self):
      SceneManager.getInstance().reloadScene()

   def gameOver(self):
      self._gameOverUI.active = True

   def onDestroy(self):
      self._gameOverUI = None
```

When MC's health fall below zero, the `Game Over` screen should appear. Edit `PlayerHealth.py` as below:

```
from igeScene import Script, SceneManager
import igeVmath as vmath
from igeScene import Script

class PlayerHealth(Script):
   def __init__(self, owner):
      super().__init__(owner)
      self.maxHp = 100.0
      self.hp = 100.0
      self.healthSlider = None
      self.damageImage = None
      self.flashSpeed = 5.0
      self.deadSfx = None
      self.hurtSfx = None
      self._animator = None
      self._audio = None
      self._damaged = False

   def onStart(self):
      self._animator = self.owner.getComponent("Animator")
      self._audio = self.owner.getComponent("AudioSource")
      self.hp = self.maxHp

   def onUpdate(self, dt):
      if self._damaged:
            self.damageImage.color = vmath.vec4(1.0, 0.0, 0.0, 0.3)
```

(continues on next page)

(continued from previous page)

```
        else:
            self.damageImage.color = vmath.lerp(self.flashSpeed * dt, self.damageImage.
↪color, vmath.vec4(1.0, 0.0, 0.0, 0.0))
        self._damaged = False

    def takeDamage(self, amount):
        self._damaged = True
        self.hp -= amount
        self._animator.setValue("hp", self.hp)
        self.healthSlider.value = self.hp
        if self.hp <= 0:
            self._audio.path = self.deadSfx
            self._audio.play()
            self.owner.getComponent("PlayerMovement").enable = False
            self.owner.getComponent("PlayerShoot").enable = False
            self.owner.getComponent("PlayerHealth").enable = False
            self.owner.scene.root.getComponent("GameManager").gameOver()
        else:
            self._audio.path = self.hurtSfx
            self._audio.play()

    def onDestroy(self):
        self.healthSlider = None
        self.damageImage = None
        self.deadSfx = None
        self.hurtSfx = None
        self._animator = None
        self._audio = None
```

Replay The Game

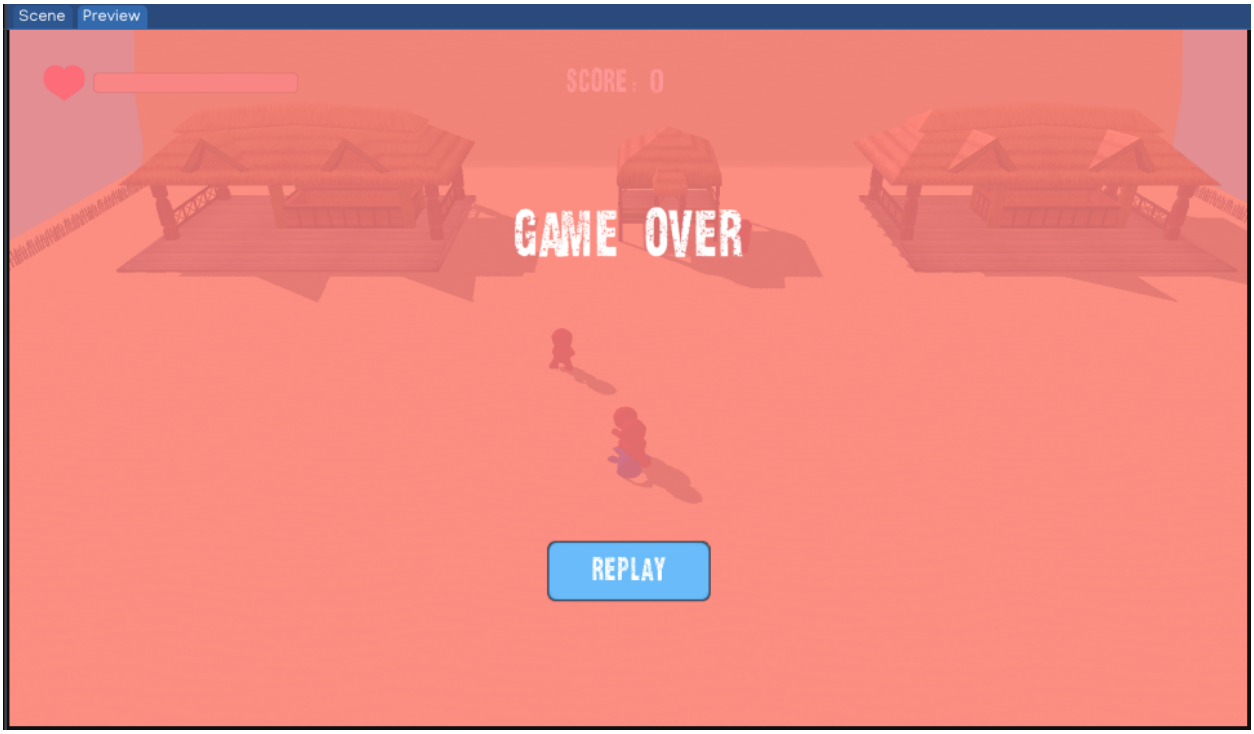
For this tutorial, replay the game is as simple as reload the scene from the beginning.

Edit `ReplayBtn.py` as below:

```
from igeScene import Script

class ReplayBtn(Script):
    def __init__(self, owner):
        super().__init__(owner)

    def onUpdate(self, dt):
        pass

    def onClick(self):
        self.owner.scene.root.getComponent("GameManager").play()
```

Play the game now, when being attacked by enemy, if the HC's health fall below zero, the Game Over screen will be shown, and user will be able to replay the game by press `Replay` button.

Checkout ige-tutorials, branch 08-game-over github repo.



1.13.11 11. Mobile Control

On mobile device, access to Keyboard is very limited. We should add UI elements to move the player, and allow shooting with touch screen.

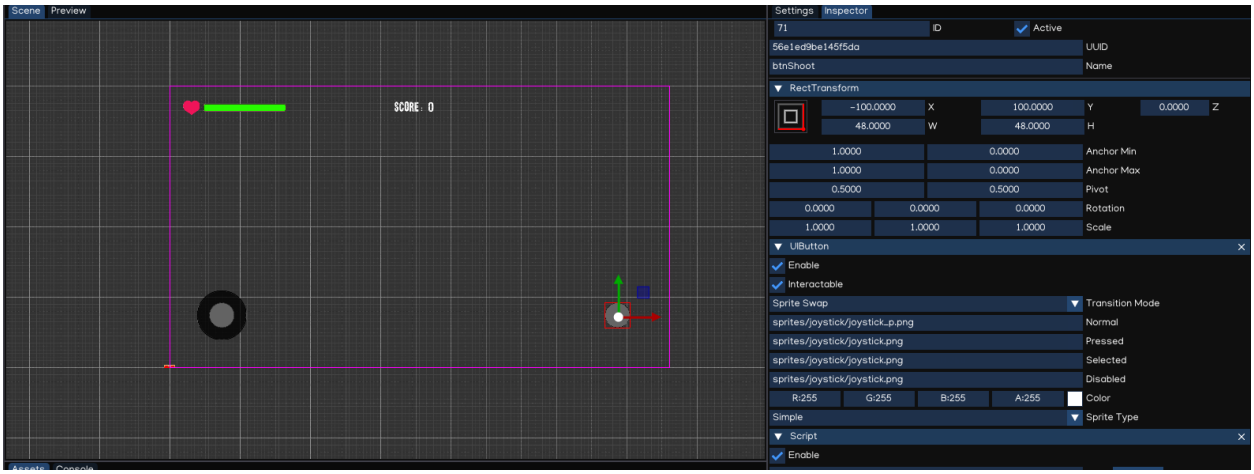
Shoot Button

- Select `Canvas` object, add new `UIButton`, name it as `btnShoot`.
- In the Inspector, change the `Transition Mode` to `Sprite Swap`.
- Set the `Normal` state to `sprites/joystick/joystick_p.png`
- Set press `Pressed` state to `sprites/joystick/joystick.png`
- Create new Script in `scripts/gui`, named `ShootBtn.py`, then attach to the `btnShoot` object.
- Adjust the RectTransform as below:

The content of `ShootBtn.py` is as below:

```
from igeScene import Script

class ShootBtn(Script):
   def __init__(self, owner):
      super().__init__(owner)
      self.player = None
      self._playerShoot = None

   def onStart(self):
      if self.player is None:
```

(continues on next page)



(continued from previous page)

```
        if self.player = self.owner.scene.findObjectByName("MC")
    if self.player is not None:
            self._playerShoot = self.player.getComponent("PlayerShoot")

  def onClick(self):
    if self._playerShoot is not None:
            self._playerShoot.shoot()

  def onDestroy(self):
    self.player = None
    self._playerShoot = None
```

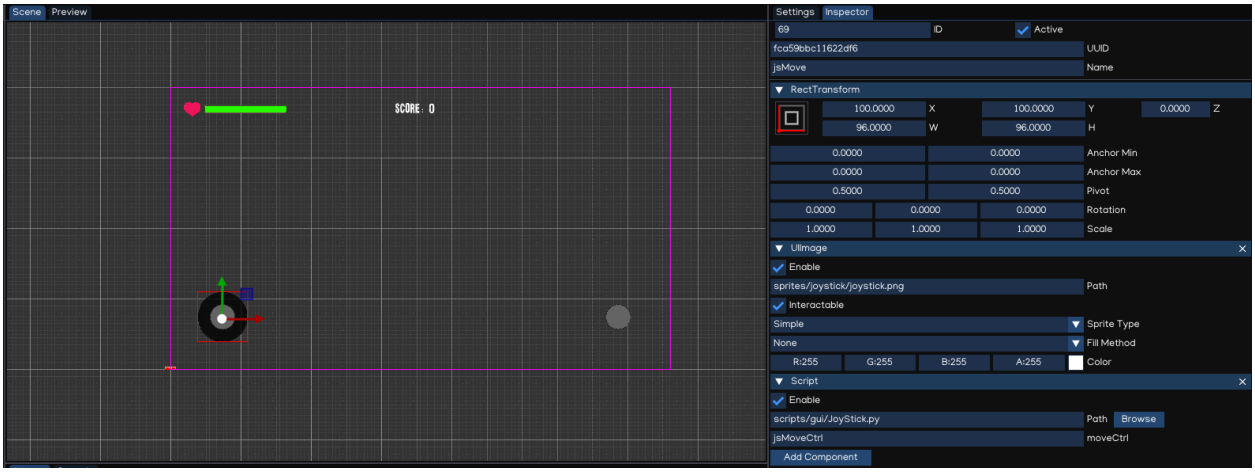
Movement JoyStick

There is no `JoyStick` component, but we can make it using `UIImage`.

- Select `Canvas`, add new `UIImage`, name it as `jsMove`.
- In the Inspector, drag `sprites/joystick/joystick.png` to `Path`.
- Adjust the size to `96 x 96` pixels.
- Adjust the RectTransform as below:
- Select `jsMove`, add new `UIImage`, name it as `jsMoveCtrl`.
- In the Inspector, drag `sprites/joystick/joystick_p.png` to `Path`.
- Adjust the size to `48 x 48` pixels.
- Create new `Script` in `scripts/gui`, named `JoyStick.py`:

```
from igeScene import Script
import igeVmath as vmath
from igeCore.input.touch import Touch

class JoyStick(Script):
  def __init__(self, owner):
    super().__init__(owner)
```

(continues on next page)



(continued from previous page)

```
      self.moveCtrl = None
      self._value = vmath.vec2(0, 0)
      self._maxSize = 0
      self._pressed = False
      self._pressedPosition = vmath.vec3(0, 1, 0)
      self._fingerId = -1
      self._transform = None
      self._scene = None

   def onStart(self):
      self._transform = self.owner.getComponent("RectTransform")
      self._maxSize = max(self._transform.size.x, self._transform.size.y) * 0.5
      self._scene = self.owner.scene
      self._value = vmath.vec2(0, 0)
      if self.moveCtrl is not None:
            self._moveTransform = self.moveCtrl.getComponent("RectTransform")

   def clamp(self, n, smallest, largest):
      return max(smallest, min(n, largest))

   def onUpdate(self, dt):
      for i in range(0, Touch.count()):
            pos = Touch.getPosition(i)
            if Touch.isPressed(i):
               hit = self._scene.raycastUI(pos)
               if hit["hitObject"].name == self.owner.name or hit["hitObject"].name ==␣
↪self.moveCtrl.name:
                  self._pressed = True
                  self._pressedPosition = hit["hitPosition"]
                  self._pressedPosition.z = 0
                  self._value = vmath.vec2(0, 0)
                  self._fingerId = Touch.getId(i)
            elif Touch.isMoved(i):
               if self._pressed and self._fingerId == Touch.getId(i):
                  hit = self._scene.raycastUI(pos)
                  newPos = hit["hitPosition"]
```

(continues on next page)

(continued from previous page)

```
                    newPos.z = 0
                    diff = hit["hitPosition"] - self._pressedPosition
                    self._pressedPosition = hit["hitPosition"]
                    if self._moveTransform is not None and self._maxSize > 0:
                        position = self._moveTransform.localPosition + diff
                        position.x = self.clamp(position.x, -self._maxSize, self._
↪maxSize)
                        position.y = self.clamp(position.y, -self._maxSize, self._
↪maxSize)
                        self._moveTransform.localPosition = position
                        self._value = vmath.vec2(position.x / self._maxSize, position.y /
↪ self._maxSize)
            elif Touch.isReleased(i):
                if self._pressed and self._fingerId == Touch.getId(i):
                    self._pressed = False
                    self._fingerId = -1
                    if self._moveTransform is not None:
                        self._moveTransform.localPosition = vmath.vec3(0, 0, self._
↪moveTransform.localPosition.z)
                        self._value = vmath.vec2(0,0)

    def getValue(self):
        return self._value

    def onDestroy(self):
        self.moveCtrl = None
        self._transform = None
        self._scene = None
```

- Attach the `JoyStick.py` to `jsMove` object, assign `jsMoveCtrl` to `moveCtrl` textbox.
- Adjust `PlayerMovement.py` as below:

```
import igeVmath as vmath
from igeCore.input.keyboard import Keyboard, KeyCode
from igeScene import Script

class PlayerMovement(Script):
    def __init__(self, owner):
        super().__init__(owner)
        self.speed = 2.0
        self.jsMove = None
        self._movement = vmath.vec3(0, 0, 0)
        self._transform = None
        self._rigidbody = None
        self._animator = None
        self._playerHealth = None
        self._jsMoveScript = None

    def onStart(self):
        self._transform = self.owner.getComponent("Transform")
        self._rigidbody = self.owner.getComponent("Rigidbody")
        self._animator = self.owner.getComponent("Animator")
```

(continues on next page)

(continued from previous page)

```
        self._playerHealth = self.owner.getComponent("PlayerHealth")
        if self.jsMove is not None:
            self._jsMoveScript = self.jsMove.getComponent("Script")

    def onUpdate(self, dt):
        if self._playerHealth.hp <= 0:
            return
        h, v = [0, 0]
        if Keyboard.isPressed(KeyCode.KEY_W) or Keyboard.isPressed(KeyCode.KEY_UP):
            v = -1.0
        if Keyboard.isPressed(KeyCode.KEY_S) or Keyboard.isPressed(KeyCode.KEY_DOWN):
            v = 1.0
        if Keyboard.isPressed(KeyCode.KEY_A) or Keyboard.isPressed(KeyCode.KEY_LEFT):
            h = -1.0
        if Keyboard.isPressed(KeyCode.KEY_D) or Keyboard.isPressed(KeyCode.KEY_RIGHT):
            h = 1.0

        if h == 0 and v == 0 and self._jsMoveScript is not None:
            mv = self._jsMoveScript.getValue()
            h = mv.x
            v = -mv.y

        if h != 0 or v != 0:
            self._movement = vmath.vec3(h, 0, v)
            self._movement.normalize()
            self._movement = self._movement * self.speed * dt
            newRotation = vmath.quat_look_rotation(self._movement, vmath.vec3(0.0, 1.0,␣
↪0.0))
            self._rigidbody.moveRotation(newRotation)
            self._rigidbody.movePosition(self._transform.position + self._movement)
            self._animator.setValue("isWalking", True)
        elif self._animator.getValue("isWalking"):
            self._animator.setValue("isWalking", False)

    def onDestroy(self):
        self.jsMove = None
        self._transform = None
        self._rigidbody = None
        self._animator = None
        self._playerHealth = None
        self._jsMoveScript = None
```

- Assign `jsMove` to `jsMove` textbox in Script Inspector.

Now, when play the game, the MC character will be able to controlled using the Move JoyStick, and he can shoot using Shoot button in the screen.

Checkout ige-tutorials, branch 09-mobile-control github repo.

1.14 Python API

igeScene	Scene management
igeCore	Core module
igeVmath	Vector math
igeSdk	Publishing SDK