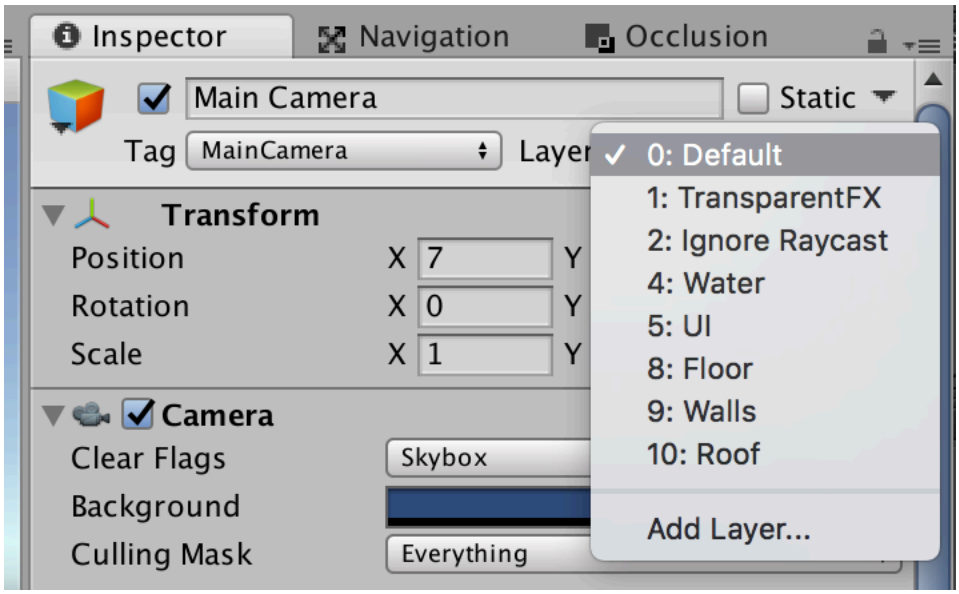
Builder Kit v3.0

About Builder Kit

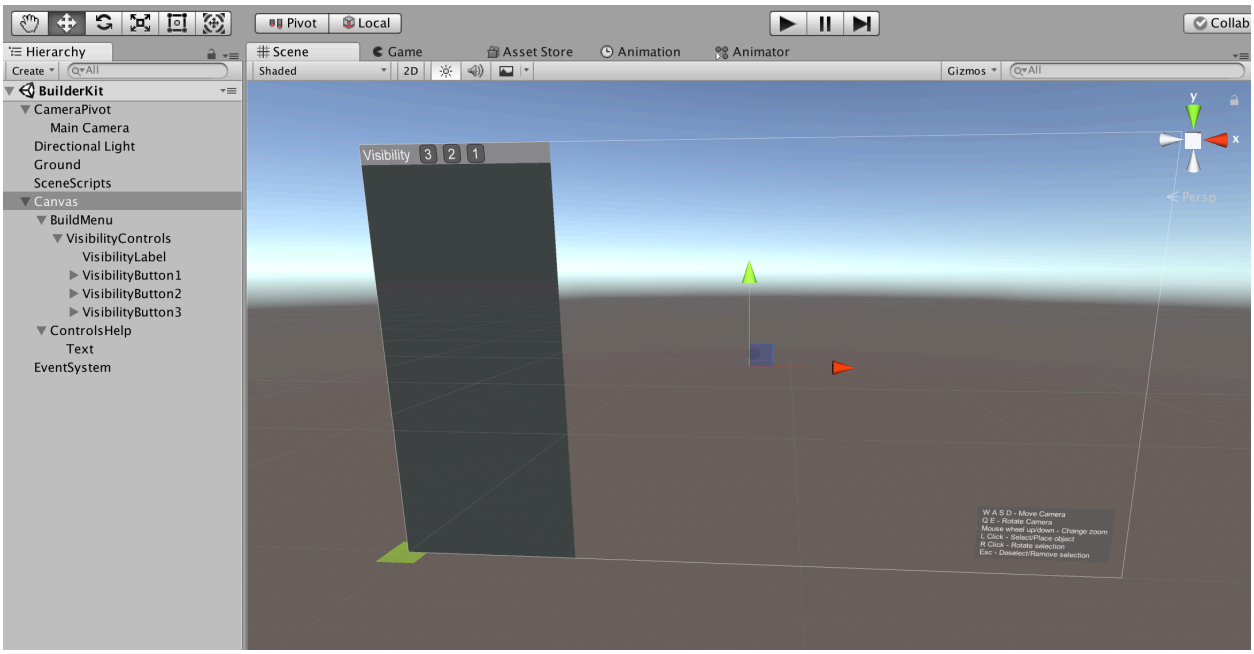
Builder Kit is a set of Scripts with a specific scene setup that serves as a base for house/office building games. It comes with a base set of low poly assets for demo purposes, but they can also be included in any finished games made with the kit.

Setting Up

The package includes a demo scene called BuilderKit, building on top of it is recommended. The only manual work needed is setting up the Floor, Walls, and Roof layers in the case of them not being already imported. This can be done by selecting any object in the scene and clicking ‘Add layers’ at the bottom of the layers dropdown, just below the object name.
Walls and roof objects have layers assigned automatically through their corresponding scripts so there’s no need to modify those.
In previous versions, the ground needed to have the floor layer assigned but this is no longer necessary, as an invisible plane with a collider it is created automatically when starting, and assigned the floor layer. You still need a ground object to visually represent the floor, make sure it has no colliders though.

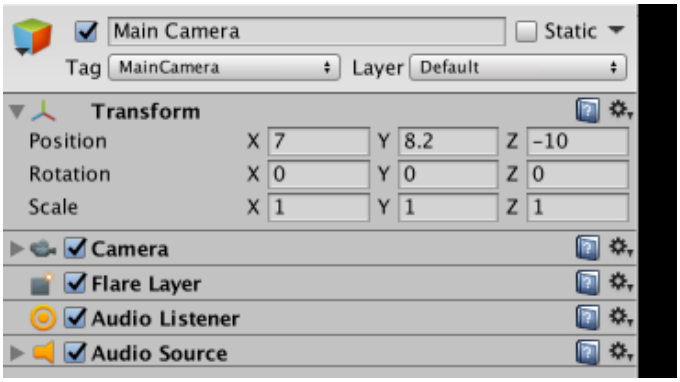


The Builder Kit Scene

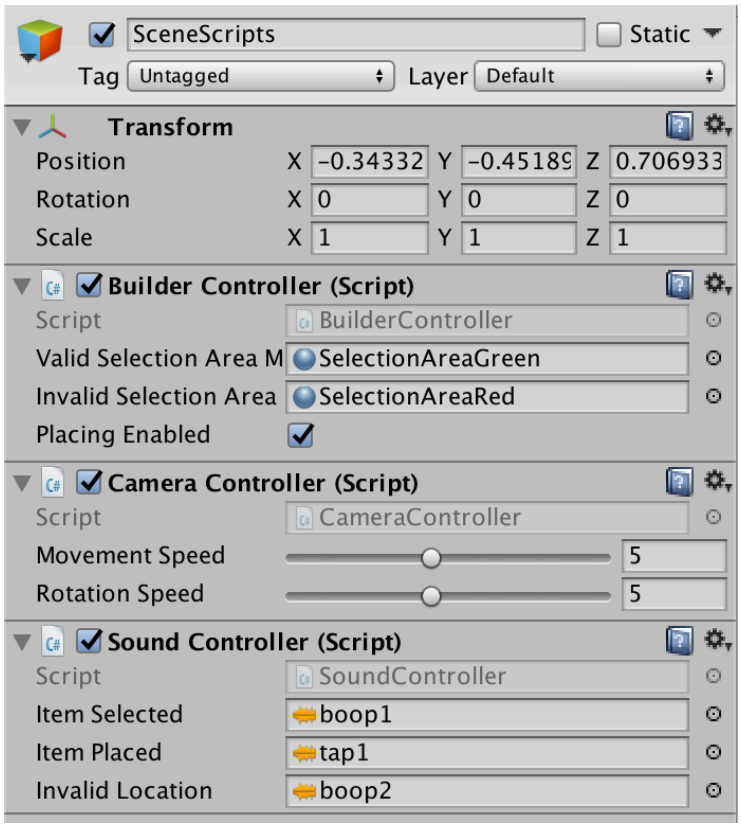


This is a quick overlook on the components and setup needed for the kit to work. The individual scripts are detailed further down in this document.

CameraPivot: Holds the main camera and is used to handle camera movement and rotations.



SceneScripts: This game object holds the Camera Controller, Builder Controller, and the Sound Controller.

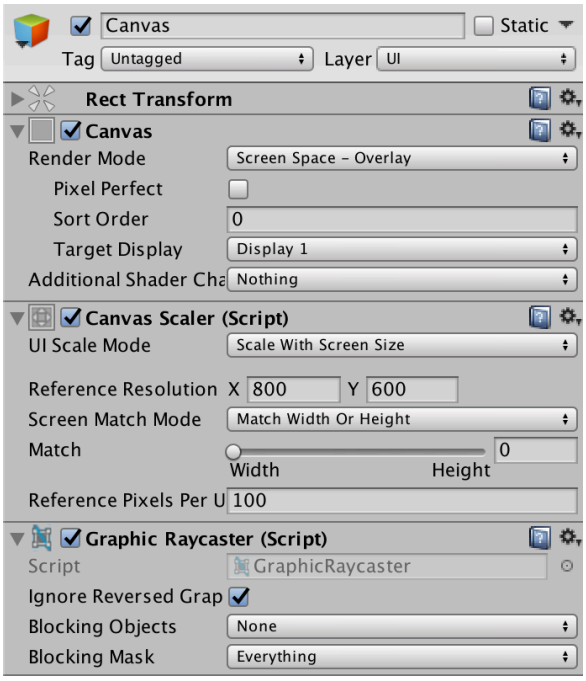


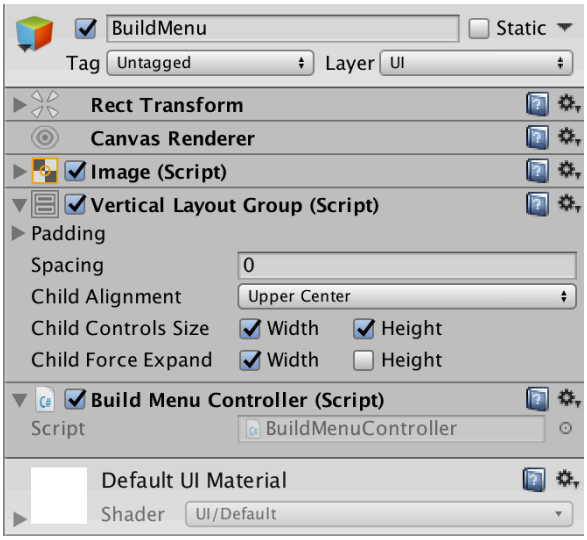
The **camera controller** manages all the camera controls by moving and rotating the camera pivot around the scene.
Builder controller is the core script, it manages most of the behaviors and interactions.
Sound controller is the script that manages the different sounds. It relies on an **Audio Source** component being attached to the main camera

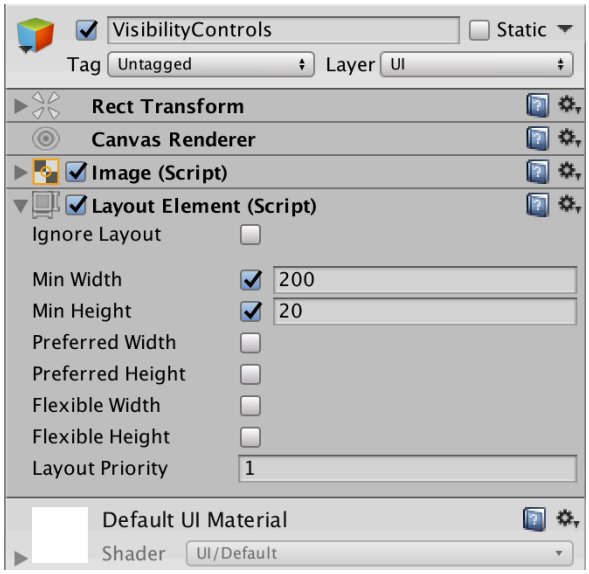
Floor (automatically created): The floor is a simple invisible plane with a **Box Collider** component attached and the **Floor** layer assigned to it. Most of the raycasting is done against the Floor layers. This is automatically instantiated from a prefab on startup.

Grid (automatically created): The grid automatically appears and disappears when you select/deselect an object. It is instantiated from a prefab on startup

Canvas: Holds all of the UI components.
The **Build Menu** has a **Vertical Layout Group** component and a **Build Menu Controller** attached. The controller script needs to belong to this game object and cannot be placed anywhere else, the reason is that it handles OnMouseEnter and OnMouseExit events.


Canvas
Static
Tag Untagged
Layer UI
Rect Transform
Canvas
Render Mode Screen Space - Overlay
Pixel Perfect
Sort Order 0
Target Display Display 1
Additional Shader Cha Nothing
Canvas Scaler (Script)
UI Scale Mode Scale With Screen Size
Reference Resolution X 800 Y 600
Screen Match Mode Match Width Or Height
Match 0
Width Height
Reference Pixels Per U 100
Graphic Raycaster (Script)
Script GraphicRaycaster
Ignore Reversed Grap
Blocking Objects None
Blocking Mask Everything


BuildMenu
Static
Tag Untagged
Layer UI
Rect Transform
Canvas Renderer
Image (Script)
Vertical Layout Group (Script)
Padding
Spacing 0
Child Alignment Upper Center
Child Controls Size Width Height
Child Force Expand Width Height
Build Menu Controller (Script)
Script BuildMenuController
Default UI Material
Shader UI/Default


VisibilityControls
Static
Tag Untagged
Layer UI
Rect Transform
Canvas Renderer
Image (Script)
Layout Element (Script)
Ignore Layout
Min Width 200
Min Height 20
Preferred Width
Preferred Height
Flexible Width
Flexible Height
Layout Priority 1
Default UI Material
Shader UI/Default

Block

This represents each building block in the area, it can hold a Wall, a Floor Tile, and a Main Object. This class has no methods because it only holds objects.

Public attributes:

Name	Class/Type	Description
wall	Wall	The wall set in this block
blockObject	PlaceableObject	The common object set in this block
floorTile	FloorTile	The floor tile at this position

Block Manager

Manages the adding and removing of objects from the blocks. Holds a dictionary that has the block positions as keys and the different blocks as values, this simplifies the lookup and eliminates the need for cpu intensive raycasting when checking if an item can be placed in a particular location.
The class has no public attributes and its public methods are static. It's basically a singleton that manages all the block operations.

Public methods:

Name	Parameters	Description
IsAreaEmpty	Vector3 start; Vector3 end	Checks if an area is empty
IsBlockPositionValid	Vector3 position	Determines if the position is within the map's boundaries
IsAreaValid	Vector3 start; Vector3 end	Determines if the area is within the map's boundaries
GetBlock	Vector3 position; bool addIfNotPresent	Gets a block at the given position, adds a new one if needed and the addIfNotPresent flag is set to true
GetWallBlock	PlaceableObject placeableObject; Vector3 position; bool addIfNotPresent	Same as GetBlock, but adjusts the coordinates to be moved by 0.5 in the corresponding direction to represent walls that exist between two blocks
PutWall	Wall wall; Vector3 position	Registers a wall object in the block at the specificed position. The position is adjusted by GetWallBlock

Name	Parameters	Description
PutWallObject	WallObject wallObject; Vector3 position	Registers an object and makes it a child of a wall game object
PutObject	PlaceableObject item; Vector3 position	Registers a Placeable Object in a block at the given position
PutFloorTile	FloorTile tile; Vector3 position	Registers a FloorTile in a block at the given position
HasTile	Vector3 position	Returns true if the block at the position has a non-blocker tile assigned
IsNextToTile	Vector3 position	Checks if a block has a tile next to it
IsNextToWall	Vector3 position	Checks if there's a wall contiguous to the given position
IsBlockerTileSet	Vector3 position	Checks if there's a blocker tile assigned as floorTile at the given position

Builder Controller

This singleton controls most of the game's behavior. Placing, moving and removing objects, handling the selected block displays, etc.

Public attributes:

Name	Class/Type	Description
validSelectionAreaMaterial	Material	The material used when the selected object can be built on the current block
invalidSelectionAreaMaterial	Material	The material used when the selected object cannot be built on the current block
placingEnabled	bool	Controls whether placing of objects is enabled or not. Used to disable this action when interacting with menu items
modalOpen	bool	When set to true, it blocks anything non UI related from happening. This is used by the Save/Load dialog.

Public methods:

Name	Parameters	Description
ToggleVisibility	bool isRoofVisible; bool areWallsVisible	Makes the roof and walls visible/ invisible
GoUp		Goes Up one floor if not at top floor
GoDown		Goes Down one floor if not at ground level
EnablePlacing		Enables the placing of objects
UpdateBlockPosition	bool forceUpdate	Updates the current block position. It only does the calculations if the mouse has moved or if forceUpdate is true
SelectObject	GameObject selection	Selects an object. This is called by the Build Menu Controller.
HideSelectedObject		Hides the currently selected object. Called by BuildMenuController
DeselectObject		Deselect the currently selected object.

Builder Kit Config

Holds configuration parameters for the kit.

Public attributes:

Name	Class/Type	Description
NUMBER_OF_FLOORS	int	Maximum number of floors available for building
MAP_WIDTH	int	Maximum map size on the x axis
MAP_HEIGHT	int	Maximum map size on the z axis
FLOOR_HEIGHT	float	This should always be the height of the tallest wall. Used to determine where to build on floors higher than ground level
GRID_FLOAT_HEIGHT	float	The distance from the floor the grid floats at. Used to avoid visual artifacts
GRID_FADING_SPEED	float	The speed at which the grid fades in and out

Name	Class/Type	Description
GRID_MOVING_SPEED	float	The speed at which the grid moves up or down when switching floors
SAVE_EXTENSION	string	The file extension used for the save files
SAVE_PREFIX	string	The prefix added to the save files
SAVE_ITEM_HEIGHT	int	The height of the save boxes in the save/load dialog

Building Floor Manager

Handles the assigning of objects to their floor parents to better control visibility when switching floors. It creates one empty BuildingFloor# object per floor specified in BuilderKitConfig.NUMBER_OF_FLOORS and sets the placed objects as children of those depending on which floor is active. It also contains a tempFloor game object that is used when moving an object that has already been placed to another floor.

Public attributes:

Name	Class/Type	Description
ativeFloor	int	Indicated which floor is currently active. 0 is ground.

Public methods:

Name	Parameters	Description
GoUp		Goes Up one floor if not at top floor
GoDown		Goes Down one floor if not at ground level
AssignToCurrentFloor	GameObject placeable	Makes the placeable object a child of the currently active floor
AssignToTempFloor	GameObject placeable	Makes the placeable object a child of the temp floor. Used when moving existing objects.

Build Menu Controller

Manages the object selection menu and the visibility controls. The objects are loaded in the menu by categories. The categories are set in folders under Prefabs.
The Prefabs folder also contains the child elements of the build menu: Menus/CategoryPanel is the layout for each category, and Menus/ObjectMenuItem if the layout for each individual object.

Camera Controller

This class handles the camera movement, rotation, and zoom. It relies on the main camera being set as the child of a game object called **CameraPivot** on the scene.

Public attributes:

Name	Class/Type	Description
movementSpeed	float	Controls the camera movement speed
rotationSpeed	Material	Controls the camera rotation speed

Sound Controller

Handles the sounds, it relies on an **Audio Source** component being attached to the main camera.

Public attributes:

Name	Class/Type	Description
itemSelected	AudioClip	The sound played when an item is selected
itemPlaced	AudioClip	The sound played when an item is placed in the scene
invalidLocation	AudioClip	The sound played when trying to place an item but the location is invalid.

Public methods:

Name	Parameters	Description
PlayClip	AudioClip clip	Plays the sound passed into it

Object Loader

This singleton loads all the prefabs under categories specified in a static attribute called *categories.* For a prefab to be loaded as a placeable object, it needs to be under one of those categories and it has to be placed in a folder with said category name under the Prefabs folder. Starting from version 3.0 it also loads two additional dictionaries, one with all the objects by name, and one with all the available materials by name. These are used by the game Load functionality and the elements of them can be searched using two methods, GetObject, and GetMaterial.

Public methods:

Name	Parameters	Description
GetObjectsByCategory		Returns a Dictionary with all the categories as keys and all their respective objects as elements of an array of GameObjects. If this method is called more than once it will always return the same dictionary.
GetObject	string name	Returns the prefab for the game object with the associated name, null if not found.
GetMaterial	string name	Returns the instanced material associated with that name, null if not found.

Placeable Object

This is the base for all objects that can be placed into the scene. It cannot be used as a component on its own, it has to be extended by the specific classes to implement the all the abstract methods instead.

Public attributes:

Name	Class/Type	Description
objectName	string	The name of the object (e.g. Comfy Chair, Blue Couch)
yPosition	float	How high off the ground the object should be

Public methods:

Name	Parameters	Description
IsValidLocation (abstract)	Vector3 start; Vector3 end	Returns true if the location for the object is valid. If the object is a single item it uses only the *end* vector as the position.
Place (abstract)	Vector3 start; Vector3 end	Places an object on the scene. If the object is a single item it uses the end vector as the position
Select (abstract)		Called when selecting an item from the menu, it can be used to perform particular actions like toggling visibility to a certain level when a particular type of object is selected.
PickUp (abstract)		Called when picking up an object that was already placed to move it somewhere else
Rotate		Rotates the object 90 degrees clockwise

Common Object
Class used for most objects, from furniture, to stairs. Extends Placeable Object and inherits all of its public methods and attributes.

Floor Tile
Class used for floor tiles, extends Placeable Object and inherits all of it's public methods and attributes. Any 1x1 plane can be converted to a floor tile by attaching a collider and this script to it and saving it as a prefab under Prefabs/Floor Tiles.
Floor tiles can be placed in entire rectangular areas instead of one by one.

Roof
Class used for roof objects, extends Placeable Object and inherits all of it's public methods and attributes. Roofs are placed in rectangular areas and their elevation (y scale) depends on the area size and an attribute called *maxElevation.*

Public attributes:

Name	Class/Type	Description
maxElevation	float	The maximum scale a roof can have on the Y axis

Wall
Class used for walls, extends Placeable Object and inherits all of its public methods and attributes. Walls can be placed individually or in straight lines in any direction.

Public attributes:

Name	Class/Type	Description
allowObjectPlacement	bool	Whether this object supports other objects being placed on it, like doors, windows, paintings. etc.
allowPainting	bool	If this is false it disables painting on the object. Useful for some items that shouldn't be painted, like steel fences.

Wall Object
Class used for objects that are placed in walls (doors, windows, paintings, etc), extends Placeable Object and inherits all of its public methods and attributes. Must be placed one by one.

Wall Paint
Class used for different wall paint colors and textures, extends Placeable Object and inherits all of its public methods and attributes. Walls can be painted in straight lines on any direction or individually.

Save
Main Serializable Class used for saving and loading.

Public attributes:

Name	Class/Type	Description
name	string	Name of the save
date	DateTime	Date and Time of the save
floors	List<FloorSave>	A list containing the save data for each floor

Public methods:

Name	Parameters	Description
Save (Constructor)	string name	Saves the current scene in a JSON file
Load		Loads the save file, replacing anything that was there before

FloorSave
Serializable class containing all the saved objects for a particular floor.

Public attributes:

Name	Class/Type	Description
floorNumber	int	Number of the floor

Public methods:

Name	Parameters	Description
FloorSave (Constructor)	int floorNumber, GameObject building floor	Generates a serializable floor save with all the objects on the passed building floor
Load		Loads the given floor

PlaceableObjectSave
Serializable class containing save data for a particular object in the scene.

Public attributes:

Name	Class/Type	Description
prefabName	string	Name of the prefab the object was instantiated from
position	Vector3	The position of the object
rotation	Quaternion	The rotation of the object
materials	string[]	The names of the materials assigned to the object
children	List<PlaceableObjectChildSave>	The children of the object (only for walls at the moment)

Public methods:

Name	Parameters	Description
PlaceableObjectSave (Constructor)	GameObject placeable	Generates a Serializable object save with the passed in game object's data
Load		Loads the object

PlaceableObjectChildSave
Serializable class containing save data for a particular child of an object object in the scene. This class is necessary to work around Unity's serialization limitations that prevent nesting objects of the same type.

Public attributes:

Name	Class/Type	Description
prefabName	string	Name of the prefab the object was instantiated from
position	Vector3	The position of the object
rotation	Quaternion	The rotation of the object

Public methods:

Name	Parameters	Description
PlaceableObjectChildSave (Constructor)	GameObject placeable	Generates a Serializable object save with the passed in game object's data

Name	Parameters	Description
Load		Loads the object

SaveManager
Class that handles save file creation and loading. It works as a singleton and doesn't need to be instantiated.

Public methods:

Name	Parameters	Description
SaveCurrent		Saves the current scene to a file and returns a Save object
GetSaves		Loads all the save objects from files and returns them in a list

SaveDialogController
Class that handles the save dialog and Saving/Loading actions.

Adding Your Custom Objects

Adding your own objects is pretty simple:

- Import your model into Unity by selecting the Objects folder, right clicking on it and selecting *Import New Asset* from the menu.
- Adjust the scale of the object using the inspector so that it fits with the rest of the objects
- Attach a **BoxCollider** component to it.
- Attach the respective script (Common object if it's something like a statue, bed, chair, etc), and set the Object's name.
- Create a prefab under the right category in Prefabs/The Category
- Select the object and drag it inside the empty prefab you just created.
- To create a thumbnail you can either take a screenshot of the prefab icon or create a small image yourself, put it under the ObjectThumbnails folder and make sure it has the same name as the prefab. Adjust the settings to your liking.
- Done!

Adding Your Custom Categories

- Create a folder with the name of the category under Prefabs/ add the objects you wish to be part of it

- Edit ObjectLoader.cs and add your new category to the **categories** static array.
- Done!

Preparing Your 3d Models

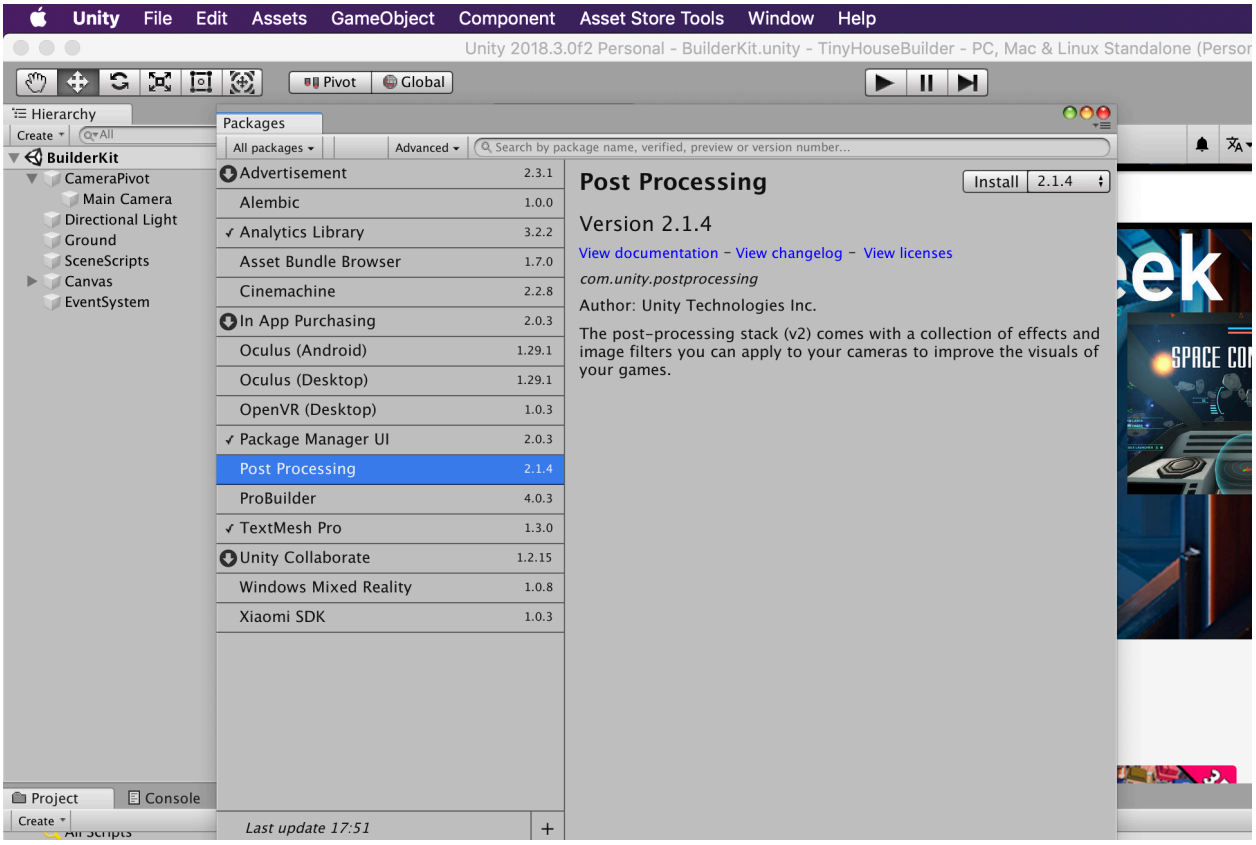
Importing models should be straightforward unless you use blender, in which case a small step is needed to correct the axes. Blender works with the Z axis representing up, Unity works with Y up. Luckily this can be solved by exporting your blender model to FBX format, selecting -Z Forward as the forward vector, Y up as the Y vector, and checking the box that says **! EXPERIMENTAL! Apply Transform.**

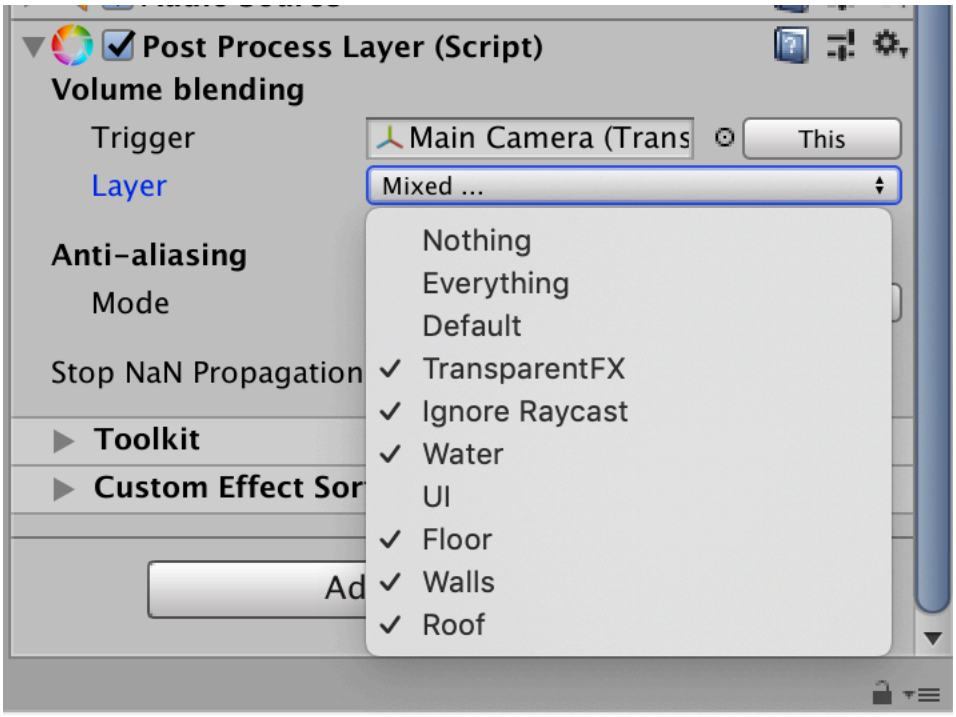
Improving image quality with the Post Processing Stack

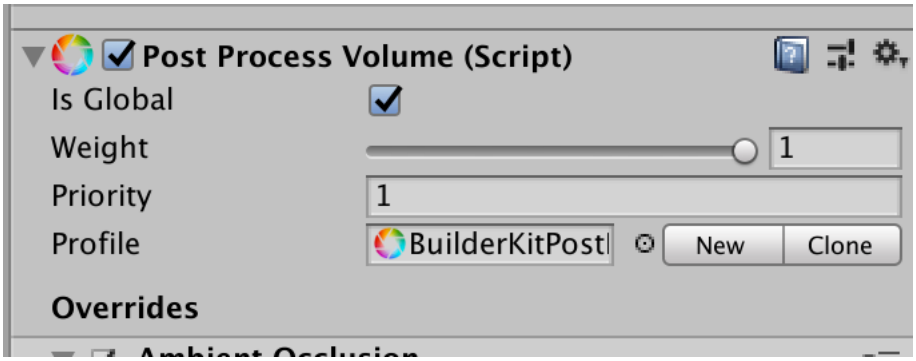
Included in the root folder of this package there's a post processing profile that can be used with Unity's Post Processing Stack.

Unity 2018.2 and newer:

Select Window -> Package Manager. Then install the Post Processing stack from that dialog.







Once it has downloaded all the files you can go to your Main Camera in the editor and add two components: A Post Processing Layer, and a Post Processing Volume.

Choose the layers just like in the screenshot above, then select the profile called BuilderKitPostProcessing that is included in this package and you are done.

Pre Unity 2018.2:

To use it, all you need to do is import the free package from the Unity asset store, attach a post processing behavior script to the main camera, and select the included profile, which is called ‘BuilderKitPostProcessing’.