City Builder Starter Kit

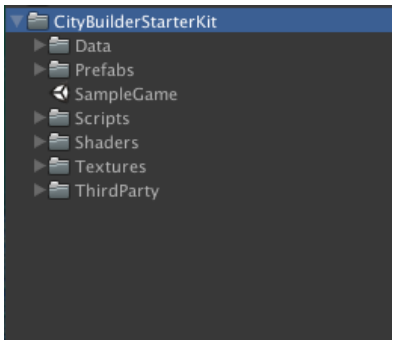
by JNA Mobile

v1.0

Table of Contents

Quick Start

To get started simply expand the CityBuilderStarterKit folder in your project window, open the **SampleGame** scene and press Play.



Due to the complexity of the configuration it is recommended that you **ALWAYS** start form the **SampleGame** scene, and then alter as desired.

Videos

JNA Mobile will be providing additional tutorials discussing how you can customise the kit via YouTube. Go here to find them:

http://www.youtube.com/jnamobilehidden

Tutorial video names will start with 'CBSK – Tutorial'.

Support

Get support via email by using the following address:

support@jnamobile.com

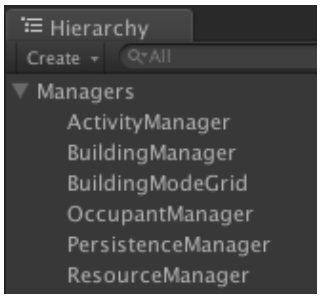
Get support on the Unity Forums by contacting JohnnyA.

Please include your asset store order number in all support requests

Structure of the City Builder Starter Kit

Manager Classes

The manager classes provide convenient methods to access various data. You should add all manager classes to your scene (or start with the sample scene).



Manager	Details
ActivityManager	Manages custom activity types which can be triggered by buildings. ActivityManager is responsible for loading the ActivityData but it does not manage the execution of individual activities. This is done by the building.
BuildingManager	Loads and manages the building types and provides high-level access to the currently built buildings. The BuildingManager also maintains the static variable ActiveBuilding, the currently selected building frequently used by the UI.
BuildingModeGrid	Manages the grid. When buildings are built they occupy space on the grid. This class lets the building manager know if a space is empty.
OccupantManager	Manages the objects that reside inside of buildings. Like the building manager it is primarily responsible for occupant types, but also provides high level access to query the exsiting occupants.
PersistenceManager	Saves the game. The current implementation provides a PlayerPrefs implementation for persistence but this can be easily extended to use another stream such as the file system or a server.
ResourceManager	Stores the gold and resources that the player owns.

Buildings

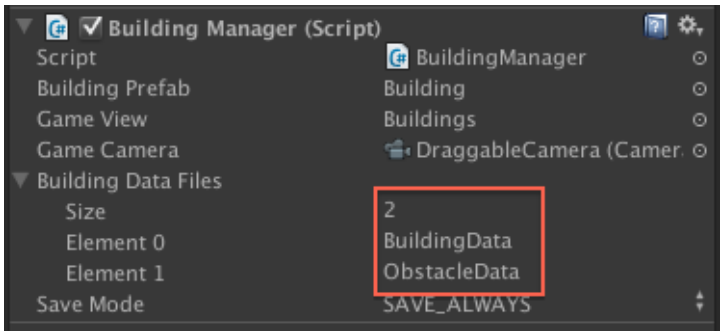
The Building class is the most important class in the kit. It manages the details of buildings, activities and occupants. It contains almost all of the data stored when the system is saved and provides public methods for interacting with buildings (such as building them, moving them, or starting new activities).

Buildings have a type and most of the behaviour of a building comes from the parameters defined in the building type.

Configuring New Buildings

Buildings are configured in an XML file. Use the sample XML file **BuildingData.xml** as your starting point. You can use any name for the building data file as long as you place it in a Unity **Resources** folder (any folder in the asset directory named Resources).

If you wish to use different XML files you can set the names in the properties of the BuildingManager object:



Note that the extension **.xml** is not included in the entry. It is however required for the filename.

Obstacles

Obstacles are also buildings but they have the special flag isObstacle set to true. For convenience the sample game places these files in a different XML (**ObstacleData.xml**) file but this is not required.

Building Type Parameters

This section lists and describes the parameters used in the XML file.

Parameter	Details
id	Unique identifier for this building. Note that the ID should be unique across buildings AND occupants.
name	Human readable name of the building.
description	Human readable description of the building.
spriteName	Name of the sprite to use for this building. Looked up in the NGUI Atlas by this name so it muct match exactly.
isObstacle	Set this to true if the building is an obstacle that is cleared instead of being built.
cost	Cost to build (or clear if this is an obstacle).
buildTime	Time to build (or clear if this is an obstacle).
allowIds	Ids of the buildings and occupants that this building allows.
requireIds	Ids of the buildings and occupants required before this building can be built.
shape	The shape of the building defined by grid offsets from 0,0 that are occupied by this building. Entries of (0,0), (1,0), (0,1), (1,1) would define a small square building.
activities	List of custom activities (defined by ID and loaded by the ActivityManager) that this building can do.
generationType	Type of reward automatically generated by this building. Ignored if generation amount = 0. For obstacles this is reward type for clearing.
generationTime	Time to generate the reward.
generationAmount	Amount of reward to generate each time interval. For obstacles this is reward amount for clearing.
generationStorage	Maximum amount of generated reward to store in this building. Acknowledgement indicator will appear once this value is reached.
occupantStorage	The space for holding occupants. Note that occupants size can be variable (for example a building could hold two tigers with a size of 1, but only one elephant which has a size of 2).

Occupants

Occupants are the objects that reside inside of buildings. They could be soldiers (like in the sample) but also dragons, cars or even furniture. The default occupant has no behaviour (other than providing the ability to train and dismiss them) but the video tutorials show you how you can add behaviour to an occupant.

Like a building the occupants behaviour is defined by occupant type which is in turn defined by a number of parameters.

Configuring New Occupants

Occupants are configured in an XML file just like Buildings. Use the sample XML file **OccupantData.xml** as your starting point. You can use any name for the occupant data file as long as you place it in a Unity **Resources** folder (any folder in the asset directory named Resources).

If you wish to use different XML files you can set the names in the properties of the OccupantManager object. Note that the extension **.xml** is not included in the entry; it is however required for the filename.

Occupant Type Parameters

This section lists and describes the parameters used in the XML file.

Parameter	Details
id	Unique id of the occupant.
name	Human readable name of the occupant.
description	Human readable description of the occupant.
spriteName	Name of the sprite to use for this occupant. Looked up in the NGUI Atlas by this name.
cost	Cost to recruit/build.
buildTime	Time to recruit/build.
recruitFromIds	List of building IDs for buildings that can recruit this unit.
allowIds	Ids of the buildings and occupants that this occupant allows. This is used for UI only, requireIds is the actual constraint.
requireIds	Ids of the buildings and occupants required before this building can be built.
housingIds	List of building type IDs for buildings that can hold this unit type.
occupantSize	Size of the occupant (see BuildingType.occupantStorage).

Activities

Activities define the things that a building can do beyond the normal activities. Buildings are associated with activities and can once built can perform them.

Configuring New Activities

Activities are configured in an XML file. Use the sample XML file **ActivityData.xml** as your starting point. You can use any name for the activity data file as long as you place it in a Unity **Resources** folder (any folder in the asset directory named Resources).

If you wish to use different XML files you can set the names in the properties of the ActivityManager object. Note that the extension **.xml** is not included in the entry; it is however required for the filename.

A note on Activity Icons

If you are using the default NGUI user interface (i.e. not rolling your own), then you will need to add an icon to the UI atlas so that the activity button can show the activity icon.

The default UI expects this sprite to be named:

<ActivityTypeInLowerCase>_icon

For example if your activity was of type **HUNT** then the sprite name should be:

hunt_icon

Activity Type Parameters

This section lists and describes the parameters used in the XML file.

Parameter	Details
type	Unique id of the activity.
durationInSeconds	Time it takes to complete the activity.
description	Human readable description of the activity.
reward	The reward given for compelting the activity. Options are from the RewardType enum. One of: RESOURCE, GOLD, CUSTOM Note that if you use reward type **CUSTOM** you will have to implement your own reward mechanism by extending the AcknowledgeActivity() method of the Building.cs class.
rewardAmount	Amount of reward received.

Building View

The building view ties a Building to its visual representation. The building view shows the building and its activities and provides the user interface by which the user can interact with the building (for example by acknowledging a compelted activity).

The current implementation of the building view relies on NGUI. In the (near) future it is expected that this dependence will be removed so that other views can be easily provided (such as a view using Unity's new built-in 2d).

Adding or Replacing Sprites

Because the building view uses NGUI it is important that you update the NGUI atlas by adding the new images. There are three options:

1. Create a new Atlas (recommended for production). To do this:
    a. Create a **new UIAtlas** and add all your building and occupant sprites.
    b. Update the Building prefab so each UISprite reference points to the new UIAtlas.
    c. Save the prefab.
    d. Make sure your sprite names in your building data file match the atlas.

2. Add sprites to the **HumanBuildingMode** atlas (quick). To do this:
    a. Use the NGUI Atlas maker to add new sprites to the existing atlas.
    b. Make sure your sprite names in your building data file match the atlas.

3. Update sprites in the **HumanBuildingMode** atlas (quickest). To do this:
    a. Name your sprites so they have the same name as existing sprites.
    b. Use the NGUI Atlas maker to update the UIAtlas with your new sprites.