Insane Systems RTS Starter Kit Guide

Note

We’re moving documentation to the new platform, you can find it here:

insanesystems.net/APIDocumentation/RtsStarterKit

If the site doesn’t open, a VPN service can help. Also this documentation will be available locally in the next asset versions.

Old documentation in the Google Docs is still available, but we do not plan to update it anymore.

Content

Introduction

RTS Starter Kit asset is designed to simplify development of RTS game. This asset is primarily designed to create military RTS, which have vehicles, bullets, etc., but you can use it in other game styles. It contains a lot of components, which you need to know to simplify your work with the asset and this guide will help you to do it. But do not worry, all these components are simple to understand.

Guide contains three main partitions:

- **Information** - information about asset objects, resources and components.
- **Settings and customization** - information about asset settings, which help you to customize examples for your RTS.
- **Programming** - this partition contains information, which will help you to implement your own functionality.

You can open this guide online in Google Docs by using this link:
https://docs.google.com/document/d/1jM-qJoNewQ2HnpzaUbwVG6VSX94sXPAGT5YRK15mUDA/edit?usp=sharing

In **Google Docs** you can quickly navigate between partitions by using **Document Structure** on the left side of the window.

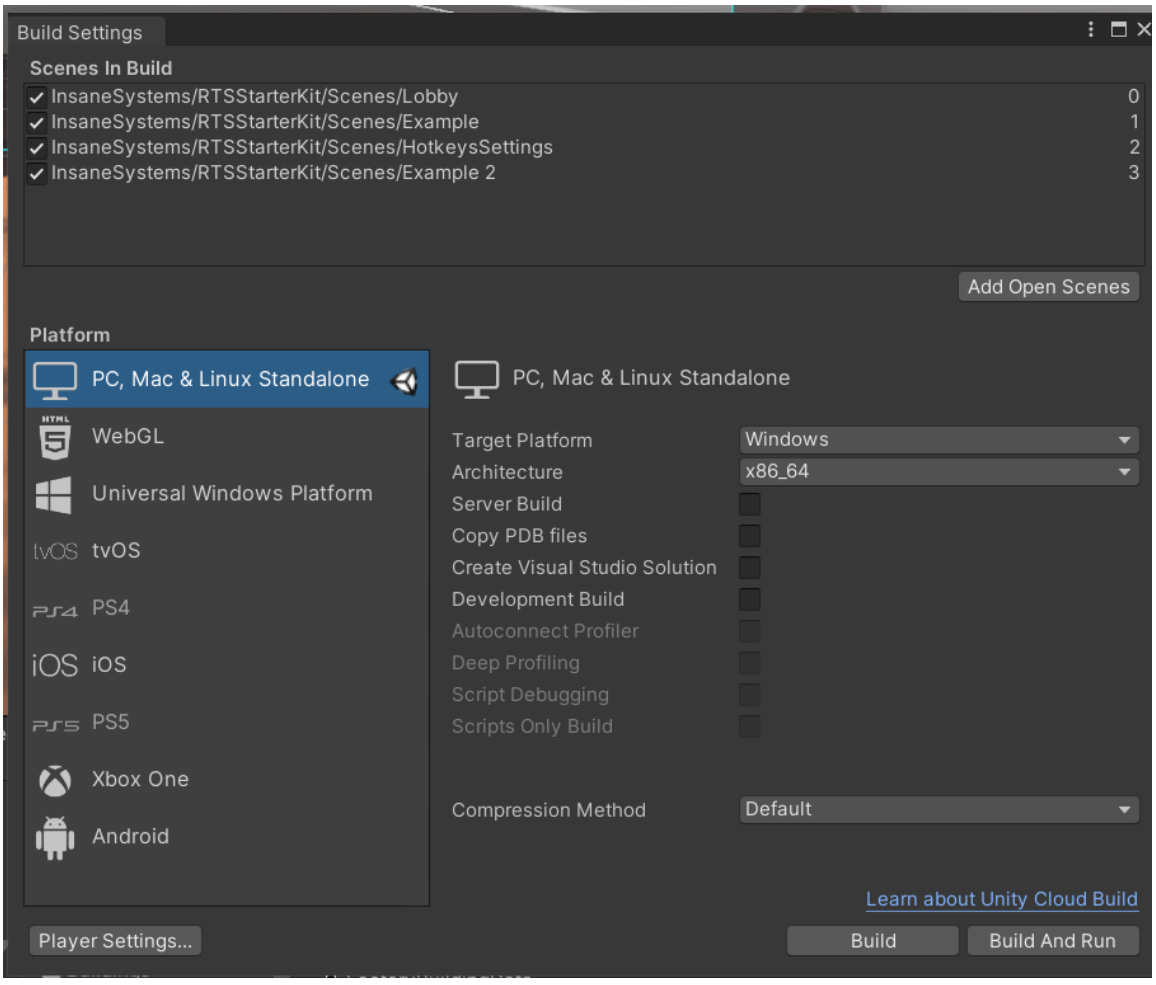
Information

Quick Start

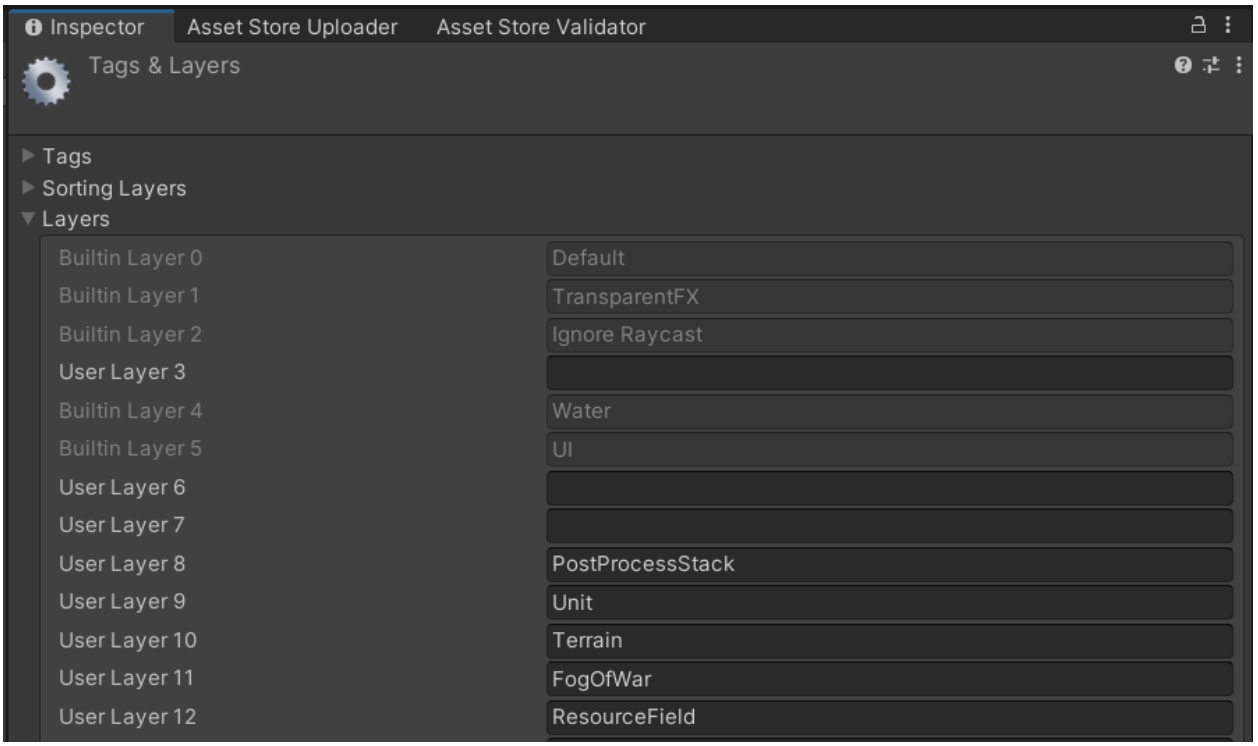
Best way to understand how it all works - check the **Lobby** and **Example** scenes. It contains all used components which are shown in a simple RTS Game prototype. Have a detailed look at it and if you have questions - go ahead and read the next sections of this guide. In this partition below you'll read some basic principles descriptions first, after it you can go to Settings and Customization partition with more details.

Project preparation

Firstly, to run the prototype properly, check that **Lobby** and **Example** scenes added to **Build Settings**. It should be done automatically by code, but if not, add it manually.



Next, you can check that the asset automatically added new layers: **9 - Unit, 10 - Terrain, 11 - Fog of War, 12 - Resource field**. First is used to indicate ground object on maps, second – for all game buildings and units, third - indicates fog of war, final is for resource fields.



These fields should be filled **automatically**, if it is not done, please, add these layers manually.

Unit modules description

RTS Starter Kit is uses module system to customize game units and buildings, and there a list of primary modules:

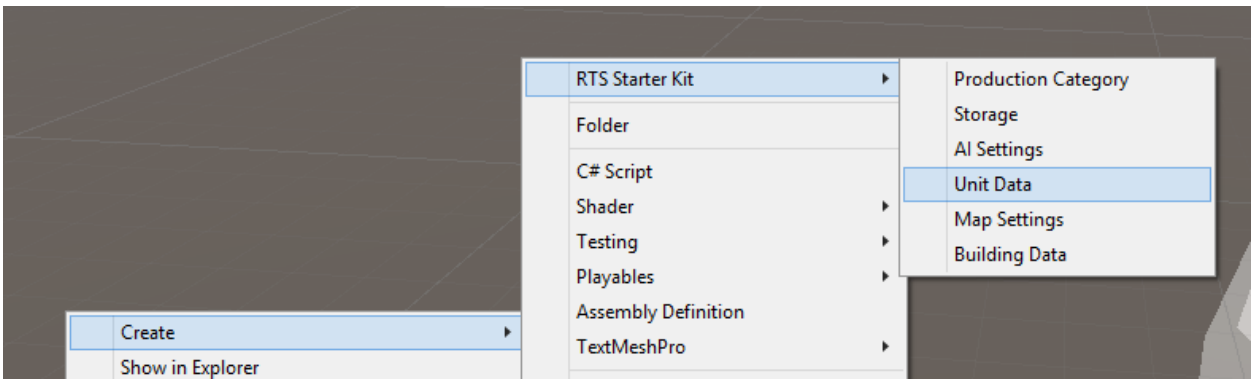
- **Unit** – core module of any unit. Contains primary unit info and links other modules together.
- **Damageable** – makes an object possible to take damage. This component contains unit health info etc.
- **Movable** – this module represents a unit moving system. If you want to allow unit to move, add this module.

- **Attackable** – attack module. Unit, which has this module, gets the ability to attack other units.
- **Tower** – this module is needed for vehicles like tanks to rotate its tower to the targets.
- **Production** – allows this object to build units or buildings.
- **Harvester** – allows unit to gather resources.

Unit module is a simple component, which can be drag n dropped to **Prefab** as any other **Unity** component or script. With automatic prefab generation (described in next partitions) all needed for unit modules will be added to prefab automatically (by its UnitData settings, read about it below). You can create your own modules and connect it to existing, if you need. This is described in the **Programming** partition.

Unit data description

Unit Data is an data file in resources (inherited from **ScriptableObject** class), which contains all settings of one unit type. You can create new **Unit Data** by right-clicking **Project Window** and selecting in context menu option **RTS Starter Kit/Unit Data**. Also it will be created automatically if you will create unit using **Unit Editor** window.

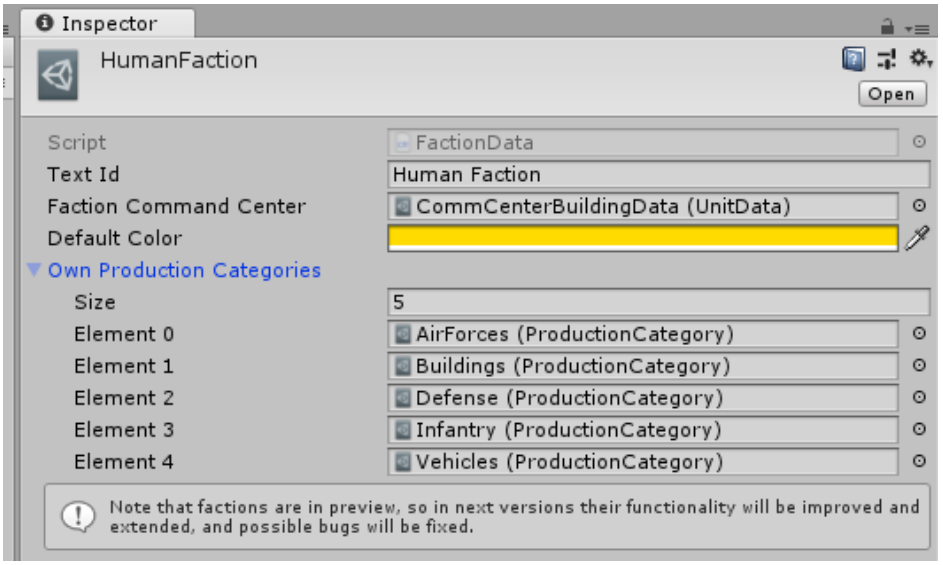


Production categories description

Production category is a list of units or buildings, which can be created by some unit or building with a **Production** module. You can create your own production categories or change existing to set up custom lists of allowed buildings or units.

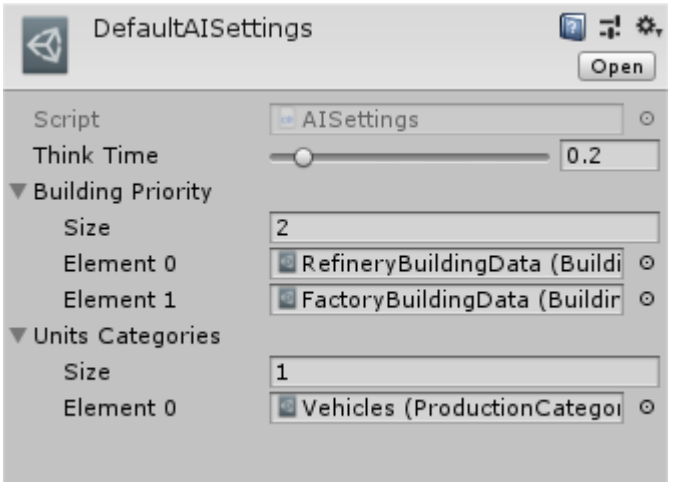
Faction description

Faction describes game faction/race. You can create **Faction Data** and setup here some custom faction parameters. Primary settings here is Command Center of faction - primary building, all next faction settings will depends on it (for example, it will contain faction-specific buildings production category, other factions building will contain faction-specific units production category etc, I think you get it).



AI Description

This asset contains bots (computer enemies). Currently their AI is simple, but flexible to an extent. Now AI Bots can create buildings, buy units and send them to attack enemies. Current **AI Settings** looks like this:



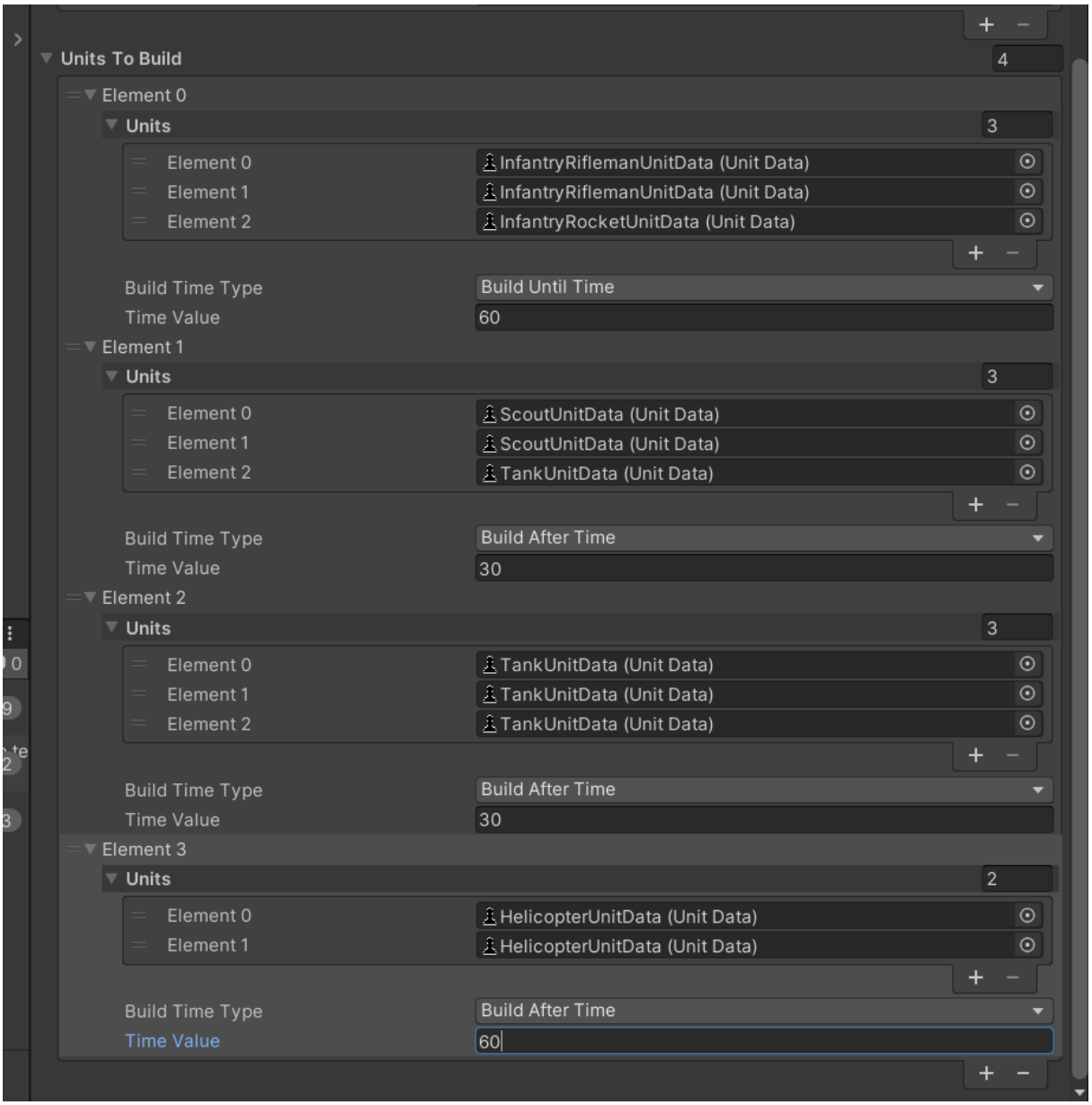
Think time is time between different AI Actions. Increase it to decrease bots actions per minute.

Building Priority is a list of buildings allowed to build for bots. Bots will create them in order of list.

Units Categories contains a list of **ProductionCategory** objects, which can be built by bots. For example, if you add default Factory Building to **Building Priority** and want bots to build units from this factory, you should add to **Units Categories** its production category. In template this category is **Vehicles**. Removed in **1.7.0**, use **Units To Build** instead.

Available from version 1.7.0

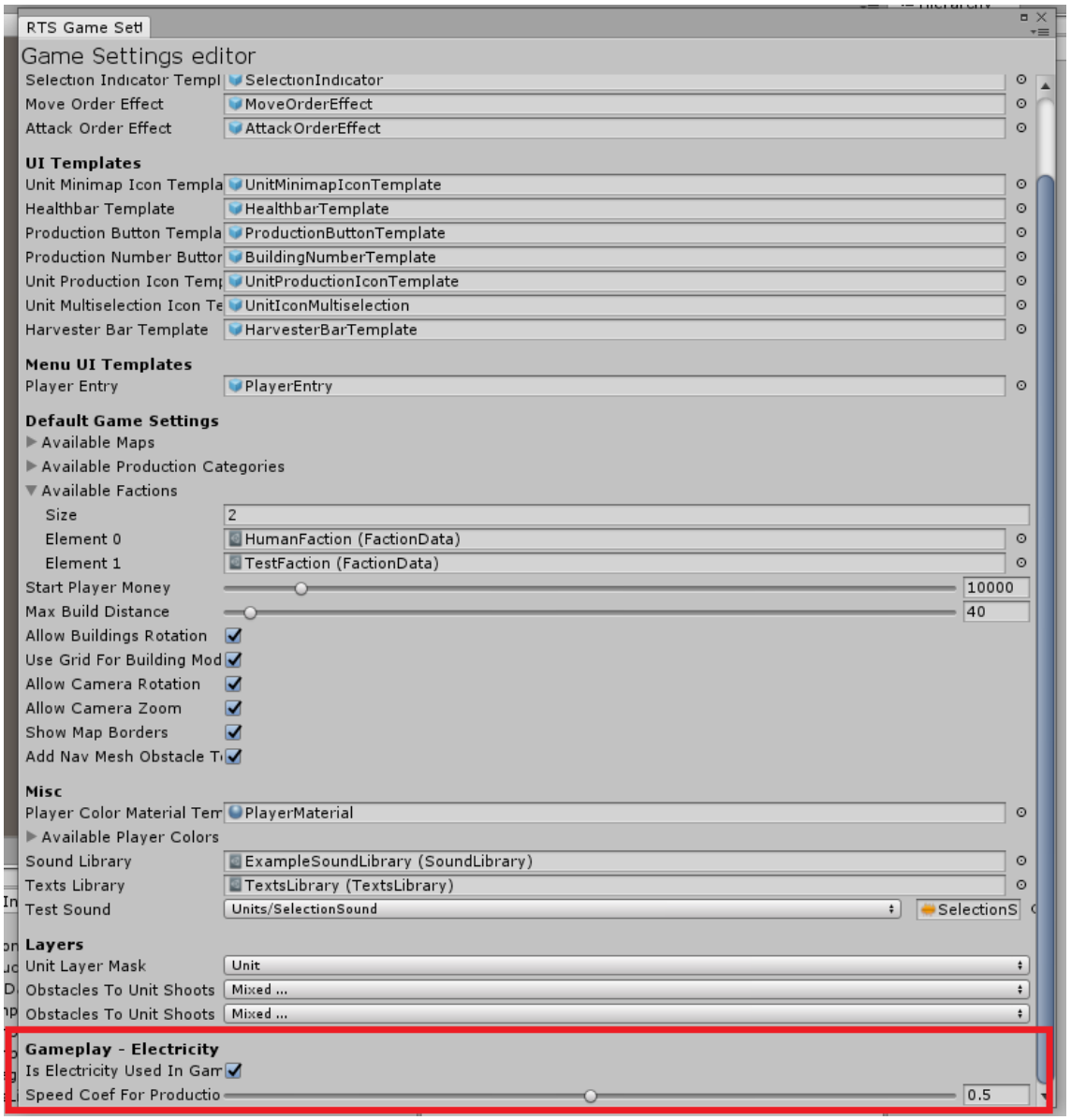
Units To Build is the most important thing here. You can setup, which units and in which order will be built by AI. You also can set time before they will be built and time after. More info you can get by pointing by cursor to its elements, there appears a detailed Tooltip.



Electricity mechanic

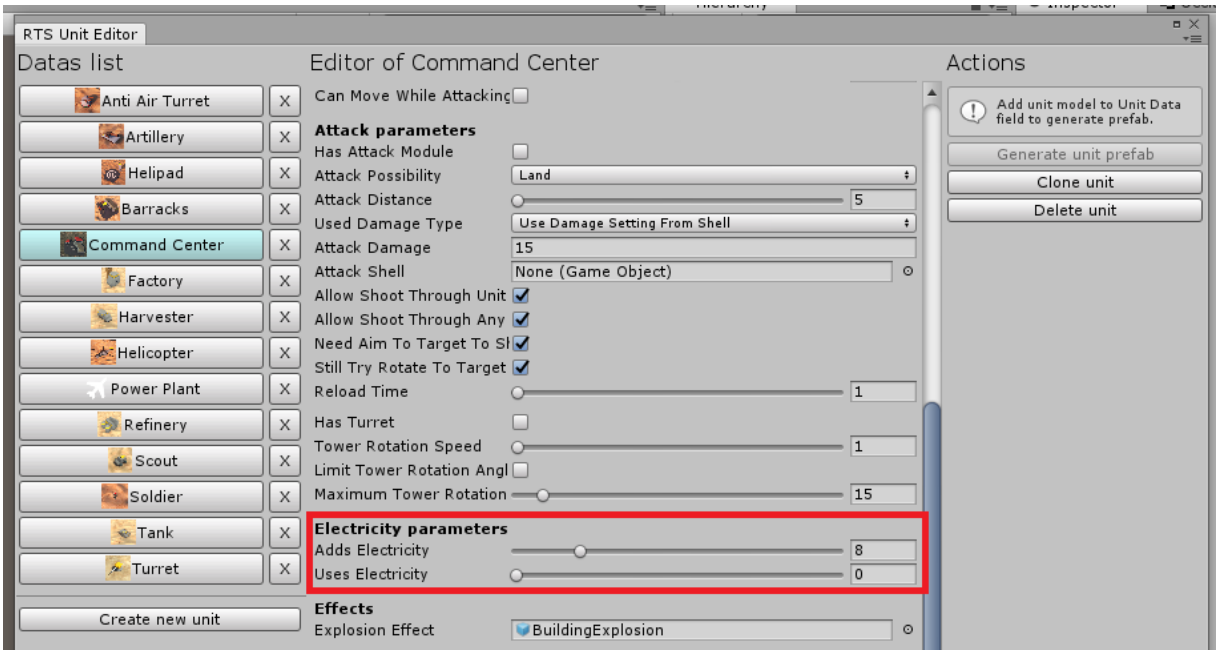
Electricity is used in a lot of classic military RTS, so we decided that it will be good, if thing like this will be implemented in our asset.

This means that some buildings will produce energy (power plants, etc), which will be used by other buildings (factories, etc). To enable/disable this mechanic, you just need to edit this toggle in **Game Settings** (you can find it in top menu - RTS Starter Kit or just search Storage file, will be described below):



So, if a player is out of electricity, his units productions will be disabled or slowed down until electricity will not be restored.

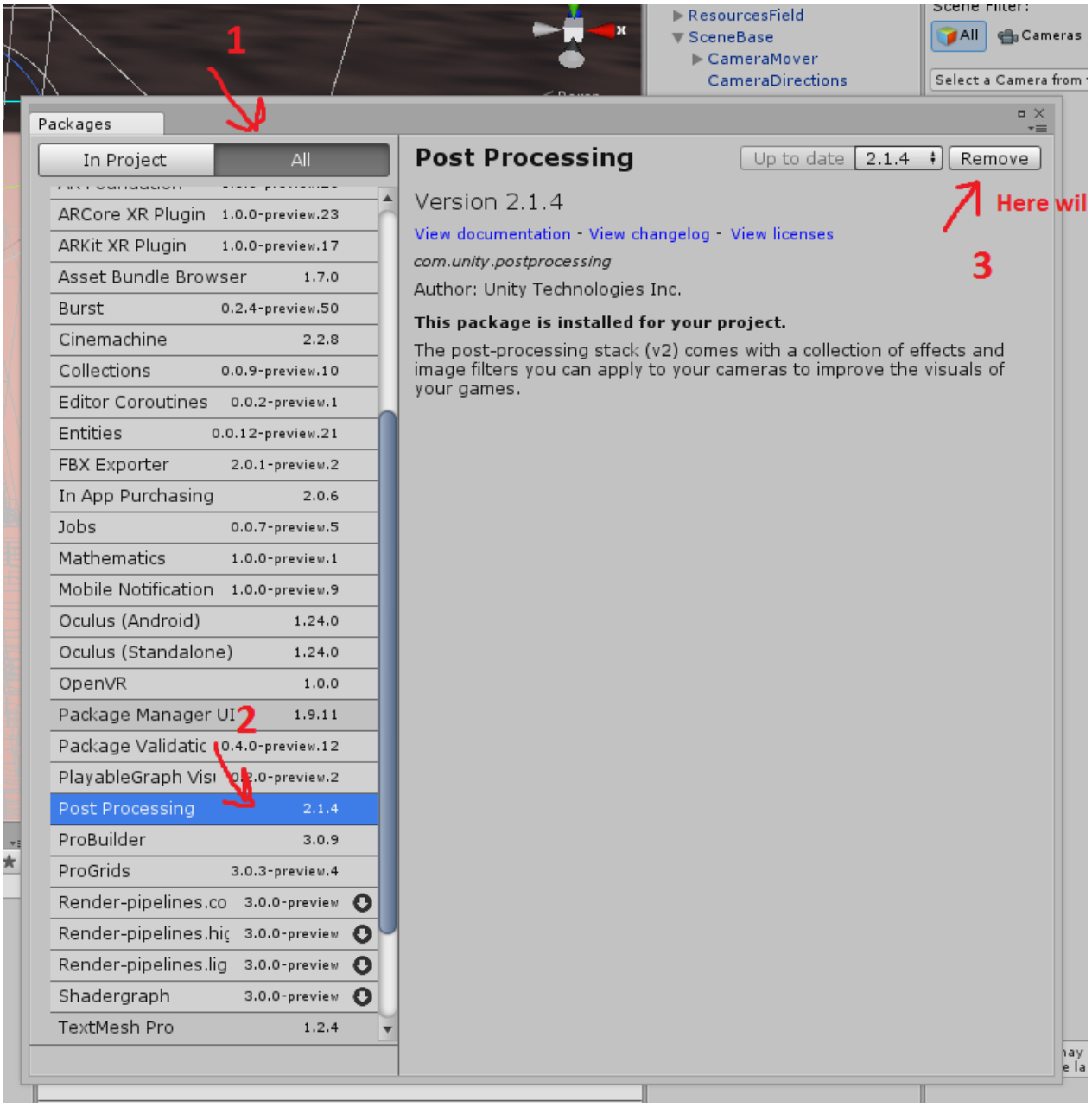
Change adding or using by units electricity you can in Unit Data of needed units:



You also need to add an **Electricity Module** to any unit, which adds or uses electricity. By generating a unit prefab it should be done automatically.

Asset dependencies

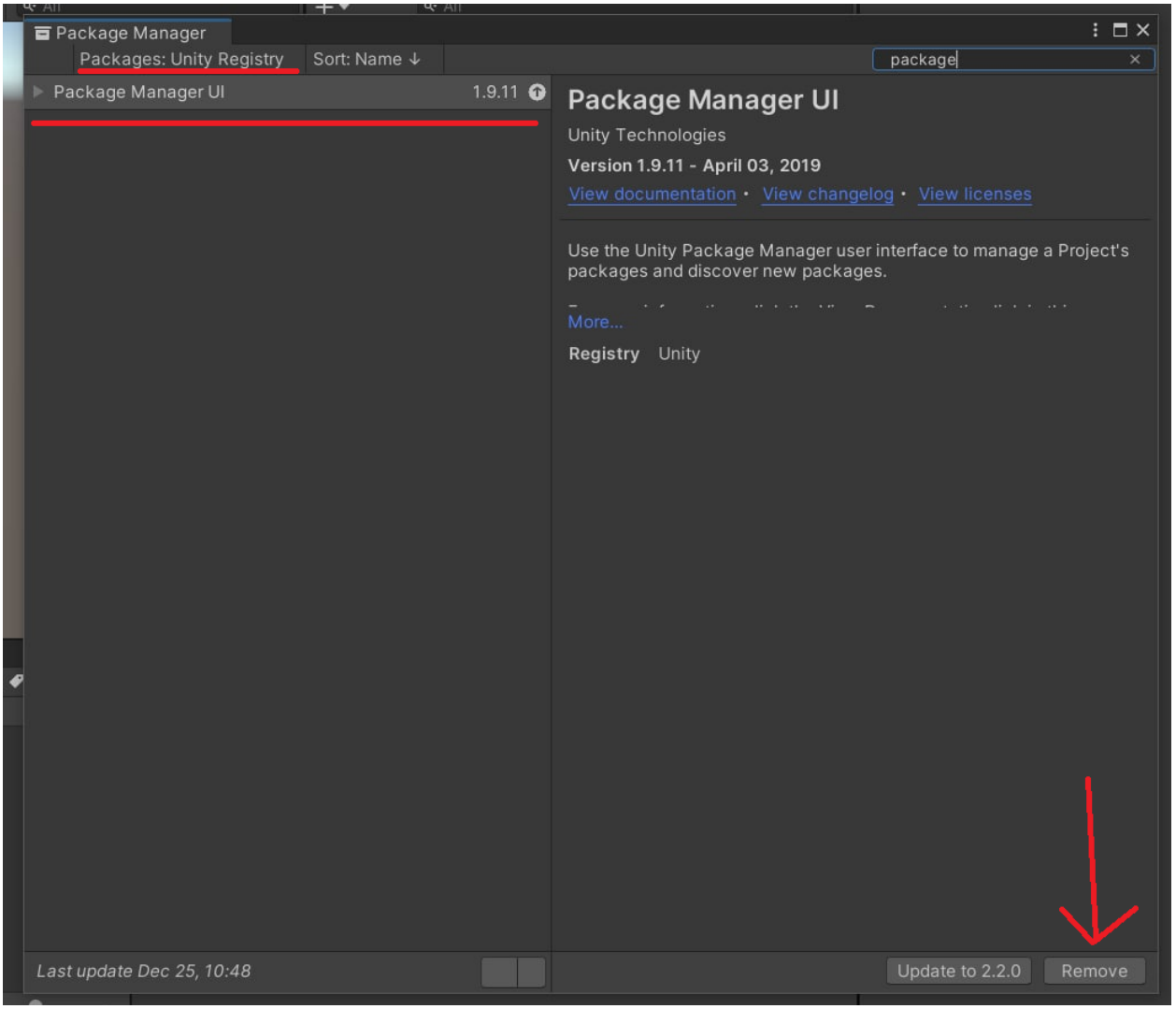
Asset uses **Post Processing Stack v2** for better graphics, so if you don't add it in **Package Manager** or by other way, it will not work. To add it from package manager, select in Unity top menu **Window** > **Package Manager**, in opened window do next steps from screenshot:



Known Issues

Package Manager UI problem (a lot of errors in console)

Caused by old dependencies (to support previous Unity version). You need just remove Package Manager UI from Package Manager:

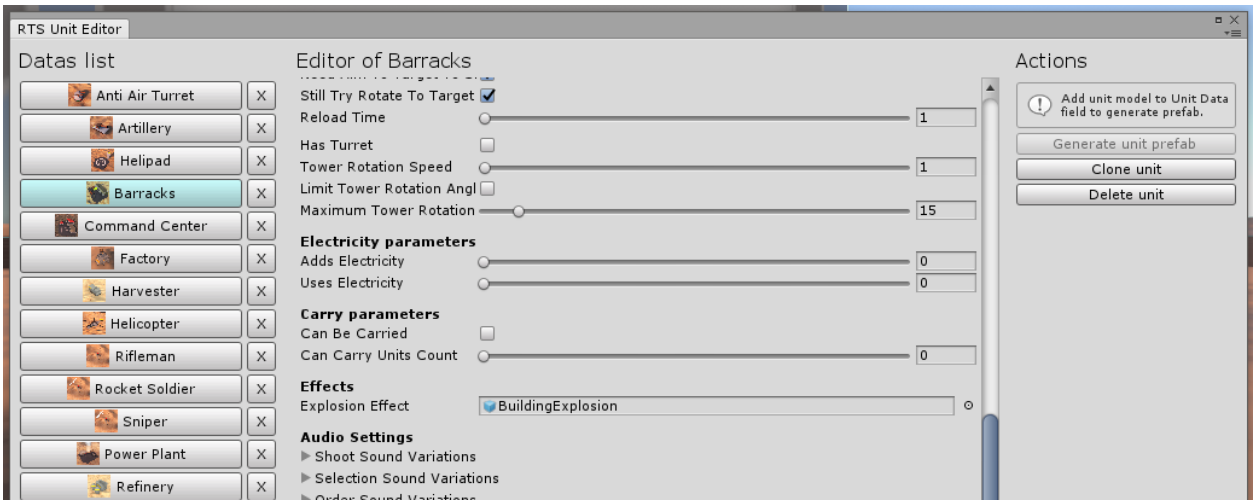


Same issue can be with the Unity Collab - remove or update Version Control package in this case.

Settings and customization

How to change unit settings?

You need to open **Unit Editor** (top menu -> RTS Starter Kit -> Unit Editor) and select wanted unit:

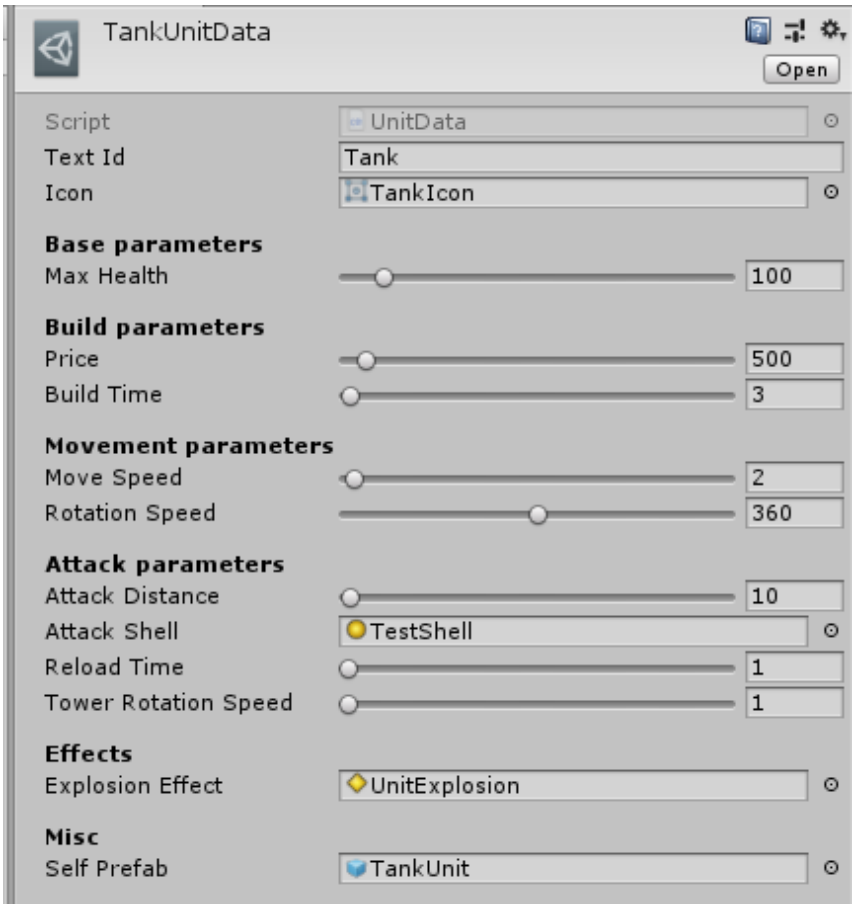


Unit Editor

Now you can edit all unit settings. Note that this all stored in **UnitData** files in **Resources/Data** and you can edit them too, it is just not so comfortable.

All settings are easy to understand, some of them have help hints (you just need to move the cursor to the setting name and wait for hint to appear) with additional info.

You can create new **Unit Data** (for new unit), if you need it. You can click **Create new Unit** in **Unit Edtior** or create new **Unit Data** object. To do it, just **right-click** in **Project Window**, and in the context menu select **Create -> RTS Starter Kit-> Unit Data**.



Selected unit data file settings example

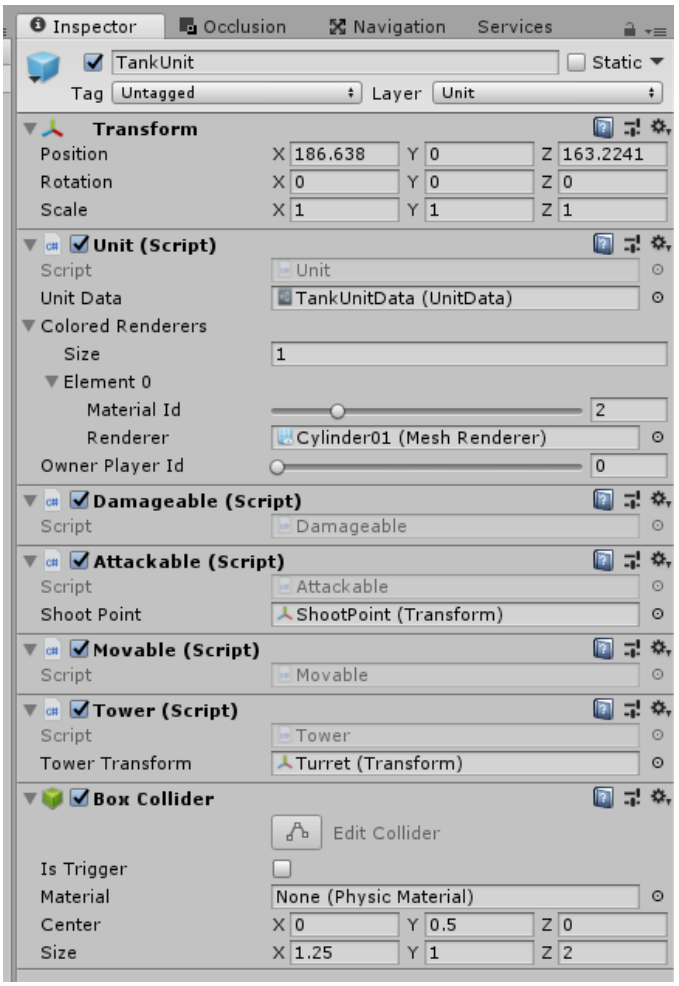
How to setup a new unit?

You need complete few steps to do it:

1. Create (or clone existing in example) prefab for unit, add **Unit** component to it. **Unit** object should be in layer **Unit**.

2. Create **Unit Data** (how to do it you can read in the previous section) with settings for this unit, set up your settings.

3. Drag And Drop this **Unit Data** to **Unit** component on your prefab in the "**Unit Data**" field.

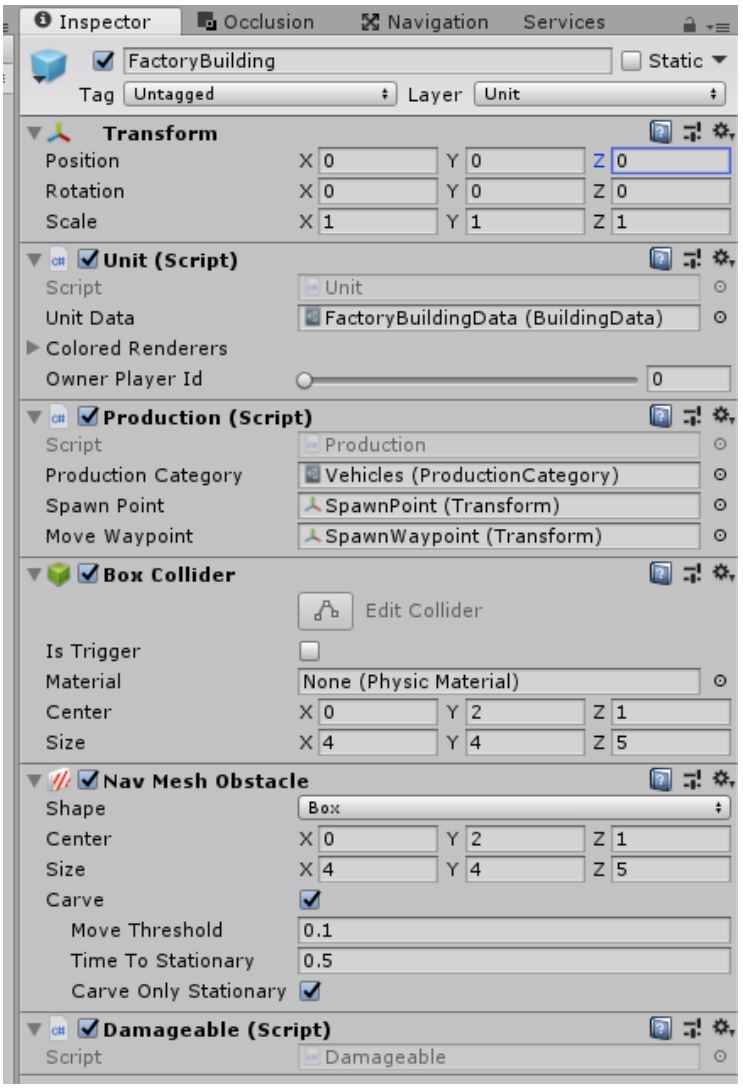


Now you can add other modules to the unit to customize it as you want.

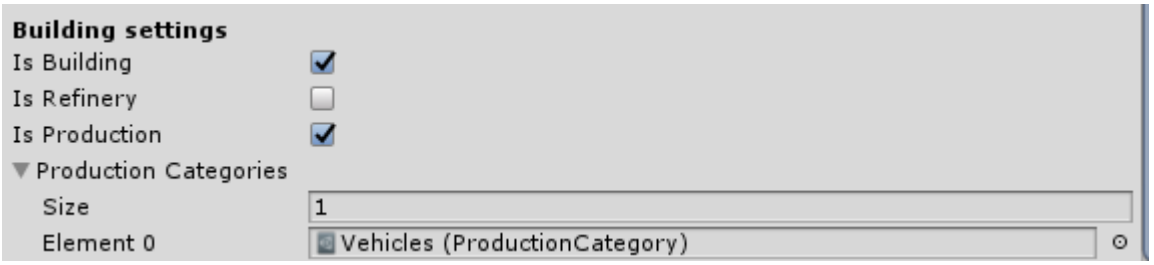
You also can **generate prefab automatically**, this is best way to setup base of unit from its data. After doing it you can customize its prefab as you want. See **Automatic unit/building setup** partition below for more details.

How to setup a new building?

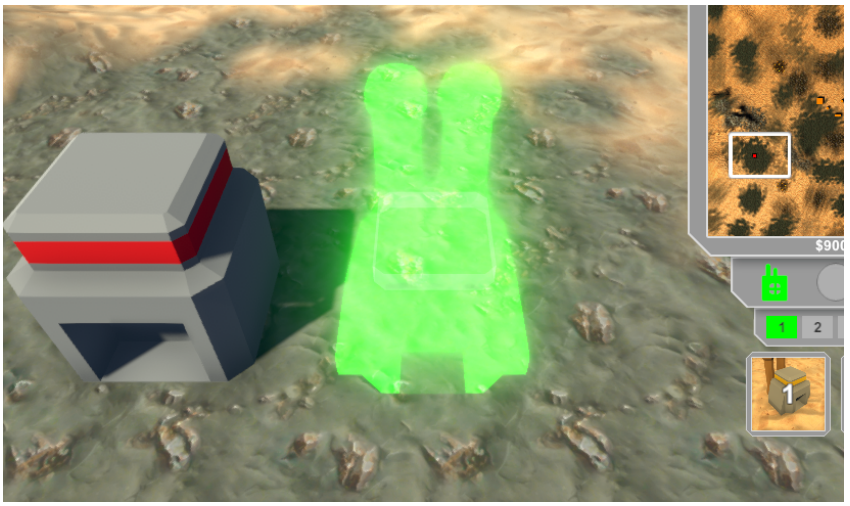
Same as usual unit, but set toggle **Is Building** to **true** in **UnitData**. You also can add **Nav Mesh Obstacle** with **Carve** parameter on to better units pathfinding. If you don’t do it, **Nav Mesh Obstacle** component will be added automatically when building spawns, and its parameters will be the same to building **Box Collider** parameters.



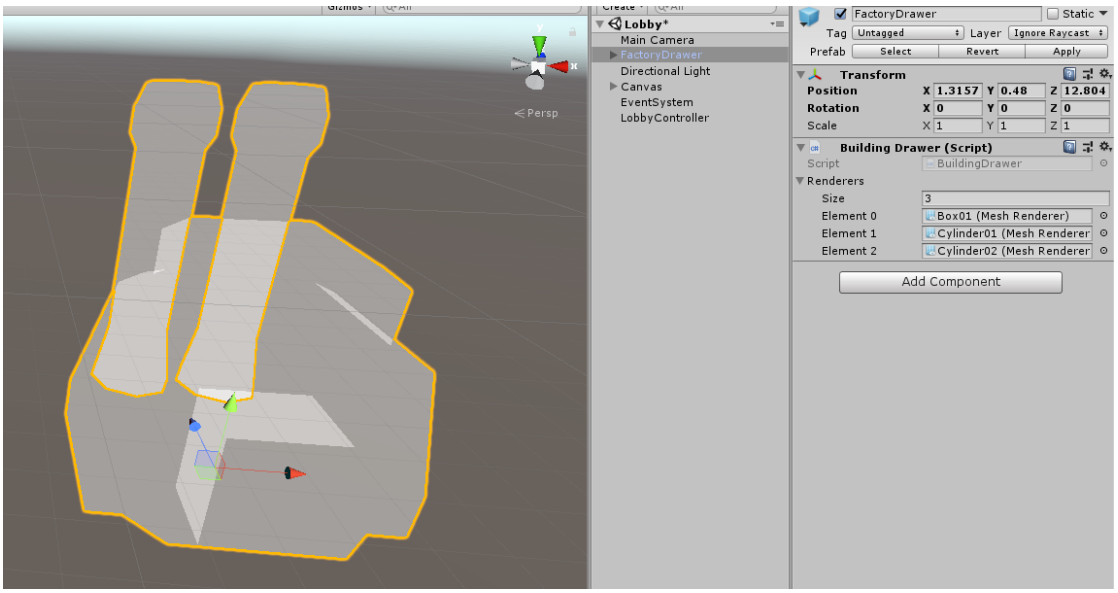
Also for buildings, which should produce units, you need to add a **Production** component and setup it with needed **Production Category** object and other settings. You need to toggle **Is Production** in **Unit Data**, and fill **Production Categories** list in this file like this:



Buildings have one feature – **building drawers**. This is object, which shown when player is in Build Mode:



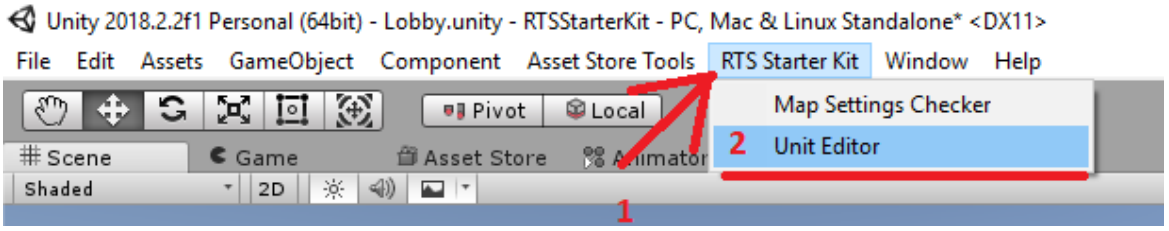
You should set up it for every building, which will be available to build by the player. To do this, you need to create an empty object, add a model of needed building to it, and add the **Building Drawer** component to it. Next, drag'n'drop all meshes of the building model to the **Renderers** field of **Building Drawer**. Finally, it should look like this:



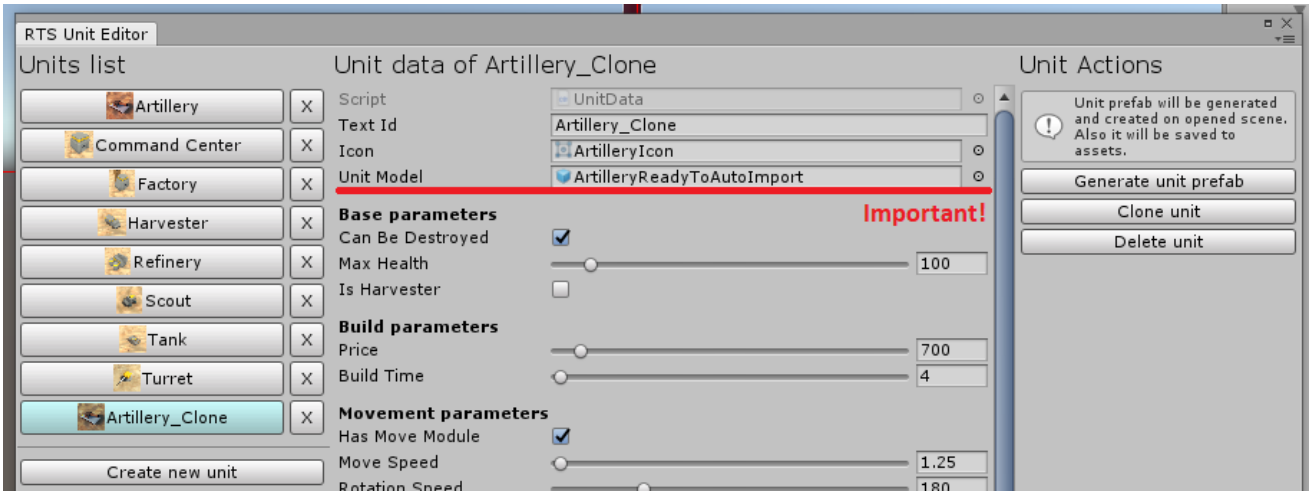
Transparent material named as **BuildingDrawer**, you can find it in **ExtraResources** folder and add to every mesh of the drawer. You can also use your own material type with another effect.

Automatic unit/building setup

Since version **1.4.0** you're able to setup units semi-automatically. It can be done using **Units Editor** window. Open this window by clicking **top menu -> RTS Starter Kit -> Units Editor**:



In the opened window select the needed unit (if you need to create a new unit, click **Create new unit** button. You can also clone another unit by selecting it and clicking the **Clone** button). Setup all parameters as you need, and don't forget to fill **Unit Model** field by model of your unit from assets:



Finally, you need to click the **Generate unit prefab** button in the right part of the window. After click, the created unit prefab will be spawned on scene and saved in **Prefabs/Units** folder. A lot of it's settings will be automatically configured, but there still some params, which you should setup manually:

1. Unit colored renderers - parts of unit, which shows its house color (read next partitions)
2. Collider size - size of your unit.
3. Fog of War module parameters - renderers/gameobjects of unit which should be hidden when unit moves to fog of war.

Other settings can be adjusted manually to, if you want.

Important thing is the auto setup of the model and bones.
First, your model will be rotated properly only if it's pivot is right for Unity (Z-axis is forward axis of your unit model).
Second, you can add several bones (empty objects or model parts) to your model, which will speed up the setup of the unit. It's namings listed below:

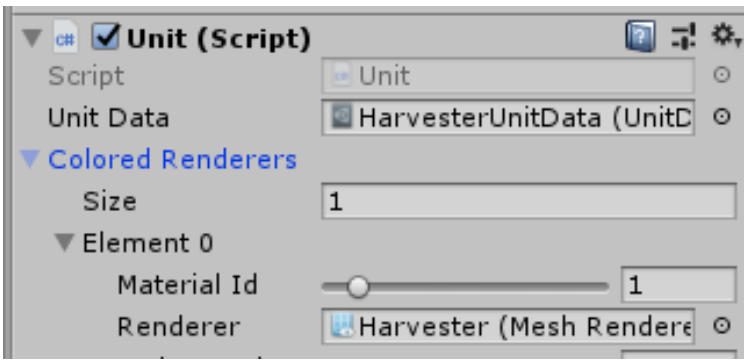
1. **WheelBone** - if a unit has wheels, name it's bones like this, it will be automatically used by scripts.
2. **ShootPoint** - points, from which will shooted bullets by unit. In most cases you need only one shootpoint.
3. **TurretBone** - if your unit has a turret, you need to name its bone like this to allow scripts to automatically find object to rotate the turret to enemies.

You can find an example model of the unit in the project, just input in search **ArtilleryReadyToAutoImport** phrase.

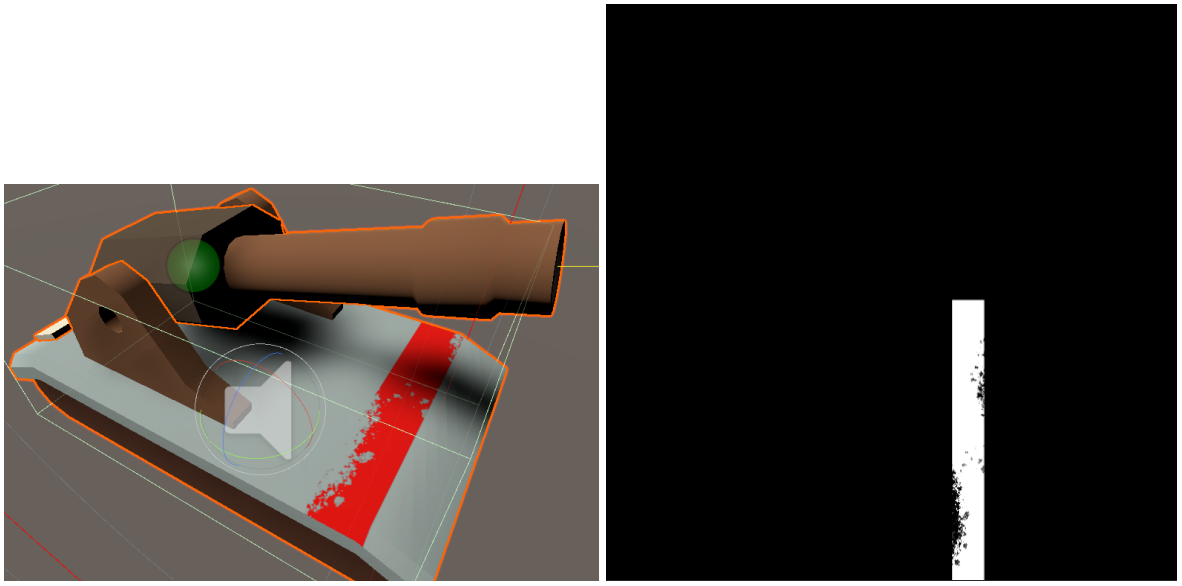
You can don't use these namings, but in this case you'll need to setup all these parameters manually (read next partitions).
In future there will be added more bones variations for automatic setup.

How to set up house color parts of units?

Unit component have field **Colored Renderers**. It is an array. Here you should add fields for all of the unit mesh renderers, which have house colors. After, add one mesh renderer per field, and if the mesh renderer has several materials, select needed material id number (number of material, which should be colored). Remember, that material numbers start from 0, not from 1.

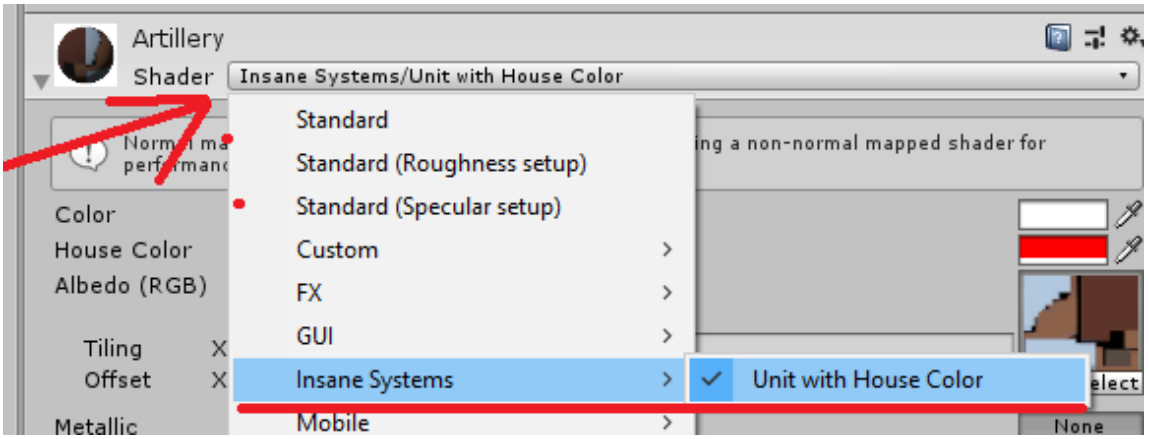


You also can use **house color texture** instead of coloring the whole mesh. House color texture is same to emission texture, but it marks unit texture parts, where should be drawn house color, for example, check Artillery unit:

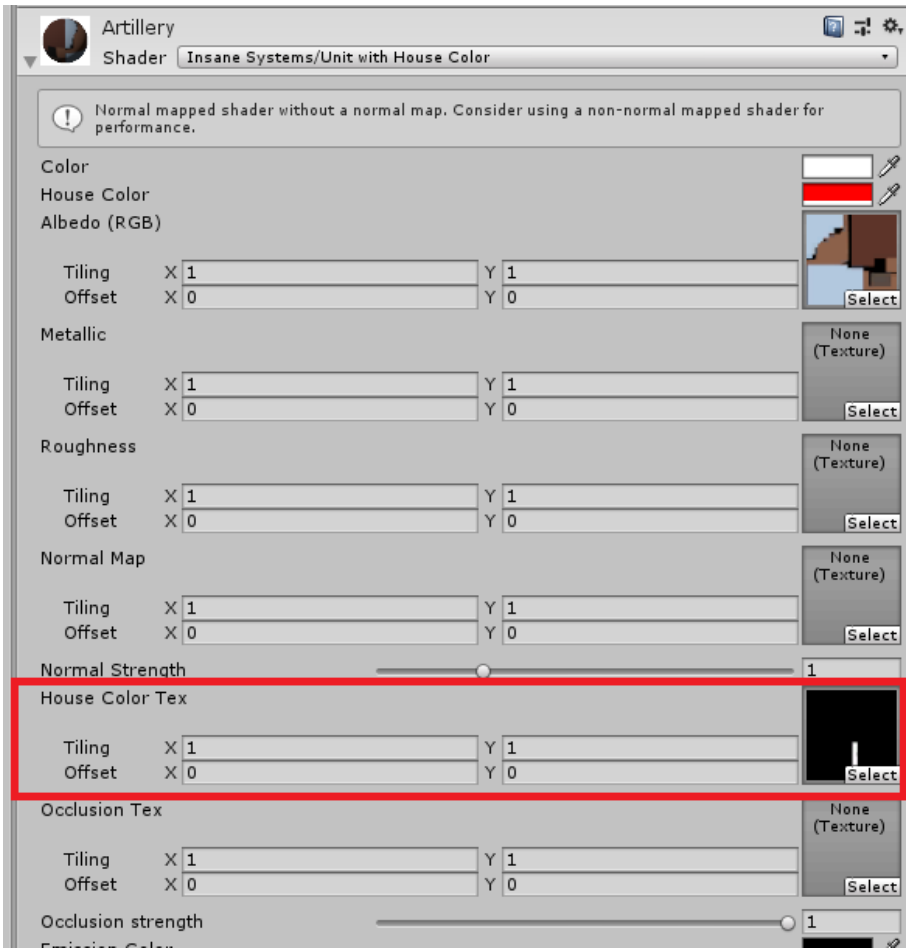


Right image - house color texture, left - how it is drawn on the unit.

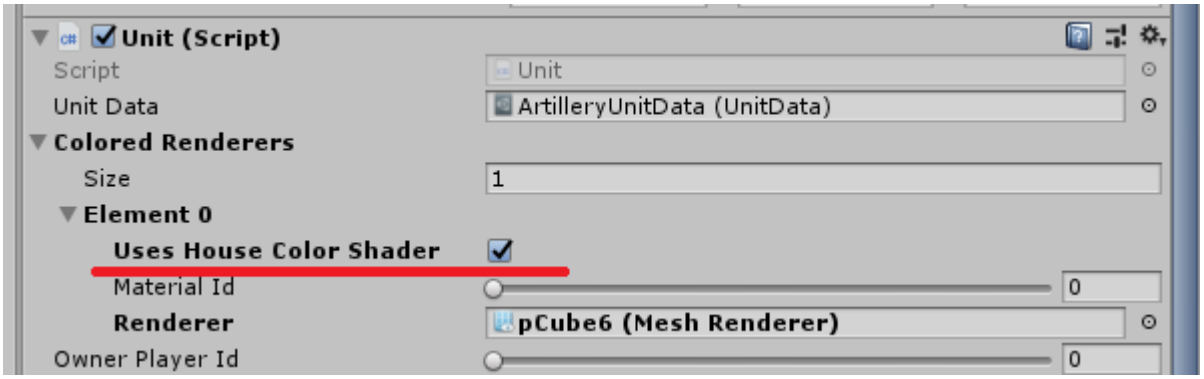
To use this type of house-coloring, you need assign unit model special shader named **Unit with House Color**:



It works in metallic + roughness flow, supports normal map, occlusion and emission. Assign your house color texture in **House Color Tex** field:



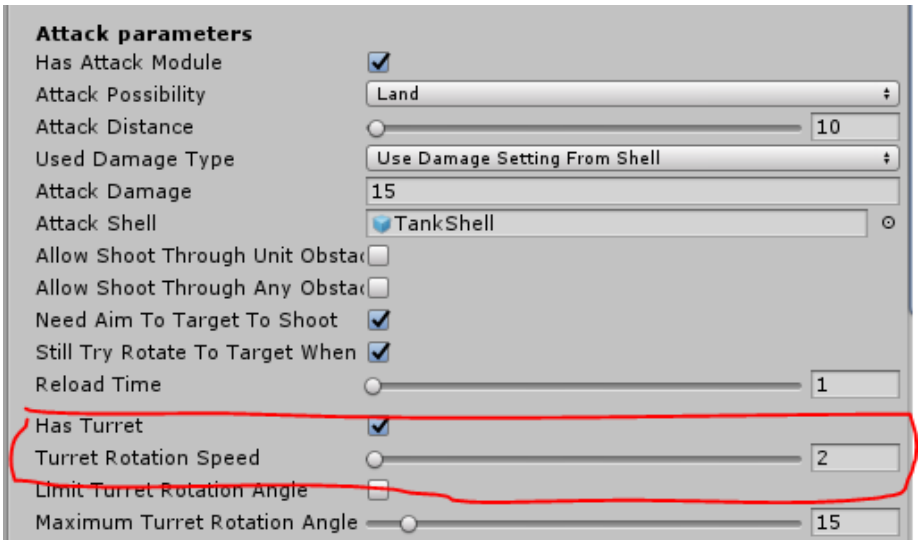
Finally, don't forget to enable toggle **Uses House Color Shader** in **Unit** script parameters:



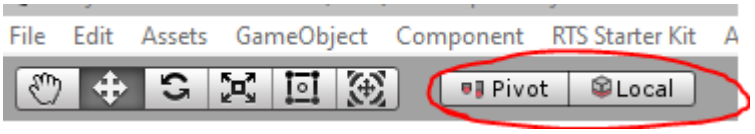
This approach can boost your performance and more flexible at all.

How to setup units with turrets?

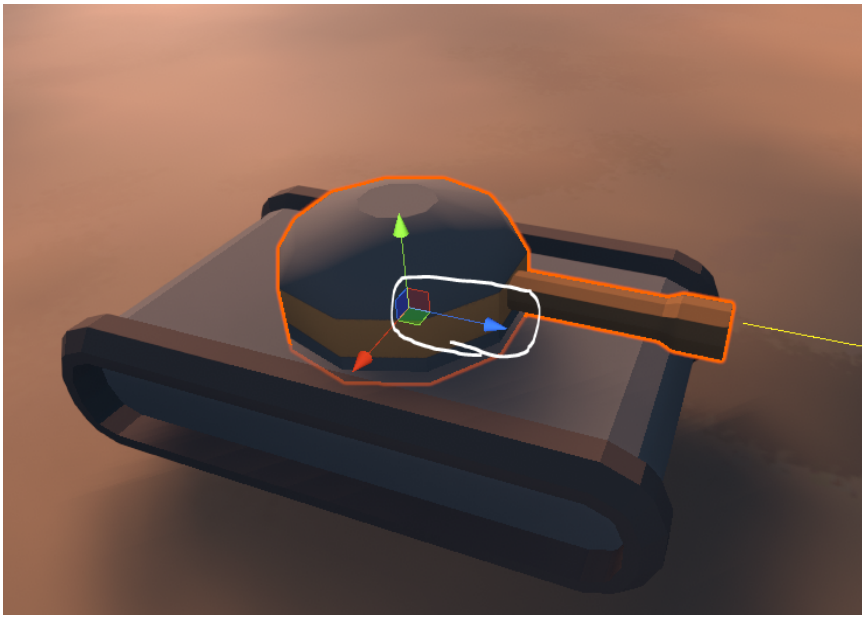
If you have a unit, for example, a tank, you, of course, want its turret to aim targets. To setup it, you need to check **Has Turret** toggle in **Unit Data**, and setup **Turret Rotation Speed** greater than 0.



Next step is setting up of the prefab. There is one important thing. Turret of your unit model should have the **correct pivot**. It should have aimed in the Z-axis direction, same to Unity. How to check it? Select turret of your unit, select **Pivot & Local** at top of the Unity Window:

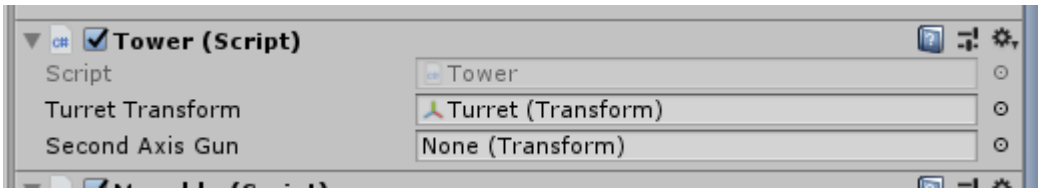


And check that turret aimed same to **blue arrow**:



If it isn't correct, you can fix it in any modelling application like Blender, 3ds Max, Maya, etc.

Next step is setting up the prefab of unit. If you're generated it automatically it should be already setted up, to validate it select prefab and check **Tower** module parameters:

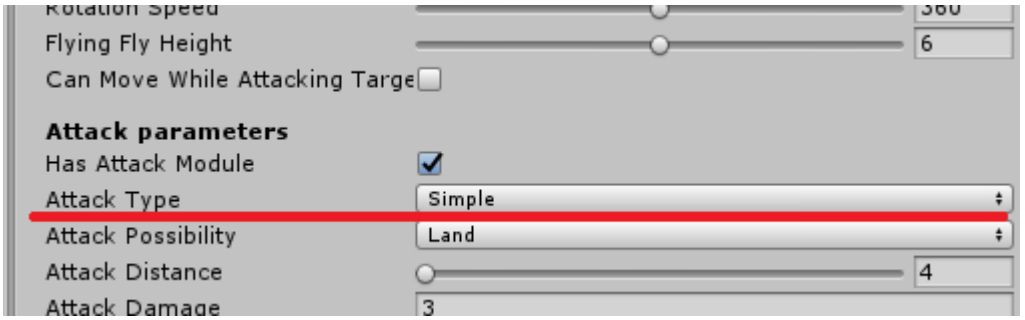


Turret Transform field should contain bone of your turret or turret model itself, depends on your modelling pipeline. That's all, now the unit turret should work. Note that **UnitData Has Attack Module** parameter also should be enabled.

Melee-ranged attack type

Available from version 1.6.10

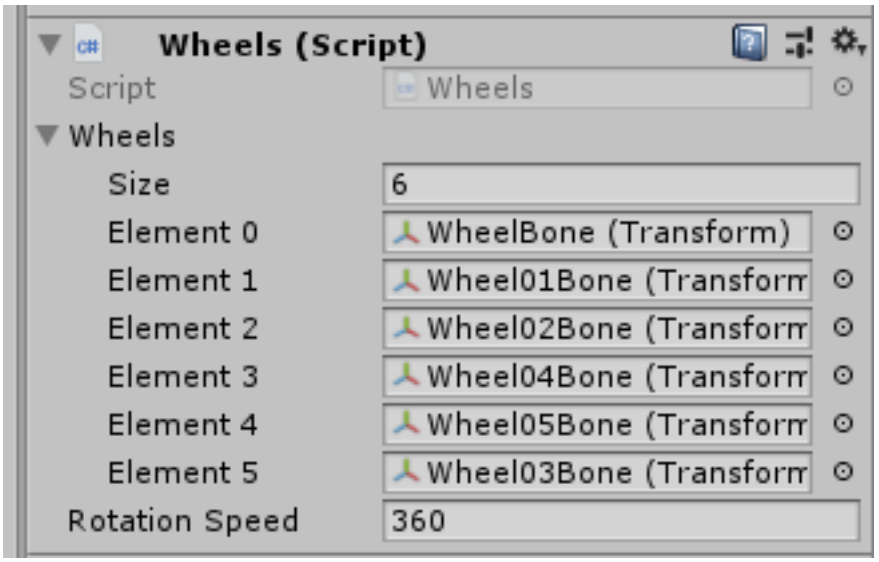
Some units (infantry usually) can have Melee-ranged attack. To setup it, you need to change the **Attack Type** parameter to a **Simple** value in the **UnitData** in **Attack Parameters** partition.



Now your unit will damage the target directly (no Bullet/Rocket will be spawned, field Attack Shell with attack prefab will be ignored). It still will use the **Attack Distance** condition, which prevents attack from the distance above this value.

How to setup wheels for vehicles?

Better way to explain it - ask you to look at the default Harvester unit. It has a fully tuned **Wheels** component.



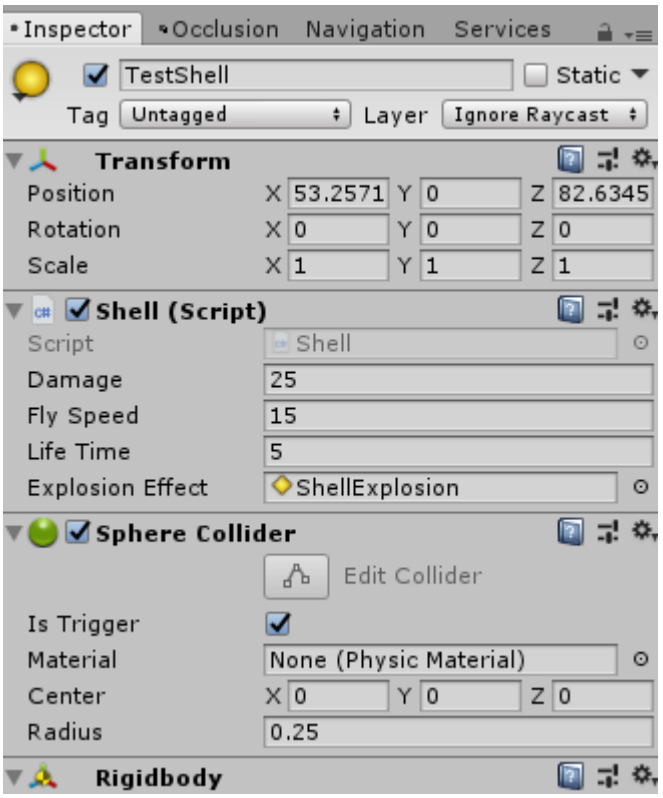
You should add all wheels to the array. If wheels have different pivot bone settings, you should add new identical bones and make each wheel child for its bone, and next add these bones to the array.

Rotation speed is in degrees in second. It means that in current settings (360) wheels will fully rotate once per second.

Shell - create custom rockets and bullets

Shell is a bullet or rocket prefab, which will be used as a unit shooting object if units have the **Attack Type** value set to **Shell**.

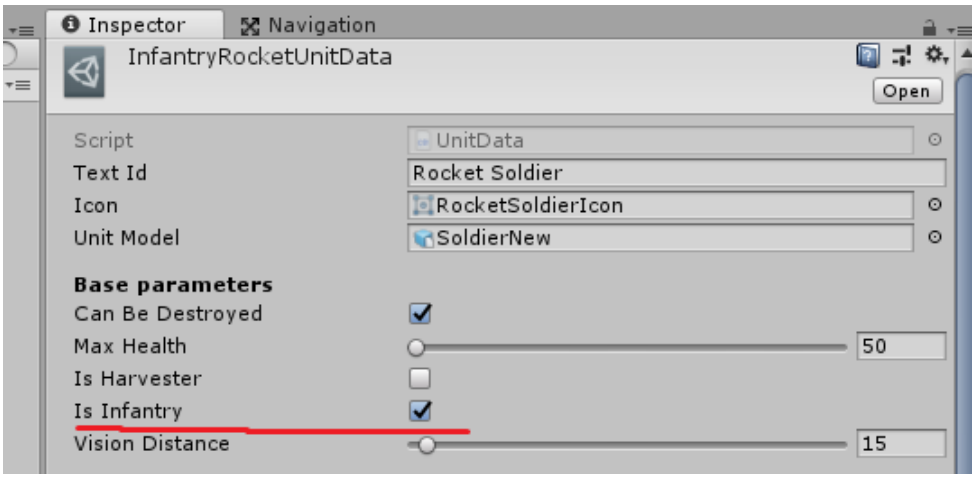
You can clone existing asset from the Prefabs/Shells folder and customize its settings as you want. Settings are easy to understand - damage value, fly speed, bullet lifetime and explosion effect:



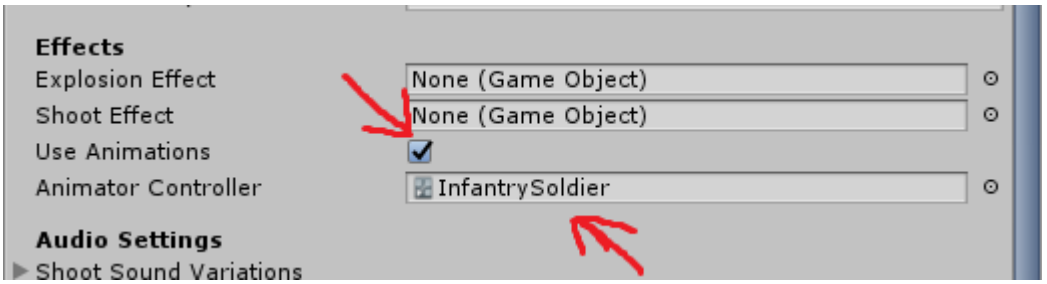
To edit bullet physical size, change **Radius** value of the **Sphere Collider**. You can use other collider types, if you need, just change the component. But this one is a better to any calculations.

How to setup infantry units

First of all, you need to setup specific fields in Unit data, it shown on screenshots below



Is Infantry simple marks this unit as infantry.



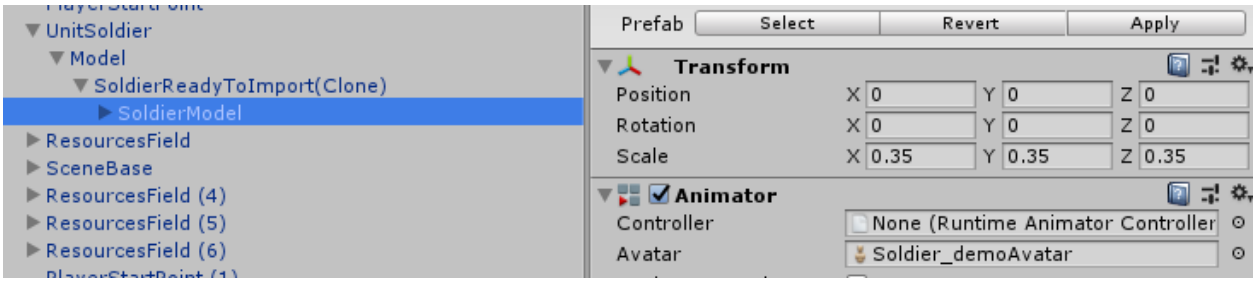
Animator Controller needed to allow animated character play its animations (our asset will setup it to Animator in prefab automatically)

Use Animations says to the code that this unit have animations like move, attack etc.

Now you need to make changes for infantry-specific needs in your unit prefab.

On automatic prefab generation it will do it, you know, automatically. :) But if something goes wrong or you want to do it manually, you need to add **Infantry** and **AnimationsModule** component to the unit prefab, and setup **AnimationsModule** field **Animator** - it will allow unit to play animations:

This animator can be found in one of childs of prefab. If there no one in your model, add it manually:



Setting up the animated character model is the same to the default Unity pipeline.

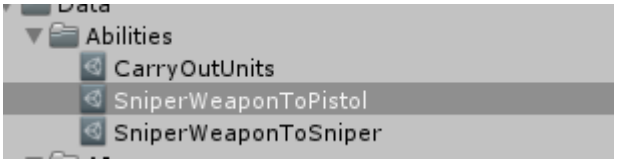
After these things are done, it should work fine and play animations, if you added it in your Animation Controller.

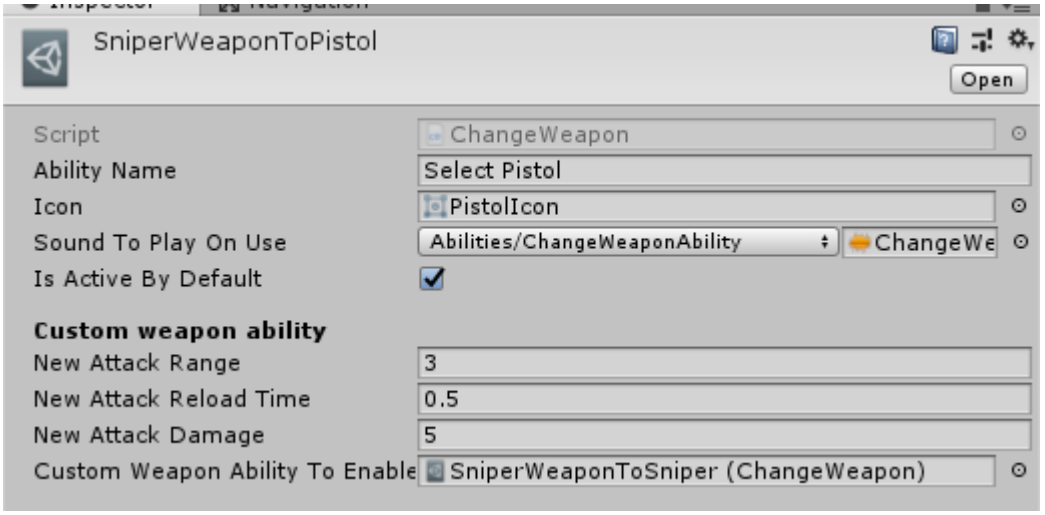
For more info you can check our asset example infantry soldier prefab, animator, etc.

Unit abilities, how to use them?

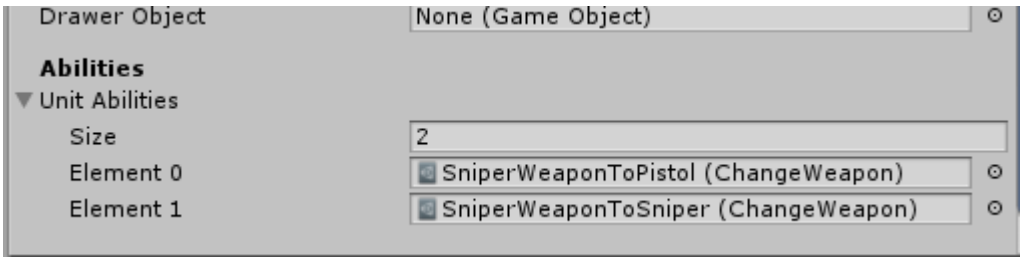
RTS Starter Kit allows you to create abilities for your units. For example, change weapon, which will allow your infantry to change weapon from rifle to pistol. Or throw a grenade, and anything else you want. In fantasy setting abilities called spells usually, so here you should totally get it.

You can look for example abilities by searching the project for Abilities folder, there will be one with scripts, and one with Data-files.





Ability script - is the core of any ability. It designed to be data-file, which automatically handled in code of unit if you add it to **UnitData** of your unit:

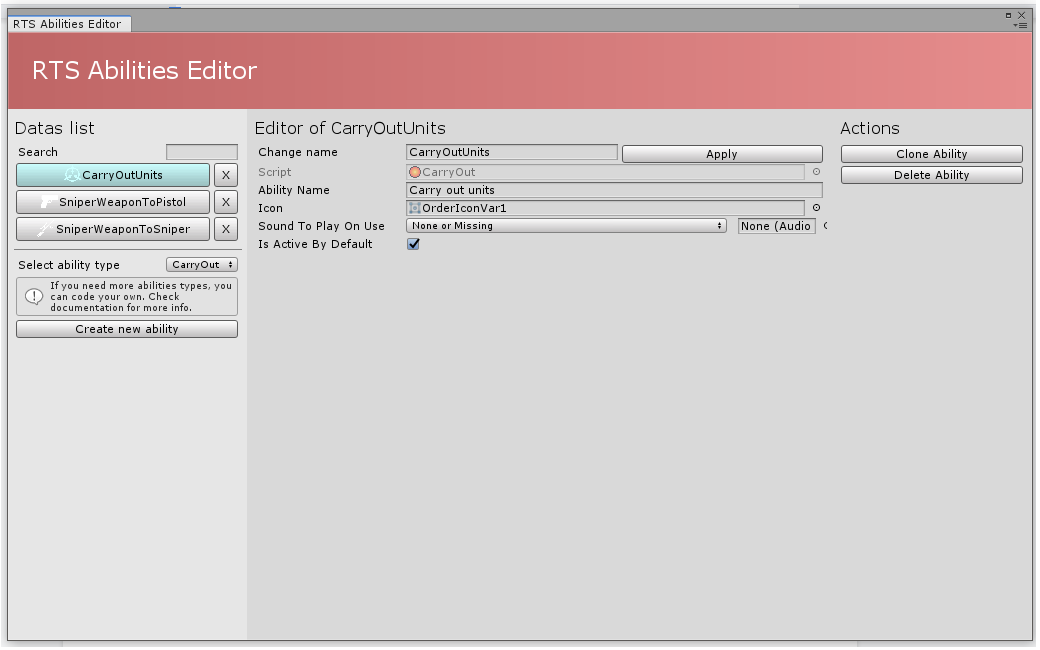


New ability data can be created by right-click in **Project Window** menu and select **Create -> RTS Starter Kit -> Abilities -> “NeededAbility”**

There are not so many ready-made abilities, because our primary target here is to give you tool to create your own. It can be done by coding custom classes using Ability as parent, you can see more info about it in the **Programming** partition. We made API easy to use, so you should have no problems with understanding it.

Implemented in version 1.6.7 and above:

Also there exists the **Ability Editor** window, which can be found in the **Top menu -> RTS Starter Kit -> Abilities Editor**. It can simplify work with your abilities for a bit.

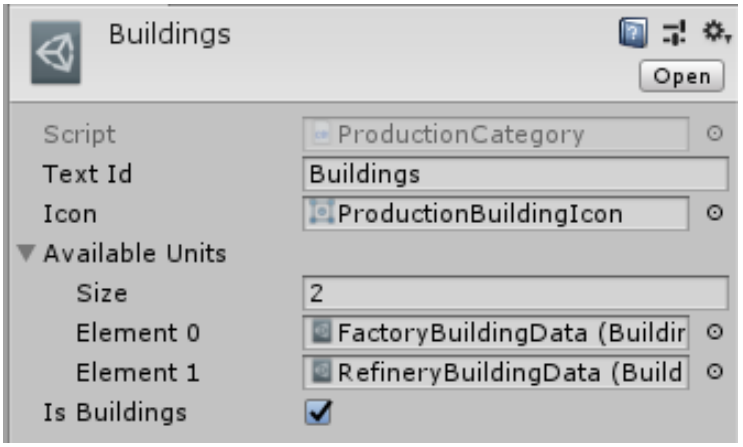


How to add or edit Production Category?

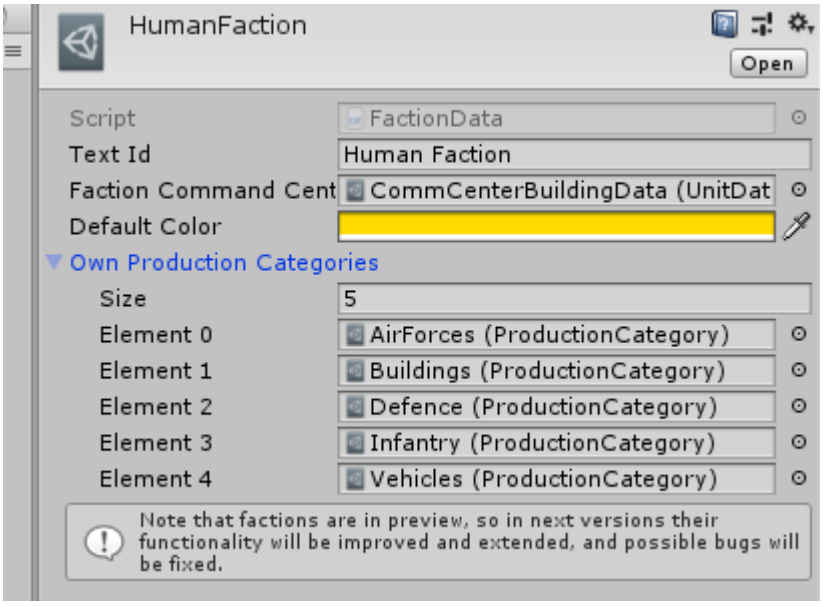
To create a new production category, you need to right-click in the **Project Window** menu and select **Create -> RTS Starter Kit -> Production Category.** But you can clone existing in example project **Production Category**. You can find it in **Resources/Data/ProductionCategories**.

Production category contains next settings:

1. **Text id**.
2. **Icon** of category. It will be drawn on UI.
3. **Available Units** - list of units (unit datas) for this category. It is all units, which will appear on UI when this category is selected.
4. **Is Buildings** parameter. Check it, if this category contains buildings (not units) objects. Otherwise remove the check mark from this toggle.



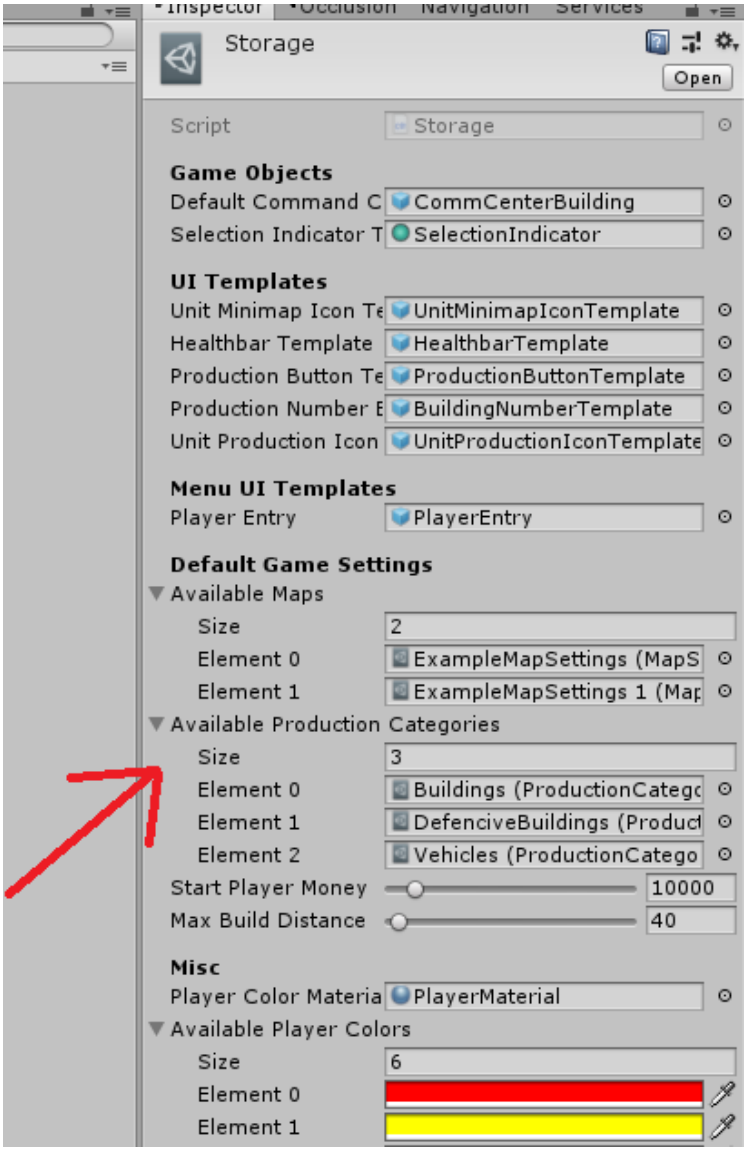
Don’t forget to add new production categories to specific faction, which should have it:



Note that different factions can have the same production category, if you need it for tests, prototyping or smth.

Also note, that in the current version it needed just to show production icons on panel ingame, but if any building of this fraction will have non-added to faction production category, on selection it still be able to buy units of this category.

After adding a new **Production Category**, add it to **Storage** object in **Available Production Categories** field.

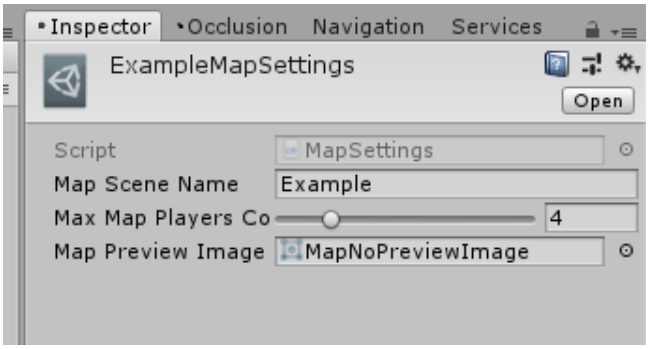


How to add and setup a new map?

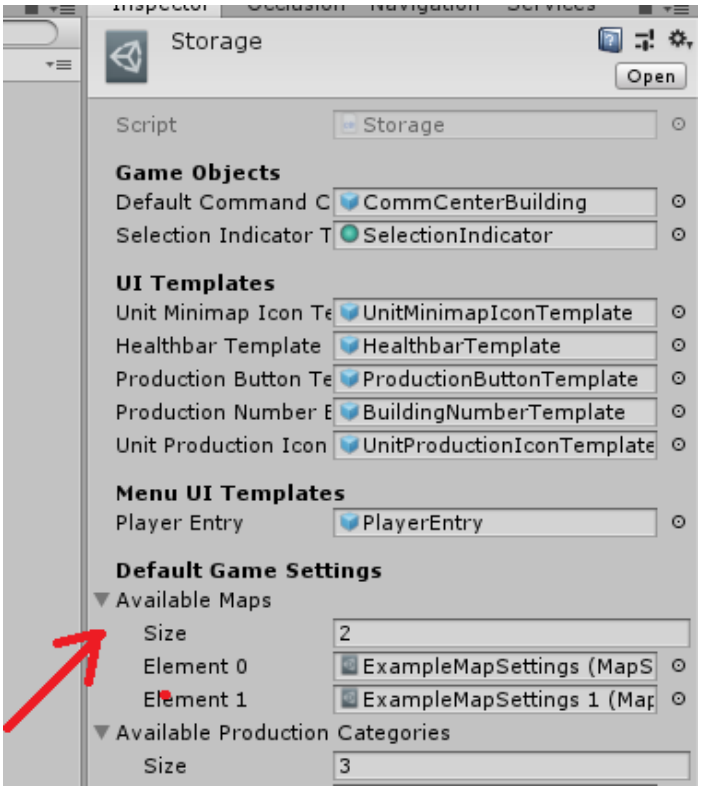
First of all, you should create **Scene** with your map. This scene should have terrain (or any object with collider and marked with **Terrain** layer), baked **NavMesh** (to allow units move) and **SceneBase** prefab from **RTS Starter Kit**. Don't forget to

place **Player Start Points** on the map (you can find this prefab in the RTS Starter Kit Prefabs folder).

Next, you should add this scene to the **Build Settings**. After it is done, create **Map Settings** (right click on **Project Window**, in the context window select **RTS Starter Kit/Map Settings**), and fill its fields (map scene name, map size, players count, etc).



Finally, drag and drop this **Map Settings** object to the field **Available Maps** of **Storage** object.

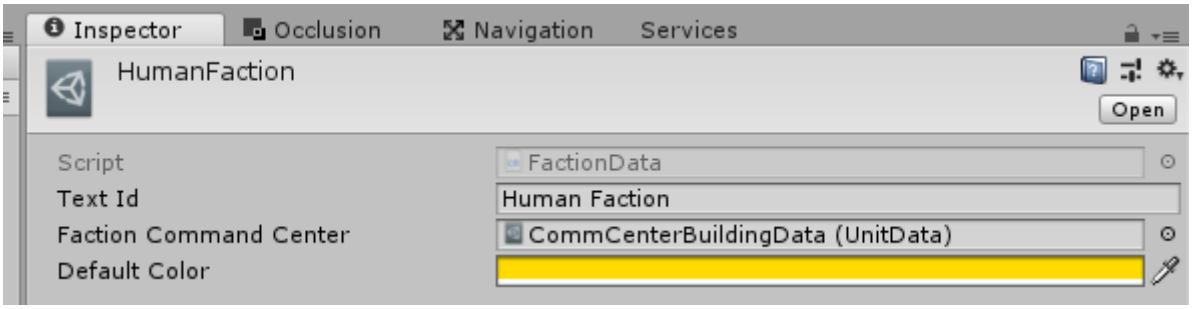


How to add a new faction (race)?

To add a new faction/race you just need to create **Faction Data** and fill it with your data.

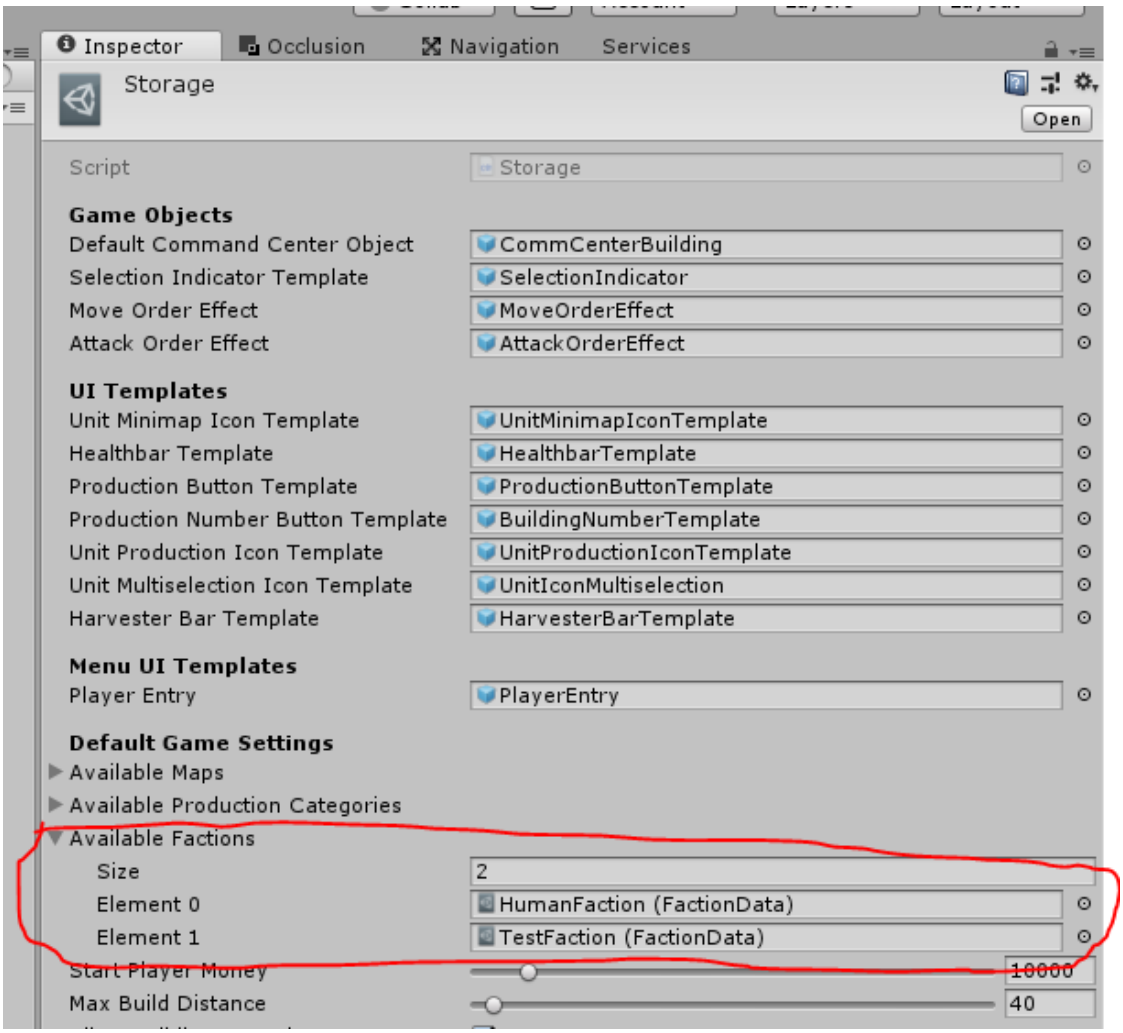
To create new **Faction Data**, you just need to **click RMB** in **Project Window**, and select **Create -> RTS Starter Kit -> Faction Data**.

In **Faction Data** you can setup parameters of your faction:



It is very simple. You write its name and add **Unit Data** of this faction **Command Center** building. All data about available units and buildings already setted up in **Unit Datas** of this faction (command center production categories -> units of these categories -> etc).

Finally, you need add it to the **Storage** into the **Available Factions** field:



Fog of War

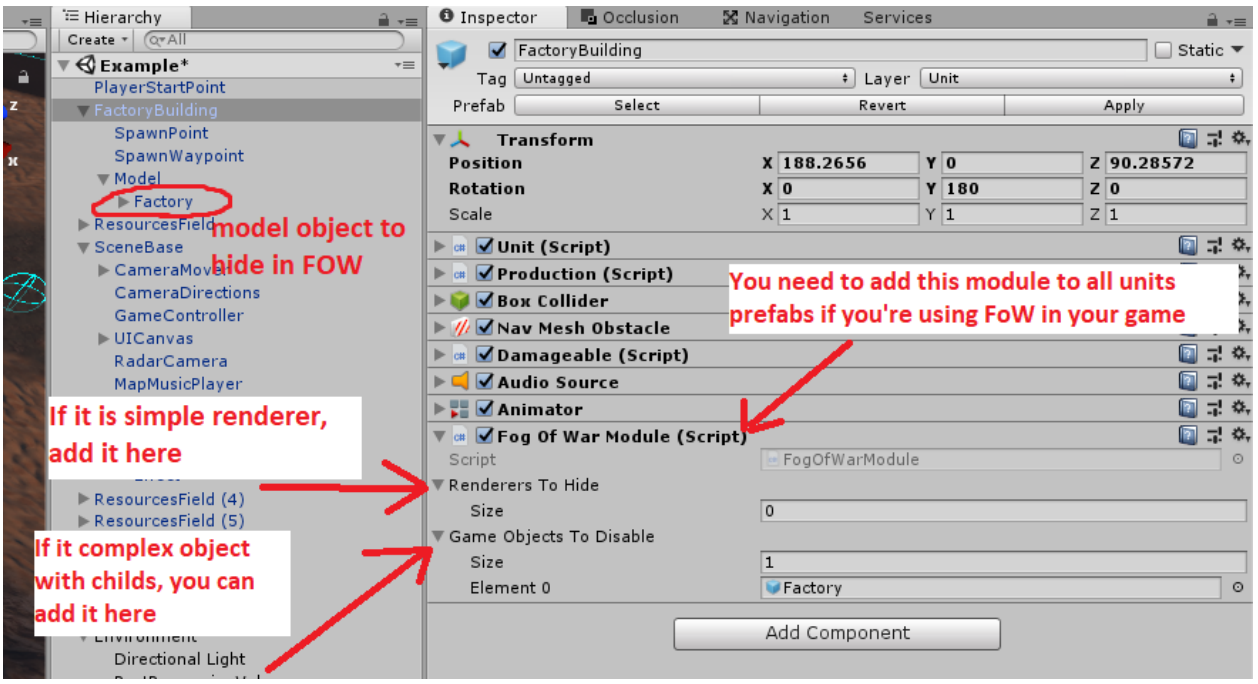
You can enable **fog of war** in the **Game Settings/Storage** asset file. Result shown on screenshot:

This is a classic mechanic for most RTS games. Hides enemies units in fog, your units have ranged visibility.

To change the visibility range of a specific unit, select it in the **Units Editor** or select his UnitData file and edit the field named **Vision Distance**.

Note that fog of war max units count (which have vision in FOW - player and his teammates units) is 1000. It can be increased in shader files manually, but not recommended.

You also need to setup all units prefabs to work in FoW. Screenshot below shows how to do it. You need add the **Fog of War Module** to the unit, next you need to select model in unit hierarchy and add it to the list of hideable objects (Do not add there unit prefab itself, it will break unit in playmode):

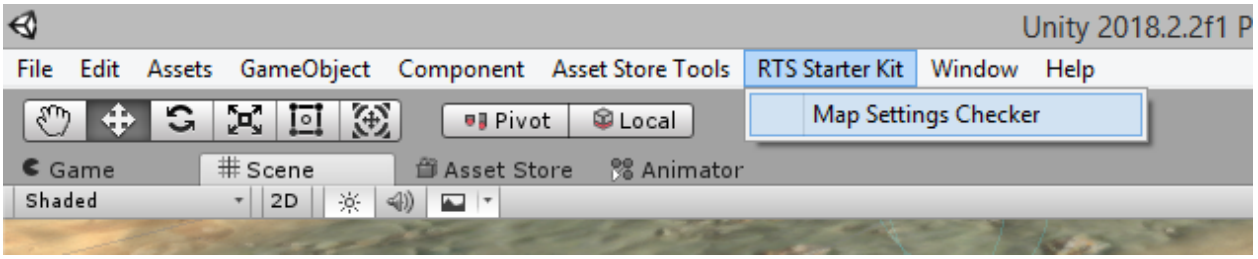


If you don’t do it or do it wrong, the enemy unit will be visible in the fog of war. If you see something like this, re-check your unit Fog of War Module settings.

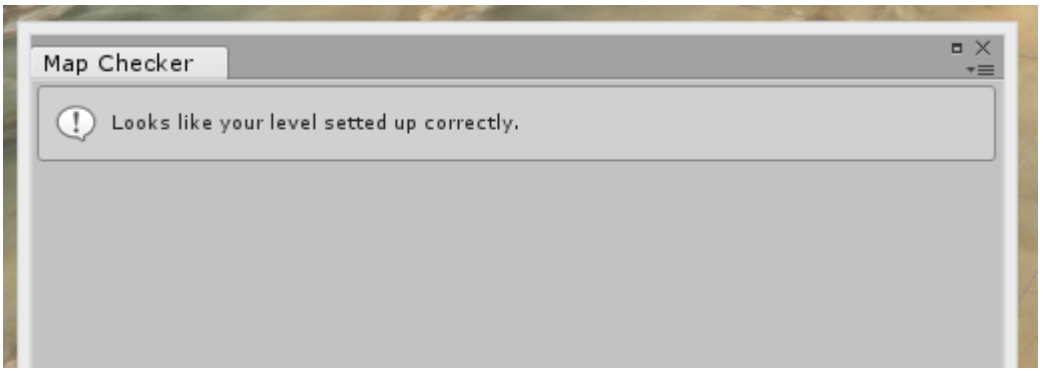
Map settings Checker

Map Settings Checker is a special window, which will check all scene settings and show warnings, if you forget to setup something. If the scene setted up correctly, it will tell you that everything is fine.

You can access this window from top Unity menu by clicking **RTS Starter Kit/Map Settings Checker**:

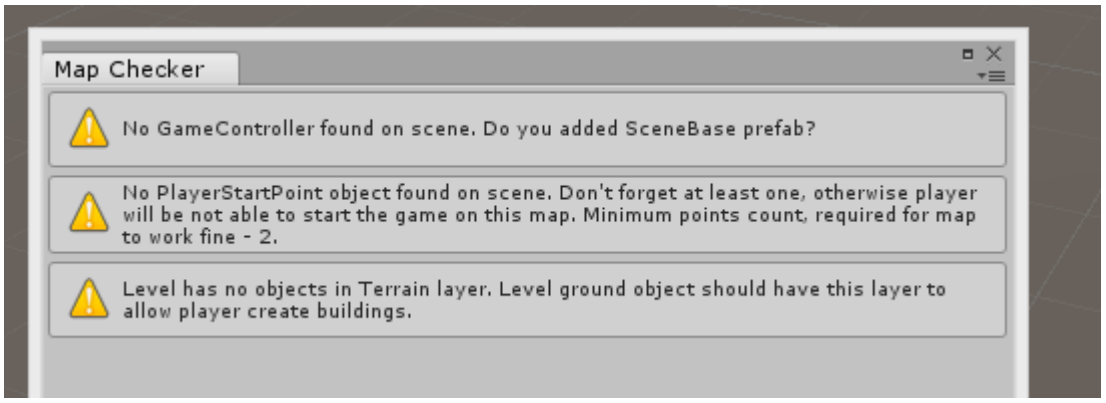


This window looks like this when scene settings is correct:



Map settings checker

Or like this, if you forget to setup something:

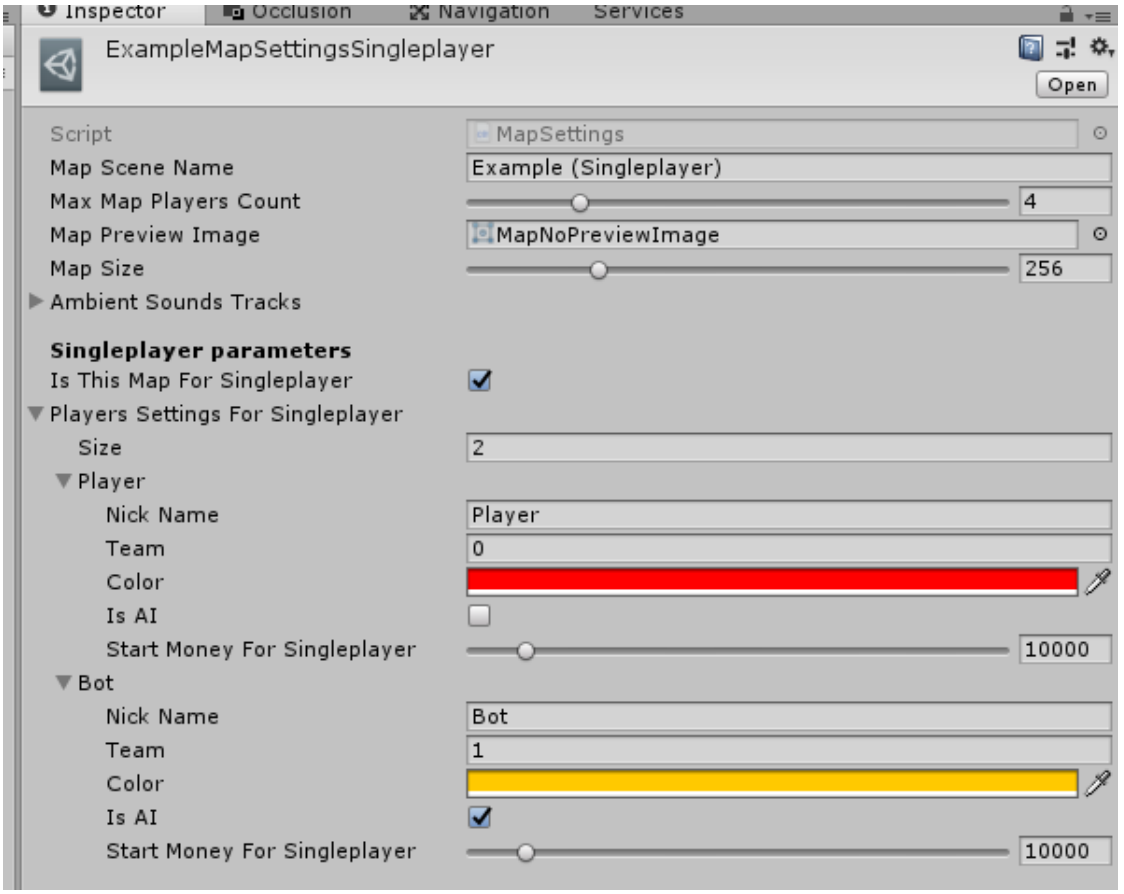


Map settings checker

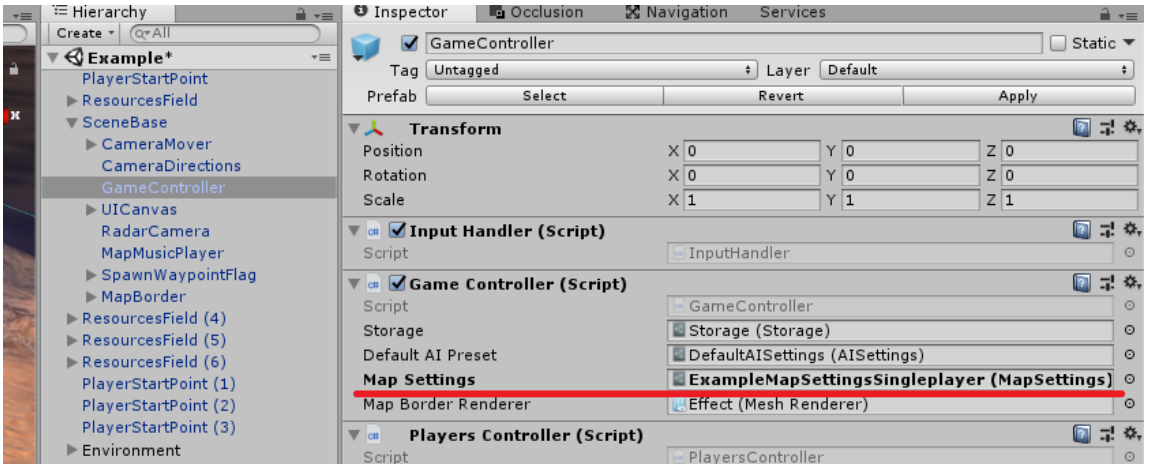
Setting up single player (campaign) map

By default, all maps settings are designed for lobby-style matches. But if you're making a single player game with a campaign, of course you don't need any lobbies.

So, you can check **Is This Map For Singleplayer** toggle in **Map Settings** and your map will work as a single player-only map.
Also you need to pre-setup this map players parameters in **Map Settings**. My example looks like this:

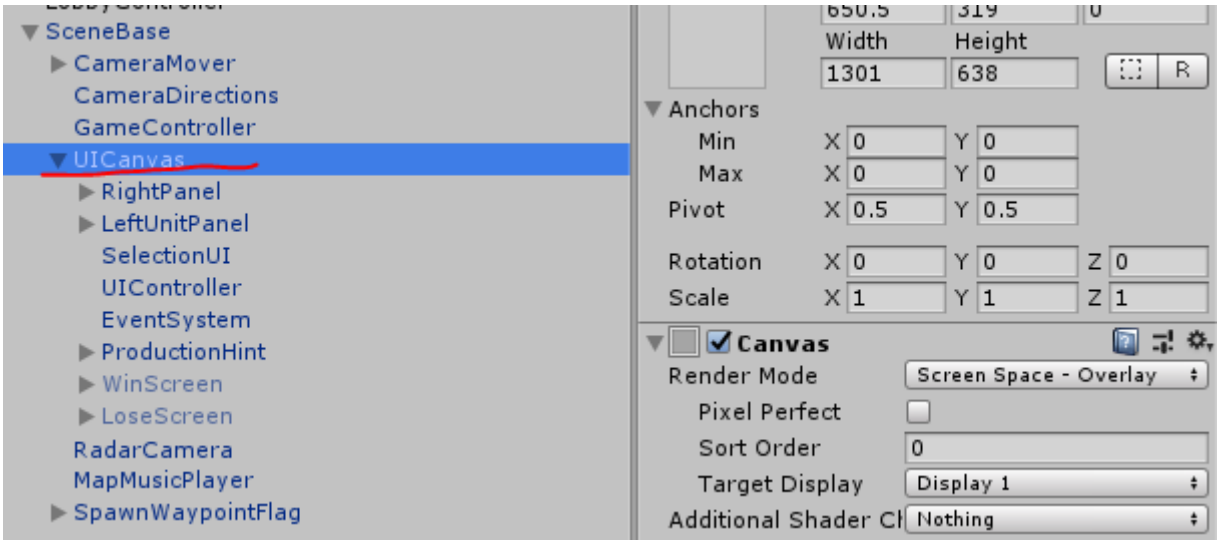


Finally, you need add this **MapSettings** asset to **GameController Map Settings** field on scene of your map:

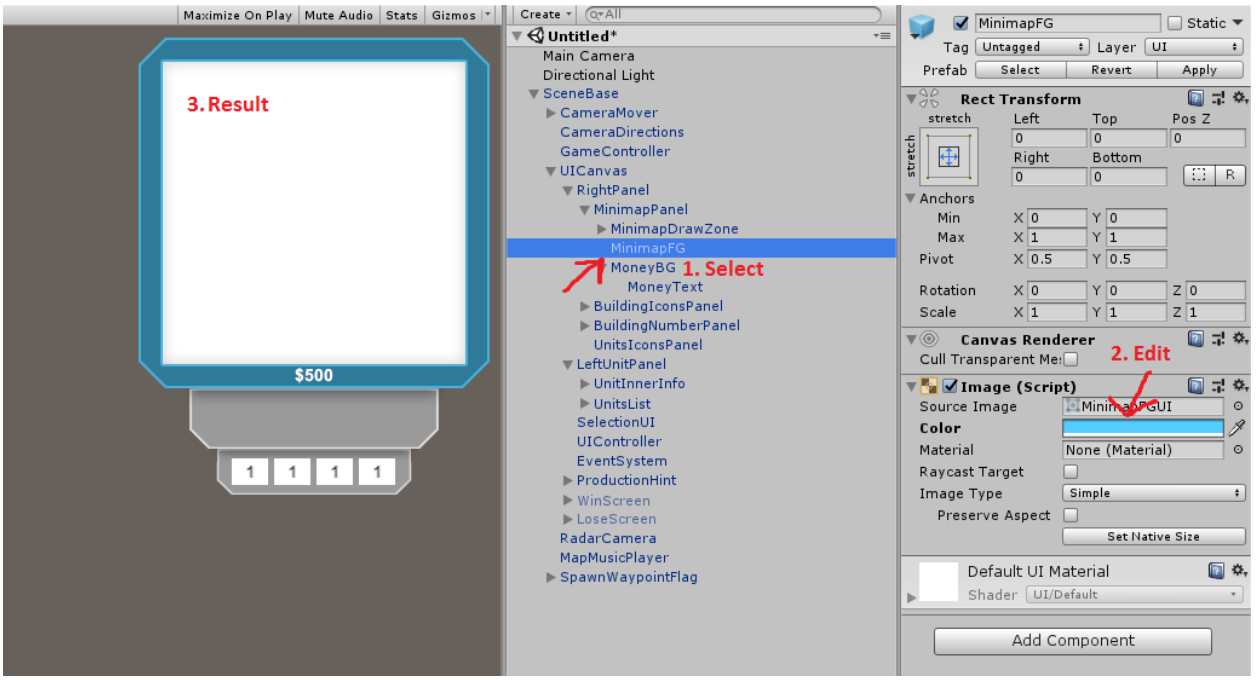


How to customize game UI?

You can easily customize game UI with your images and colors, just drag'n'drop **SceneBase** prefab to **Scene** (or open in Prefab Editor, if you're use 2018.3 or newer version), and edit **UI Canvas** child UI Elements:



For example, I'd changed color of one UI windows:

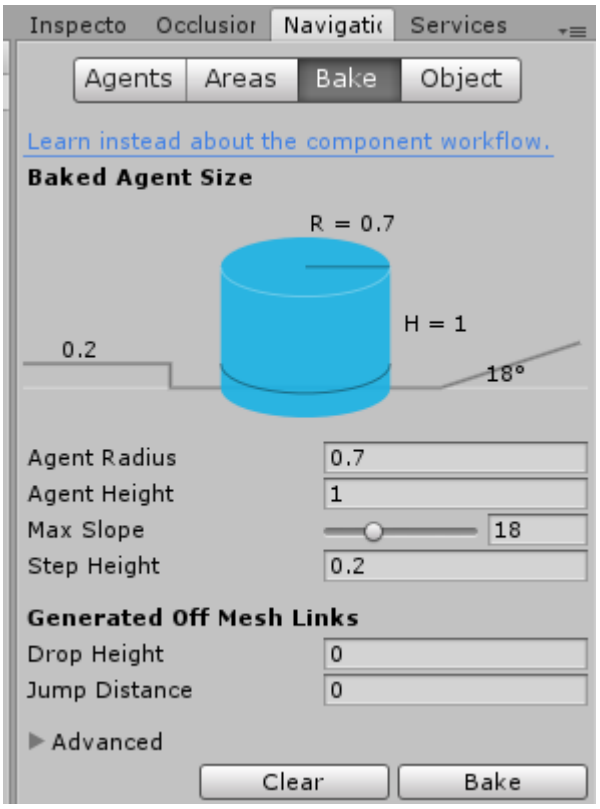


Don't forget to save the prefab after your changes.

You can also change the position of elements, images, size etc. But a lot of these elements work with code, so don't remove any of them from prefab, it can cause errors.

Best settings for NavMesh

In current version asset uses these NavMesh settings:



It is good, because it disallows units to move at very high angles of terrain. But it works fine only for example units scale. If the scale of your units is bigger or smaller, you need to find your own best settings of NavMesh.

But you always can look at our settings and make something like this, but with proportion to your scale.

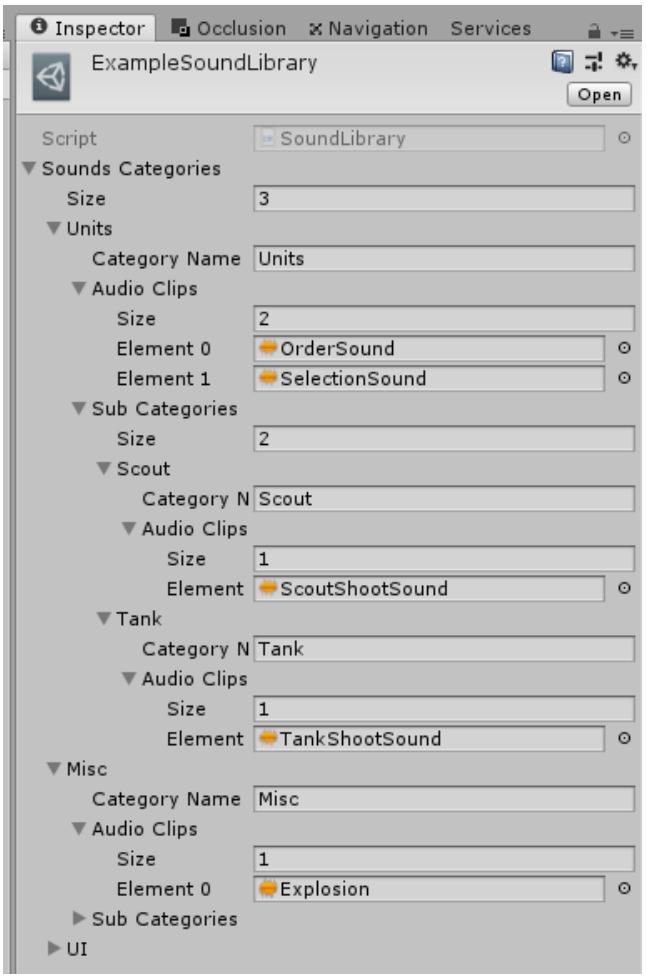
Sound Library and Sound Editor

Sound Library is a resources asset file like **Storage** (described before). It used to manage all project sounds in a more convenient way than default. It allows you to create sounds categories (you can divide units sounds and UI sounds for example) and subcategories (for example, subcategories for different units). Next, in **Unit Data** settings you can simple select the needed sound from the dropdown. It is simpler than searching them through all project.

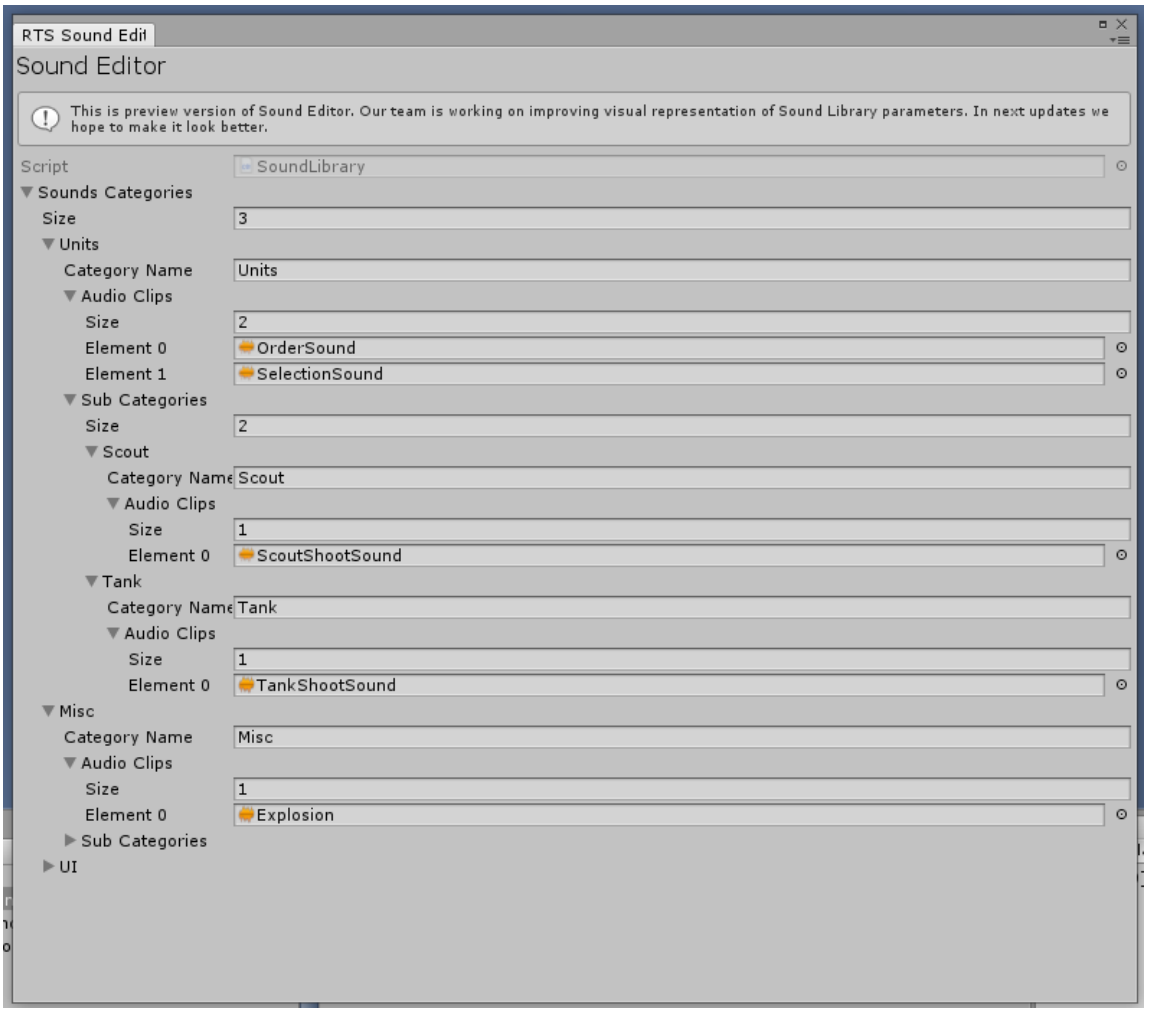
Example project already contains example **SoundLibrary** asset, and it also setted up in **Storage** (you need drag and drop here **SoundLibrary** asset which you want

to use, or left it as it is and edit example **SoundLibrary** when you want to add new sounds).

Currently it looks not the best, but can be edited as any array in Inspector: to add new categories or sounds - extend the array size. After it is done, setup newly appeared field as you need.



Sound Editor is just a quicker way to access currently used in project **Sound Library**, you can open it from the **top menu -> RTS Starter Kit -> Sound Editor**.



If you want to use Sound dropdown in your code, just write **[Sound]** before your **Audio Clip** variable.

Triggers

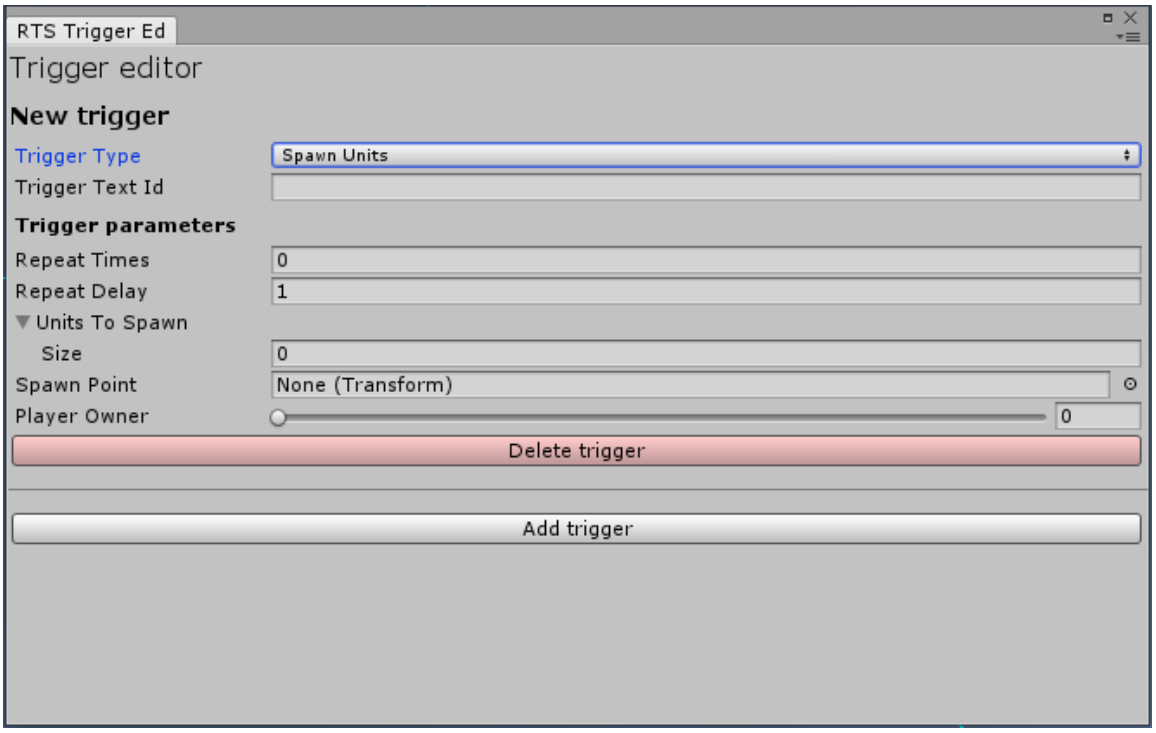
Triggers allow you to setup custom gameplay situations on your maps. It is a very useful thing for singleplayer and campaign games (can be used in your multiplayer implementation too, but of course requires improvements in code).

For example, imagine the situation, when computer AI should start attacking player units after them moving in some zone setted up before. With trigger you simple select which units should attack the player and select, where it should move to meet player units.

Or, another example. Player units reaches some destination, where player should build the base. You can give player a money with trigger or you can spawn Command Center (or another building which allows to create other buildings), so player continue playing, but now have the possibility to build the base.

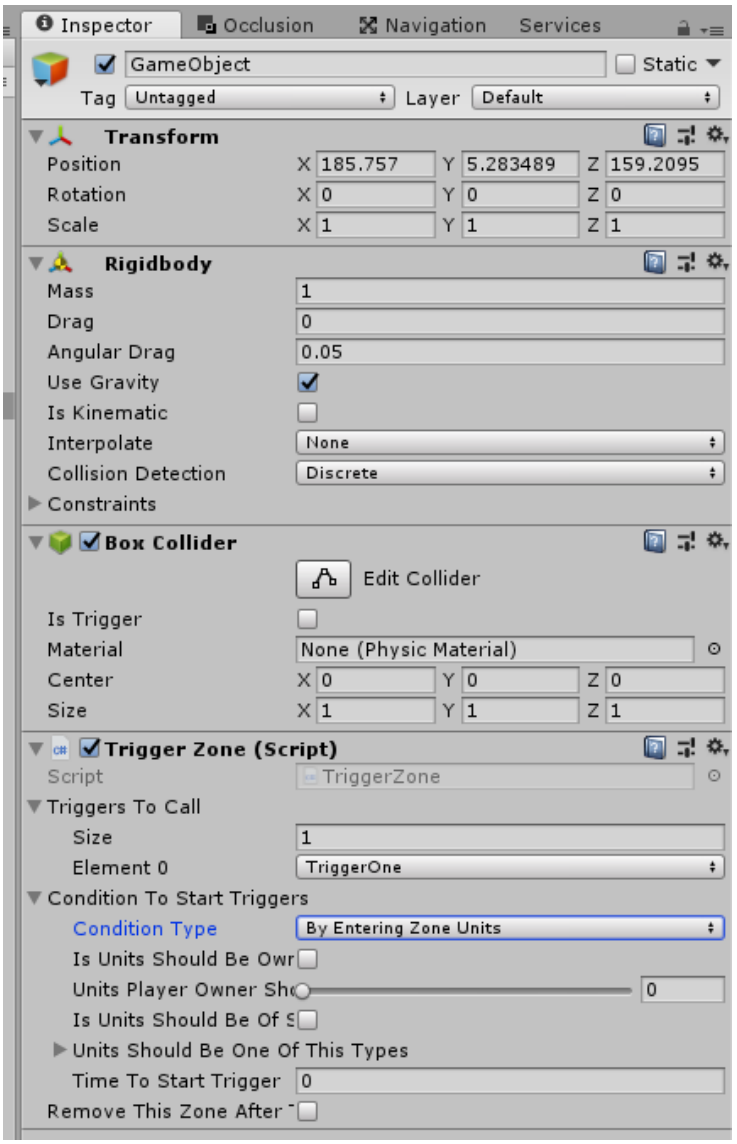
Our asset includes implementation of such triggers.

To start working on triggers, open **top menu -> RTS Starter Kit -> Trigger Editor**. You should do it on your map, not in the Lobby scene etc.



Trigger Editor allows you to manage map triggers, add new and setup all their parameters. If you already worked in other editors like this, it should be very intuitive for you.

Trigger Zone component allows you to create trigger zones mentioned before. Just add it to the empty **GameObject**, setup **Box Collider** of trigger zone (size etc). After this is done, in the **Trigger Zone** component you can select triggers which should be called when this zone is being triggered. Do it in **Triggers To Call** field.

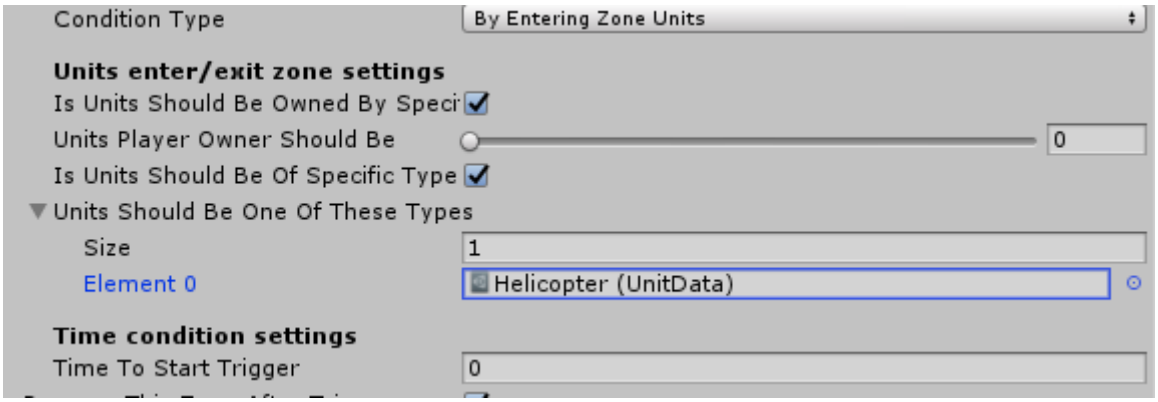


Note that if you will rename triggers, you need to update parameters in **Trigger Zones** too.

There also exists a **Trigger Condition**, which allows you to filter things which should call these triggers.

So **Trigger Zone is not** only used for check entering units, but you also can use it to make trigger which wait some time after level start.

This is an example condition of triggering zone:



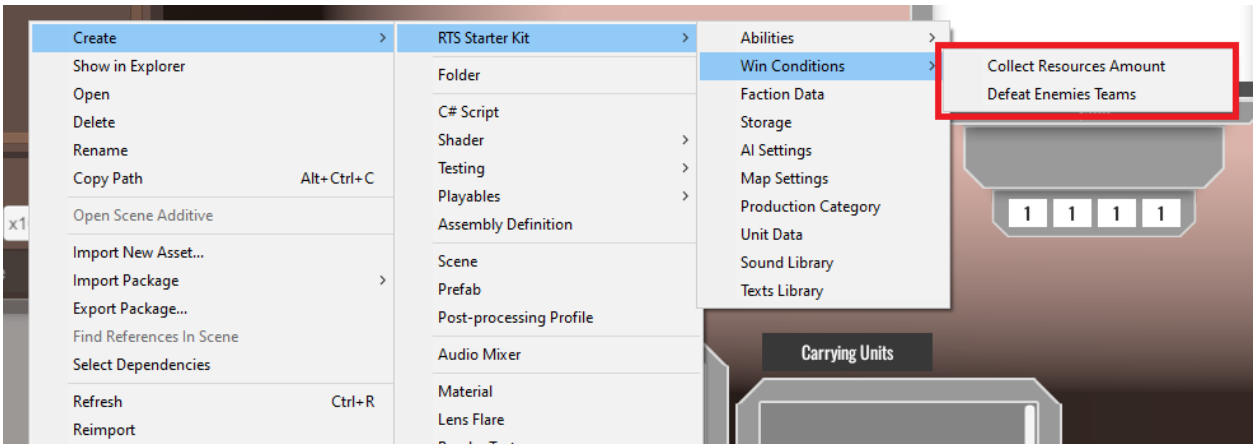
This zone **will be triggered only if** a unit of type **Helicopter** owned by **player 1** (with index 0) **enters the zone**. It is very easy to use, isn't it?

Our asset already have some ready-made triggers, conditions and we will add new ones in updates. You also can code your own triggers (use existing as example), it is not too complicated. Don't forget to add your trigger name in the **TriggerType** enum. Note that it also should end with **Trigger** word, but you don't need to include this **Trigger** word to a **TriggerType** name of trigger.
Example: trigger named **TestTrigger** should be added to a **TriggerType** as a **Test.**

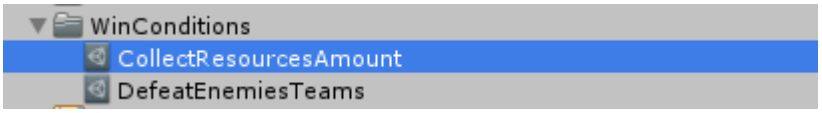
Win Conditions

Available from 1.6.8 version

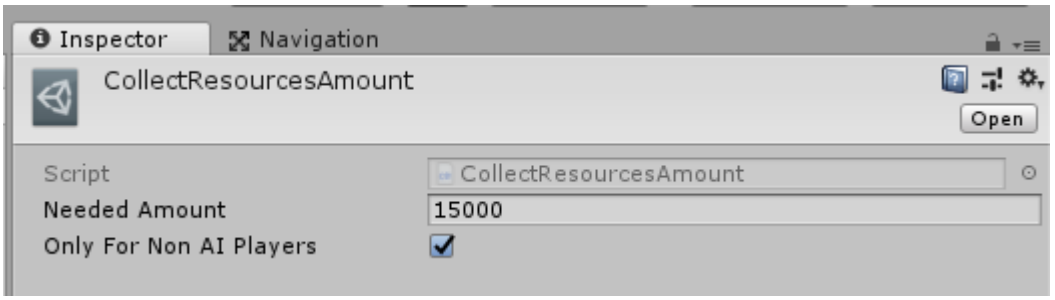
Win conditions mean condition when one of players teams wins. By default in asset used classic RTS **Destroy All Buildings** condition. But there also exist **Collect Resources Amount** condition. And you can easily code your own, check the **Programming** partition.
To create new variation of existing condition, use this menu (right-click on **Project Window**):



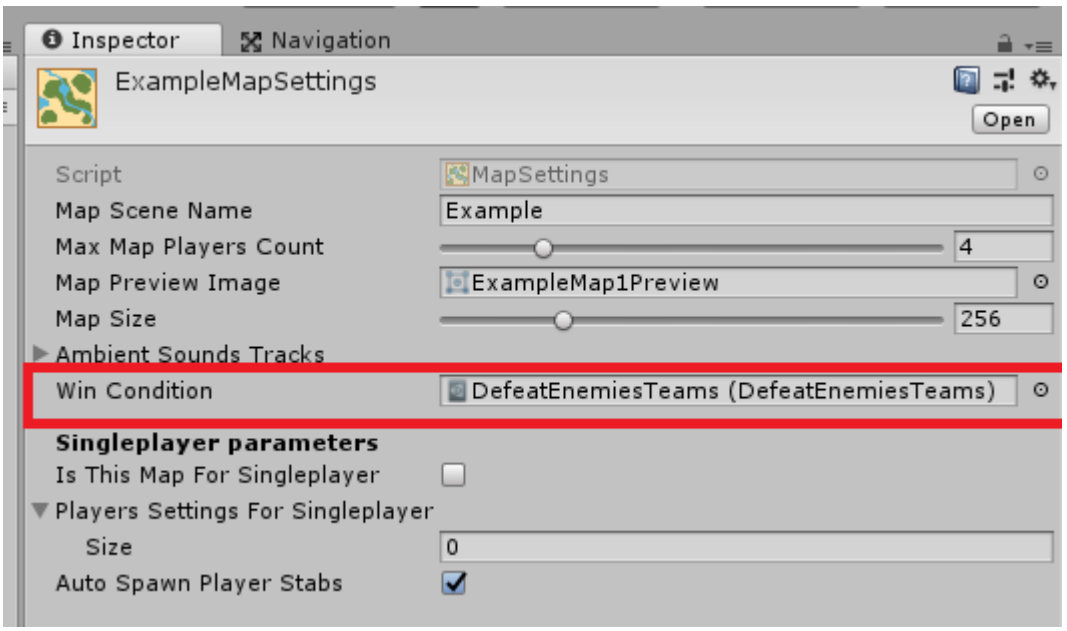
Will be created file like this:



If it customizable, it will have additional parameters:



To setup win condition on your map, open Map Settings and add it to this field:



Programming

How to create my own module?

All what you need is create new **C# Script**, and place your logics here. If you want, you can inherit it from the **Module** component – this is the basic component for all Unit modules in **RTS Kit**. It contains link to main **Unit** component, so you're can easily access to it from your own module, for example:

```
public class CustomModule : Module
{
    void Start()
    {
        Debug.Log(selfUnit.OwnerPlayerId);
    }
}
```

How to use features of RTS Starter Kit modules in my own?

If you inherited your custom module from core **Module** class as it shown in previous section, you can get access to the other modules by **Unit** component, it contains links to all other default modules, if it added to the unit:

```
public class CustomModule : Module
{
    void Start()
    {
        selfUnit.movable.MoveToPosition(Vector3.zero); // access to the Movable module

        Debug.Log(selfUnit.attackable.attackTarget); // access to the Attackable module

        selfUnit.damageable.TakeDamage(15); // access to the Damageable module

        Debug.Log(selfUnit.production.unitsQueue.Count); // access to the Production module
    }
}
```

You can check the accessible module for null, if you are not sure that it exist on a unit object.

To send a new order to a unit, use **selfUnit.AddOrder** method. You can send move, follow and attack orders:

```
    void Start()
    {
        var moveOrder = new MovePositionOrder();
        moveOrder.movePosition = Vector3.zero;

        selfUnit.AddOrder(moveOrder, false);
    }
```

Using of RTS Starter Kit events

Rts Starter Kit code contains a lot of events, which you can connect your code to. It simplifies development of new features very well.

For example, you can do something when any unit spawned. You can receive info about spawn by connecting to **Unit.UnitSpawned** (previously Unit.unitSpawnedEvent).

```
using UnityEngine;
using InsaneSystems.RTSStarterKit;

public class ExampleDebug : MonoBehaviour
{
    void Start ()
    {
        Unit.unitSpawnedEvent += OnUnitSpawned; // now every time when any unit being spawned,
                                                // OnUnitSpawned method will be called and...
    }

    void OnUnitSpawned(Unit unit)
    {
        Debug.Log("Spawned a new unit" + unit.name); // ... this message will appear in Console.
    }
}

```

Full list of events can be found in online API Documentation.

Creating new unit Abilities

This is available from 1.6.6 version

RTS Starter Kit contains an API for ability creation. You can make your own abilities for units with custom logic. You need to create a new class inherited from the **Ability** class. After implement all needed methods with **override** directive like this:

```
using InsaneSystems.RTSStarterKit.Controls;
using UnityEngine;

namespace InsaneSystems.RTSStarterKit.Abilities
{
    [CreateAssetMenu(fileName = "CarryOut", menuName = "RTS Starter Kit/Abilities/Carry Out")]
    public class CarryOut : Ability
    {
        protected override void StartUseAction()
        {
            if (Selection.selectedUnits.Count == 0) return;

            for (int i = 0; i < Selection.selectedUnits.Count; i++)
            {
                var carryModule = Selection.selectedUnits[i].GetModule<CarryModule>();

                if (carryModule) carryModule.ExitAllUnits();
            }
        }
    }
}
```

You have 3 methods to override:

InitAction - being called once when the unit spawns and ability initializes.

StartUseAction - this being called when the player clicks the ability icon and it is possible to use it (ability icon active etc).

UpdateAction - being called every frame with the usual MonoBehaviour Update cycle (from **AbilitiesModule** of specific unit).

Don’t forget to add a custom **CreateAssetMenu**, which will allow you to create a data-file of your ability.

Abilities API is very “raw” and doesn’t contain a lot of useful functions like reloading, targeting etc for the 1.6.6 version. You can try to implement it yourself, or wait for our updates.

If you need to pass to your ability some data from the unit owning it, for example, child transform or something, you need to create a new script and add public fields in it. Will name it **AbilityAdditionalSettings** for example.

```
using UnityEngine;

namespace InsaneSystems.RTSStarterKit.Abilities
{
    public class AbilityAdditionalSettings : MonoBehaviour
    {
        public Transform customTransform;
        public GameObject customGameObject;
    }
}
```

Add this component to your unit prefab and setup your fields as you need.

Next, in ability you can get this component using this code:

```
namespace InsaneSystems.RTSStarterKit.Abilities
{
    [CreateAssetMenu(fileName = "CarryOut", menuName = "RTS Starter Kit/Abilities/Carry Out")]
    public class CarryOut : Ability
    {
        AbilityAdditionalSettings additionalSettings;

        protected override void InitAction()
        {
            additionalSettings = unitOwner.GetComponent<AbilityAdditionalSettings>();
        }
```

It will once load component from unit owning this ability, and next you can get any of your data in another ability code like this:

```
[CreateAssetMenu(fileName = "CarryOut", menuName = "RTS Starter Kit/Abilities/Carry Out")]
public class CarryOut : Ability
{
    AbilityAdditionalSettings additionalSettings;

    protected override void InitAction()
    {
        additionalSettings = unitOwner.GetComponent<AbilityAdditionalSettings>();
    }

    protected override void StartUseAction()
    {
        additionalSettings.customTransform.position += Vector3.forward;
        additionalSettings.customGameObject.SetActive(false);
    }
}
```

Creating new Win Condition

Available from 1.6.8 version

To create a new win condition, create a new class and derive it from **WinCondition**.

There is an abstract method **bool CheckCondition(out int winnerTeam)**, you need to override it with your own. It should return true if one of the teams wins. Also should fill the winnerTeam variable, so the asset will know which team wins.

Don't forget to add **CreateAssetMenu** since this is a Scriptable **Object** and you will need a way to create asset of your **WinCondition** type.

Final code can look like this:

```
namespace InsaneSystems.RTSStarterKit
{
    [CreateAssetMenu(fileName = "CollectResourcesAmount", menuName = Storage.AssetName + "/Win Conditio
    public class CollectResourcesAmount : WinCondition
    {
        public int neededAmount = 15000;
        [Tooltip("If AI player should not win if he collect such money amount, check this true.")]
        public bool onlyForNonAIPlayers = true;

        public override bool CheckCondition(out int winnerTeam)
        {
            var playersIngame:List<Player> = GameController.Instance.PlayersController.PlayersIngame;
            winnerTeam = -1;

            for (int i = 0; i < playersIngame.Count; i++)
            {
                if (onlyForNonAIPlayers && playersIngame[i].IsAIPlayer)
                    continue;

                if (playersIngame[i].Money >= neededAmount)
                {
                    winnerTeam = playersIngame[i].TeamIndex;
                    return true;
                }
            }

            return false;
        }
    }
}
```

You can add any parameters and it will be shown in the asset of your win condition.

Creating custom AI

Available from 1.6.9 version

You can derive from **AIController** class and override its virtual methods with your own logic. You can still use default logic and add only new, if you'll keep some of virtual methods not overridden (for example, HandleUnitsBuilding method, by default it handles units production by AI, if you override it and don't use base.HandleUnitsBuilding(), AI will stop buying new units).

Our coding conventions

Actual from 1.7.0 version

We trying to be consistent with Microsoft C# Convention.

But our code has some own conventions:

- We do not use a private modifier, since it gives no useful info to the user, but increases code size and reduces readability in our opinion
- We do not use any prefixes in code like "_" or "m_", since they reduce code readability in our opinion.

Some code places can be inconsistent, because the first version of the asset had bad conventions, similar to Unity's ones, and it hard to change without breaking a project for users (for example, ScriptableObject serialized fields, FormelySerialized doesn't fully save from serialization problems).

Online API Documentation

You can get much more info about **RTS Starter Kit Asset** source code in our **online API Documentation**. API Documentation available here:

Actual from 1.7.0 version

https://insanesystems.net/APIDocumentation/RtsStarterKit

We often update it, and are still working on improving the documentation.

There is also exist old documentation, which will be removed in future:

https://insanesystems.net/APIDocumentation

To get access to the **RTS Starter Kit** partition in the old documentation, you'll need to enter this token: **67gfx17nj**. Using the link above should automatically give access to this partition.

Roadmap

Possible roadmap for the next versions. You can read it, if you are looking for some feature, but the RTS Starter Kit asset doesn't have it now.

Relevant roadmap can be found here: https://trello.com/b/50aM7pH1

Contacts

You can ask your questions or send your suggestions to us. ☺

To contact us use email **godlikeaurora@gmail.com**

Credits

RTS Starter Kit by **Insane Systems**.

Used free content for preview screenshots and demo:

Environment Terrain textures by **Yughues** - http://patreon.com/Yughues

Low Poly Soldiers Demo by **Polygon Blacksmith -**

http://assetstore.unity.com/publishers/17894

Not included in the asset, but you can download it free.