# TurnBased-ToolKit

## Documentation for Unity3D

Version: 2.0
Author: K.Song tan
LastUpdated: 3rd December 2014
Forum: http://goo.gl/ZBwsth
WebSite: http://goo.gl/aUXnrS
AssetStore: http://goo.gl/PQveIB

Thanks for using TBTK. This toolkit is a collection of coding framework in C# for Unity3D. This toolkit is designed to cover most of the common turn based tactic game mechanics. Bear in mind TBTK is not a framework to create a complete game by itself. It does not cover elements such as menu scenes, options, etc.

The toolkit is designed with the integration of custom assets in mind. The existing assets included with the package are for demonstration. However you are free to use them in your own game.

If you are new to Unity3D, it's strongly recommend that you try to familiarised yourself with the basic of Unity3D. If you are not familiar with TBTK, it's strongly recommended that you look through this video series for a quick tutorial. Once you grasp the basic, the rest should be pretty intuitive.

You can find all the support/contact info you ever needed to reach me on '**Support And Contact Info**' via the top panel. Please leave a review on AssetStore if possible, your feedback and time are much appreciated.

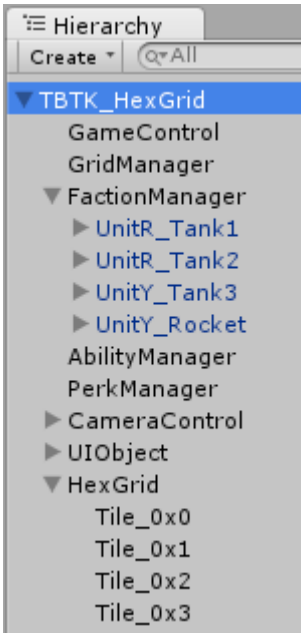
# Important:

**Update Note**
If you are updating an existing TBTK and don't want the new version to overwrite your current data setting (unit, ability, damage-table, etc), Just uncheck '*TBTK/Resources/DB_TBTK'* folder in the import window.

Please note that TBTK2 is not backward compatible with earlier TBTK release.

# OVERVIEW

## General




Basic components: - A basic TBTK scene will need following components:

- **GameControl/TurnControl/AIManager -** control all the general game logic (win/loss state, turn-logic, AI, etc)
- **GridManager –** responsible to generate, manage and edit the grid.
- **FactionManager** – contains all the information of the factions and associate units in the scene.
- **Unit** – the unit on the grid. Unit can be deployed to the grid both before or during the game is playing. In later case, the unit is spawned during runtime
- **AbilityManager/EffectTracker** – component that controls manage all the in game abilities, (both unit and faction) as well as tracking and managing the cooldown and duration of effects
- **HexGrid/SquareGrid** – the object that contain the actual grid itself.
- **Tile**_NxM – the individual tile within the grid
- **UI** – Not required in order for the game to run. However without a working ui, player has no way to interact with certain game function like end turn or activate abilities. The default UI is based on uGUI and is built to support all features of TBTK. You can replace the default TBTK UI should you want.

Prefab spawned in Runtime: - These are the prefabs that spawn in run time

- **ShootObject** – the 'bullet' that was shot from any unit in an attack

Optional components: - These are the optional component, the absence of these component wont break the core game

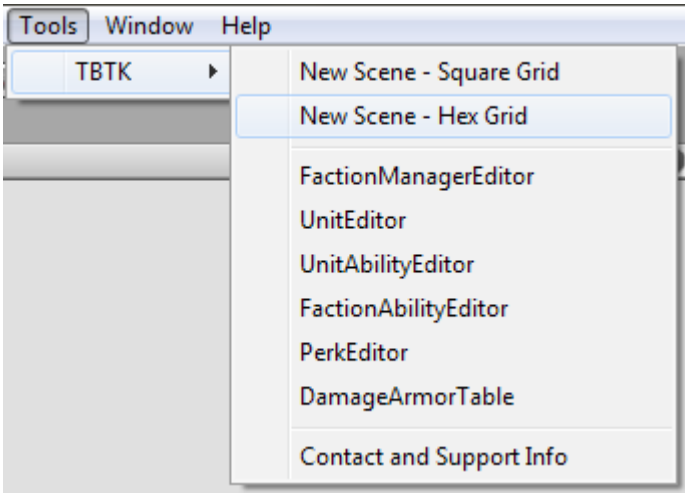
- **PerkManager** – governing the perk system. Without it, perk system are disable in game.

Misc: - These are peripheral/utility component that either support core component or enhance the game

- **AStar** – Component that handle the path-finding of the toolkit
- **CameraControl** – Provide camera control for user
- **AudioManager** – Component that manage all the sound effect and music
- **ObjectPoolManager** – Use to pre-instantiate recycle objects that needs to be instantiate regularly

# HOW TO:

## Create A TBTK Scene

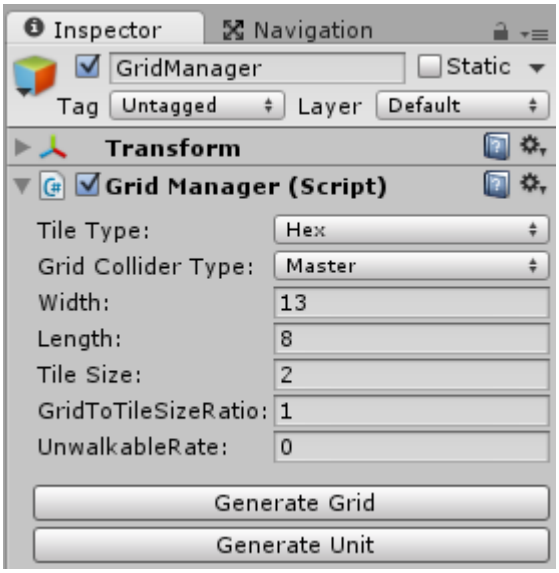



Assuming that you have some basic know how about Unity, It's very easy to create a functioning game using TBTK. To create a new TBTK scene, simply access the top panel '***Tools/TBTK/New Scene - Square Grid***' or '***Tools/TBTK/New Scene - Hex Grid***'. The former should setup a scene with square-grid and the later should setup a scene with hex-grid.

Then default scene as it's, is void of unit and therefore not playable. You will need to add at a few units for each of the 2 default faction. One of the default faction is set to be a player faction (controlled by human player) and the other an AI faction (controlled by AI).

You can add unit to the grid by directly deploying it to the grid (either hand place them or procedurally generate them). Alternatively you can assign unit to be spawned in runtime or even load from saved data from previous scene. For more information about setting up units, please refer to the setting up units section.

## Generating Grid




Generating a grid is pretty straight-forward. Simply select GridManager gameObject in the hierarchy. In the inspector, you can specify the tile of grid you want to generate, the dimension, tile size and so on. When you are happy, just click the '**GenerateGrid**' button. Generating a large grid may take sometime.

The note that when generating a grid, the new grid overwrite existing grid in the scene. Any unit on the grid will be removed.

## Edit The Grid


You can edit the grid directly via SceneView while you have the GridEditor component active (when you have GridManager object selected in Hierarchy).

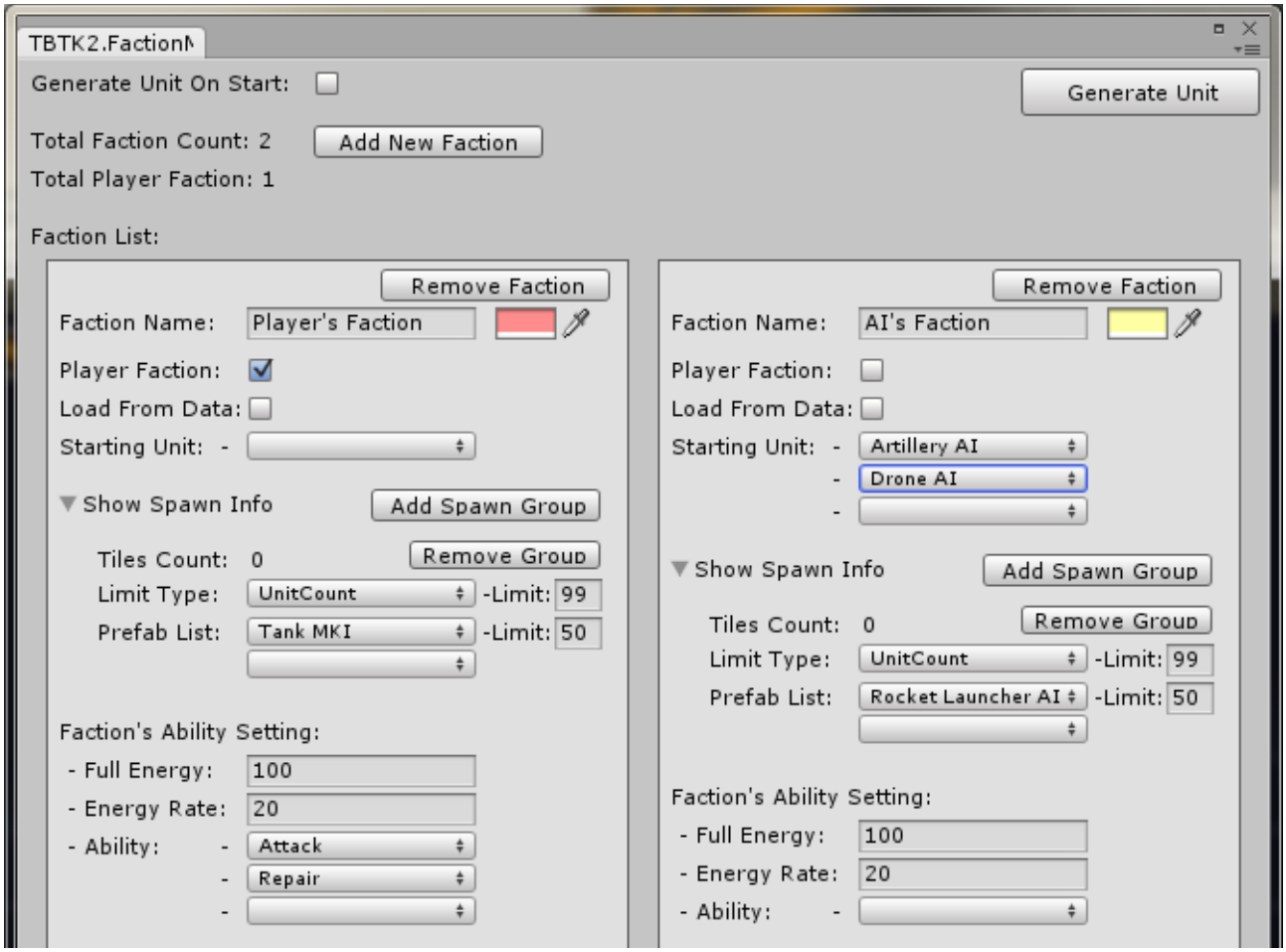
You will need to make sure the GridEditor is enabled and have the 'Edit Type' set to Tile as shown in the image. Then simply select a tile state and click on any of the tile on SceneView to apply the selected tile state.

**Unwalkable**: the tile cannot be occupied by a unit and will be set to invisible (by default)
**Default**: just a normal tile
**Wall**: an obstacle between 2 tile which stop unit from unit between those tiles
**Obstacle**: an obstacle that occupied the entire tile
**SpawnArea**: tile which unit can be spawned on when procedurally generating unit
**Deployment**: tile which unit can be deployed on

***Note**:
obstacle (both wall and full obstacle) is used for LOS calculation when Fog-of-war is enabled
obstacle type (half/full) is used to determined cover type when cover system is enabled
SpawnArea and Deployment tile are tied to specific faction specified in FactionManager

## Setting And Edit Factions

A functional scene will need at least 2 factions. You can setup the faction using FactionManagerEditor, which can be open via the TopPanel, or selecting FactionManager in Hierarchy. There's no any limit to how many faction you can have in a scene. You can also set a multiple faction to be player faction for a hotseat game. Any non-player faction will be controlled by AI during runtime.




Each unit can has it's own starting unit, deployed unit, spawn/deployment area on the grid as well as Ability.

**StartingUnit** is the unit to be spawned at the start of the game. These unit will subsequently be deployed on faction's deployment area.

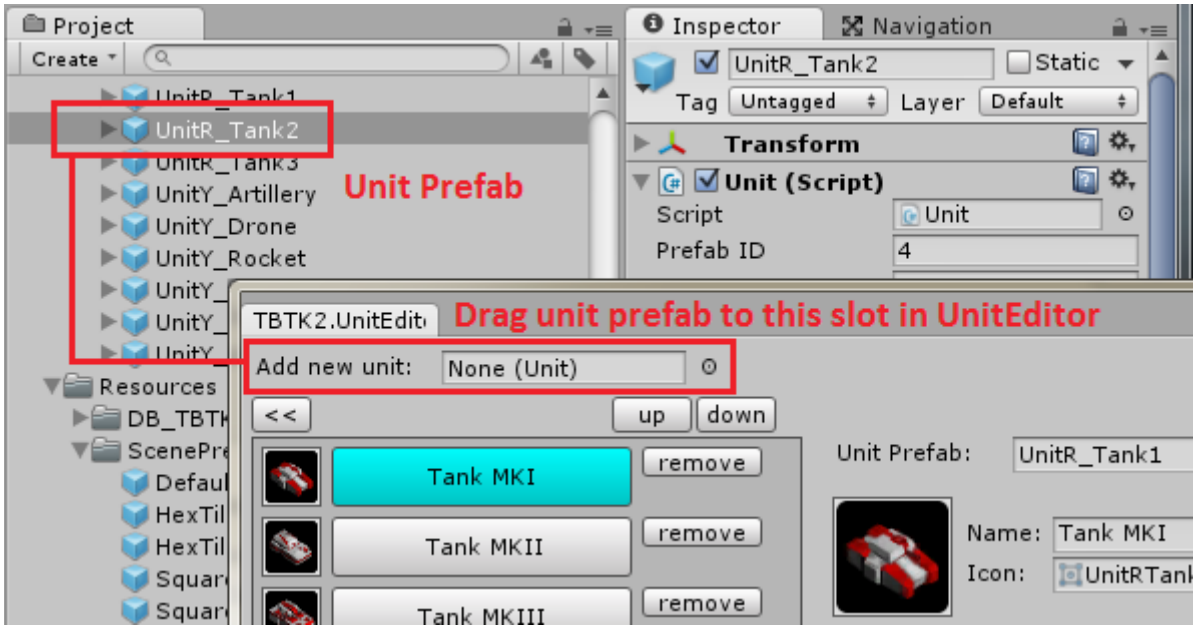
**Deployed unit** are the units that has been deployed on the grid prior to the game start using GridEditor or procedurally unit generation,. The placement of these unit are not restricted to the faction's deployment area.

**SpawnArea** is the area on the grid where the the faction's unit can be spawned on. Each spawn area can have it's own allowed spawn unit, spawn limit and so on.

**DeploymentArea** is the area on the grid where the faction's starting unit can be deployed on.

## Adding New Unit Prefab To The Game

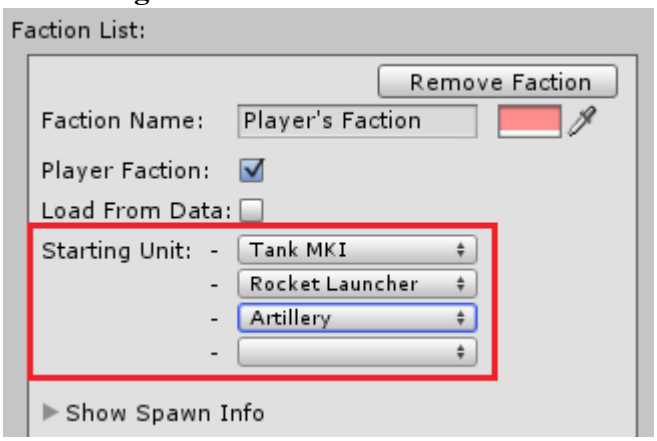
First you will need to create the unit prefab (refer to unit for how to create a prefab) and add it to UnitEditor. Once a unit prefab is in the UnitEditor, the unit should appear in other editor where unit prefab selection applies.




## Adding Unit To The Grid

There's a few way a unit can be add to the grid. In all case, the unit will need to be associate with a specific faction set in FactionManagerEditor

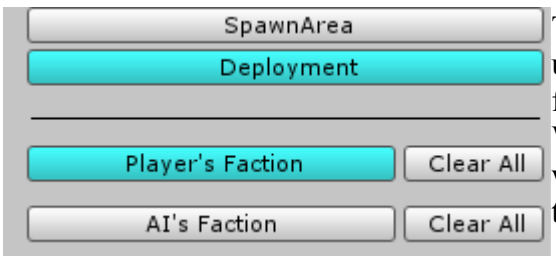
**As Starting Unit**




SartingUnit is the unit to be spawned at the start of the game. These unit will subsequently be deployed on faction's deployment area.

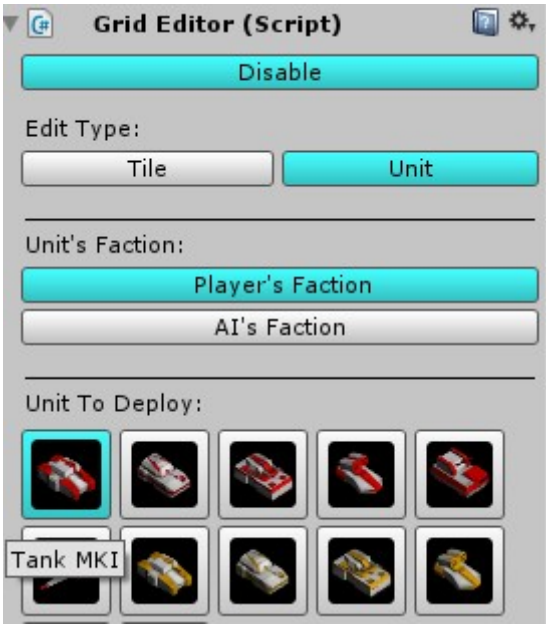
Each item in the list will be spawn as a unit instance so you can have more than one instance of the same unit prefab on list.

As you set unit to the StartingUnit list, you will need to setup sufficient deployment area on the grid so that these unit can be placed on grid on runtime.




To setup deployment area for a faction, you will need to use the GridEditor. Make sure you select the right faction in the GridEditor (by matching faction's name). When you click on the tile in the SceneView, a cross with the faction's color will show up on the tile to mark the tile as a deployment tile for the faction

**As Deployed unit on the grid via UnitEditor**



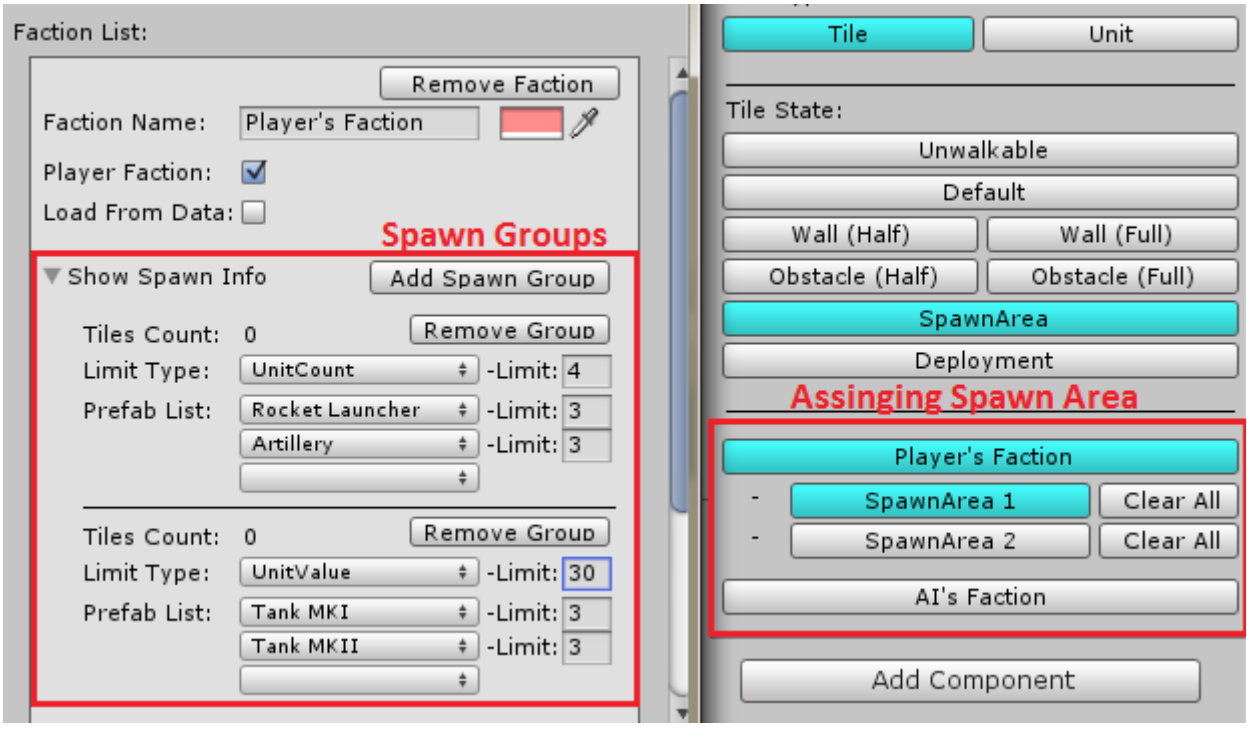

You can directly place a unit onto the grid via the GridEditor. Just Set the EditType on the GridEditor to Unit and you will see the all the unit prefabs show up in the Editor. You can select any of them and simply click on any tile on the grid and the an instance of the selected unit prefab will be spawned on the tile. Right-click on a tile with a unit on it will remove the unit.

Please note that the faction of the unit being deployed is based on the faction selected in the Unit's Faction Tab. The select option in there is based on the faction set in FactionManagerEditor.

Tips: when placing unit the grid, the facing of the unit is depends on the position of the mouse to the center of the grid

**As Deployed unit on the grid via Procedural generation**

The proecdural unit generation algorithm spawn unit on the grid based on spawn information specified on each of the faction. So to enable procedural generation, you need to first setup those information.

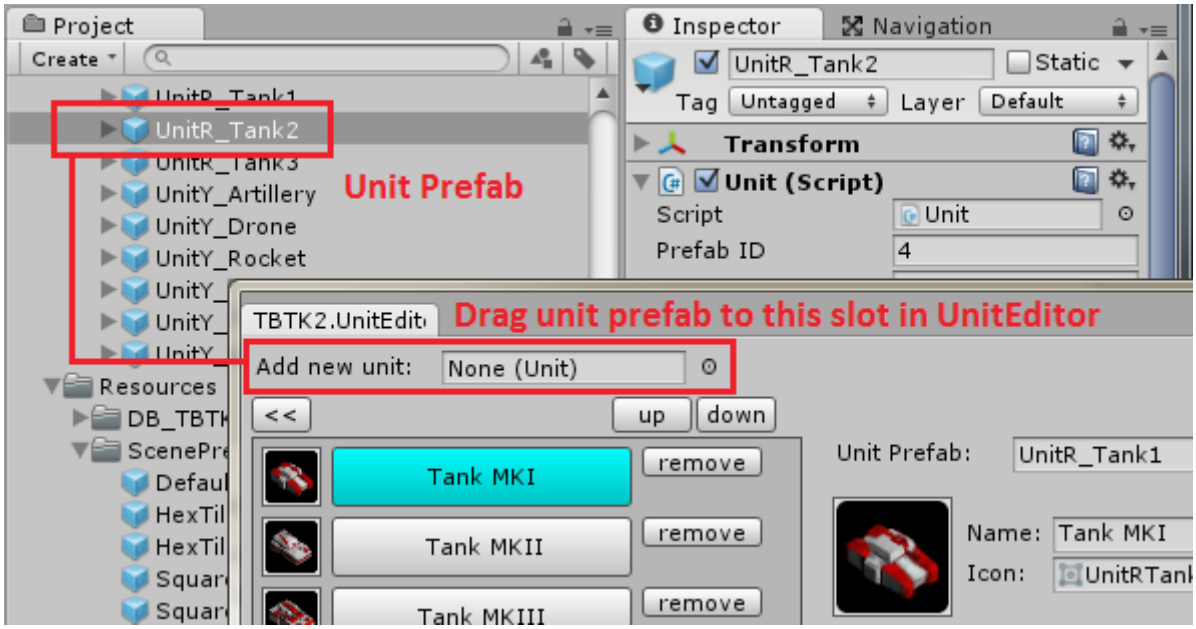



SpawnInfo for a faction takes from a SpawnGroup. Each group have its own spawn limit type, unit prefab available for spawn. Each group is also corresponded to an independent set of SpawnArea (a group of tiles on the grid where the unit can be spawned on) which can be set via SpawnEditor.Once the information is setup properly, simply use the “Generate Unit” button on either GridManager or FactionManager.

**Load From Data**
Finally the unit can be load from a previous scene. To do that, you will need to check the "Load From Data" flag for the faction and give it a unique dataID. And of course, it will be up to you to assign the data to load. This can only be done by code. You can refer to script "*Campaign.cs*" for how it's done.

## Add New Unit Prefab To The Editor

First you will need to create the unit prefab (refer to "Unit Prefab" in "How things work" section for how to create a prefab) and add it to UnitEditor. Once a unit prefab is added to the UnitEditor, the unit should appear in other editors.



# HOW THINGS WORK:

## Prefab Management

TBTK uses a centralized database to store all the information of the unit prefabs and in game item (ability, perk, damage and armor type). The in game item can be created with just a simple click of a button in their associated editor. The unit prefab however, needs to be create manually before they can be added to the database. Once added, they can be accessed and edit via editor. Unit prefab that is not added to the database will not show up as option in any editor.

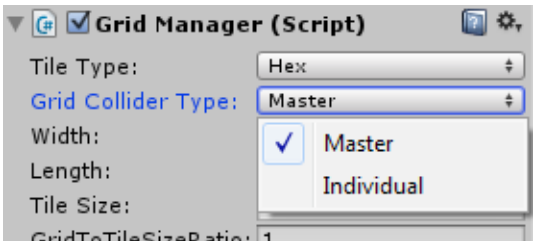
## Global Setting & GameControl

You may want to have all the scene in your project to using a similar rules set. In that case, check the UseGlobalSetting flag on GameControl. When the flag is checked, instead of configuring the local variable on GameControl, you will be configuring the global setting, which is applicable to any scene that has its the UseGlobalSetting flag checked.

## Grid

Each tile on the grid is an individual gameObject. The prefab for each tile object is located in "/TBTK/Resources/ScenePrefab/" and named HexTile and SquareTile respectively. The prefab with "_Collider" prefix are the version of prefab with collider. You can change the appearance of the tile by changing the prefab appearance. Just make sure you keep the same name and of course in the case of the tile with collider, make sure there is a matching collider on them or the grid may not work properly. These prefab are loaded and used in the grid generation every time a grid generation is called.

### Collider on Grid




To enable detection of cursor on tile in runtime, the grid will need to have collider(s) on it. There are two mode which can be set on GridManager – GridColliderType.. The first one is for the whole grid to use a single master collider (thus using the tile prefab with no collider). The second one require each individual tile to have a collider (thus using the tile prefab with collider).

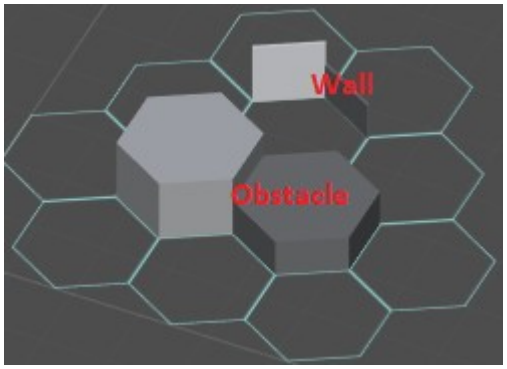
Using master collider would allow a significant performance boost and a much larger grid. However the tile position cant be adjust in anyway. On the other hand, individual collider has a much higher performance cost but the position of individual tile can be adjusted manually. In short, individual collider mode are used when the position of the individual tile needs to be customized (ie. Height adjust, position shift, etc.)

### Customizing the details

The appearance of the cursor/target indicators, as well the the appearance of the tile indicating various status of the tile can all be customized. The indicators are basically a gameObject prefab with particleSystem of various appearance. The tile appearance however, are depend on the material assigned to them. You can make your own and simply replace the defaults using GridManager.

## Obstacle And Wall Prefab

Obstacle and wall are object placed in the grid to block unit movement, block line-of-sight (if fog-of-war is enabed), provide cover bonus (if cover system is enabled). There are two type of obstacle, half cover and full cover. Only a full cover obstacle would block line-of-sight.




Obstacle are placed on top of a tile to stop the tile from being occupied by any unit. Walls are placed between two tiles to block access between those two tiles in question.

Both wall and obstacle prefab can be placed on the grid via GridEditor. The prefab requires:

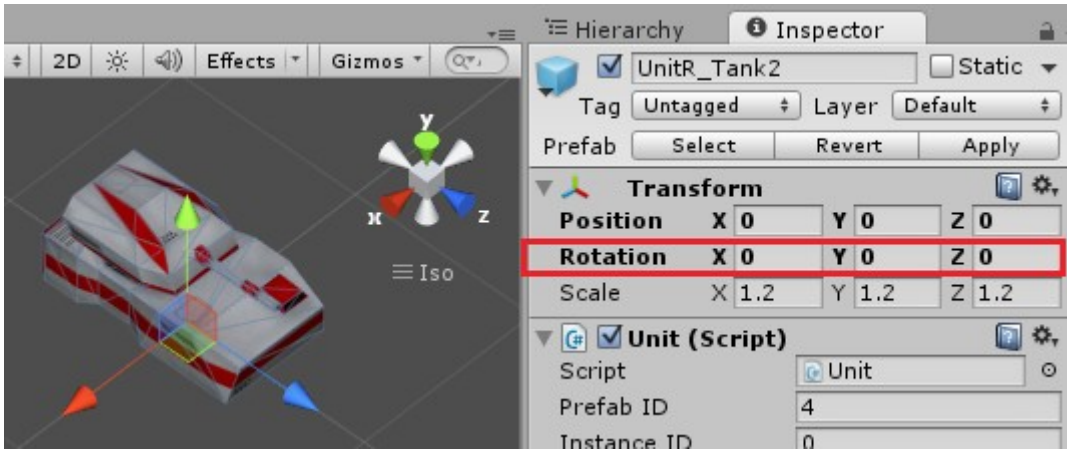
- a collider
- layer assigned to ObstacleFullCover/ ObstacleHalfCover (26/27 by default)

The type of cover is depend on the layer assigned to the prefab.

## Unit Prefab

A primitive unit prefab can be an empty gameObject with just the script *Unit.cs* attached on it. Any mesh/modal is entirely optional (an invisible unit can function just fine).

It's recommended that any mesh is place as the child transform of the unit prefab. When placing the mesh, make sure it's facing the +ve z-axis when the unit rotation is at (0, 0, 0) otherwise the unit may not be facing the correct way in runtime.




To configure a unit, it's recommended that you do so via the UnitEditor. Which can be access via the top panel. Note that to assign abilities to unit, you must first create the abilities in UnitAbilityEditor. Once an ability is created, it will show up in the Ability tab in UnitEdtitor.

A unit prefab has to be add to the UnitEditor before it can be accessed in editors.

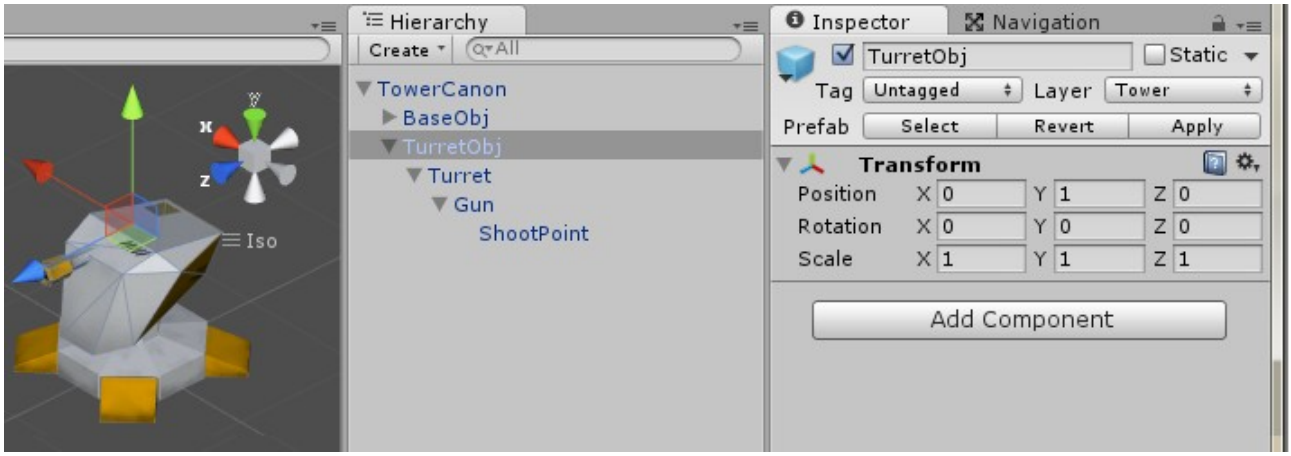
## Shooting At Target

If the unit is to fire shootObject at its target (in a normal attack or when casting unit's ability), it requires

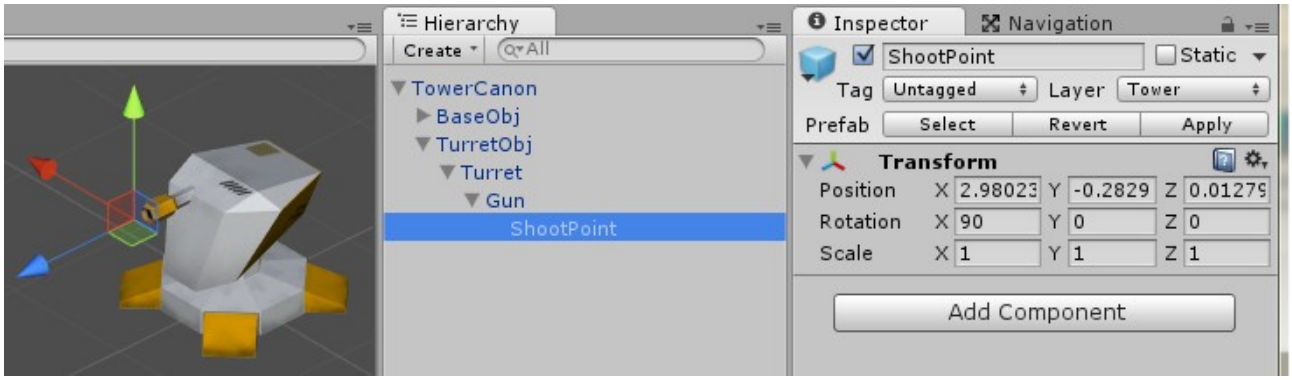
- a shootpoint
- a shootObject
- hitChance value is set to >0

For such unit, you might want it or certain part of it aims at the target (call that turret) and fires shootObject from the barrel of the turret. In that case, you will need to assign the *turretObject* (the object that do the aiming), *shootPoint* (a reference transform at the end of the barrel where the shootObject can be spawned) and *shootObject* (the bullet). All of these are optional, the framework will auto-assign them if they are left empty.

*TurretObject* is one the pivot transform in the hierarchy of the unit prefab. There are two aiming mechanism supported, one's where the whole turret rotates in both x and y-axis, and there's another one where the turret rotates in y-axis and the barrel rotates in x-axis (default example – UnitR_Tank3). In the former, you will need to assign the the turretObject. In the later, you will need to assign the *barrelObject* too. For any setting, all the turretObject and barrelObject must have a localRotation of (0, 0, 0) when aiming at +ve z-axis. For the shootPoint transform, the +ve z-axis must be points outward from the barrel.




TurretObject, note that the barrel is pointing at +ve z-axis and the rotation is (0, 0, 0)

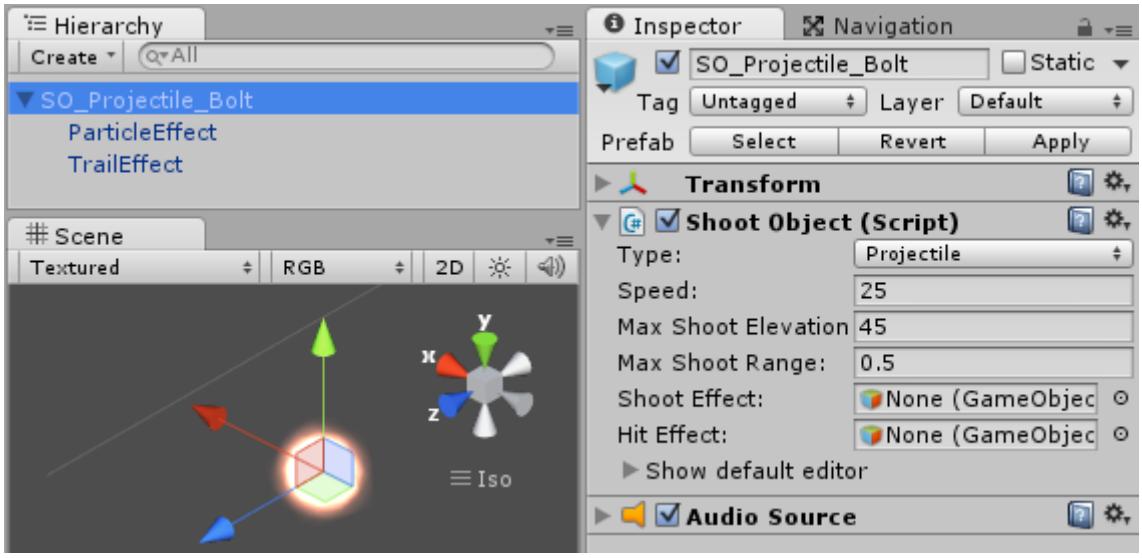



ShootPoint, note that it's pointing in the direction of the barrel

For shootObject, please refer to **ShootObject** section.

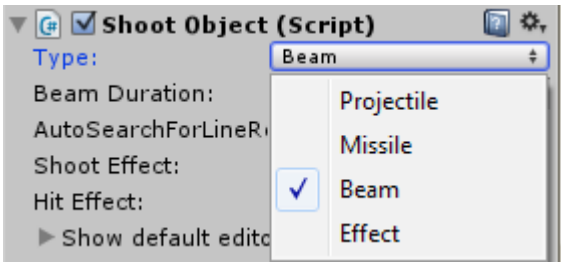
## ShootObject

ShootObject are the 'bullet' object that is fired from a unit in an attack. A primitive shootObject can be an empty gameObject with just the script ShootObject.cs attach on it.




A shootObject prefab. Note that like the example prefab, you can add AudioSource to the shootObject to play the shoot sound

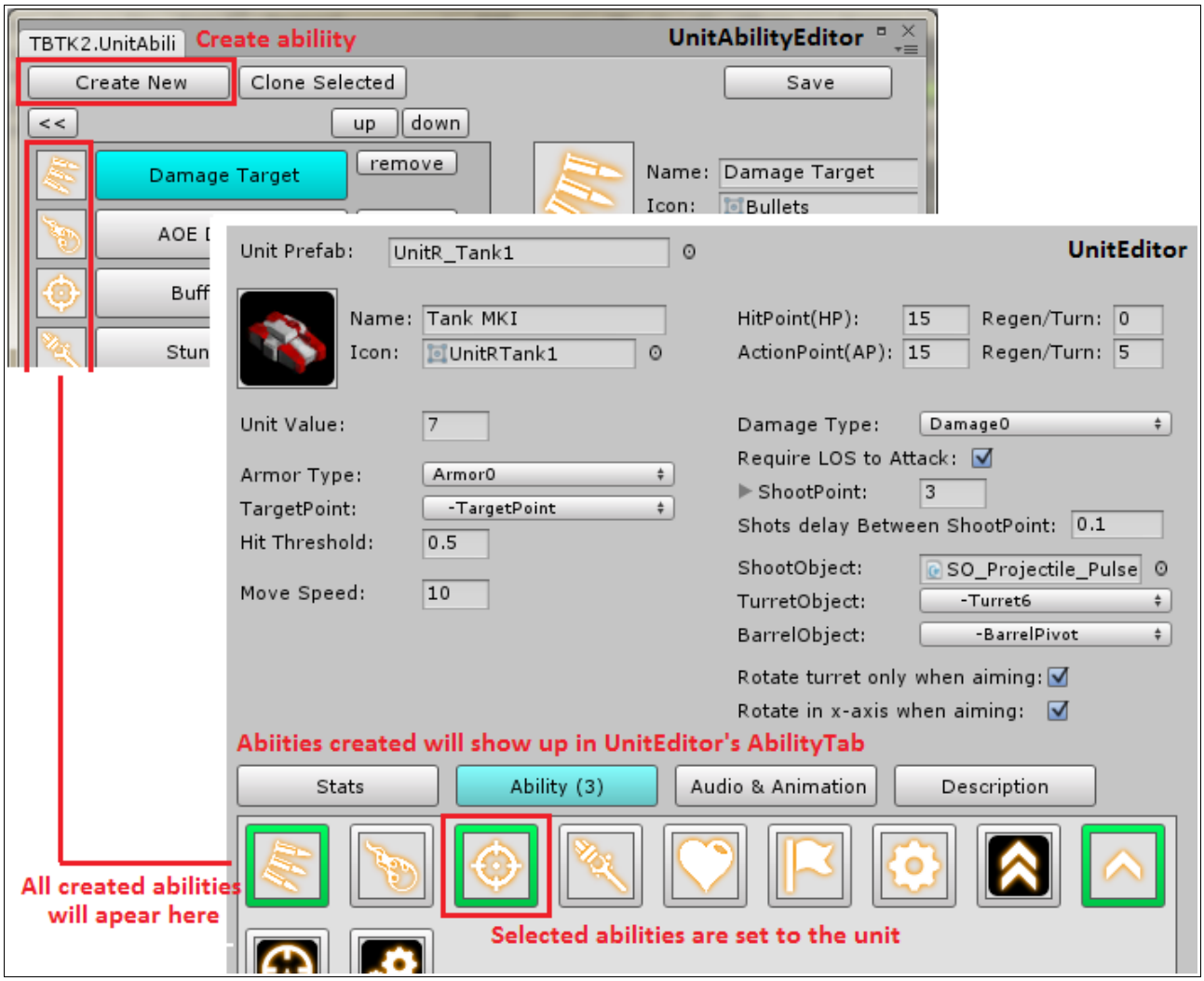
ShootObject type break down:




**projectile**: a normal shootObject which travel in a fix trajectory from shootPoint to targetPoint. A projectile shootObject is considered hit when its reach it's target point.
**missile**: a shootObject which travel from shootPoint to targetPoint but in a random trajectory that simulate a self-propelled projectile. A missile shootObject is considered hit when its reach it's target point.
**beam**: a shootObject which use LineRenderer that drawn from shootPoint to targetPoint. A beam shootObject must have a LineRenderer within it's hierarchy. A beam shootObject uses a timer to determine when it hits the target
**effect**: Does nothing mechanically, it simply waits for a timer before it hits the target. Used to melee attack or range attack that doesnt require to show the shootObject trajectory.

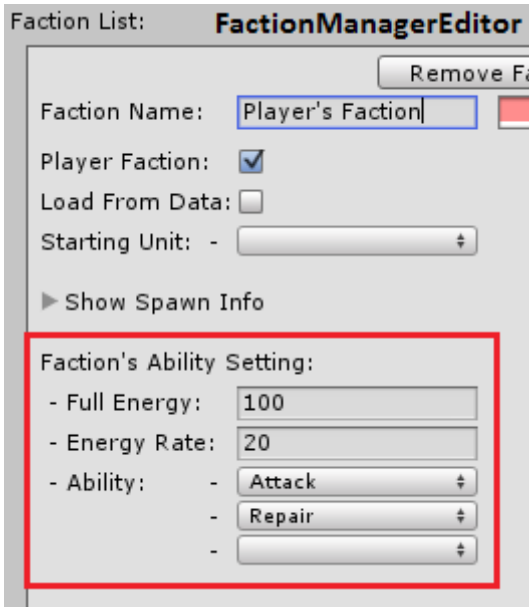
## UnitAbility

Unit abilites are abilities that ties to unit. They can be customized to do a variety of things. You can create unit abilities in UnitAbilityEditor and assign them to each unit in UnitEditor (both can be access via top panel). Once created, a unit-ability can be shared across multiple unit.




Create UnitAbility in UnitAbilityEditor and add it to assigned it to a unit in UnitEditor

## FactionAbility




FactionAbility are similar to UnitAbility except they are tied to faction. They can be created and edited via FactionAbilityEditor, can be accessed from the top panel. Like UnitAbility, once created, a FactionAbility can be assigned to any faction and each faction can has each own set of FactionAbilities. Ability assignment to each faction can be done in FactionManagerEditor.

Each of the faction has also been given a energy pool which is use to cast FactionAbility. You can specify the maximum eneregy capacity and their recharge rate in FactionManagerEditor.

## Perk System

Perk system are optional extra to the framework. Perks are upgrade item that can be purchased during runtime to give player a boost in various means. This include modifying a units stats, add new unitAbility to a specific unit, modify the stats of a ability, etc.

You can create a perk much like how you create an ability, via PerkEditor. Which again can be accessed from the top panel. Once you have create a perk, it should show up in the PerkManager, allow you to set it status as enabled/disable/pre-purchased in the game.

PerkManager can either be used in a self-contained level where any purchase progress made in the PerkManager are discard as soon as a new level is loaded. Or across the game where the progress is persistent until the game is quit. You can switch between this be checking/unchecking the "SingleLevelOnly" flag in PerkManager. Please note that since there can only be an instance of PerkManager at any given time, when using persistent mode, only the first instance of the PerkManager from the starting scene and its setting will take effect. Any PerkManager instance in any subsequently loaded level will be discarded, rendered the setting on them useless.

When using persistent mode (with "SingleLevelOnly" unchecked), you can enable saving of perk progress. When enabled, the progress is kept even when the player exit the game. You can call following function to clear all progress made:

**PerkManager.ClearPerkProgress()**
*public static void ClearPerkProgress()*

Please note that PerkManager only support game setting where there's on one human player as all the non AI unit/faction will gain the perk bonus

## Fog-of-War

Fog-of-war is a built in system in TBTK to hide any hostile unit that is out of unit's sight. There are two ways in which a unit can be concealed from the hostile.

- Being out of all hostile units sight.
- Has no direct Line-of-Sight with any hostile unit (blocked by a obstacle with full cover)

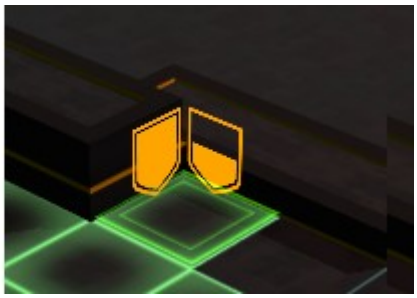
A unit cannot attack a hostile that is concealed.

## CoverSystem & Obstacle

CoverSystem is a built in system in TBTK to emulate the cover system seen in game like XCom. Basically it tries to simulate a real life situation where a combatant is harder to hit when he is hiding behind cover. A unit is considered in cover when it's behind and adjacent to an obstacle with respect to it's attacker. A unit behind in cover gains a cover bonus which gives them a bonus in dodging incoming attack. Any unit attacking a target not in cover will gain a bonus is critical.

There are two tier of cover bonus, half and full cover.  Each tier of cover provide different bonus value which can be set in GameControl. A cover is determined by raycasting against Obstacle or Wall object. The layer assigned to the obstacle or wall object determine if the cover bonus is half or full. An shield icon overlay will show up when unit is about to move into a tile with cover.





## Flanking

When flanking is enabled, an unit that is attacked from behind suffer more damage. The amount of excess damage is depend on the FlankingBonus on GameControl as well as the FlankingModifier on the unit and FlankingBonus on the attacker. You can adjust the FlankingAngle in GameControl to determine when a unit is considered flanked.

## DamageArmorTable

DamageArmorTable is modifier table used to create a rock-paper-scissors dynamic between units. You can setup various damage and armor type using DamageArmorTableEditor (access from top panel). Each damage can act differently to each armor. (ie. damage type1 would deal 50% damage to armor1 but 150% damage to armor2). Each unit can be assigned to a use a specific damage type and a specific armor type.

# INTEGRATING TBTK TO YOUR GAME:

**Unit integration**
TBTK is made to support custom integration, that is to carry your unit into a TBTK scene and provide you the unit information when the battle is done. An example to do this has been setup in the form of a mini campaign. You may refer to the script Campaign.cs for example code. The script is fully commented to help you understand it.

The class that handle the integration are Data. You can look at the class defination at TBTK_Class_Data. The function to set the unit to a battle or to retrieve unit remain after a battle is pretty straight-forward. To set the unit, it's:

> **Data.SetLoadData()**
> *public static void SetLoadData(int dataID, List<UnitData> unitDatalist)*

In the TBTK scene where it's suppose to use the data, you will have to configure the faction in question to load from data, with the matching dataID you pass to the function. The faction will load the data and use the unit as it's startingUnitList. Note that although you can use any figure for dataID as long as it match in the code and the faction setting, it's recommended that you keep the figure low (0, 1, 2, etc) so that it takes less time for the code to search fro the matching ID. To retrieve the unit in the next scene after the battle(TBTK scene) is done, you can use following function:

> **Data.GetEndData()**
> *public static List<DataUnit> GetEndData(int ID)*

The key information is **DataUnit** class, which store the information of a particular unit and it's intended stats(for going into a battle) and it's remaining stats (after a battle). The class definition can be find in Data.cs.

Essentially before loading into a TBTK scene, you compile a list of DataUnit of the units to be set as a faction startingUnitList. You can modify each unit stats based on your game intepretation (for instance, adjust the unit stats based on the level). Then you set the list to data. After the TBTK scene, you can retrieve the DataUnit list. The stats in the DataUnit in this case are the active stats on the unit when the battle end so you can tell if a unit is damaged. Units that are destroyed in a battle will not be appear on the list

You can use following function to check if there has been any previous battle

> **Data.EndDataExist()**
> *public static bool EndDataExist()*

**Custom Saving and Loading of Perk Progress**

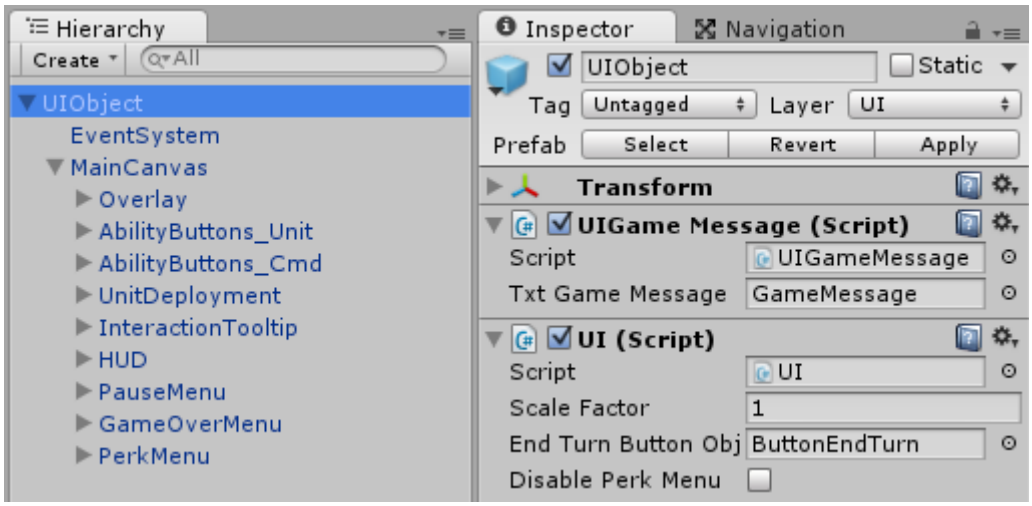
In some case you might want handle save/loading the perk progress with your own method. It's recommended that you save the perk progress as a list of integer, with each integer being the ID of the perk being unlocked. To access the perk and check their purchase state, you can use something like this

```
List<Perk> perkList=PerkManager.GetPerkList();
for(int i=0; i<perkList.Count; i++){
        if(perkList[i].purchased){
                //add perkList[i].prefabID to your list
        }
}
```

When loading the game, you can simply replace the *purchasedIDList* in PerkManager with your own integer list in PerkManager.Init(), just before the perks are loaded. Node that you will have to set “SingleLevelOnly” flag on PerkManager in order for this to work.

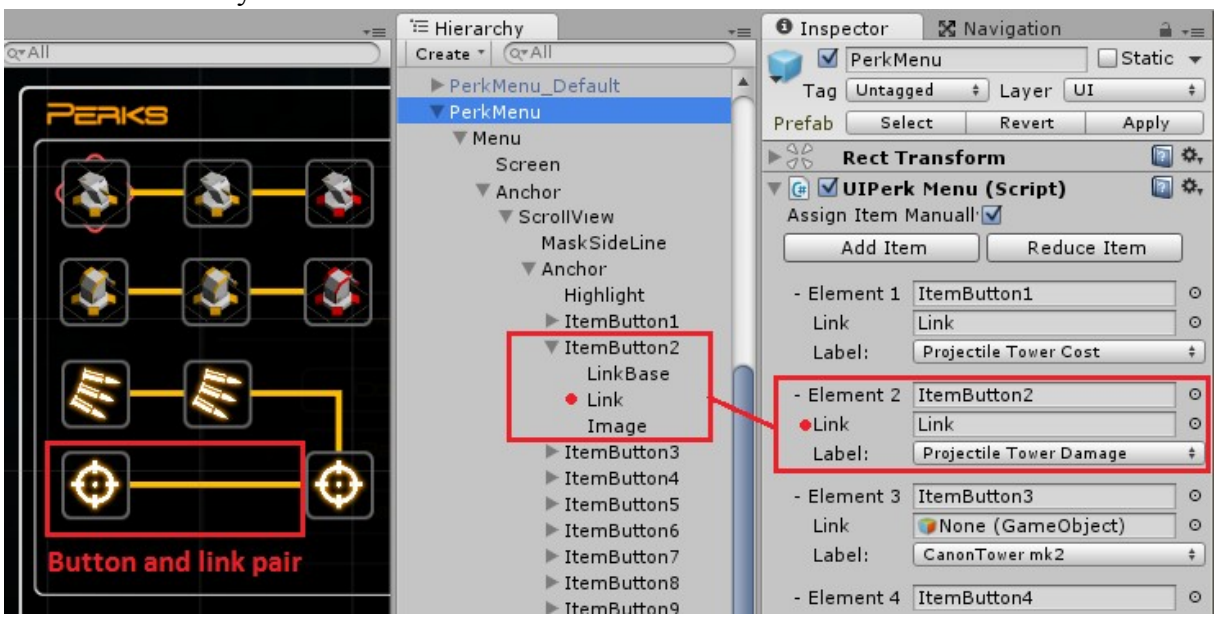
# ABOUT USER INTERFACE:

User Interface is very much an integral component of the package although it's not a core component. It's built to be modular and thus can be replaced with any custom solution. The interface is based on uGUI, thus you can simply adjust the various element in it to change it appearance. However it's recommend that you get yourself familiar with uGUI, understand how it works before you start tinkering it. There's a prefab that act as a UI template, located in 'TBTK/Prefabs/UI/UIObject'. Certain parameters in the default UI prefab that are made to be configurable.



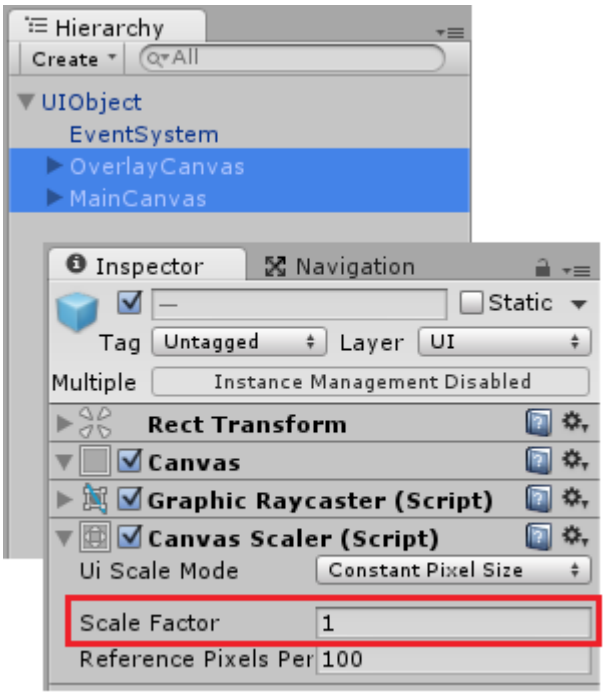

**PerkMenu**

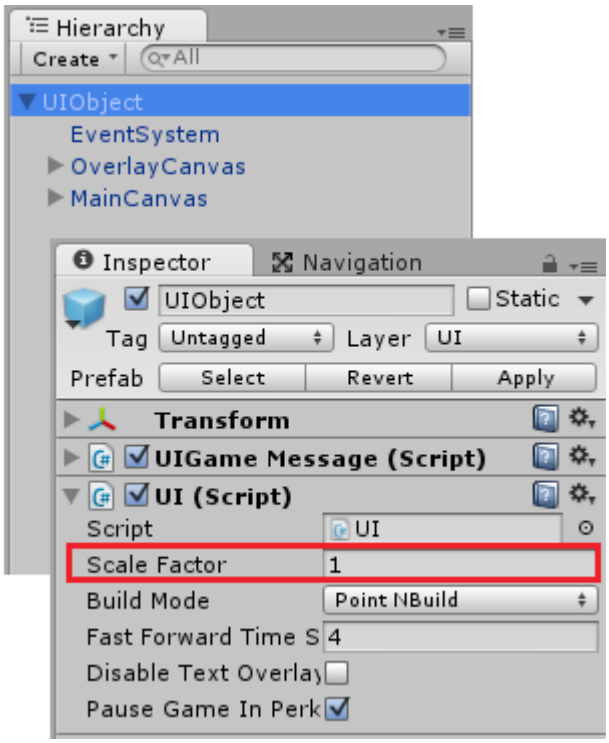
PerkMenu is the special UI object that support customization, mainly to provide a mean to build custom tech-tree in conjunct with the perk system. When the flag '*AssignItemManually*' is checked. You can arrange the button anyway you want, you just need to assign them as item to UIPerkMenu component as shown in the image below. Each of the button can be given a perk to associate with. The link object will be activate when the assocaite perk is purchase and deactivated if otherwise. When the flag '*AssignItemManually*' is off, all the perk in available AbilityManager will be assigned a button automatically.



**Scalling the UI**

When building for mobile device with high screen resolution and relatively small physical size, you might want to scale up the UI in case they are too small on screen. To do that, you can tweak the value of the ScaleFactor on CanvasScalar component on each of the canvas. At the same time, you will need to change the ScaleFactor on UI component to match.



# THANK-YOU NOTE & CONTACT INFO

Thanks for purchasing and using TBTK. I hope you enjoy your purchase. If you have any feedbacks or questions, please dont hesitate to contact me. You will find all the contact and support information you need via the top panel "***Tools/TBTK/Contact&SupportInfo***". Just in case, you can reach me at k.songtan@gmail.com or TBTK support thread at Unity forum.

Finally, I would appreciate if you take time to leave a review at AssetStore page. Once again, thankyou!