TurnBased-ToolKit

Documentation for Unity3D

Version: 3.0 f5
Author: K.Song tan
LastUpdated: 19th July 2019

Forum: http://goo.gl/3hntLv
WebSite: http://songgamedev.com/tbtk
AssetStore: http://u3d.as/5ed

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.

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 Note:

TBTKv3.x is a new version redesigned and written from scratch therefore it's not backward compatible with earlier version of TBTK.

If you are updating an existing TBTKv3.x and don't want the new version to overwrite your current data setting (towers, creep, damage-table, etc), Just uncheck '*TBTK/Resources/DB_TDTK'* folder in the import window.

OVERVIEW

How Things Work

A working TBTK scene has several key components to run the basic game logic and user interaction. These are pre-placed in the scene. They have various setting that can be configured and the setting dictates how the scene is play. For instance, what type of grid to use, how the unit move order is decided, can unit-A attack unit-B and so on.

Then there the prefab like units and item like abilities. These are the object to form a valid game scene. Each of these has their corresponding control component on them. The stats of and behaviours are configured via those components. These prefabs and item are assigned to a series of database. The key components will access the database to access these prefab. So the game know which unit to spawn, which unti has access to which ability and so on.

Basic Components And Class

These are the bare minimal key component to have in a basic functioning TDTK scene. You will need at least one of these in a working scene. There can be only one instance of these component in the scene.

GameControl and TurnControl Control the game state and major game logic.

AI Runs the AI algorithm.

GridManager Store the grid information and runs all the logic on the grid.

GridGenerator Generate the grid.

GridIndicator Control all the visual indication of the game state. It's not mandatory but without it the game would certainly be unplayable.

UnitManager Store the information of all the units.

AbilityManager Runs the logic for all abilities.

CollectibleManager Runs the logic for all collectibles.

Attack An abstract class that runs all the logic during an attack (or when an ability hit)

AStar An abstract class that runs all the logic for path-finding

Prefab Components And Class

These are the component that interacts with each other in runtime to form the gameplay.

Unit* The component on each unit.

Collectible* The component on each collectible.

ShootObject* The component on objects fired by unit during an attack to hit the target.

Node The individual node on the grid.

Ability The ability item.

Effect The effect item that can be applied to a unit during runtime.

Perk The item used to altered item (unit, ability, effect) based stats during gameplay

- These are prefabs.
-

Optional/Support Components And Class

These are the optional component for a TBTK scene. There can be only one instance of these components in the scene.

PerkManager Control the logic for perks system and contain the information of usable perks. It's required only if perk system is used in the level.

ObjectPoolManager* The component responsible for all the logic for object-pooling.

CameraControl The control component for camera.

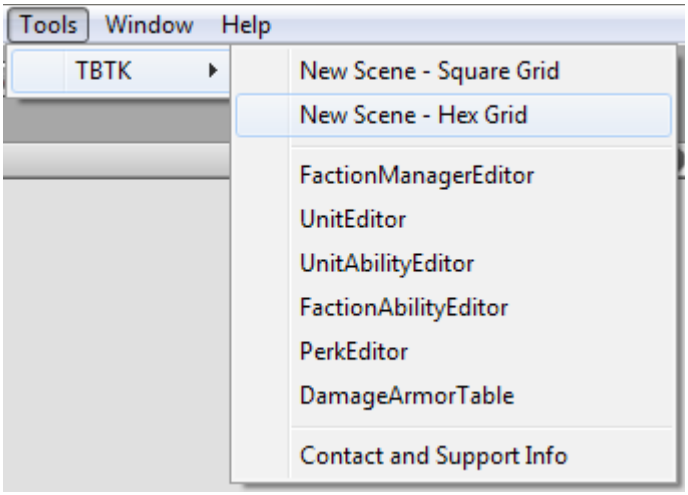
* These will be created in runtime automatically if they are required but are not present in the scene

UI

Although the UI is required for a level to be playable, it's a modular extension of the framework. You can have the base logic running without it. That means you can delete all the UI component and replace it with your custom UI.

HOW TO:

Create A TBTK Scene

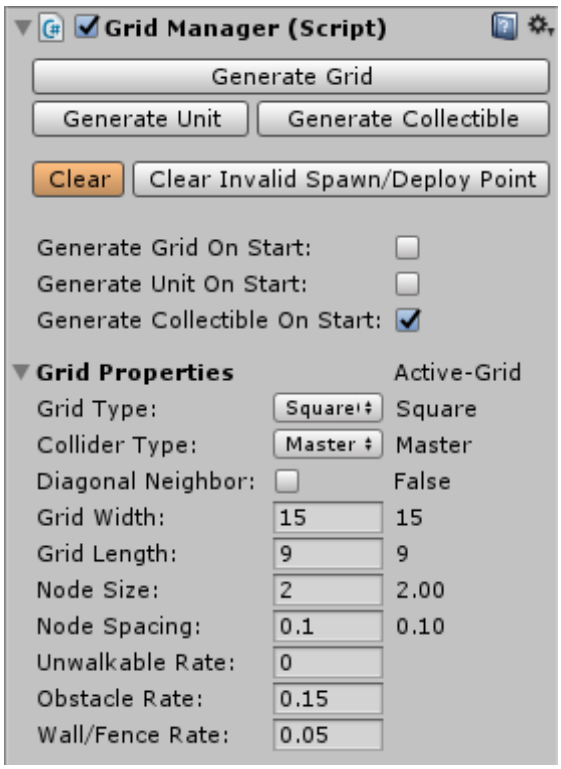


Assuming that you have some basic know how about Unity, It's very easy to create a functioning game using TBTK. To create a new TBTK scene, simply access the top panel '***Tools/TBTK/New Scene - Square Grid***' or '***Tools/TBTK/New Scene - Hex Grid***'. The former should setup a scene with square-grid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

The scene is setup with some 2 faction (1 player and 1 AI) and ready to be played from the get go.

It's a recommended that you start with these default scene and configure it to your liking instead of starting from a blank scene. This way you don't need to worry about getting all the setting correct before the scene can run.

Generating Grid

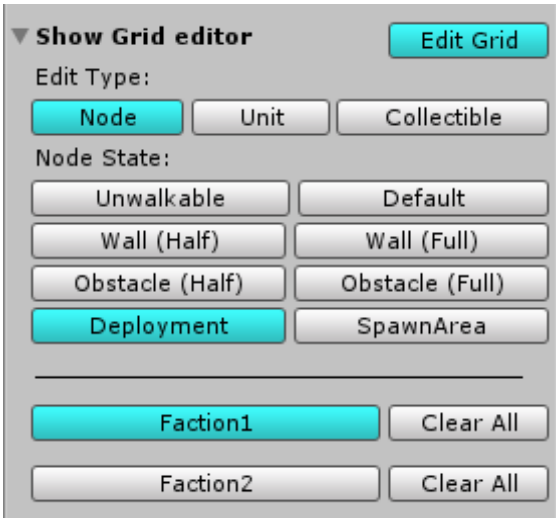


Generating a grid is pretty straight-forward. Assuming you are starting with the default scene. Just select the game-object 'GRID_MANAGER_GENERATOR' in the hierarchy.

In the inspector, you can specify the type of grid you want to generate, the dimension, node size and so on. When you are happy, just click the '**GenerateGrid**' button. Please note that generating a large grid may take sometime.

The note that when generating a grid, the new grid overwrite existing grid in the scene. Any item 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 'GRID_MANAGER_GENERATOR' object selected in Hierarchy).

You will need to make sure that you have the button 'Edit Grid' enabled. From there it, simply select what you want to be changed/added to the grid and then click on the grid in SceneView.

As seen in the image, there are 3 main categories which you can edit (Node, Unit and Collectible). Unit and Collectible are self explainatory. It's for adding/removing units and collectibles. Node are for editing the grid itself. Here's a brief explanation of what each option does.

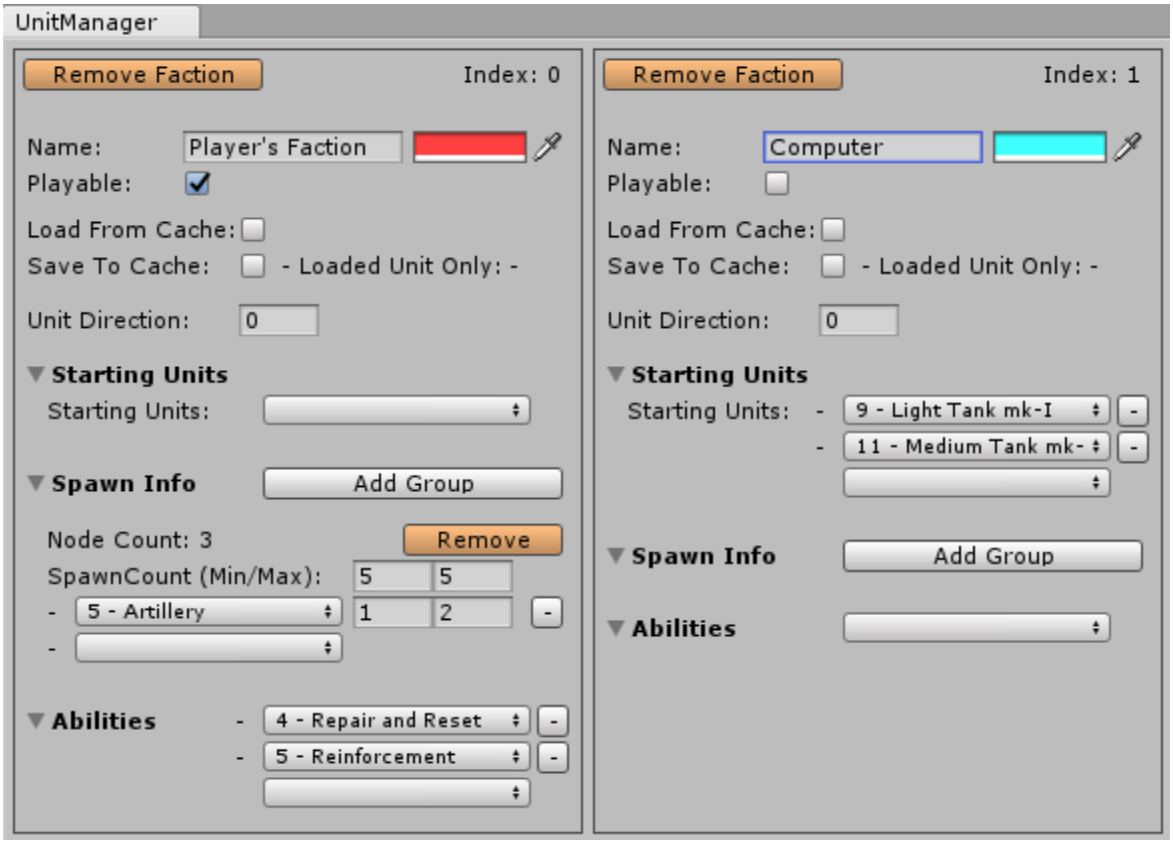
- **Unwalkable**: the node cannot be occupied by a unit and will be set to invisible (by default)
- **Default**: just a normal node
- **Wall**: an obstacle between 2 nodes which stop unit from unit between those nodes
- **Obstacle**: an obstacle that occupied the entire node
- **Deployment**: node which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

You can right-click on the grid in SceneView to remove the corresponding item selected in editor on the node. For instance, right-click on a node when the unit tab is selected will remove the unit on the node. Right-click on a node when 'Deployment' tab is selected will clear the node as a deployment node.

***Note**:
obstacle and wall 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 node 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 UnitManagerEditor, which can be open via the drop down menu, or selecting UnitManager 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.



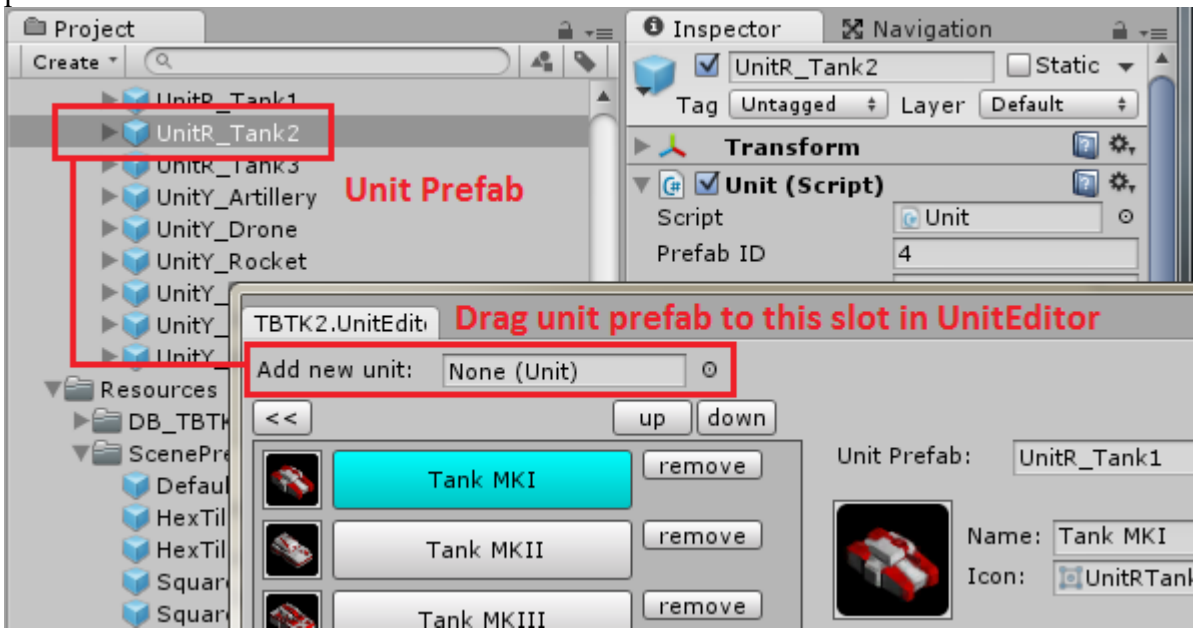
Each unit can has it's own starting unit, deployed unit, spawn/deployment area on the grid as well as Ability.

StartingUnit are the fixed set of units the faction possesses at the start of the game. These units will be deployed on faction's deployment area.

Spawn Info are information to be used for procedurally unit generation. These unit will be spawned as soon as the grid is generated. You can set the placement of these units by specifying spawn area on the grid using GridEditor. Note that you can have multiple spawn groups to create group of units that scatter across the grid.

Adding New Unit Prefab To The Game

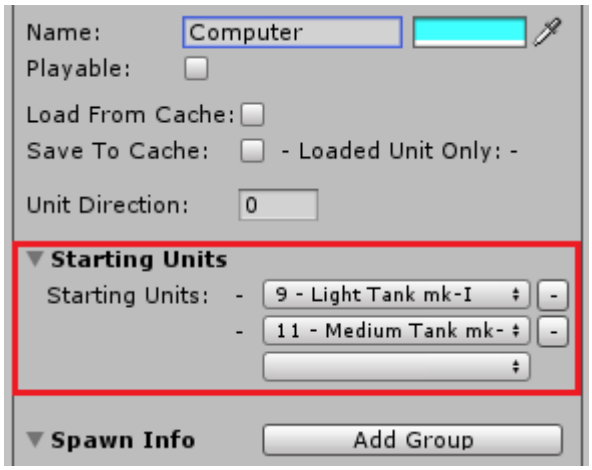
First you will need to create the unit prefab (refer to unit for how to create a prefab) and add it to UnitEditor. Once a unit prefab is in the UnitEditor, the unit should appear in other editor where unit prefab selection applies.



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

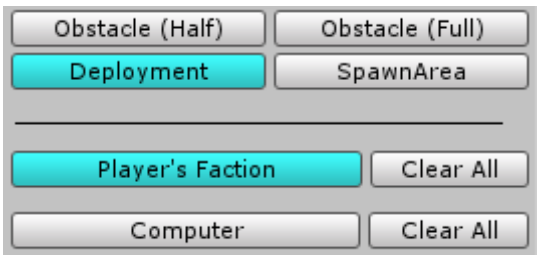
As Starting Unit



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 node in the SceneView, a cross with the faction's color will show up on the node to mark the node as a deployment node for the faction

As Deployed unit on the grid via UnitEditor

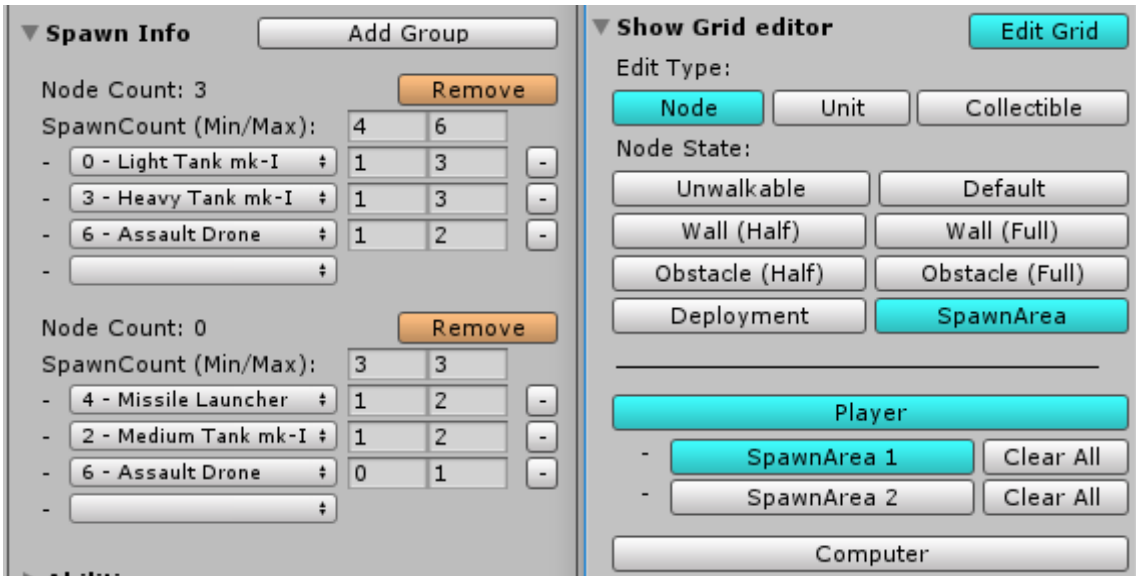
You can directly place a unit onto the grid via the GridEditor. Just Set the EditType on the GridEditor to Unit and you will see the all the unit prefabs show up in the Editor. You can select any of them and simply click on any node on the grid and the an instance of the selected unit prefab will be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

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 nodes 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 UnitManager.

Load From Cache
Finally the unit can be load from a previous scene. To do that, you will need to check the “Load From Cache” option for the faction. By doing that the faction will load any unit in the runtime cache as the starting unit. You can either assign unit to be loaded using code or using the 'Save To Cache' option.

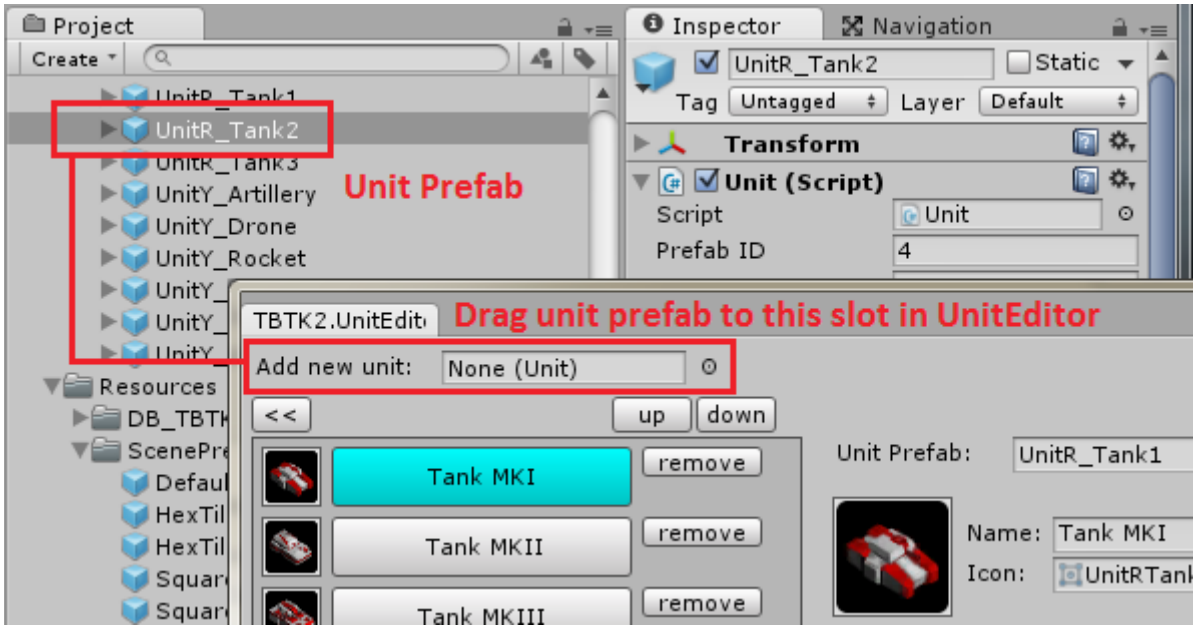
Enable the 'Save To Cache' option will save the remaining unit in the current scene to the cache. So if the next scene has 'Load From Cache' enabled, the remaining unit from will scene will be used as the starting units.

Refer to the section 'Integrating TBTK To Your Game' for more information

Add New 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.

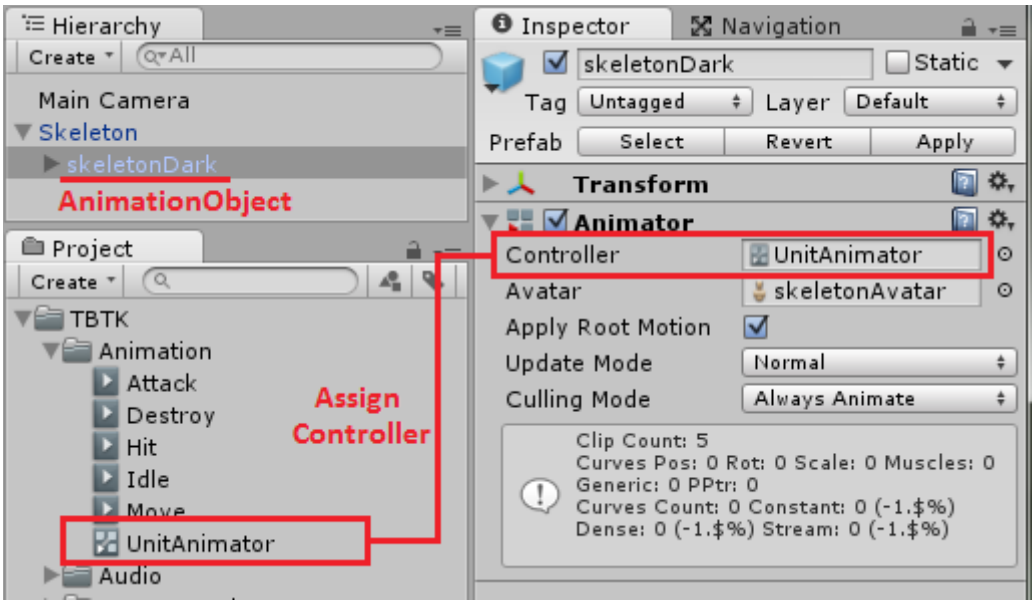
Collectible prefab can be created and added to Collectible editor in the same way.



Add New Animation To The Unit Prefab

TBTK uses mecanim animation system. The animation to be used for a unit in various event can be assigned in UnitEditor. However before you can do that you will need to setup the prefab properly. First you will need to make sure there's an Animator component on prefab. This usually comes with the animated model. You will need to assign the *UnitAnimator* animation controller as the controller for the *Animator* component as shown in the image below. Once done, you should be good to assign the animation clip you want to use in the UnitEditor.

*The system works with legacy animation clip, you can use it as it's with legacy animation clip.



Work With Collectible

The work flow of creating collectible prefab and adding them to the game is very similar to the work flow of working with unit prefab. The only difference of the setting of collectible generation on the grid is done on CollectibleManager.

To create new collectible item, simply create new object and attach the script Collectible.cs to it. Like unit, any visible mesh/effect on the prefab is optional. Once it's made into a prefab, it's ready to be added on to the grid.

There's two way of adding collectible to the grid. Manually, you can use GridManager and plop individual item on the grid. Alternatively you can let the procedural generation do the work. The parameters for procedural setting can be found on CollectibleManager.

HOW THINGS WORK:

Item DB (DataBase)

TBTK uses a centralized database to store all the information of the prefabs (units and collectibles) and in game item (ability, perk, damage and armor type, etc.). The in game item can be created with just a simple click of a button in their associated editor. The prefabs however, needs to be create manually before they can be added to the database (see 'Add New Prefab to The Editor' for more information). Once added, they can be accessed and edit via editor. Prefabs that is not added to the database will not be appear in the game.

Abstract item like ability and effect can be created in their corresponding editor. Once created, they can be accessed in all other editors. Unit-ability will shows up in UnitEditor and so on.

Perk can be individually enabled/disabled in PerkManager via Inspector. Disabled perk won't be available in the scene.

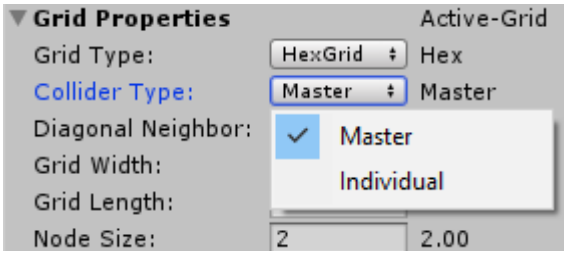
Global Setting & GameControl

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

The grid are made out of a series of game-object. Each node and item on the grid is a prefab. These prefabs are assigned in GridGenerator. You can change the appearance of the node by changing the switch the default prefab to a custom prefab. Note that you can assign multiple prefabs for the obstacle and wall.

Collider on Grid



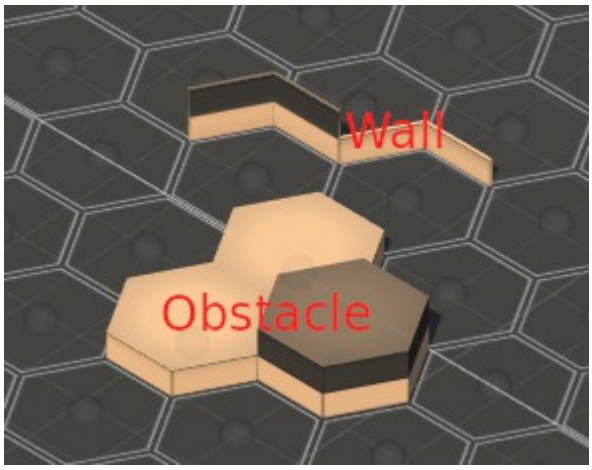
To enable detection of cursor on node in runtime, the grid will need to have collider(s) on it. There are two mode which can be set on GridManager – ColliderType. The first one is for the whole grid to use a single master collider (thus using the node prefab with no collider). The second one require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

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

customized (ie. Height adjust, position shift, etc.)

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.



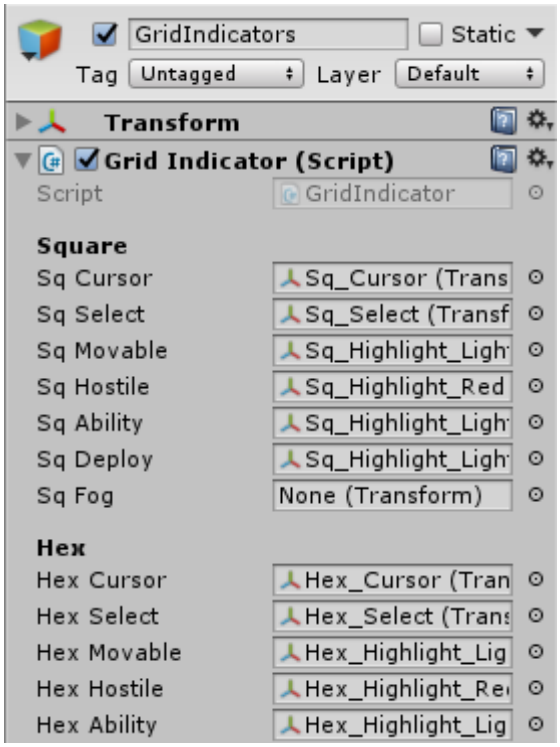
Obstacle are placed on top of a node to stop the node from being occupied by any unit. Walls are placed between two nodes to block access between those two nodes in question.

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

- a collider
- layer assigned to CoverFull/CoverHalf (27/28 by default)

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

Visualization & Indicators

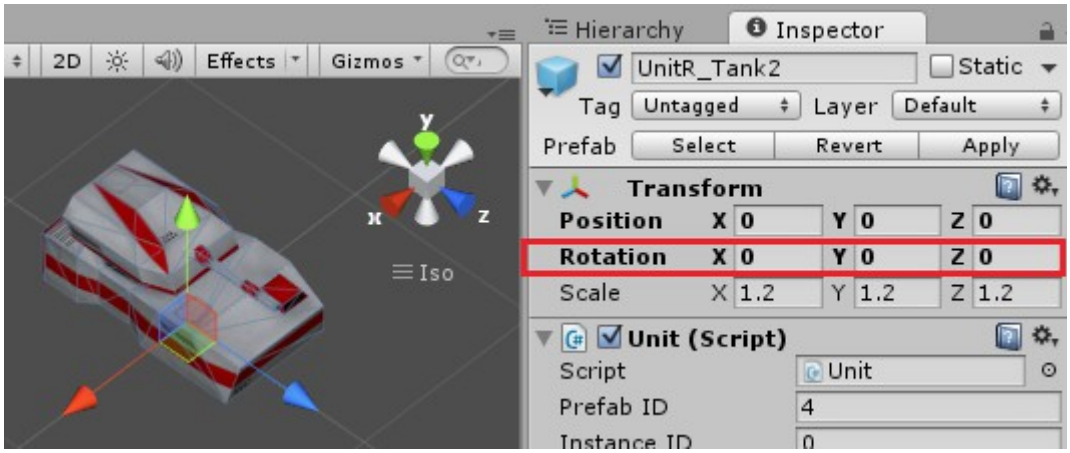


The appearance of the cursor/target indicators, as well the the appearance of the node indicating various status of the node can all be customized. These indicators are basically gameObject prefabs of various appearance. They are assigned to GridIndicators (on the game-object GridIndicators). You can change the appearance of a certain state of the node or an indicator by changing the prefabs.

Unit Prefab

A primitive unit prefab can be an empty game-object 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.

Hybrid Unit (range/melee Attack)

It's possible to have unit that switch between range and melee attack depends on the target range. To do that, simple check the “Hybrid Unit” option in UnitEditor. This will open up another set of parameter attributed to the unit melee attack (note that a melee only unit will use the default attribute, not the extras). You can then set the melee range for the unit. In runtime the unit will switch between melee and range attack depending on target range.

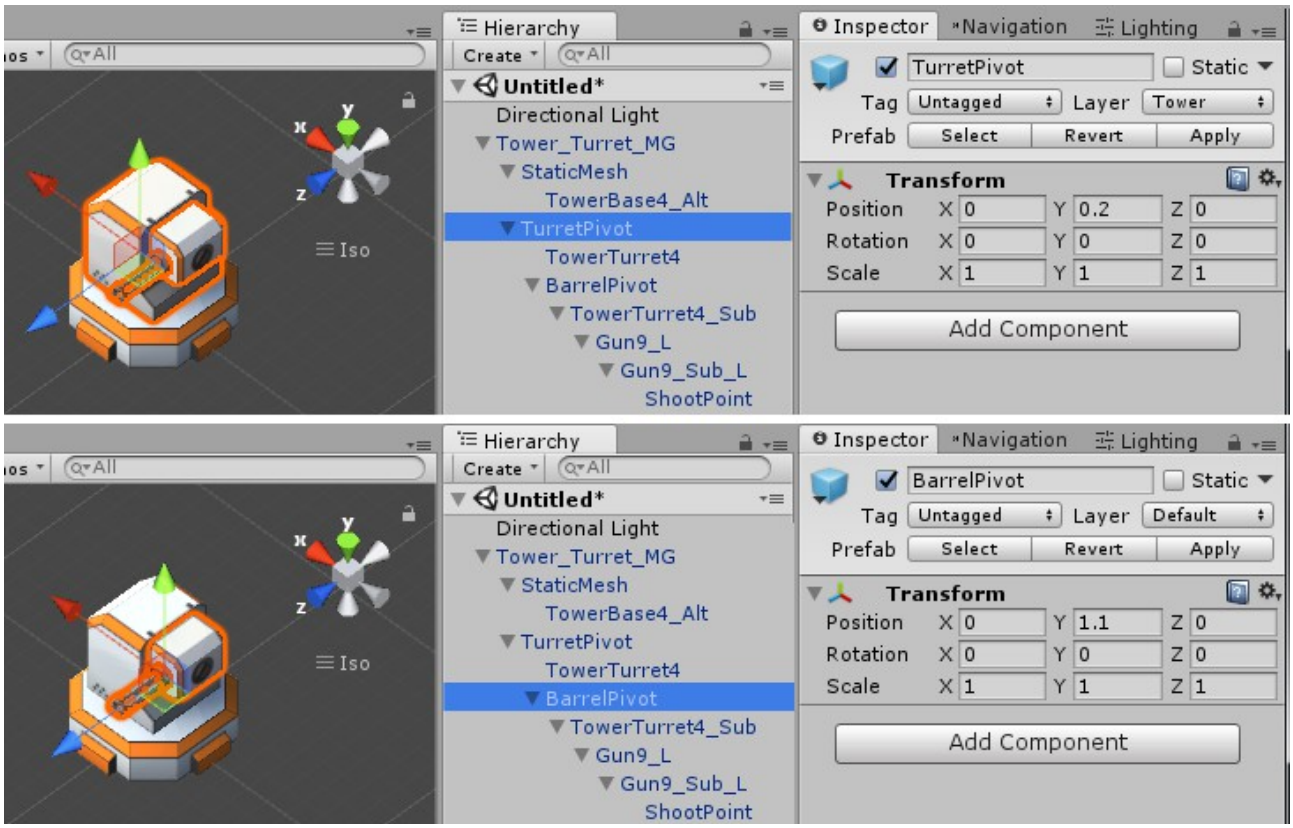
A hybrid unit will have 2 sets of stats (damage, hit chance, etc.) for normal attack. One for range and one for melee. Each attack type will have their respective shootObject, audio and animation.

Get Unit To Aim And Fires At Target

Turret type unit attack target directly by firing shoot-object at them. In many case, you might want the unit to aim towards the target and have the shoot-object fired from the correct position in relation to the model/mesh.

For aiming, the code will rotate the assigned transforms 'Turret-Pivot' and 'Barrel Pivot' in the unit's transform hierarchy to face the unit current active target. The way the model works, 'Turret-Pivot' is the main pivot that can be rotate in both x-axis and y-axis. 'Barrel-Pivot' on the other hand is optional and expected to be anchored as a child to 'Turret-Pivot'. 'Barrel-Pivot' will only be rotated in x-axis.

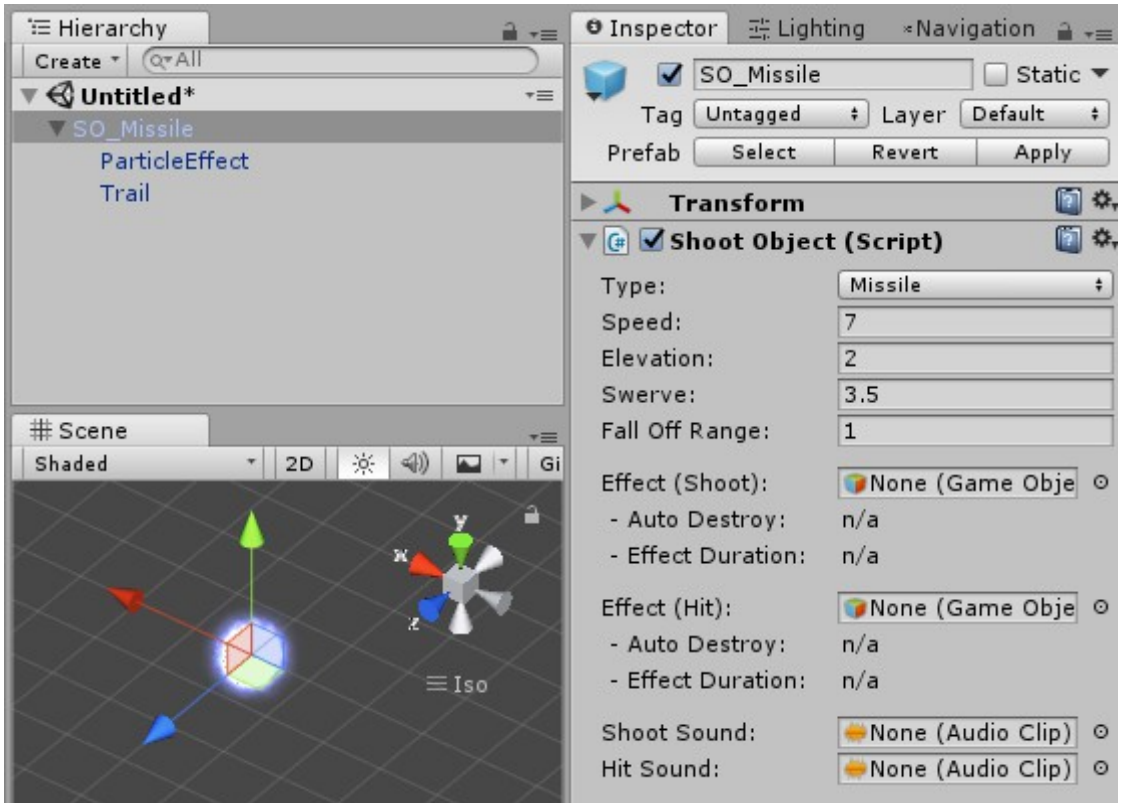
To make sure the turret aim in the right direction, you need to make sure that the rotation of both 'Turret-Pivot' and 'Barrel-Pivot' is at (0, 0, 0) when aiming towards +ve z-axis as shown as the image below. If you are not sure, please refer to default tower prefab. To understand why this is setup the way it's, it's strongly recommended that you go through this tutorial video to get the basic understanding of hierarchy, parent-child relation.



Shoot-point are the reference transform in the hierarchy of a unit to indicate the position where shoot-object should be fire from. To have it work with the aiming, anchor it as a child object of 'Turret-Pivot' or 'Barrel-Pivot', if there's one. Again, please refer to default tower prefab. You will find that they all positioned at the tip of the model's barrel.

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.



Shoot-object are objects that fired by turret type tower or creep during an attack to hit the target. A working shoot-object can be as simple as a empty game-object with the script ShootObject.cs attached on it. This would enable the shoot-object to be assigned to any turret type unit and be fired upon attack despite it's not visible. Any mesh/model/visual-effect on the shoot-object is optional.

There are three type of shoot-object and each of them serve a different purpose.

Projectile A typical object that travels from firing point toward the target before it hits. A projectile shoot-object can be configured to simulate a curved trajectory.

Missile Similar to 'projectile' type shoot-object except it also curve along y-axis

Beam Used for a beam effect where LineRenderer is used to render a visible line from shoot-point to the target. A timer I used to determine how long the target is hit after the shoot-object is fired.

Effect For attack that doesn't require the shoot-object to travel from shoot-point to target. A timer I used to determine how long the target is hit after the shoot-object is fired. Used to melee attack.

Unit And Faction Ability

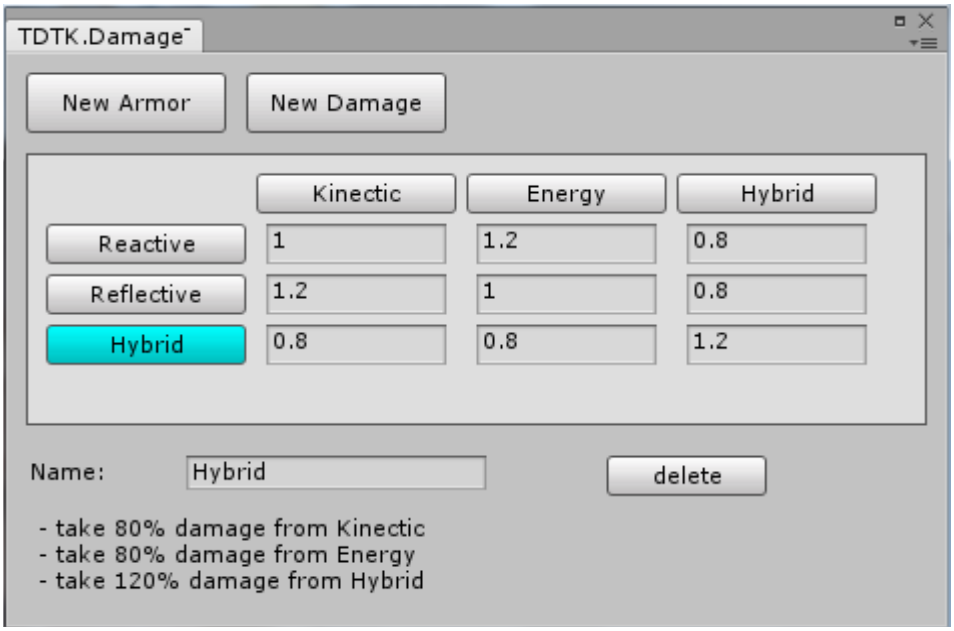
Unit abilites are abilities that ties to unit. They can be customized to do a variety of things. You can create unit abilities in UnitAbilityEditor and assign them to each unit in UnitEditor (both can be access via top panel). Once created, a unit-ability can be shared across multiple unit.

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 UnitManagerEditor.

Damage Table

DamageTable is multiplier table used to create a rock-paper-scissors dynamic between units. You can setup various damage and armor type using DamageTableEditor (access from the drop down menu). Each damage can act differently to each armor. (ie. damage type1 would deal 50% damage to armor1 but 150% damage to armor2).

Each unit's attack, ability and effect that could cause damage on target can be assigned a damage type.



Perks And PerkManager

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 drop down menu. 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.

You can enable persistent progress of the perk in a game session by enable the option 'Save To Cache On End' on PerkManager. This will cache any progress made with the perk (unlocking perk). If any subsequent level has a PerkManager with 'Load From Cache On Start' enable, the PerkManager will retrieve the cached progress so any perk unlocked in the previous level would have been unlocked. Note that this is not saving, the progress only last through the game session. Once the player exit the game, the progress is lost.

Please note that PerkManager only support playable faction and the perk bonus will be apply to all playable faction

Fog-of-War

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

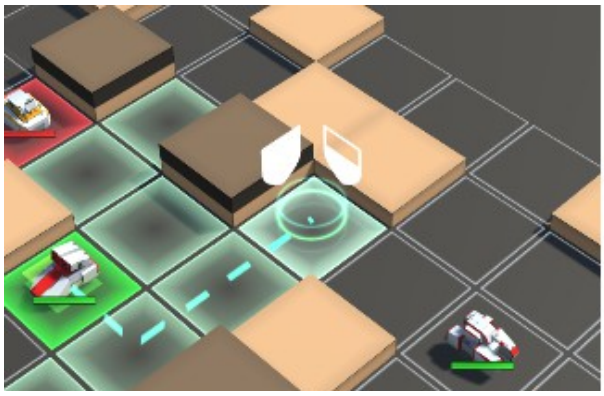
- Being out of all hosnode units sight.
- Has no direct Line-of-Sight with any hosnode unit (blocked by a obstacle with full cover)

A unit cannot attack a hosnode that is concealed.

CoverSystem & Obstacle

CoverSystem is a built in system in TBTK to emulate the cover system seen in game like X-Com. 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 node with cover.



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 scene Demo_Persistent (PreGame and Game). You can refer to the script Demo_Persistent_PreGame.cs for example code. The script is fully commented to help you understand it.

Before diving into the code, you will need to enable some setting on UnitManagerEditor. For each faction, there's a option for LoadFromCache and SaveToCache. Enable LoadFromCache indicate the faction will be loading the starting unit list from cache and enable SaveToCache will have the UnitManager to cache the remaining units on the grid to be load in the next scene.

To cache the starting unit for a particular faction in the next game scene, simply call the function:

> **UnitManager.CacheFaction ()**
> *public static void CacheFaction(int factionIndex, List<Unit> unitList)*

The factionIndex are the index of the faction in the intended scene. The value is displayed on UnitManagerEditor. The unitList is simply the list of the unit prefab to be used. The unit specified will be use to override the starting unit.

To retrieve the units from the cache, simply call the function:

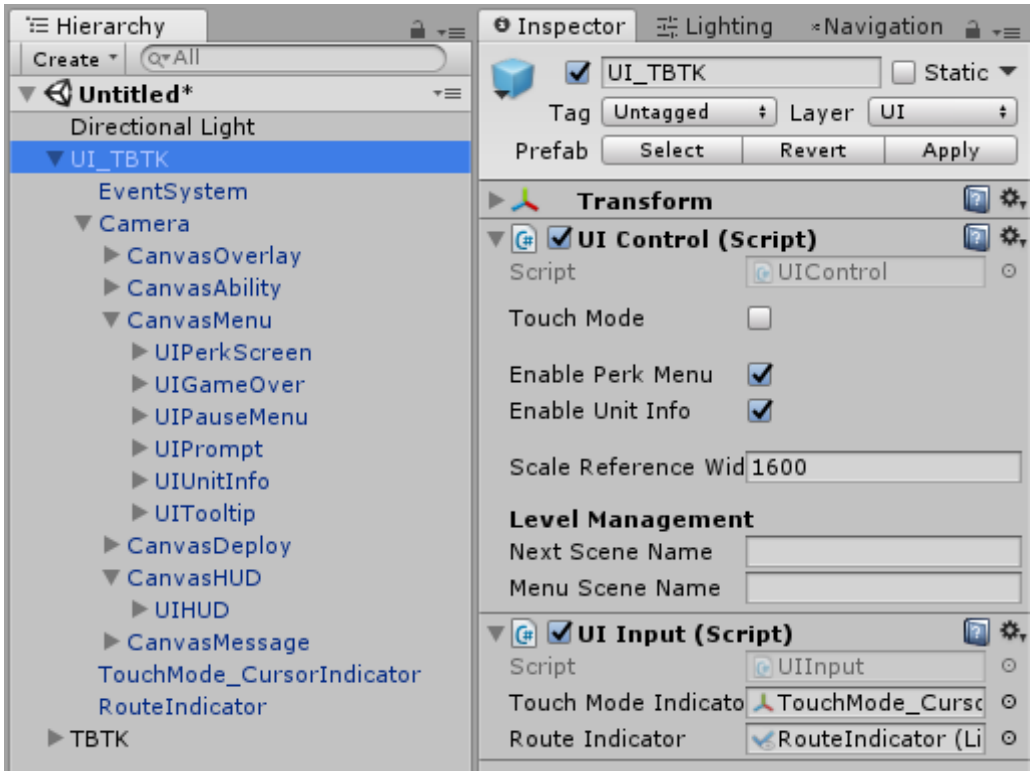
> **UnitManager.*GetCachedUnitList* ()**
> *public static List<DataUnit> GetCachedUnitList(int factionIndex)*

The function will return a list of unit being cached.

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 thinkering it. There's a prefab that act as a UI template, located in 'TBTK/Resources/NewScenePrefab/UI_TBTK'.

Certain parameters in the default UI prefab that are made to be configurable. They are immediately visible on UIControl component when you select the UI_TBTK prefab (as shown in the image below). In most case you won't need to bother with other setting of other UI component unless you want to modify the UI.



The UI are made to be touch control compatible. For touch interface on mobile device. It's recommend that you have the 'Touch Mode' enabled. Touch Mode s a special mode that when enabled, require double tap on button to trigger build/ability button (first tap to show the tooltip, second to confirm). This is specifically made to solve the issue where on touch device, it's impossible to hover over a button with mouse cursor. The UI should scale automatically in most case. However you might want to disable 'Limit Scale' in case the UI elements gets too small on device with smaller screen.

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. You can refer to how this is done in the demo campaign menu.

When the flag '*Custom Layout*' 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. As well as specifiy the ID of the perk the button is associated to. When the flag '*Custom Layout*' is off, all the available perk enabled in PerkManager will be assigned a button automatically.

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!