TopDownShooter-ToolKit

Documentation for Unity3D

Version: 1.3
Author: K.Song tan
LastUpdated: 15th December 2016

Forum: http://goo.gl/HVsZQ9
WebSite: http://songgamedev.com/tdstk
AssetStore: http://u3d.as/mqT

Thanks for using TDSTK. This toolkit is a collection of coding framework in C# for Unity3D, designed to cover most, if not all of the common top down shooter ("TDS") game mechanics. Bear in mind TDSTK is not a framework to create a complete game by itself. It does not cover elements such as menu scenes, options, progression saving 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. Once you grasp the basic, the rest should be pretty intuitive. Almost all of the editable elements of the toolkit comes with tooltip in the editor. You can just hover over the label to get an idea of what a particular setting is about. For that reason, most of this documentation only explains the general concept of the toolkit and things that are either less obvious or aren't included in the tooltip.

Finally, 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 TDSTK and don't want the new version to overwrite your current data setting (units, weapons, collectibles, etc), Just uncheck '*TDSTK/Resources/DB_TDSTK'* folder in the import window.

OVERVIEW

General

Basic components. All component are optional with the exception of GameControl and UnitPlayer. Special case being AbilityManager, which is required if the player uses any ability.

- **GameControl/DamageTable* -** control all the general game logic (life, hit logic, etc.)
- **ObjectiveTracker*** – a component to track and the objective in the level
- **UnitPlayer*** - the component on player, handle everything the player do
- **PlayerProgression** – the component that hold and process the level information of player unit
- **PlayerPerk** – the component that hold and process the perk information of player unit
- **AbilityManager*** – component that govern player's ability
- **DropManager*** – handle the spawning of collectible(bonus) item when hostile unit is destroyed
- **UnitSpawner** – the component that handle procedural spawning of additional AI unit
- **CollectibleSpawner** – the component that handle procedural spawning of collectible item
- **Trigger** – component that is used to trigger various event when activate by player unit
- **TDSArea** - a component that is used to define effective area for various spawner/trigger

***There cannot be more than one of these component in the scene at any given time

Runtime component: - These are in game object that can be pre-placed or spawned during runtime

- **UnitAI** – component that drive each individual AI unit
- **Weapon** – the weapon of the player, spawned when the level is loaded
- **ShootObject** – the 'bullet' object fired by all units
- **Collectible** – item that can be collect by player for various bonus effect

Support components: - These secondary supports the mechanic of the game so that it's playable

- **CameraControl** – handle the in game camera
- **AudioManager** – component that handle the audio aspect of the game
- **UI** – a series of component that handle the User interface

Demo And Examples

The framework comes with sets of example scene and prefabs that show case how the toolkit works. These scenes are what you would have seen in the playable web demo. The scenes can be located in '***TDSTK/Scenes/***'. There's another sets of similar scenes with the UI and input set to be mobile ready located in '***TDSTK/Scenes/Mobile***'.

BASIC CONCEPT:

Player Unit – In Game Object

The unit controlled by the player. You can assign multiple weapons and abilties to a player unit, which can be switched during runtime. The player primary attack will depend on the weapon being selected.

AI Unit – In Game Object

AI unit is pretty much any unit (not necessary hostile) that can be attacked by player in the game. They can be setup with various behaviour and attribute to accommodate various role (a stationary neutral target, an aggressive hostile that shoots, a proximity mine, etc...)

Weapon – In Game Object

Weapon defines that attack of the player unit. Each weapon has it's own stats and shoot-object. Each weapon can also be assigned a unique alt-attack mode, in the form of an ability.

ShootObject – In Game Object

Shoot Object are 'bullet' object that is fired by a player's weapon/ability or AI unit. There are different kind of shoot-object, namely simple, homing, beam and point. Please refer to section for how each shoot object works.

Collectible – In Game Object

Collectibles are bonus item can be collected by player unit in game for various effect.

Effect

Effects a buff/debuff that can be applied to units. This effects can be applied through a normal weapon attack, an ability, a colletible.

Ability

Ability are special action available for player. They can be used for various effect like a direct attack, a game wide debuff, a self buff and so on.

UnitSpawner

Spawner that procedurally spawns AI unit during runtime.

CollectibleSpawner

Spawner that procedurally spawns collectible item during runtime.

DamageTable

DamageTable is multiplier table used to create a rock-paper-scissors dynamic between attacks. You can setup various damage and armor type using DamageTableEditor (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 attack (from weapons, abilities, etc.) can be assigned to use a specific damage

type and each unit can be assigned a specific armor type.

ShootPoint (for Weapon And AI Unit)

Shoot points are reference transform assigned to weapon and AI unit to indicate the position where the shoot-object should be fired from and the direction they are facing. For a typical weapon which shoots straight, it's important that the z-axis direction of the shoot-point transform is aligned with the barrel of the weapon.

Triggers – In Game Object

'Triggers' are object used to trigger various event. Essentially they are just invisible area/switch in the scene. Upon triggered by player or AI unit (by moving into the area), they perform what that is required depends on the type of trigger used. Please refer to 'Making & Using Triggers' section for more information.

Objective & ObjectiveTracker

Each level in TDSTK can be set to have different objective. This objective can include destroying a certain group of units, fight until all the spawner has exhausted the available spawn, get to a series of check point, timed survival and so on.

Most of the objective parameter can be set in ObjectiveTracker component with the exception of timer (which can be set using GameControl) and win trigger (which is a trigger placed manually into the scene)

Level Up And Perks (PlayerProgression & PlayerPerk)

Player can gain experience and level up. As player level up, they can gain various stats bonus. The experience cap required to level and the bonus stats gained can be configured in ProgressionStatsEditor window.

Along with that, there's the perk system. Perks are unlockable and purchasable item that grant player various things like stats bonus, new ability /weapon or modify existing ability/weapon. Perk system can be tied to leveling system where player can get new perk when level up.

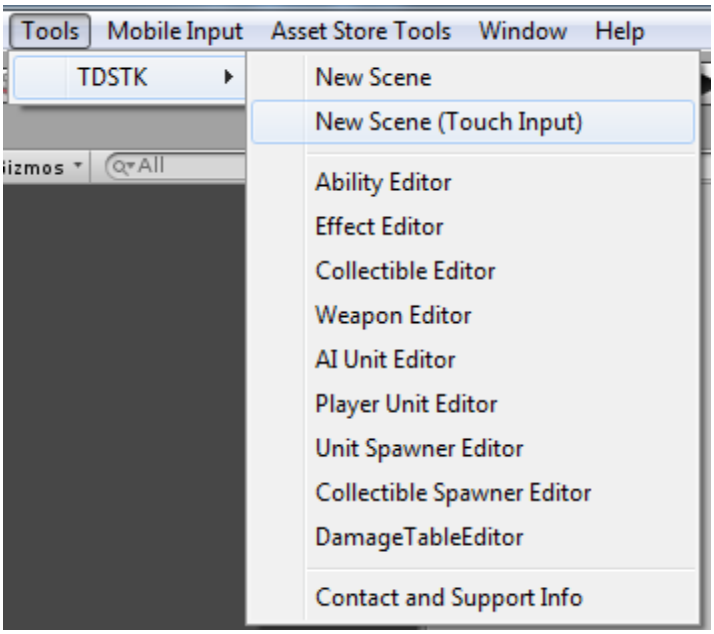
Both system can be configure to use in adjunct with each other create different synergy in game. Alternately they can be used independently of each other. Both are optional.

Database

TDSTK uses a centralized database to store a reference to some of the prefabs (unit, weapon, collectible) and in game item (effect and ability). 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. Once added, they can be accessed and edit via editor.

HOW TO:

Access The Editors



The first thing in TDSTK that you should be aware of is the drop down menu in the top panel.

From this menu you can create a simple starter scene, access various editor menu, as well as find the contact info for further support.

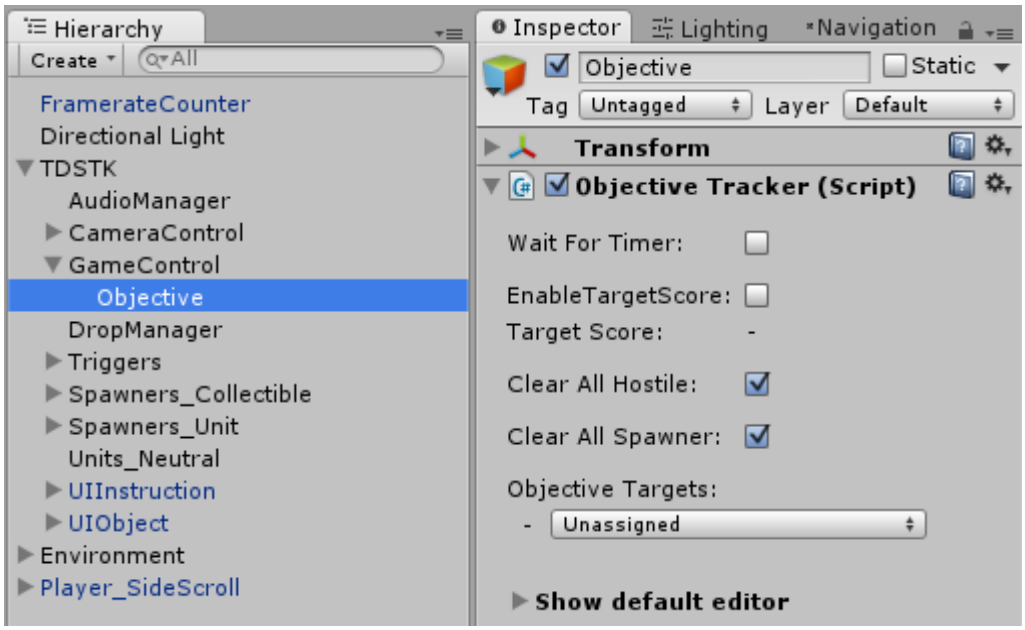
Getting Started - Create A TDSTK Scene And Modify It

To create a new TDSTK scene, simply access the top panel '***Tools/TDSTK/New Scene***'. This should set up a game ready scene. '***Tools/TDSTK/New Scene (Touch Input)***' would setup a game ready scene with user interface for touch input.

The scene contains all the elements you can potentially use. It's recommended that you always start with this template and then further customize the level to your liking. Change the player prefab, modify the objective, add trigger to spawn more unit and so on.

Setting Up Objective And Level Fail/Complete State

There are several criteria that could trigger a level complete or level fail state when using TDSTK. The first thing you should be aware of is the ObjectiveTracker component. You can assign one to the GameControl and the setting on the component would be used as the condition for level completion. You can enable as much condition on the objective component as you want and player would need to full fill all of those condition to complete the level. Each of the setting available on the component has a tooltip explaining what are they so please mouse over the setting in Inspector for more information.



On top of the ObjectiveTracker, there's a game timer on GameControl. When enabled, this timer limit the length of the level with the time specified. When the timer runs out, the level is considered completed if all the objective in the ObjectiveTracker has been completed. Otherwise the level is considered failed. The level can be completed in advance of the timer when the objectives are completed if wait-for-timer flag in ObjectiveTracker is disabled.

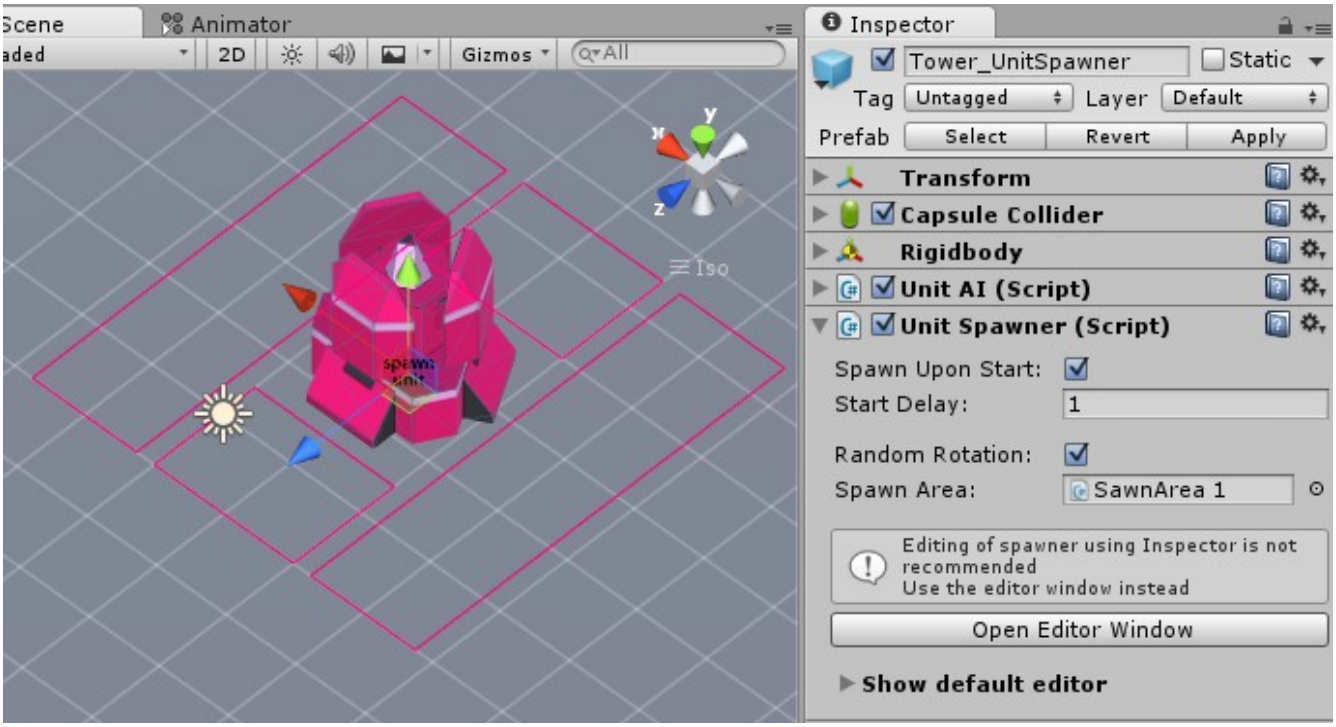
To fail a level, player either have to be destroyed and used up all their respawn, or fail to complete all the specified objective before the timer runs out.

Work With Unit Spawner

When designing level, you can place specific AI unit into the scene, or use UnitSpawner to spawn unit procedurally during runtime. You can have multiple instance of UnitSpawner in a single scene, each with their own spawn criteria and information. For instance, spawn 10 unitA when player gets to room1, then spawn another 15 unitB when player gets to room2. The exact spawn information can be configured using SpawnEditorWindow, where the settings are pretty much self-explained. Further more, there's tooltip to each of the settings.

It's worth mentioning that you can either set the spawner to start spawning automatically (with option to time it), or use trigger. When using trigger, the spawner will remain in dormant state until the trigger is 'triggered' by the player. For instance, place a trigger at the entrance of a room, player enters room hit trigger and the spawner in the room start spawning unit.

You can place a UnitSpawner either as a empty gameobject in the scene, or as a Unit. The later of course can be destroyed by the player. The exmaple prefab Tower_UnitSpawner does just that.



An example of UnitAI being used as a UnitSpawner

Work With Collectible Spawner

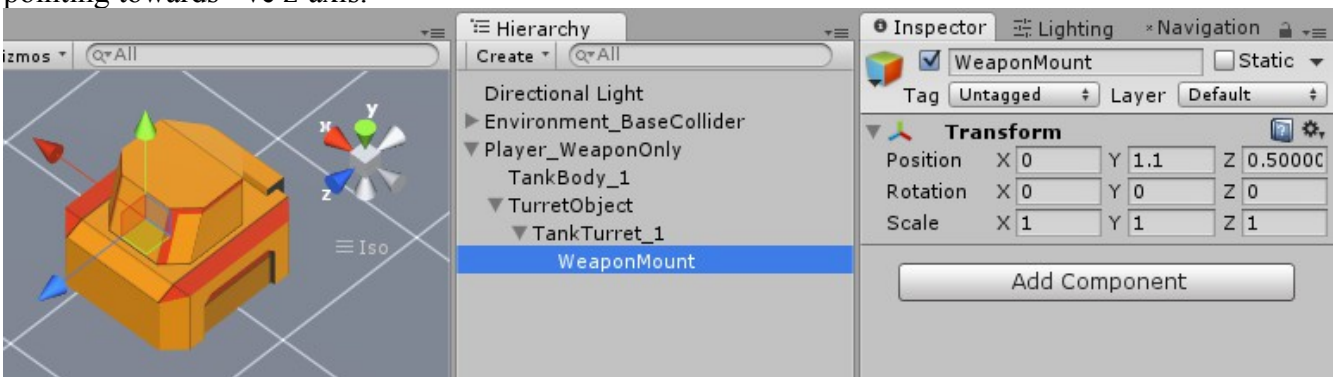
Collectible spawner is the collectible counterpart for UnitSpawner. It works pretty much like UnitSpawner in every way except it doesn't support a wave based spawn, for obvious reason.

Make A UnitPlayer Prefab

Making a player unit can be as simple as attaching UnitPlayer.cs script to an gameObject. The script will automatically add all the required components (Rigidbody and Collider) to the gameObject.

Of course to have the unit looks good in game or at least, have it aim towards where it's firing. For that, you will have to give it a turret object. Turret object is typically a child object within the player unit transform hierarchy which always aim towards where the cursor is pointing. When setting up turret object, it's important to make sure that the transform aim towards +ve z-axis when the rotation is (0, 0, 0). You can refer to the default unit player prefabs.

Moving on, you might want the player to be able to switch weapon during runtime and have the switched weapon show up in the right place. For that, you will need to assign a weapon mount point (in the form of a transform), where all the weapon transform will be anchored on. Typically you will want the weapon mount point to be a child to the turret object so that it follows the aim direction of the turret. And again, make sure that the transform aim towards +ve z-axis when the rotation is (0, 0, 0). You can refer to the image below, note that the rotation of the weapon mount point is (0, 0, 0) and it's pointing towards +ve z-axis.



It's vital that you keep all the child transform of the unit free of any collider to avoid any error in hit detection of shoot-object or collectible.

Finally it's recommended that you keep the root transform of the unit free of any mesh renderer. Instead, make any mesh a child object like the example prefab. That way you have the freedom to adjust the position of the mesh.

Make A UnitAI Prefab

Making a AI unit is very similar to making a player unit. You simply attach UnitAI.cs script to a gameObject and the script will do the rest.

Of course the same rules applies for other setting such as turret and mesh. Please refer to 'Make a UnitPlayer Prefab' section.

Make A ShootObject Prefab

A shoot object is just any game object with ShootObject.cs script attached on it. There are four distintive type of shoot object, which all work in very different manner and may require different setup.

Simple: a simple projectile shoot-object which just shoots forward from it's shoot direction. A simple shoot object will require a collider and a rigidbody attached to it. These component are optional however since the ShootObject component will automatically add them if there isn't any.

Homing: a homing shoot-object which tracks a specific unit, or a point if no unit is specified. In term of setup, a homing projectile is similar to a simple shoot object.

Beam: a special heat-scan based shoot object which instantly hits target in the line of fire based on it's shoot direction. In term of setup, a beam will require a LineRenderer component to render the line of fire.

Point: a special shoot object which uses cursor scan and hits directly (and instantly) at any object cursor position. That means it can shoot pass the any obstacle and hit a specific spot. This type of shoot object have no prior requirement in order for it to work correctly. However it's recommend that you at least give it a hit effect so that the hit is visible.

Make A Collectible Item

A collectible is just any game object with Collectible.cs script attached on it. You can just attach Collectible.cs on any gameObject and it would start working as an collectible item. To avoid any error in hit-detection, avoid having any collider in any of the child transform of the collectible item except the root object

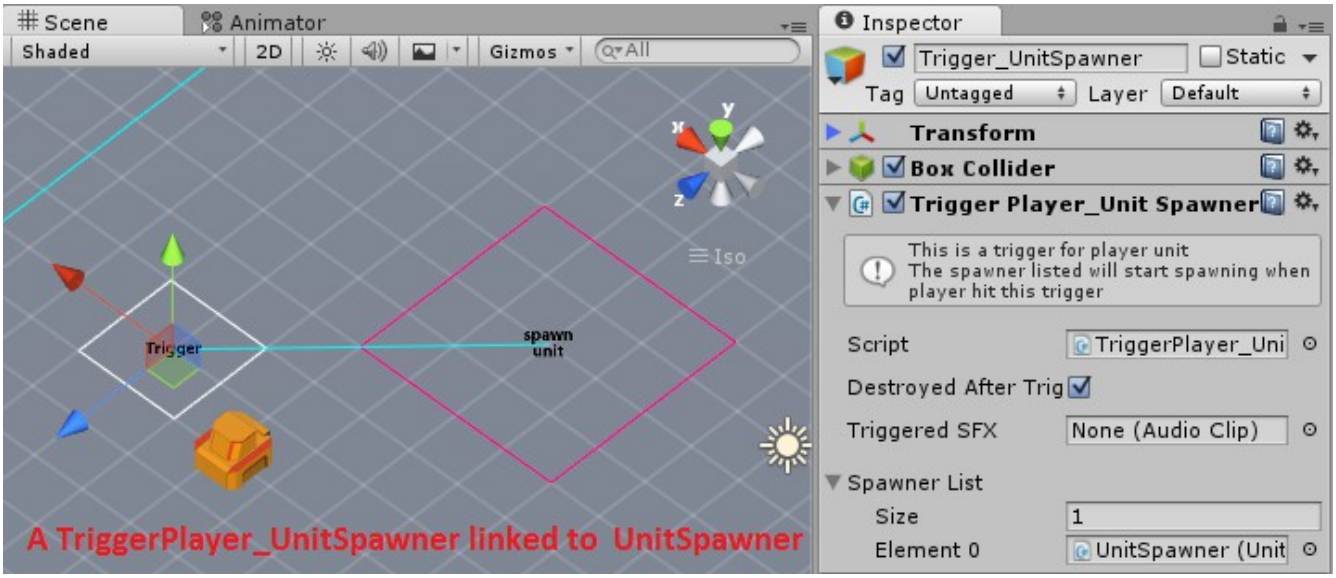
Making And Using Trigger

A trigger is simple an empty game object with script “Trigger” attached on it. The trigger area of trigger is depend on the scale of the game object. You can adjust them as you see fit. The gizmo sould show up the area cover as you adjust the scale.

Each type of trigger sever different purpose. The type of trigger available in current version are:

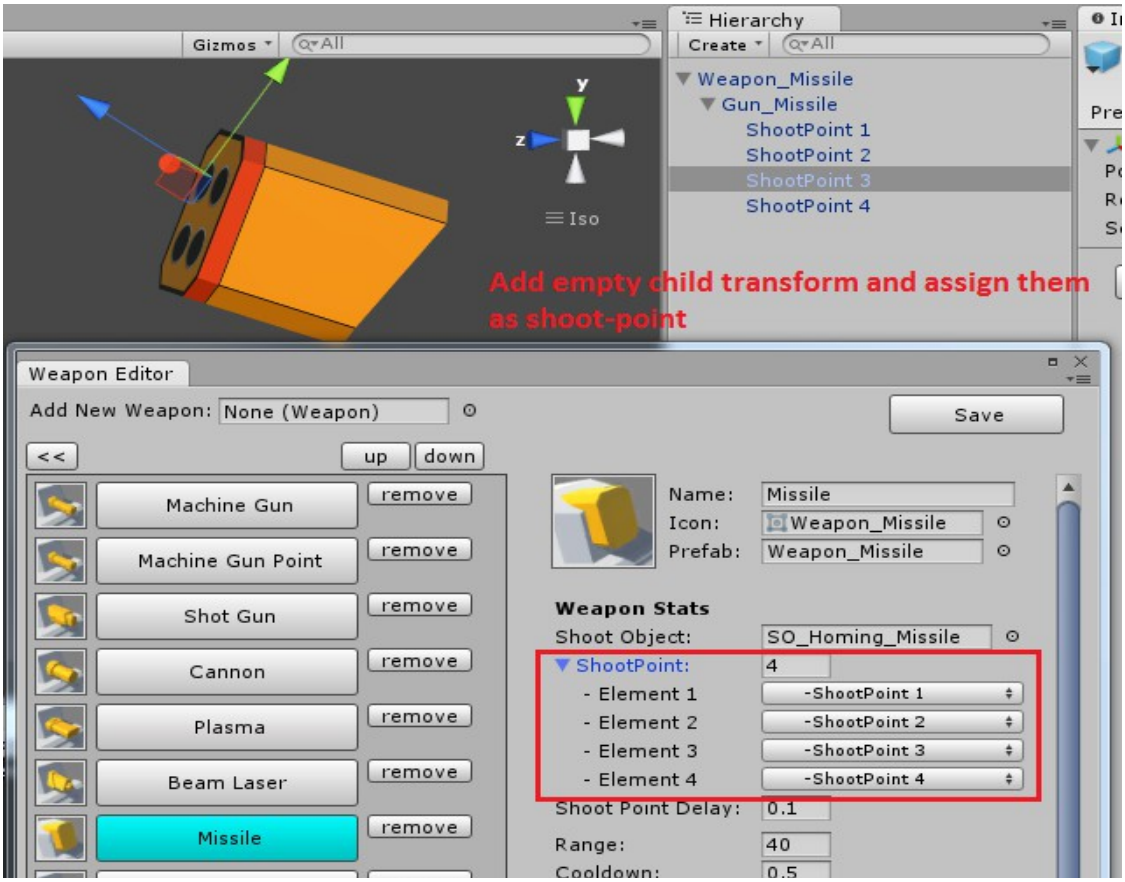
For hostile unit (to be triggered by hostile unit)
TriggerHostile_DamagePlayer – damage player unit
TriggerHostile_Kill - destroy the hostile unit that trigger it

For player unit (to be triggered by player)
TriggerPlayer_ActivateUnit – activate designated inactive units
TriggerPlayer_CollectibleSpawner – prompt designated collectible spawners to start spawning
TriggerPlayer_Damage – damage player unit
TriggerPlayer_Objective – complete certain objective
TriggerPlayer_PlayerSwitch – switch current active player to another player prefab
TriggerPlayer_RepawnPoint – activate the position as the new player respawn point
TriggerPlayer_SaveProgress – save the player current progress
TriggerPlayer_Teleport – teleport player to a designated position
TriggerPlayer_UnitSpawner – prompt designated unit spawners to start spawning
TriggerPlayer_Win – wins the level instantly



Make A Weapon

A basic weapon prefab can be an empty game object with the script “Weapon.cs” attached on it. However in most case, you can give it a unique appearance by adding custom mesh as one of the child object. Should you do that, you may want the shoot-object to be fired from specific position in sepcific direction, the barrel for instance. To do that you will want to assign specific shoot-point to the weapon in the form of empty child transform. These empty transform will simply be used as a dummy so that the weapon script knows where to spawn the shoot object and in what direction. You will need to assign this shoot-points to the weapon component. You can refer to the example weapon prefab how the shoot-point is arranged with respect to the mesh of the weapon.



Firing multiple shoot object per shot

There's two way to do this. First by increase the spread value on the weapon setting. However with spread, you are restricted in the sense that all the shoot-object is going to be fired from the same shoot-point and subject to a pre-defined, uniformly scattered angle. Alternately you can assign multiple shoot-point to the weapon. A shoot-object will be fired from each shoot-point, in accordance to their position and direction. To get a clear example of these difference, please refer to the shotgun and missile of the default weapon prefab. Please note that this will work on top of spread value, so if you have 3 shoot-points assigned and have a spread value of 3. A single shot from the weapon will fire 9 shoot-objects (3 from each shoot-point.

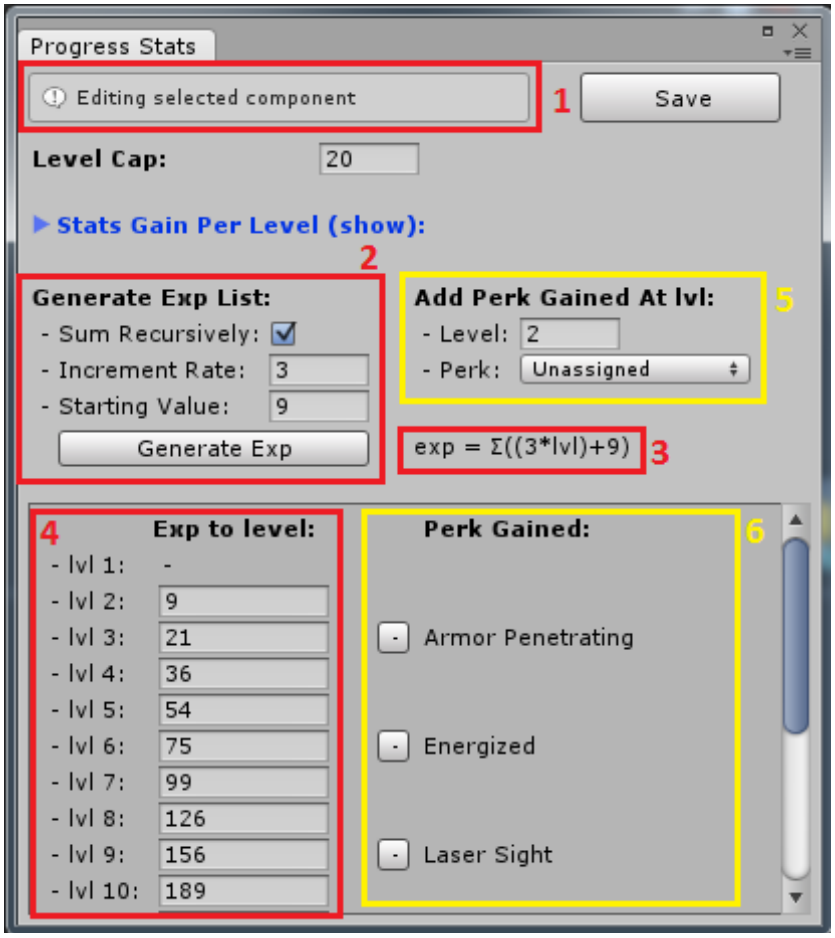
Since a weapon prefab is eventually going to be attached on a player unit as a child object. It's vital that it doesn't have any collider component on it to avoid hit-detection error.

Leveling And Perk System

Leveling System

Leveling system are optional extra where player can gain experience and level up. Leveling up grants player various stats bonus. To enable player leveling, you will need to attach the script PlayerProgression.cs to the player unit in the game. To configure the various stats in regards to leveling, you can use the ProgressStatsEditor window in the top panel. With the editor, you can procedurally generate the experience required for each level or manually edit each of them.

For the stats, there's a global value which can be load and used in every level. Alternatively you can disable the 'UseGlobalStats' option in PlayerProgression component so that it use stats store locally in that component. You can edit the local stats by selecting the player game object in hierarchy or project tab with ProgressStatsEditor opened.



1. indicate if the editor is editing the global or local stats
2. generating the experience list procedurally using a linear equation
3. the equation used to generate current experience list
4. the experience list indicate experience cap required for each level. You can edit each field directly
5. Add perk to gained at specific level. The item added will shows up in 6. You can add more than 1 item to a level.
6. Indicate the perk player gain at what level. Each item can be removed by clicking on the '-' button.

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 existing stats, add new weapon or ability, modify the stats of a ability, etc. It's possible to add your own custom effect.

To use the perk system, you will need to attach the script PlayerPerk to the player unit in the game. You can then 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 PlayerPerk component, allow you to set it status as enabled/disable/pre-purchased in the game. There are several way to use the perk system.

These are a few examples supported by the default UI:

- **Stand alone** – Player gain perk currency through various in game event like defeating boss or reach check point. The perk currency is then used to purchase various perk
- **Passive** – Ties to leveling system. Player gain perk automatically by reaching certain level. These depends on the setting in ProgressStatsEditor (refer to last section 'Leveling System').
- **Attribute** – Ties to leveling system, perks are treated like attributes and can be repeatably purchased. Each level grant perk point where player can choose to to which attribute to upgrade. An example would be the attribute system in diablo.
- **Skill/Tech-tree** - Ties to leveling system, player gain perk currency through leveling which can then be spent on the perks. An example would be the talent tree system in mmo like world of warcraft.

Obviously some of the example mentioned are not mutually exclusive. You can infact use all three 'passive', 'attribute' and 'skill-tree' setup together. However to reduce the complexity, the default UI only supported one of them at any given time. Due to the open nature and the synergy possibility between the leveling and perk system, the potential of the leveling and perk system can only be fully utilise if you customise your UI.

For more information about how to setup the UI for leveling and perk system. Please refer to the UI section 'Level and Perk Menu'

Saving and Loading Perk Progress

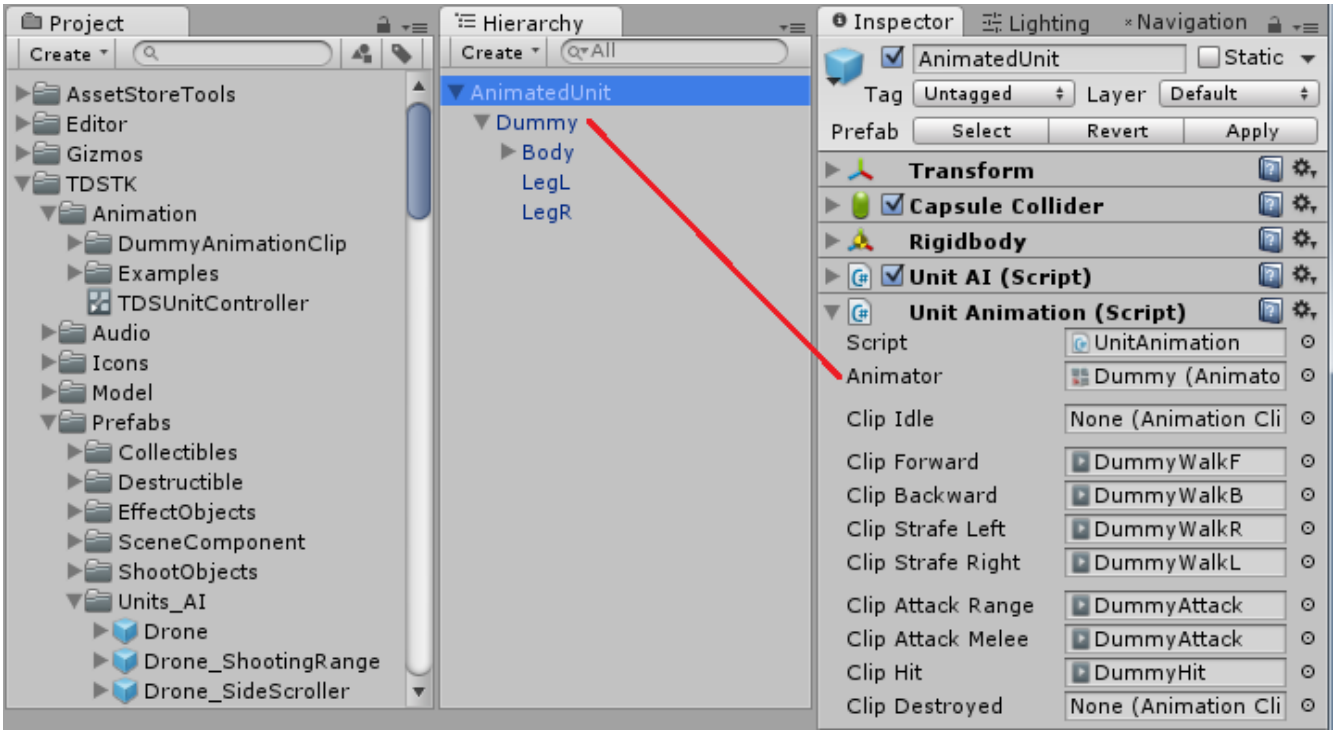
It's possible to save the progress of player level and perks and load them in future game session. The progress can then be load when a new level is loaded or when player start the another game session. To do that, you simply have to check the 'LoadProgress' and 'SaveProgress' option on the UnitPlayer.

When save is enabled, there's a few way where the progress would be saved. You can either check the 'SaveUponChanged' option on UnitPlayer, that would prompt a save whenever something changes in PlayerPerk or PlayerProgression. Alternatively you can use trigger so that the progress is only saved when player reach a certain check point in game.

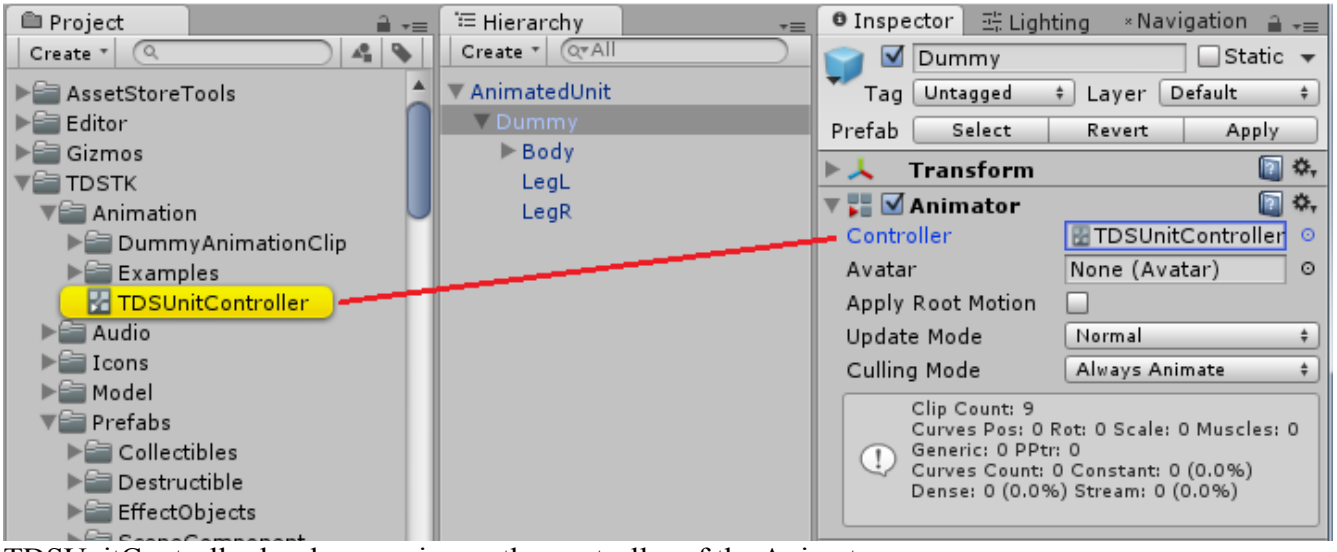
Animation

TDSTK has a system supports animation on units, both AI and player. To Add animation to a unit, you will need to:

- add UnitAnimation.cs to an unit prefab, along side UnitAI.cs or UnitPlayer.cs.
- assign the animator component on the unit model to the Animator Slot of the UnitComponent
- assign 'TDSTK/Animation/TDSUnitController' as the controller of the animator component.



Unit Animation component is added to the unit prefab. Animator component on the 'Dummy' game-object is being assigned to the Animator slot of UnitAnimation.



TDSUnitController has been assign as the controller of the Animator

Once that is done, you can assign the animation clip wanted to each corresponding animation slot in the UnitAnimation component. They are all optional so you can leave the slot where there's no animation clip to play empty.

You can refer to the two example prefab about how this is done. There's one for player unit and there's one for AI unit. They are both located in 'TDSTK/Animation/Exampes'.

PLAYER CONTROL & USER INTERFACE:

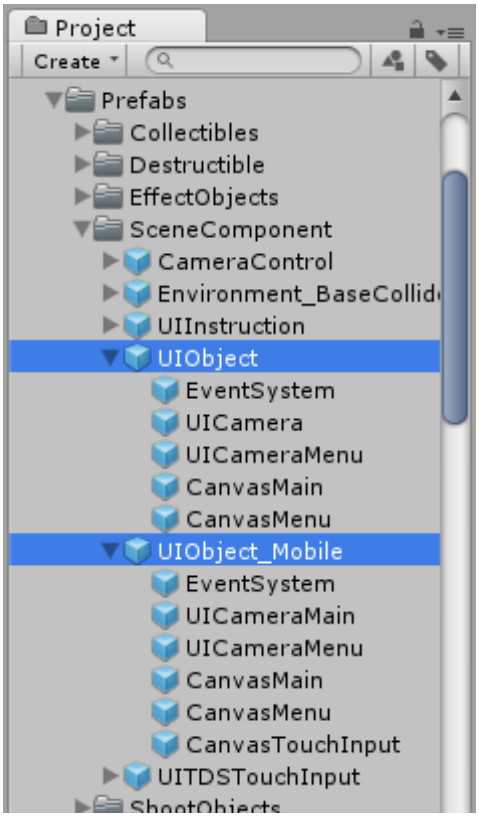
Player Control

There are several control option that might be enabled/disabled based on the setting you use.

First movement. There's two type of movement, rigid and free-form. Rigid movement is very much a binary movement where player either stay stationary or move at max-speed. Free-form is a more natural movement mode with acceleration and deceleration. With free-from movement mode, the player unit will have movement momentum and may carry on moving even when there's no move input. In this case, 'Space' key can be used to apply braking to player unit. In either mode, 'Left-Shift' can be used to boost player speed at the expense of energy.

Abilities and weapon alt-fire mode both shares the same input, right-mouse-button. However that is only true when only one of them is enabled in the game. When both abilities and weapon alt-fire mode is enabled, the activation of abilities is switch to middle-mouse button. You can enable/disable either abilities of alt-fire mode in GameControl.

User Interface



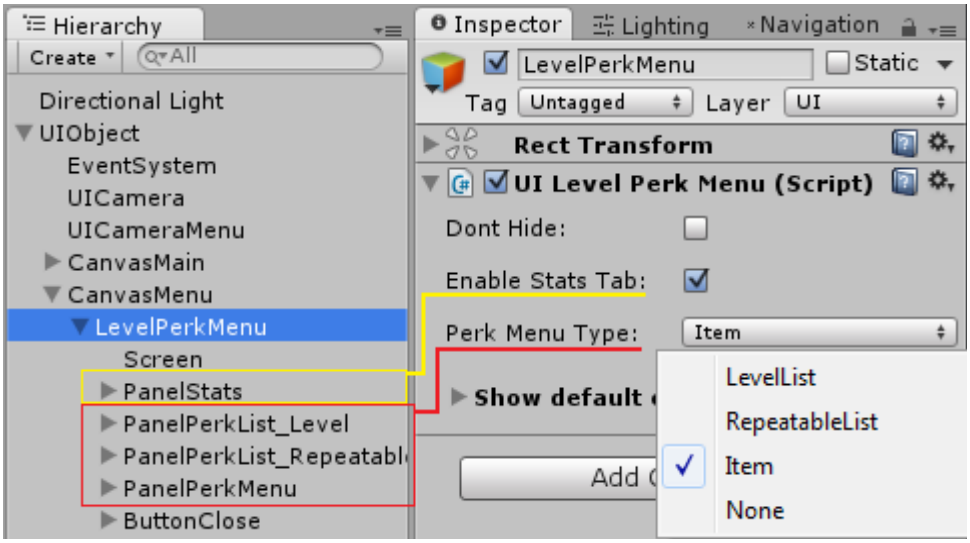
User Interface is very much an integral component of the package although it's not a core component. It's built to be modular and thus can be replaced with any custom solution.

The default interface is based on uGUI. There are two pre-built prefabs that act as a UI template which you can drag into any custom scene. One for mouse and keyboard input and one for touch input (mobile platform). They are both located in '*TDSTK/Prefabs/SceneComponent/*', named respectively as *UIObject* and *UIObject_Mobile*. If you create a new scene using the top panel. The new scene will take consideration of the type of scene you want to create a use the appropriate UI prefab automatically.

Certain parameters in the default UI prefabs that are made to be configurable (ie. Enable-overlay, list all ablities, etc) for ease of use but for the most part, you will need to thinker with it to get the setting/appearnce you want. You can also adjust the various element in it to change it appearance. You can also deactivated certain UI element if they are not needed in the scene like how it's done in the demo scene. However, it's recommend that you get yourself familiar with uGUI, understand how it works before you start tinkering it.

Level And Perk Menu

LevelPerkMenu is the special UI object that support customization, mainly to provide a mean to build custom tech-tree in conjunct with the perk system.



In a nutshell, LevelPerkMenu is the main controller for all the UI regarding level stats and perk. As shown in the image, the option 'EnableStatsTab' is related to PanelStats (which show player level information amd it's non-interactable) and 'PerkMenuType' is related to PerkMenu. You can only choose to show or not to show the level stats, but you can choose which type of perk menu to use.

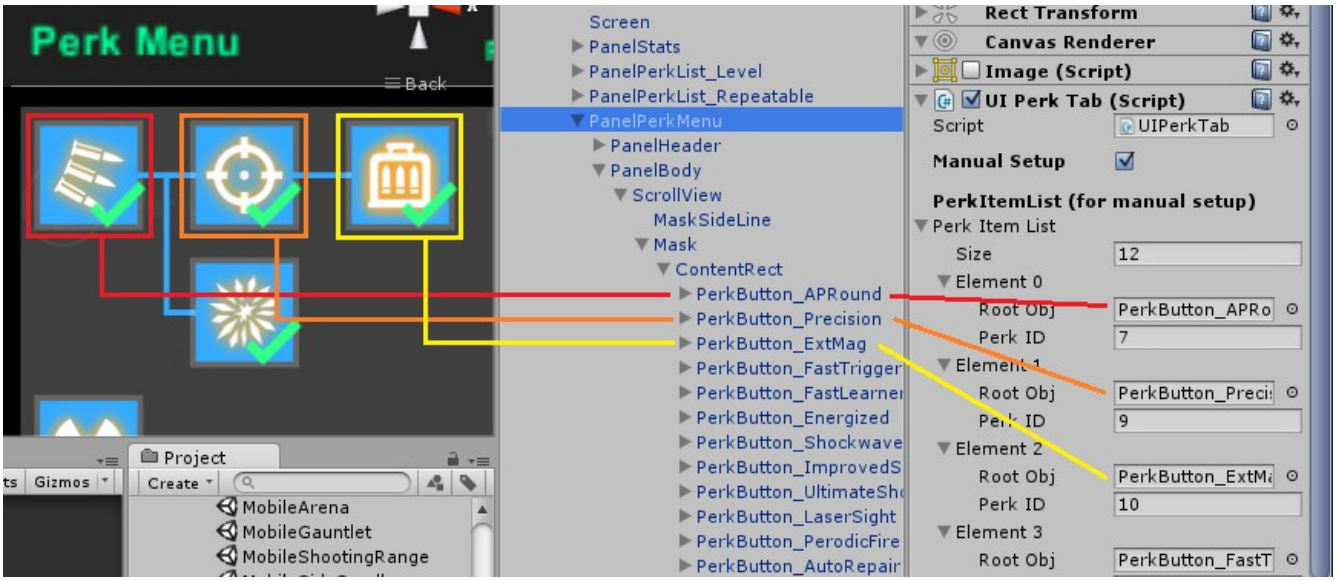
The option of perk menu type are as follow:

- **LevelList** – have the perk UI to show a list of perk to be unlocked upon reach certain level, as demo in demo scene 'DemoShootingRange'. Correspond to the child object PanelPerkList_Level.
- **RepeatableList** – have the UI show a list of perk that would be presented as attributes and can be repeatably purchased. as shown in demo scene 'DemoGaunlet'. Correspond to the child object PanelPerkList_Repeatable.
- **Item** – have the UI show a customizable skill-tree (or just a grid of items), as shown in demo scene 'DemoArena'. Correspond to the child object PanelPerkMenu.
- **None** – don't how any perk menu.

***PanelPerkList_Repeatable - y**ou can choose which perk to show up in this menu by selecting it just like you would with the available perk selection in PlayerPerk.

***PanelPerkmenu**

When the flag 'ManualSetup' is checked. You can arrange the perk item anyway you want, you just need to assign them as item to UIPerkMenu component as shown in the image below. As well as specify the ID of the perk the button is associated to. When the flag 'ManualSetup' is off, all the available perk enabled in PlayerPerk will be assigned a button automatically. You can future specify if the perk should show up in the UI



Touch Input And Mobile Support

The framework is mobile ready and comes with a preset touch input for that purpose. Just like the default UI and input for mouse and keyboard, the touch input is independent from the the core framework. You can always implement your own touch input should you wish.

By default the touch input is integrated with the UI prefab for mobile platform. It's for all intend and purpose the same as it's mouse and keyborad counterpart apart from the additional of UI element for touch input (on screen button and joystick).

Please note that the default visual effects used for the demo are not exactly optimized for mobile. So is the level design of the demo scene (some spawn too many units). Therefore you might experience some frame drop when using some of the demo scene as it's.

THANK-YOU NOTE & CONTACT INFO

Thanks for purchasing and using TDSTK. 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/TDSTK/Contact&SupportInfo***". Just in case, you can reach me at k.songtan@gmail.com or TDSTK support thread at Unity forum.

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