

TowerDefense-ToolKit

Documentation for Unity3D

Version: 4.0.5 f1
Author: K.Song Tan
LastUpdated: 28th September 2020

Forum: <http://goo.gl/Wz94gv>
WebSite: <http://songgamedev.com/tdtk>
AssetStore: <http://u3d.as/QV3>

Thanks for using TDTK. This toolkit is a collection of coding framework in C# for Unity3D, designed to cover most, if not all of the common tower defense (“TD”) game mechanics. Bear in mind TDTK 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 TDTK, it's strongly recommended that you look through [this video](#) 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 '***Tools/TDTK/Support And Contact Info***' via drop down menu from UnityEditor. Please leave a review on AssetStore if possible, your feedback and time are much appreciated.

Important Note:

TDTKv4.x is a new version redesigned and written from scratch therefore it's not backward compatible with earlier version of TDTK.

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

OVERVIEW

How Things Work

A working TDTK scene has several key components to run the basic game logic and user interaction. These are pre-placed in the scene. They have various settings that can be configured and the setting dictates how the scene is played. For instance, how many waves of creeps are there, how is the path layout, how many resource players have and so on.

Then there are the prefabs like towers and creeps which are spawned during game play. Each of these has their corresponding control component on them. The stats and behaviours are configured via those components. These prefabs are assigned to a series of databases. The key components will access the database to access these prefabs. So the game knows which tower can be built, which creep should be spawned and so on.

For most of the settings concerning a specific level (with the exception of spawn information), you can configure them on their relevant components in the scene using Inspector. The rest you can configure using a custom editor window which could be accessed via the drop-down menu on the top of the Unity Editor, *'Tools/TDTK'*.

Basic Components And Class

These are the bare minimal key components to have in a basic functioning TDTK scene. You will need at least one of these in a working scene.

GameControl* Control the game state and major game logic.

RscManager* Contains all the code and logic for resources.

SpawnManager* Control the spawn logic and contain all the spawn information for a scene.

TowerManager* Control the build logic and contain the information of buildable towers.

SelectControl* Control the logic for selecting a tower or a valid build point.

Path Used to specify the waypoints that form the path and store the information of the constructed path during runtime.

BuildPlatform Attached on a game-object with collider to specify where and what tower can be built. A build-platform also forms a walkable grid which the creep has to navigate through via path-finding when it's assigned as a waypoint on a path.

* There can be only one instance of these components in the scene.

Prefab Components And Class

These are the component attached on prefab to be spawned on runtime.

UnitTower The component on each tower object that control the behaviour of the tower.

UnitCreep The component on each creep object that control the behaviour of the tower.

ShootObject The component on objects fired by tower or creep during an attack to hit the target.

Optional/Support Components And Class

These are the optional component for a TDTK scene. With the exception of PathIndicator, there can be only one instance of these components in the scene.

AbilityManager Control the logic for abilities system and contain the information of usable abilities. It's required only if the ability system is used in the level.

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.

AStar* Contain all the logic for path-finding.

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

CameraControl The control component for camera.

AudioManager The component that contain and play all the sound effect.

PathIndicator Used to visualize the creep's active route from a starting path.

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

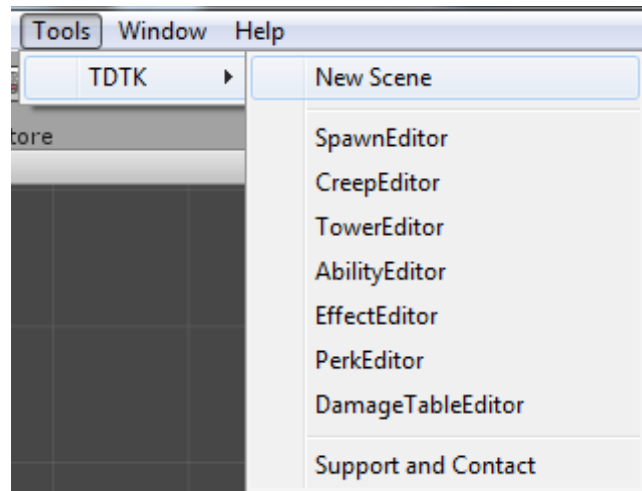
UI

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:

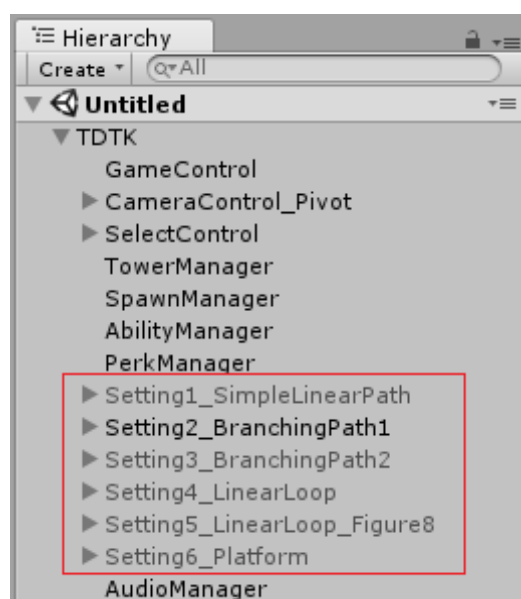
Create A TDTK Scene And Modify It

To create a new scene, simple use the drop down menu '*Tools/TDTK/New Scene*' and s default template scene will be created automatically. The scene should be game ready.



From the default setting, you can further customize things to your preferred design. You can configure the game setting and rules by configuring the active component in the scene via Inspector. There are also various editor windows that would help you to configure the more complicated setting such as tower's stats and spawn info. All the editor windows can be access via the drop down menu.

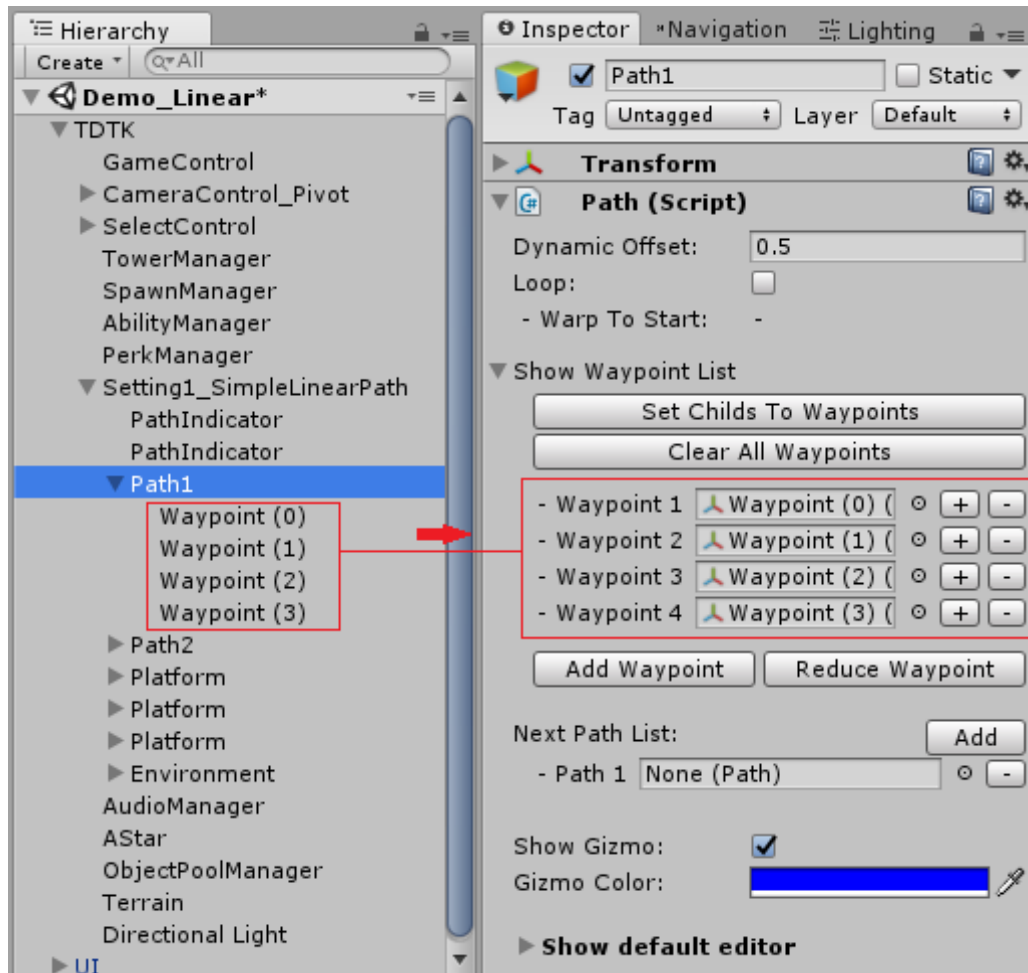
Note that the template scene comes with a few preset setting (as highlighted in the image below). You can switch between them by deactivate the active game-object setting and activate the setting you want. Each of them provide example for a specific setting (linear path, branching path, looping path, custom platform, etc.). Some of them has been used in the demo scene.



Working With Path

Basic

You can modify existing path by just changing the position of individual waypoints in the path. To extend the path, just create a new empty transform to the scene and add it to the path component as the extended waypoint. The easier way to add a new path would be duplicate an existing one (select it in HierarchyTab and press Ctrl-d) and modify it.



Path with Walkable BuildPlatform

You can create a path that move through a walkable build-platform by simply assign a BuildPlatform as one of the waypoint. The build-platform will automatically forms a grid in which the creep needs to navigate through using the path-finding. Note that build-platform cannot be assigned as the first waypoint of the path.

Branching Path

You can assign next path for a path, which acts as the continuation of the path when it ends. When multiple paths are assign as the next path, a branching path is created. When a creep reach a branching point, it will automatically select the shortest path. Branching path is only useful when the branched out path contain walkable build-platform where the length of the path can be altered during runtime. Otherwise the creep will consistently follow the shorter path, making the branching meaningless. An example of this setup can be found in '*TDTK/Scene/Demo_BranchingPath*'.

That said, you could probably add some custom script to manipulate the waypoint position during runtime, altering the path length (just some idea).

Note On Path-finding and Branching Path

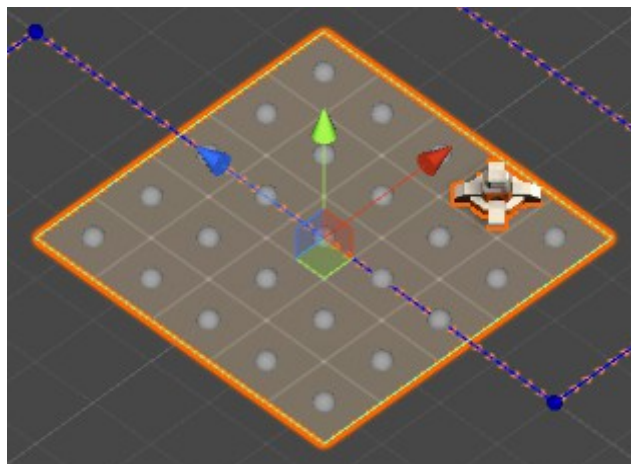
There are two things you need to know when configuring path involve path-finding or branching.

- The creep will always try to get to the nearest available destination
- The game will always make sure there's at least one available path

That means if there's only one path in the scene, the game will not allow the player to block the path entirely. In branching path setting though (there's more than one path), the player can block the path entirely provided there's still another alternative path to get to another destination.

Add And Modify New BuildPlatform (where Tower Is Built On)

You can modify existing platform by simply changing it's position or scale (note – platform cannot be a slope). The scale of the platform will determine how many towers can be built on it (note – scale will be adjusted to fit the grid-size, as specified in TowerManager). To add new platform, the easiest way would be duplicate existing one (select it in HierarchyTab and press Ctrl-d).



A simple walkable build-platform in runtime in SceneView. Each tile can accommodate one tower and each white sphere can be act as a waypoint for creep.

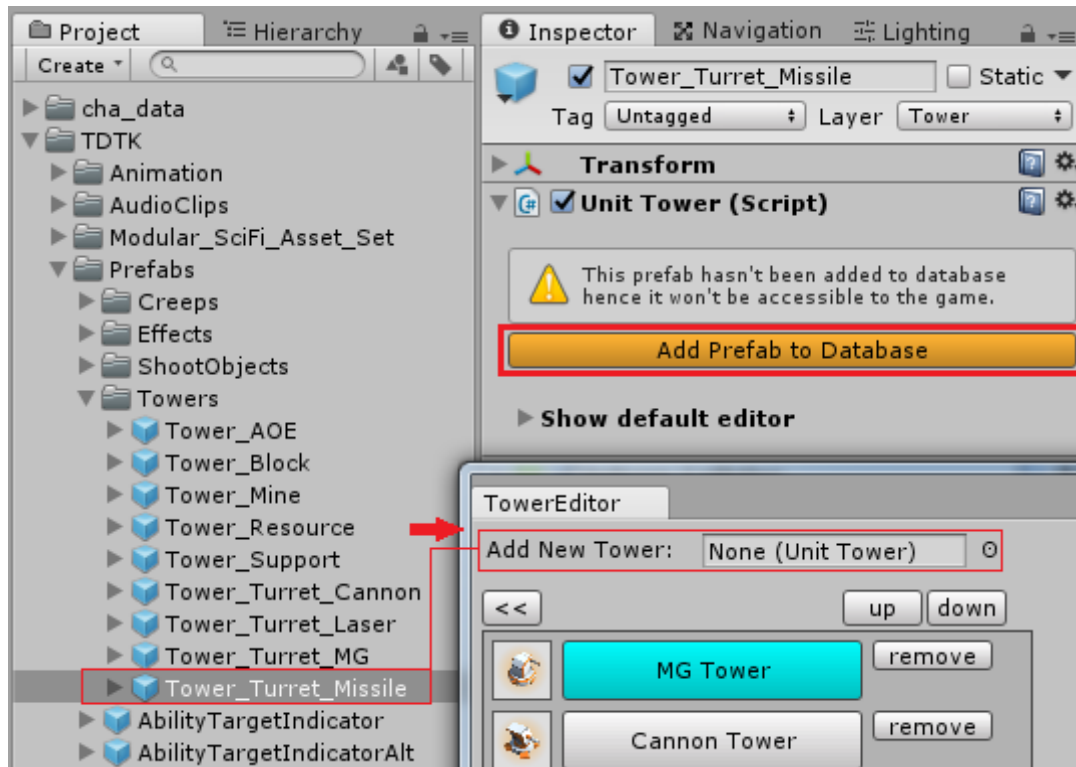
The default build-platform are created using quad with a box-collider. It's recommended that you keep the same template as the code will always assume a box collider and scale. You can disable the mesh renderer and replace it with your own visual element.

To visualize the grid, a special material is used on the platform renderer. The tiling of the material will be adjusted in runtime so that the grid shows up correctly. This is all done automatically if you are using the default platform. Of course you can disable this (by unchecking the 'Auto Adjust Texture' option in TowerManager) and add your own visualization.

It's possible to create build-platform with custom shape and layout. You can place various collider with layer 26 - ('Obstacle') to block out specific tiles on the grid. Blocked tile cannot have tower built on them or creep traverse through them. You can also create 'no build zone' by placing collider with layer 25 - ('No Build Zone') on specific tiles. The player will not be able to build tower on 'no build zone'. However the creep will still be able to traverse through it. For build-platform that is not a square, you can simply disable the mesh renderer on the platform itself and add in your own custom mesh. There an example for all this in the example scene '/TDTK/Scene/Demo_Platform'.

Add New Tower To The Game

To add a new tower to the game, first you will need to create the tower prefab (refer to section [Tower Prefab](#) for how to create a prefab). Once you have a prefab, you can add it to TowerEditor. You can either manually drag the prefab to TowerEditor window (as shown below) or use the 'Add Prefab to DataBase' button on the UnitTower prefab. Once a tower prefab is in the Editor, the tower should appear in TowerManager and available to all scenes. You can disable a tower in any particular scene by disabled it in TowerManager.



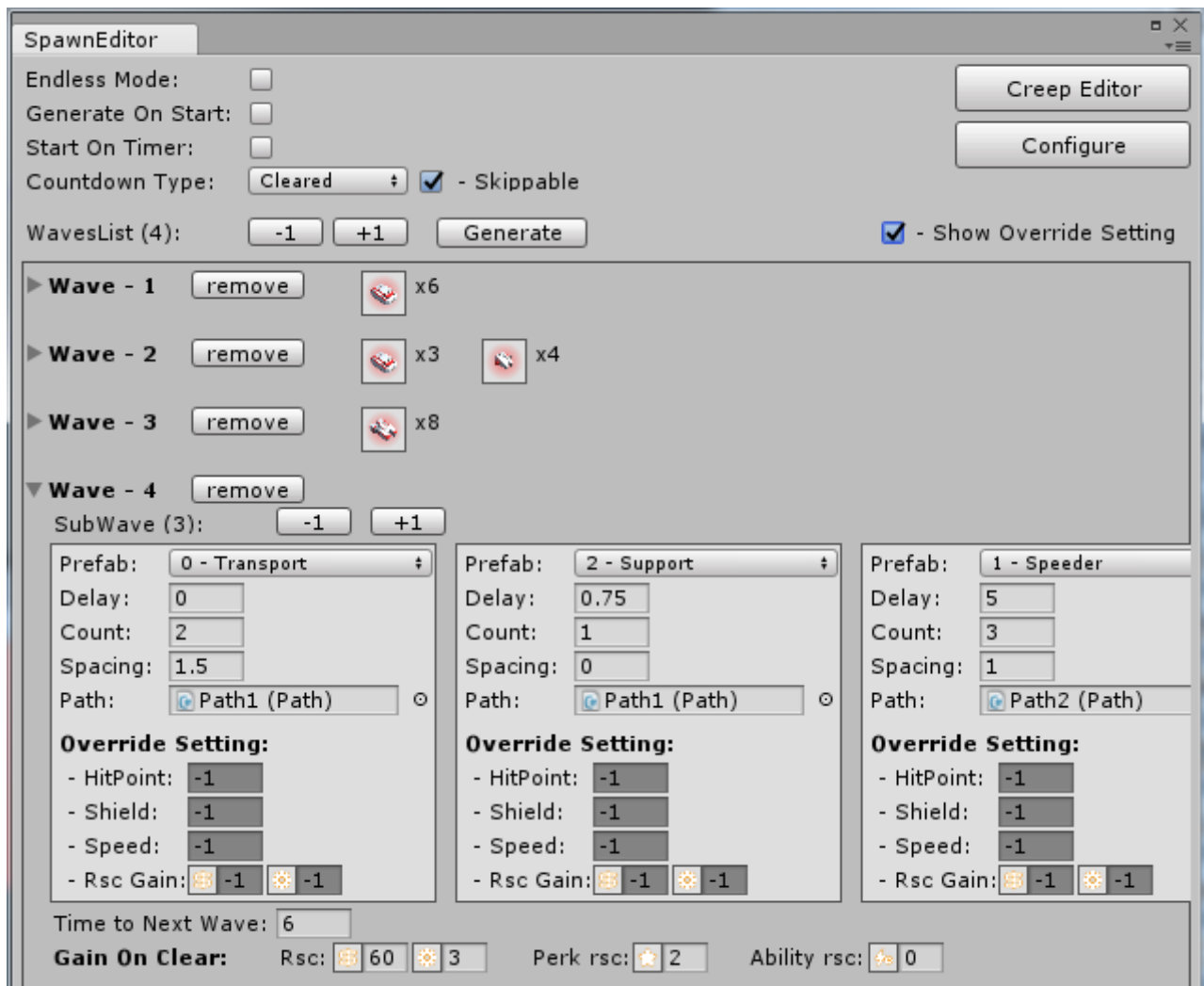
* You may want to take a look at section [ItemDB](#) to understand why this is required.

Add New Creep To The Game

To add a new tower to the game, first you will need to create the new creep prefab (refer to section [Creep Prefab](#) for how to create a prefab). Once you have a prefab, you can add it to CreepEditor. The process is similar to how to add new tower to the game. Once a creep prefab is in the Editor, the option to spawn that particular unit should appear on SpawnEditor.

Editing Spawn Information

To change the spawn information, you will have to open SpawnEditor. You can use the drop down menu 'Tools/TDTK/SpawnEditor' or use the 'Open SpawnEditor' in SpawnManager gameObject. Once the editor is opened, it's pretty intuitive from there on. As shown in the image below, you can specify the number of wave to spawn as well as the detail information for each wave.

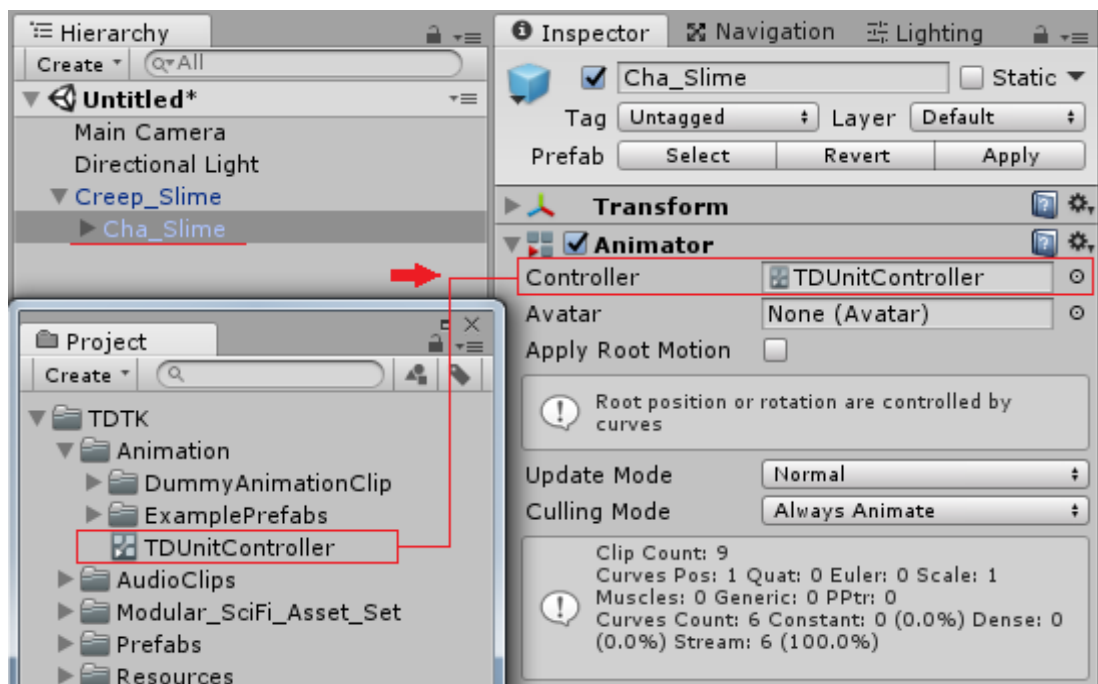


There are also options to use procedural generation for the spawn info (by either enabling 'Endless Mode' or enabling 'Generate On Start'). When that is enabled, you configure the input parameters of the algorithm rather than specific waves. You can find out more about this in section 'Endless Mode and Procedural Generation Spawn Info'.

Add New Animation To The Tower And Creep

TDTK uses mecanim animation system. The animation clip to be used in various event can be assigned in the editor (TowerEditor for tower and CreepEditor for creep). 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 *TDUnitAnimator* as the controller for the *Animator* component as shown in the image below. The images shows the controller for creep animator but the process is similar to tower. Once done, you should be good to assign the animation clip you want to use in the editors.

Note that in the editor window, there's a slot for '*Animator Object*'. That the object within the prefab hierarchy that holds the Animator component. 'Cha_Slime' in the case shown in the image below.



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

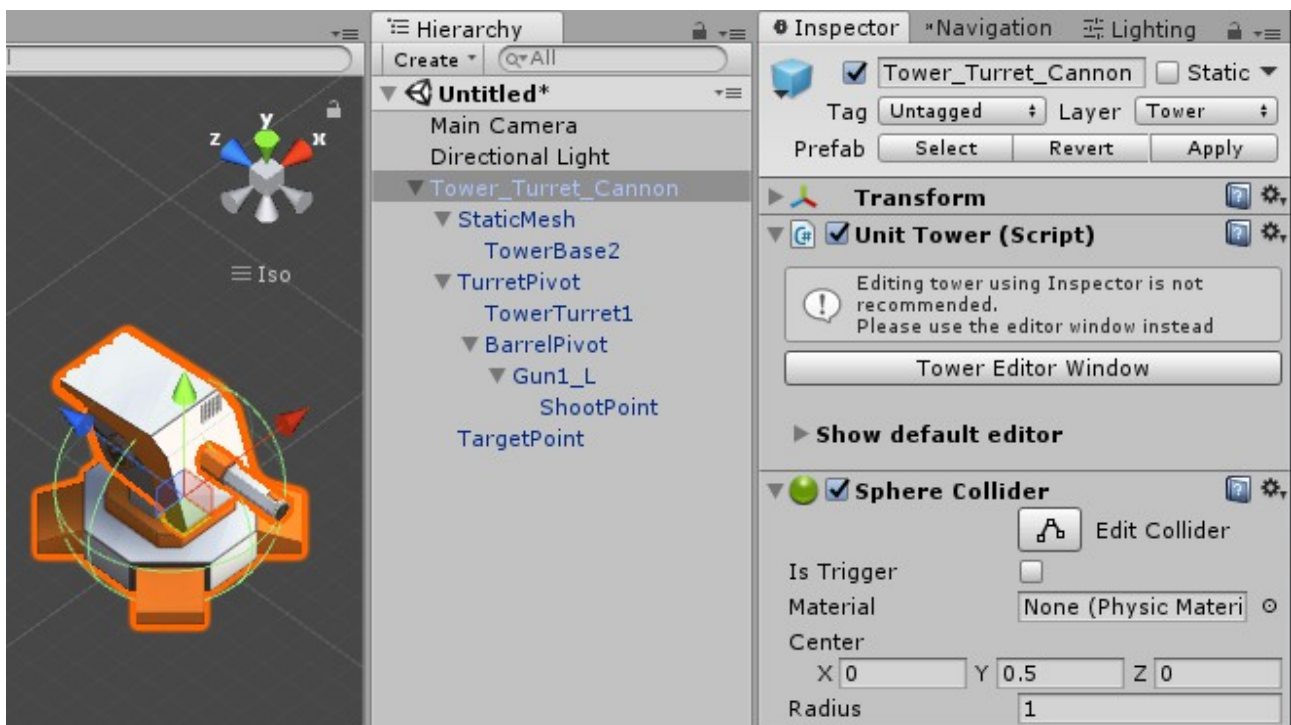
ABOUT PREFABS:

Tower Prefab

A working tower prefab can be as simple as a empty game-object with the script UnitTower.cs attached on it. This would enable the tower to be built and function as usual despite it's not visible. Any mesh/model on the tower is optional.

To make the tower targetable in ability targeting phase, it would require a collider. To make sure the collider is not in conflict with other component, the layer of the tower game-object will need to be set to 'Tower' (layer 28 by default). The size of the collider should be set to just encapsulates the mesh/model of the tower.

Finally, it's recommended that you follow the hierarchy structure used by the default tower prefab. Especially when it comes to 'Turret' type tower that involve aiming. All the static mesh are placed in the 'StaticMesh' transform and all the aiming parts in placed under 'TurretPivot', where it will be assigned as the pivot for aiming. For more about getting a tower to aim correctly, please refer to section 'Unit (Turret) that Aims and Fires at Target'.



Creep Prefab

A working creep prefab can be as simple as a empty game-object with the script UnitCreep.cs attached on it. This would enable the creep to be spawned and move along the path as usual despite it's not visible. Any mesh/model on the tower is optional.

To make the creep targetable in ability targeting phase, it would require a collider. To make sure the collider is not in conflict with other component, the layer of the tower game-object will need to be set to 'Creep' (layer 27 by default). The size of the collider should be set to just encapsulates the mesh/model of the creep.

Finally, it's recommended that you follow the hierarchy structure used by the default example creep prefab.

**A creep prefab draw many parallel to a tower prefab in term of hierarchy arrangement. You can always refer to the tower section for image reference.*

ShootObject Prefab

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.

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.

HOW THINGS WORK:

Item DB (DataBase)

TDTK uses a centralized database to store all the information of the prefabs (towers, creep and shoot-object) and in game item (resource-type, 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 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. Prefabs that is not added to the database will not be appear in the game.

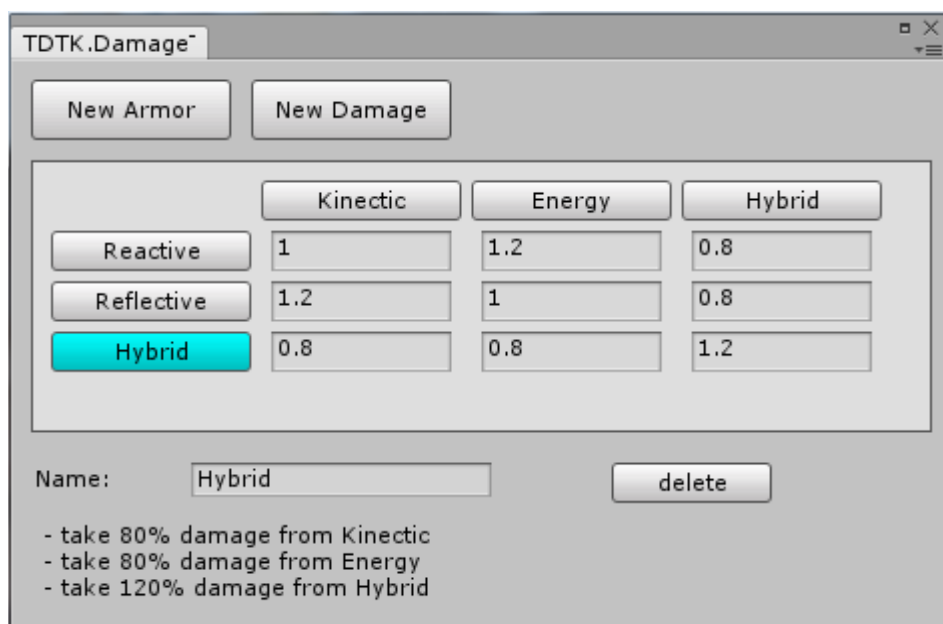
Tower prefab can be individually enabled/disabled in TowerManager Inspector. Disabled tower won't be available in the scene. However a disabled tower can be added to the scene during runtime via PerkSystem. Tower prefab can also be individually disabled in Platform. When a tower is disabled on a platform, players won't be able to built that tower prefab on that platform.

Just like tower, ability can be individually enabled/disabled in AbilityManager Inspector. Disabled ability won't be available in the scene. However a disabled ability can be added to the scene during runtime via PerkSystem.

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

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 can be assigned a specific armor type and each attack/effect can be assigned a specific damage type.



The screenshot shows the TDTK.DamageTableEditor window. It has a title bar with 'TDTK.Damage' and standard window controls. Below the title bar are two buttons: 'New Armor' and 'New Damage'. The main area contains a table with damage types as columns and armor types as rows. The 'Hybrid' row is highlighted in blue. Below the table, there is a 'Name:' field with 'Hybrid' entered, a 'delete' button, and a list of damage types: Kinectic, Energy, and Hybrid. The table data is as follows:

	Kinectic	Energy	Hybrid
Reactive	1	1.2	0.8
Reflective	1.2	1	0.8
Hybrid	0.8	0.8	1.2

Below the table, the 'Name:' field is set to 'Hybrid'. To the right of the name field is a 'delete' button. Below the name field, there is a list of damage types: Kinectic, Energy, and Hybrid. The list shows the following damage types and their corresponding damage values:

- take 80% damage from Kinectic
- take 80% damage from Energy
- take 120% damage from Hybrid

Upgrading System For Tower

There are two ways to upgrade a tower. The first being stats upgrade where the game-object remain but the stats is changed. The second being a complete upgrade where the existing tower is removed and replaced by a new tower.

In TowerEditor, you will find that you can add multiple level for each tower prefab (as shown in image below, section-B). The first entry is always the base stats used when the tower is built. Any subsequent level are the for upgrades. A tower undergo stats upgrade means they will use stats value from next subsequent level.

You can also assign different tower prefabs as the 'next upgrade tower' for a tower in TowerEditor (as shown in image below, section-A. This allow the tower to be upgraded to the assign 'next upgrade tower'. In runtime, the existing tower being upgraded will destroy itself to give way to the tower specified in its 'next upgrade tower'. This is the option to go for if you want to change the tower appearance on upgrade.

▼ Tower Stats And Upgrade

Next Upgrade: - 1 - Cannon Tower -

- 2 - Laser Tower -

Add New: -

Add Level

Level 1 remove

Cost (Rsc): 20 0

Sell Value (Rsc): 20 0

Build Duration: 1.5

Sell Duration: 0.5

HitPoint: 10

Shield: 0

- Regen Rate: -

- Stagger Duration: -

Level 2 remove

Cost (Rsc): 10 0

Sell Value (Rsc): 30 0

Build Duration: 1.5

Sell Duration: 0.5

HitPoint: 10

Shield: 0

- Regen Rate: -

- Stagger Duration: -

The tower upgrade methods are not mutually exclusive. A tower can have both stats upgrade and next tower upgrade. However, a stats upgrade will always come first. That means a tower cannot be upgraded to next tower unless it already uses up its available stats upgrade.

You might also noted that you can assign more than one 'next upgrade tower'. Doing so will allow the tower to branch out to different upgrade. The default TDTK UI will automatically present user with multiple upgrade buttons (one for each prefab in 'next upgrade tower').

Example:

TowerA has 3 level of stats specified and has TowerB assigned as the 'next upgrade tower'. The upgrade path for TowerA will be towerA (stats-level-2), then towerA (stats-level-3) then TowerB.

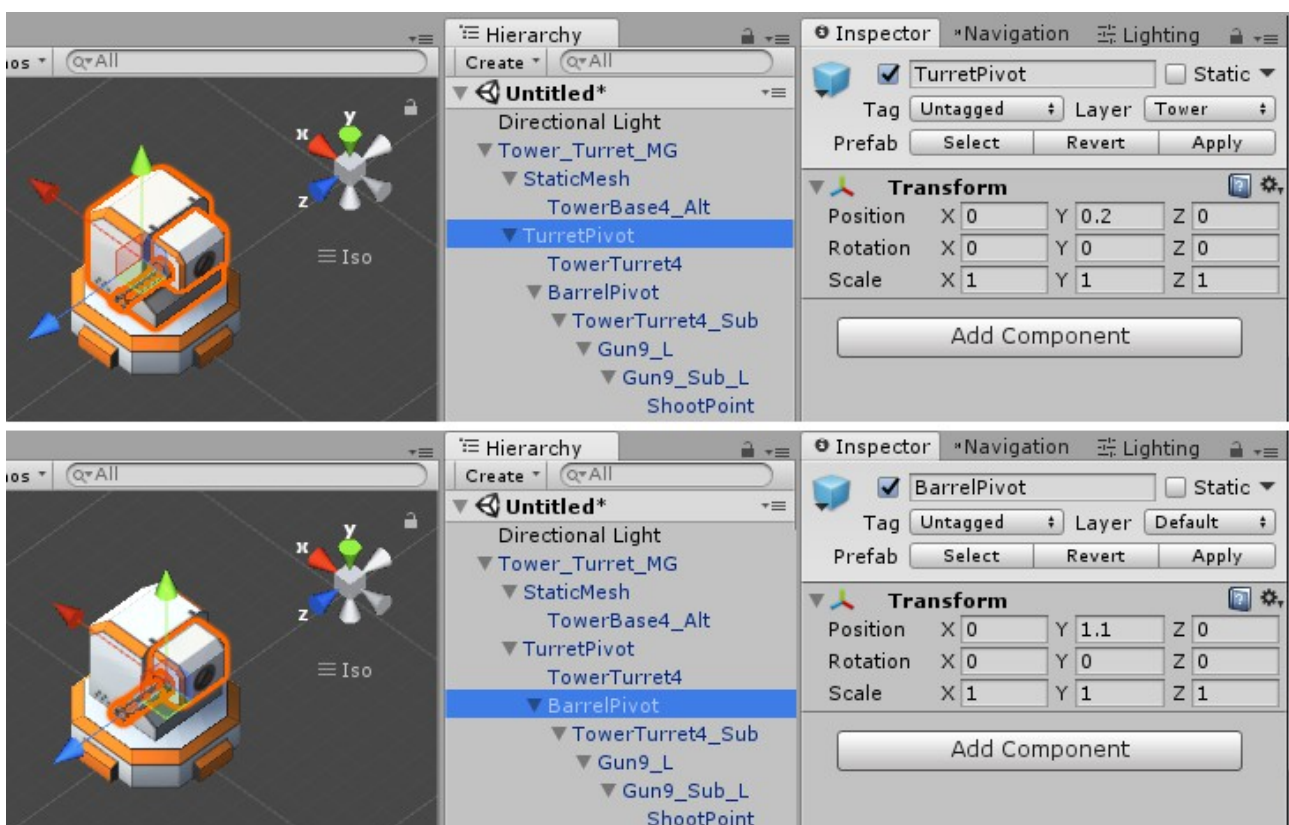
TowerC has 1 level of stats specified. TowerD and TowerE is assigned as the 'next upgrade tower'. TowerC can be either upgraded to TowerD or TowerE.

Units (Turret) That 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.

Ability And Perk System

Ability is are actions that can be performed by player during runtime to achieve various means like damage creeps or buff towers.

You can create individual ability item in AbilityEditor (accessed via the top down menu). Once created, the ability will appear on AbilityManager, ready to be used in game.

Ability are entirely optional, you can disabled all ability in a level by simple remove the AbilityManager game-object.

Perk System

Perk system are for all intent and purpuse, a customizable upgrade system. You can use it to create various upgrade system for the game. It can be as simple as a linear upgrade or a full on tech tree.

You can create individual perk item in PerkEditor (accessed via the top down menu). Each item can be set to do different things upon purchased on player. For instance, unlock a new tower/ability, modify the attribute of an existing tower/ability, change resource gain rate to name a few. Each item also has individual unlock criteria like minimum wave, prerequisite perk.

Perks are entirely optional, you can disabled it in a level by simple remove the PerkManager game-object.

Effect

Effects are buff/debuff that applies to individual unit. Effects can be applied through a normal unit attack or an ability.

To add effect of an attack or ability, you will first need to create the effect using EffectEditor (accessed via the top down menu). You can configure the effect in the editor. Once an effect is created, you can set it to an attack or ability using the appropriate editor.

Note that effect can be set as modifier or multiplier. Modifier effect add their value to the target. Multiplier effect multiply the existing value on the target with their own value.

Using Custom Model And Add Background Environment

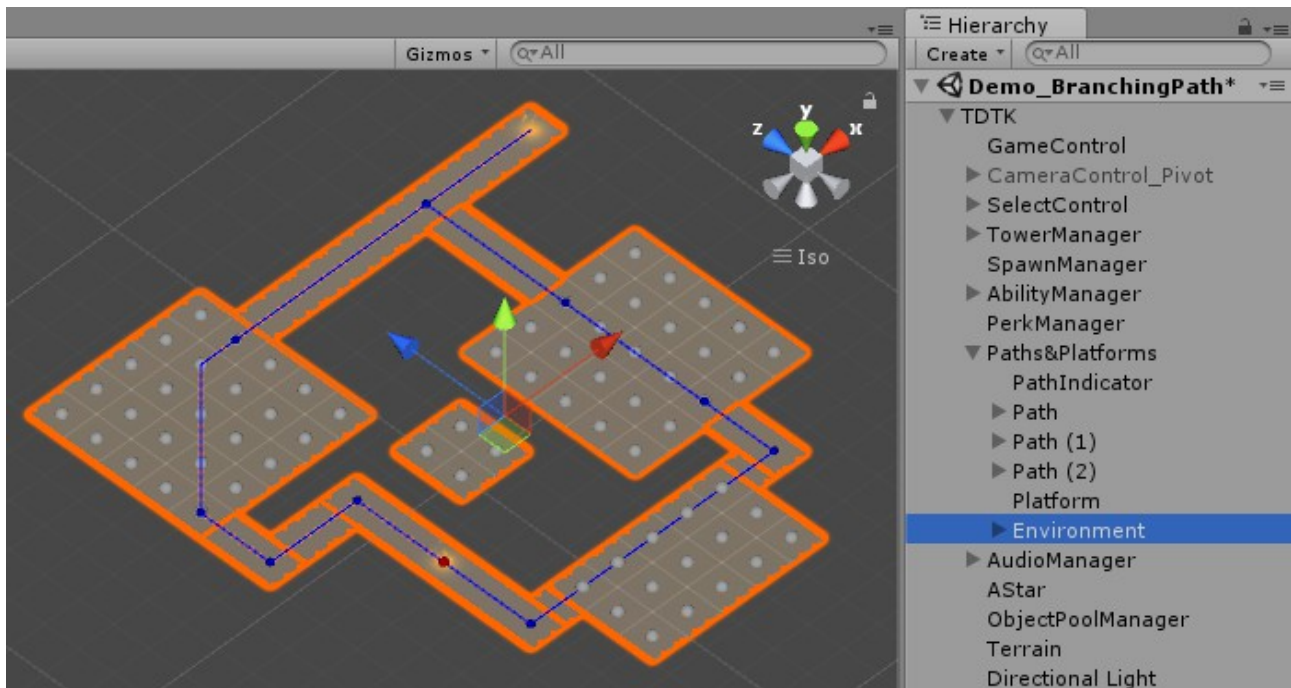
Custom Model

At it's minimal form, TDTK doesn't require any visual element at all. All the components can run just fine without any visual element. They are only necessary otherwise the player wouldn't be able to see anything. All the model used in the default prefab is meant for that purpose. Therefore you can remove the default model and replace it with whatever you need.

Background Environment

You can put a TDTK scene on top of just about any background environment. The only requirement is that they are free of any collider component. If you must have collider component in the background game-object for what ever reason, you can set those game-object to layer-31 (Terrain). These would have the code recognise the collider as a background and not obstacle for walkable platform.

You can refer to the any of the demo scene to see how this in practice. As shown in the image below, all the highlighted game-object (the game-object 'Environment' and it's child object) are there for visual purpose. You can deactivate the game-object if you want, remove all visual-element on the level except the grid.



Endless Mode & Procedural Generated Spawn Info

TDTK support procedural generation for the spawn information. It's a mandatory for endless mode since there's no way to manually setup an infinite amount of spawn info. The procedural spawn generation can also be used for normal non-endless mode level.

You can configure the outcome of the procedural spawn generation in SpawnEditor. The algorithm is based on a set of attribute like subwave count in each wave, total unit count in each wave, how likely a particular prefab is feature in a wave, etc.. Each attributes is defined by a linear equation based on a set of value. These values are what you can configure to adjust the output of the generation. The calculation for each attributes goes like this:

$$\text{base value on wave-}N = (N \times \text{IncRate} + \text{StartValue}) * (1 + (\text{Random}(-\text{Deviation}, \text{Deviation})))$$

The final value is then limit to within the value of **Limit(Min)** and **Limit(Max)**

Wave Interval (Min/Max):		5	10
Use all path:		<input type="checkbox"/>	
One SubWave per path:		<input type="checkbox"/>	
Mirror SubWave:		<input type="checkbox"/>	

Wave Setting:		Gain On Wave Cleared:	
SubWave Count:	Total Unit Count:	Life Gain:	Cash
- Start Value: 2	- Start Value: 5	- Start Value: 0	- Start Value: 10
- Inc Rate: 0.5	- Inc Rate: 1	- Inc Rate: 0	- Inc Rate: 2
- Deviation: 0.5	- Deviation: 0.25	- Deviation: 0	- Deviation: 0
- Limit (Min): 1	- Limit (Min): 1	- Limit (Min): -1	- Limit (Min): -1
- Limit (Max): 4	- Limit (Max): -1	- Limit (Max): -1	- Limit (Max): -1

<input checked="" type="checkbox"/>	Transport	Wave (Min/Max):	0	-1	
Odds:	Interval	HitPoint	<input checked="" type="checkbox"/>	Shield:	<input checked="" type="checkbox"/>
- Start Value: 1	- Start Value: 1	- Start Value: 1	- Start Value: 1	- Start Value: 1	
- Inc Rate: 1	- Inc Rate: 1	- Inc Rate: 1	- Inc Rate: 1	- Inc Rate: 1	
- Deviation: 0	- Deviation: 0	- Deviation: 0	- Deviation: 0	- Deviation: 0	
- Limit (Min): -1	- Limit (Min): -1	- Limit (Min): -1	- Limit (Min): -1	- Limit (Min): -1	
- Limit (Max): -1	- Limit (Max): -1	- Limit (Max): -1	- Limit (Max): -1	- Limit (Max): -1	

Persistent Progress

It's possible to carry forth the player resource and perk progress to the next level.

For player resource, you will need to check '*Carry Over*' option on RscManager in the level. When the option is checked, RscManager will attempt to check if there's any prior level that have player resource saved and used that saved value as the starting value for current level. If there's not saved value, RscManager will use the default specified value instead. Upon finishing the level, the player resource value will once again be saved for any subsequent level that have the option saved. Please note that the resource value will only be saved if the level is beaten. Failing the level or restarting will not count.

Similarly for perk progress, you can check the '*Carry Over*' option on PerkManager. However there's the difference ends. For perks, any progress made are saved, no matter the level is finished or not. And the first PerkManager instance that has the option checked will be made persistent throughout the game, overriding PerkManager instance in any subsequent level.

It's also possible to run a PerkManager with perk menu only scene for persistent perk progress while disable the access to perk UI in normal game play level, creating a 'off gameplay level' upgrade system. You can refer to the sample scene '*TDTK/Scene/Example_PerkMenu*' to see how a PerkManager only scene with just perk screen.

Important:

Persistent progress in this context doesn't mean save game. The progress only last for a single game session. All progress is cleared as soon as you stop in play (in Unity Editor) or exit the game (in actual build). You will have to add your own code to actual save the game. There are some reference code that might be useful in '*TDTK/Scripts/Support_NotInUsed/TDSave*'.

Other Things You Should Know

Almost every item in the TDTK custom editor has tooltip. If you are unsure about something, just hover your cursor over the label.

You can have pre-built tower in a scene by placing tower prefab in the scene. The tower will automatically be adjusted to the nearest tile on the nearest build-platform when possible.

You can check the '*Hide In Inspector*' option for a tower/ability. This will disable the tower/ability in question from being available in game by default. This is useful for towers and abilities that can only be unlocked using perks

You can script your custom ability effect and attached it to the '*SpawnOnActivate*' object to create your own custom ability.

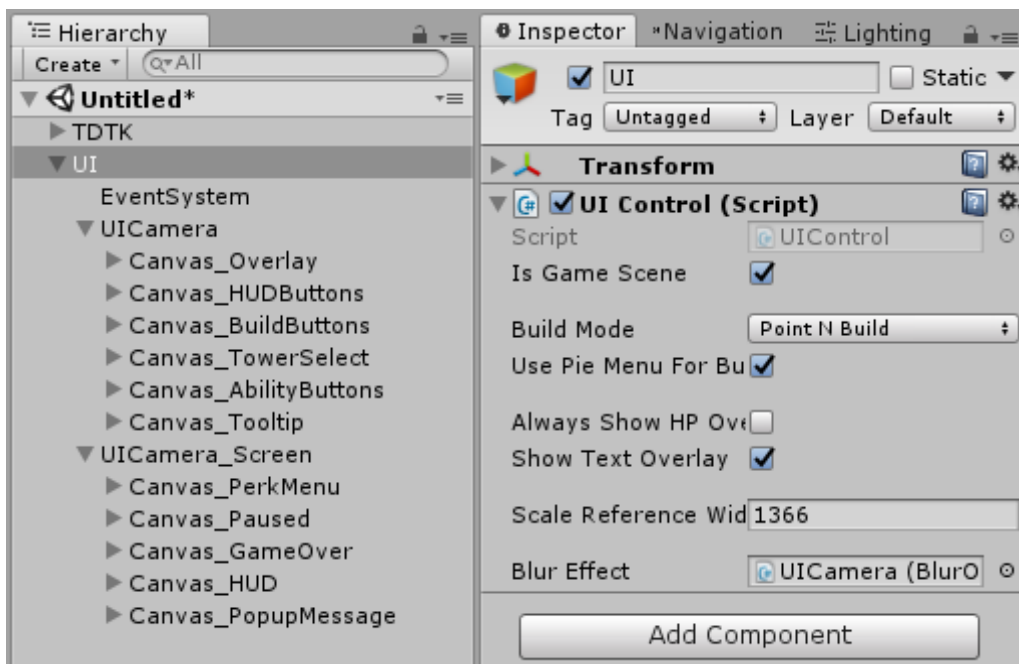
The new scene from the drop down menu are created using prefabs located in '*TDTK/Resources/NewScenePrefab*'. You can edit those to change the default new scene setting.

You can check the 'Flying Bypass' option in AStar to have flying unit ignore obstacle/tower on a walkable platform.

ABOUT USER INTERFACE:

Configure The Default UI

The default UI is added to the scene everytime you create a new scene via the drop down menu. You will find all the UI related stuff placed under the heirarchy of the game-object 'UI'. Although there are quite a lot of script governing the UI logic. Most of the important setting you need can be accessed on UIControl, the main control component attached on the game-object 'UI'. The only exception being PerkMenu, which is talked about in later section on this documentation.



Of course you can further customize the UI should you wish, however it's strongly recommended that you familiar yourself with unity UI system and how they work before you proceed.

Replacing The Default UI

The default UI is made with modular design and functionality in mind. it covers almost everything you need in a basic gameplay and provide a way to interact with every available mechanic in the framework. It also mean that it can be replaced with your own solution if you want. You can delete the game-object 'UI' in any TDTK scene to get rid of the default UI.

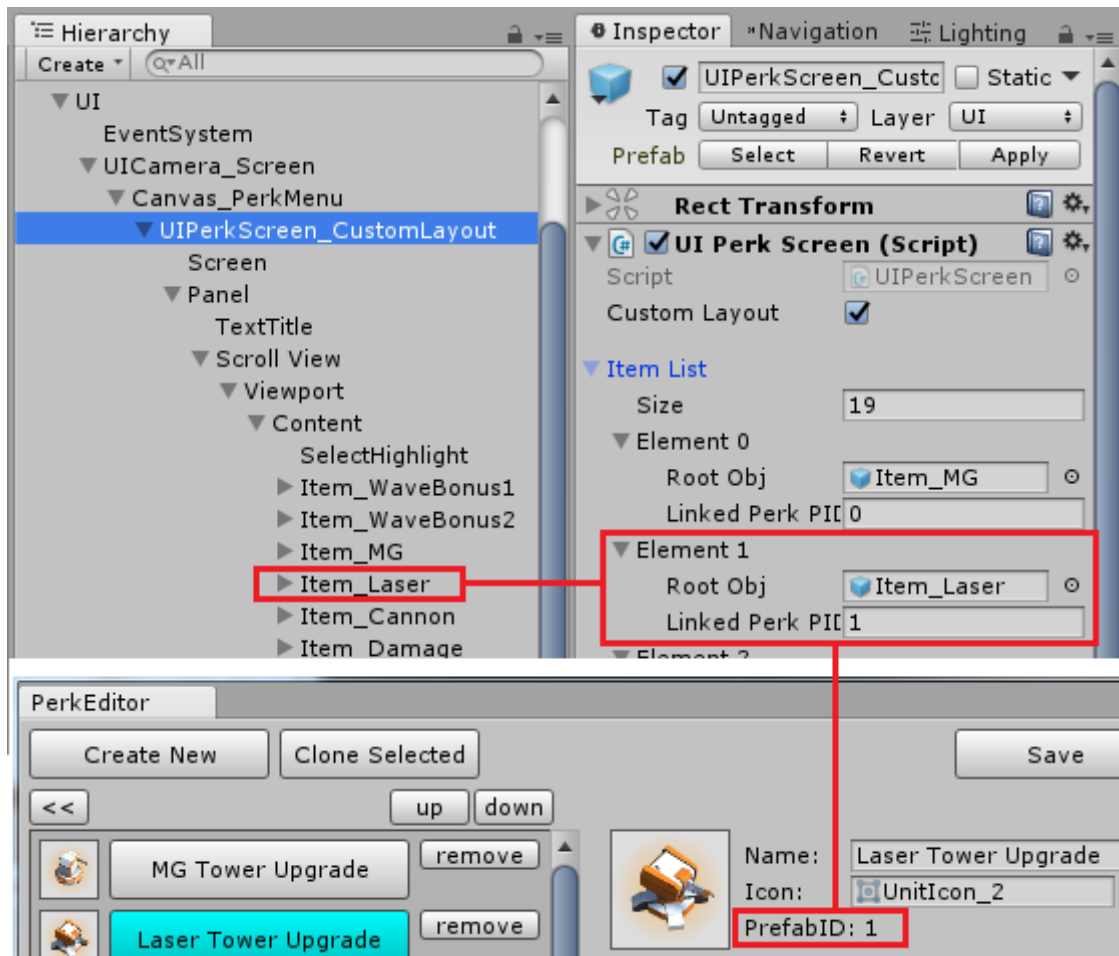
You can refer to the code for example of interacting with the core component when coding your own UI solution. You will find all the relevant scripts in '*TDTK/Scripts/UI*'. They are appropriately named so you can tell what is each scripts responsible for.

Please note that UI is also used to provide an overlay for the unit HP and tower construction status. Removing default UI will get rid of those as well.

Perk Menu

By default, perk menu will simply list all the available perk from PerkManager on the menu. However, you can override that and arrange your own custom layout just like the one shown in the demo scene.

To do that, first check the '*Custom Layout*' option on UIPerkscreen. Then you can add your own button in the ScrollView and assign them to the '*Item List*' in UIPerkscreen. Note that each item will have a '*Linked Perk PID*', the prefabID of the perk the item correspond to. You will need to manually match that to the PrefabID of the perk in PerkEditor, as shown as the image below.



Note that you can have connector that indicate the upgrade path of a perk if there's any. The connector object must be name 'Connector' and 'ConnectorBase'. The 'Connector' game-object will be set to inactive if the perk has not yet been purchased and active if the perk has been purchased.

You can refer to the perk menu in any of the demo scene to find out how all this setup comes together.

Perk Menu Only Scene

As mentioned before, it's possible have a perk menu only scene for 'off-gameplay-level' upgrade system. To do that you will need to uncheck the 'In Game Scene' option on UIControl and remove other UI element that are not required. This will make sure the perk menu always stay on screen. Obviously, you will still need a PerkManager in the scene and possibly the RscManager. Please refer to the sample scene '*TDTK/Scene/Example_PerkMenu*'.

THANK-YOU NOTE & CONTACT INFO

Thanks for purchasing and using TDTK. I hope you enjoy your purchase. If you have any feedbacks or questions, please don't hesitate to contact me. You will find all the contact and support information you need via the drop down menu panel "***Tools/TDTK/Contact&SupportInfo***". Just in case, you can reach me at k.songtan@gmail.com or [TDTK support thread at Unity forum](#).

Finally, I would appreciate if you take time to leave a review at [AssetStore page](#). Once again, thank you!