TowerDefense-ToolKit

Documentation for Unity3D

Version: 3.0f3
Author: K.Song tan
LastUpdated: 6th January 2015
Forum: http://goo.gl/iHokzS
WebSite: http://goo.gl/GJMnrc
AssetStore: http://goo.gl/RBoSxo

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 '**Support And Contact Info**' via the top panel. Please leave a review on AssetStore if possible, your feedback and time are much appreciated.

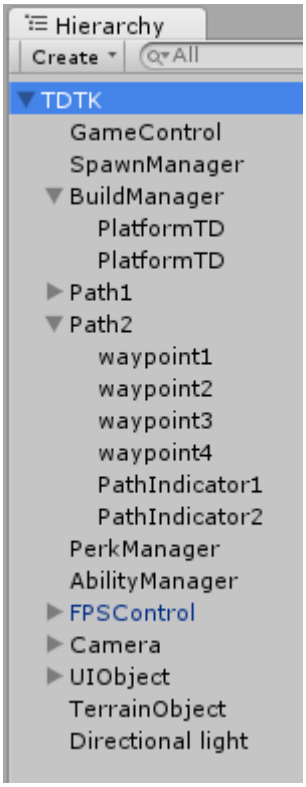
Important:

Update Note
If you are updating an existing 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.

Please note that TDTK3 is not backward compatible with TDTK2

OVERVIEW

General



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

- **GameControl/ResourceManager/DamageTable -** control all the general game logic (life, resource)
- **PathTD -** a list of waypoints positioned in 3D-space resembling a path. There can be multiple pathTD objects in a scene
- **SpawnManager** to spawn the creep which moves along **PathTD** SpawnInfo (what, which, how many creep to spawn) are specified in SpawnManager.
- **BuildManager** to govern the construction of towers
- **PlatformTD** – an object used to define a rectangular area which depend on the area size, act as a single or a grid of build-point(s) for tower. There can be multiple PlatformTD objects in a scene.
- **UI** – Not required in order for the game to run. However without a working ui, player has no way to interact with the game. The default UI is based on uGUI and is built to support all features of TDTK. You can replace the default TDTK UI should you want.

Note (to build scene with mazing platform):
PlatformTD can be assigned as a waypoint in a PathTD. If the platformTD has a grid dimension larger than 1x1, it will automatically act as a maze area. Within a maze area, each build point on the grid will act as a waypoint and creeps will try to find the shortest path through the grid. Tower built on the grid will block off the waypoint for grid, forcing creep to find alternative route.

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

- **UnitTower** – The towers
- **UnitCreep** – The creeps
- **FPSWeapon** – Weapon to be used in FPS-mode
- **ShootObject** – the 'bullet' that was shot from any unit (tower/creep) in an attack

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

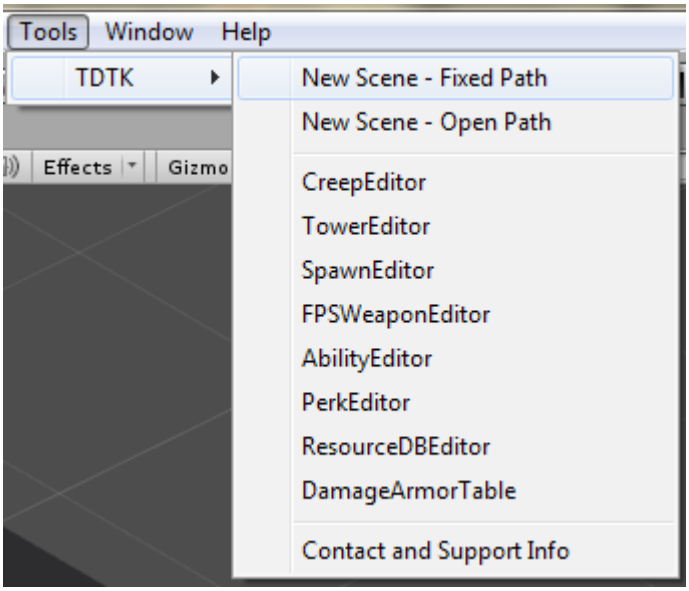
- **PerkManager** – governing the perk system. Without it, perk system are disable in game.
- **AbilityManager** – governing the ability system. Without it, ability are disabled in game
- **FPSControl** – governing the fps system. Without it, fps mode are disable in game
- **TerrainObject** – an object with collider and layer 'Terrain' (layer26 by default) to add as the terrain. It's for the most part act as the horizontal plane where the ability and drag-and-drop tower can target on.

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

- **NodeGenerator** & **PathFinder** – Component that handle the path-finding of the toolkit
- **ObjectPoolManager** – Use to pre-instantiate recycle objects that needs to be instantiate regularly
- **CameraControl** – Provide camera control for user
- **AudioManager** – Component that manage all the sound effect and music
- **PathIndicator** – Special particle system used specifically to visualize path.

HOW TO:

Create A TDTK Scene And Modify It

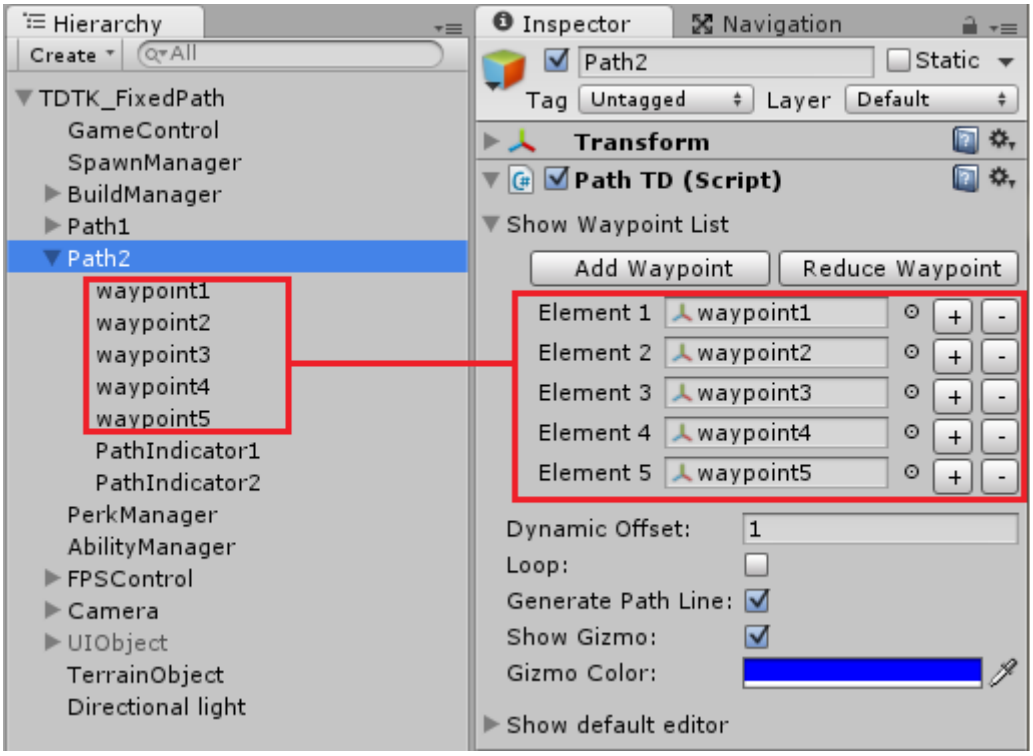


Assuming that you have some basic know how about Unity, It's very easy to create a functioning TD game using TDTK. To create a new TDTK scene, simply access the top panel '***Tools/TDTK/New Scene - Fixed Path***' or '***Tools/TDTK/New Scene - Open Path***'. This should set up a game ready scene.

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 top panel.

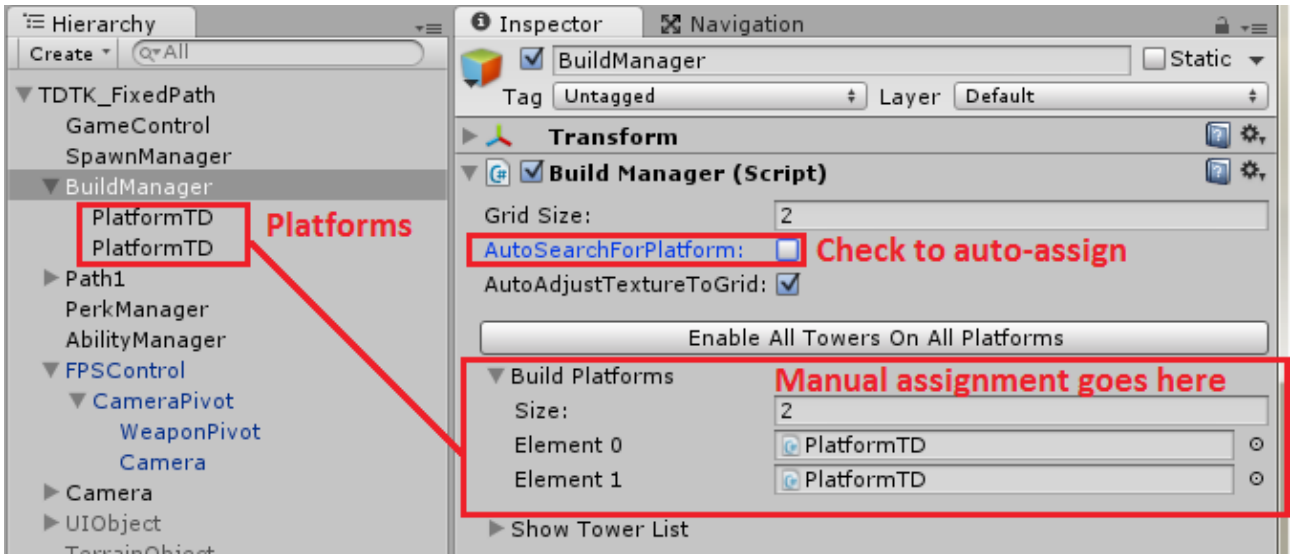
Modify Existing Path Or Add New Path.

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 key Ctrl-d) and modify it.



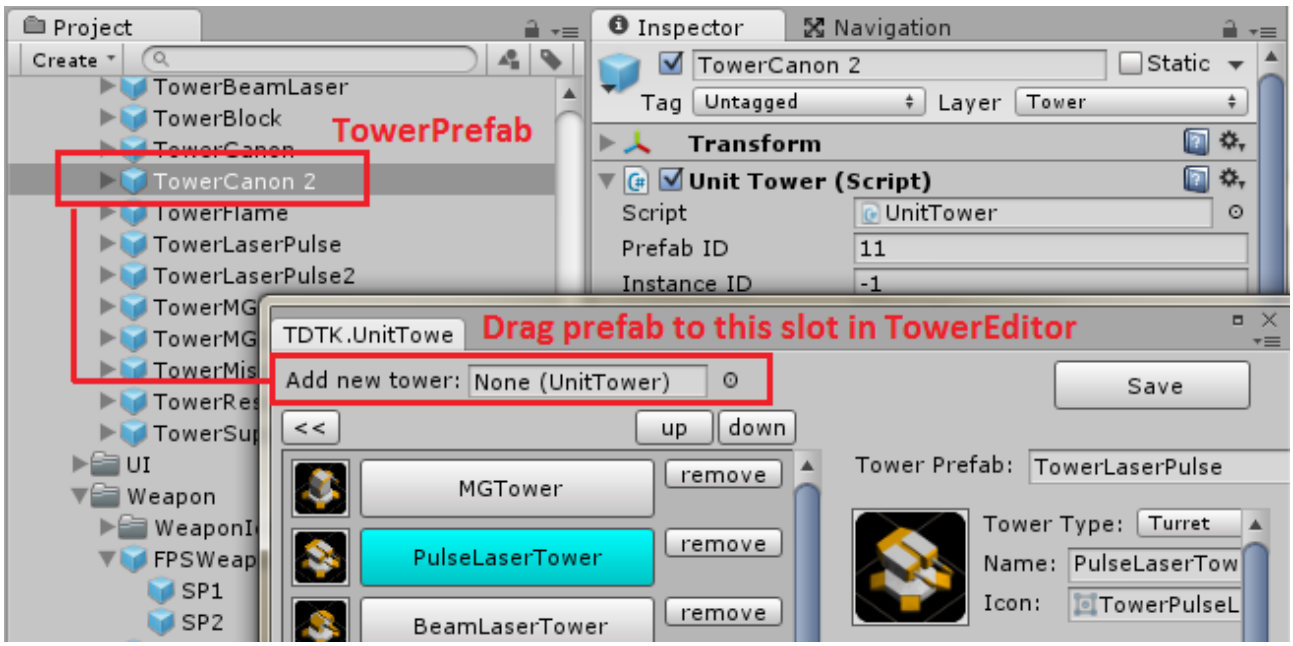
Modify Existing Platform Or Add New Platform (where Tower Is Built On)

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). To add new platform, the easiest way would be duplicate existing one (select it in HierarchyTab and press key Ctrl-d). All the platforms in the scene needs to be assigned to BuildManager, you can either do this manually by drag and drop them to BuildPlatform list, or let BuildManager auto look for all the platforms in the scene by checking the flag in BuildManager.



Add New Tower To The Game

First you will need to create the tower prefab (refer to tower for how to create a prefab) and add it to TowerEditor. Once a tower prefab is in the Editor, the tower should appear in BuildManager and available to all scenes. You can disable a tower in any particular scene by disabled it in BuildManager.



Add New Creep To The Game

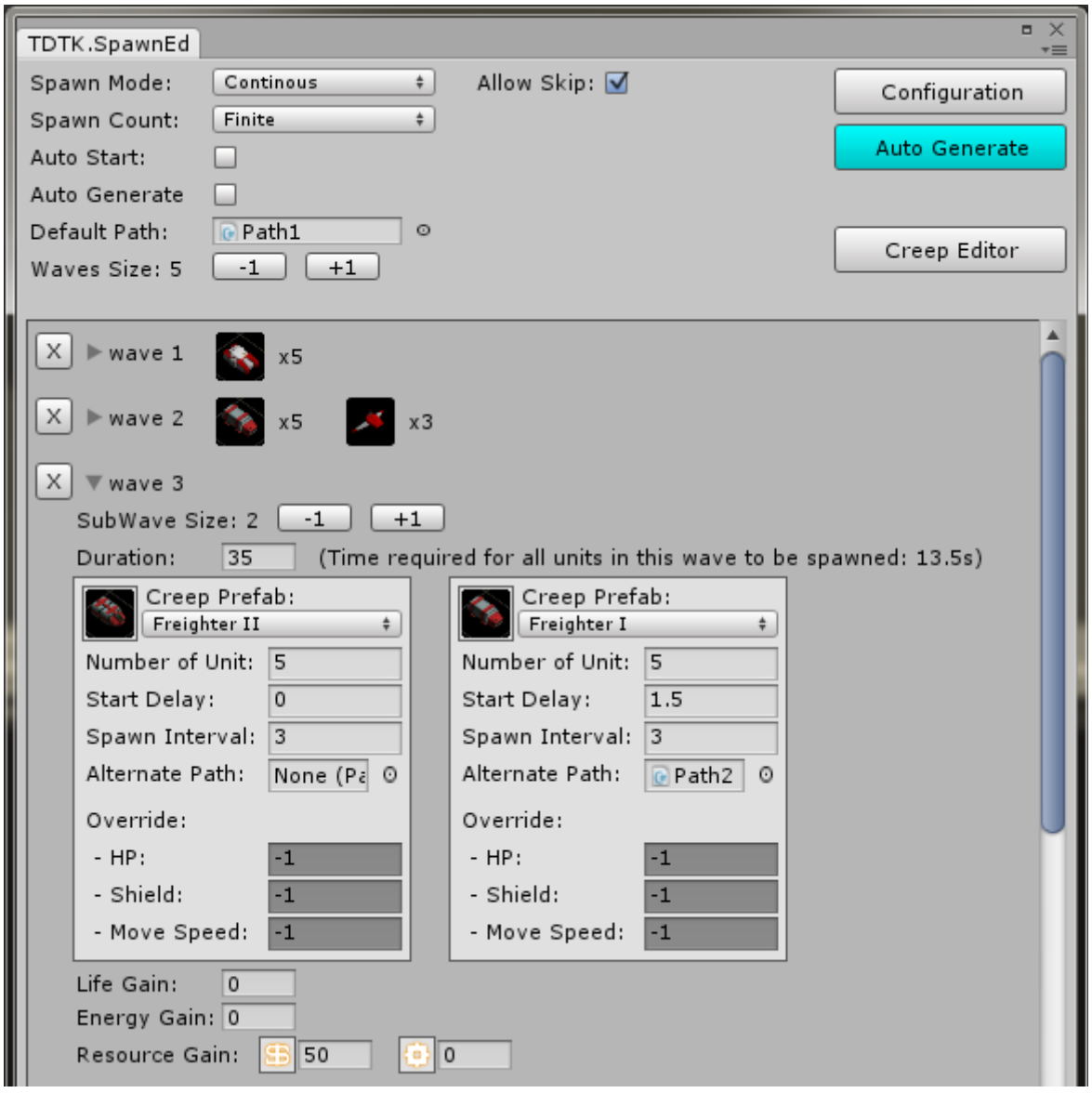
First you will need to create the new creep prefab (refer to creep for how to create a prefab) and add it to the CreepEditor. This is similar to how to add new tower to the game. Once a creep prefab is in the Editor, the option to spawn that particular unit should appear on SpawnEditor.

Add New FPS-Weapon To The Game

First you will need to create the new weapon prefab (refer to FPSWeapon for how to create a prefab) and add it to the FPSWeaponEditor. This is similar to how to add new tower to the game. Once a weapon prefab is in the Editor, the option for the weapon will shows up in the editors where appropriate

Change Spawn Information

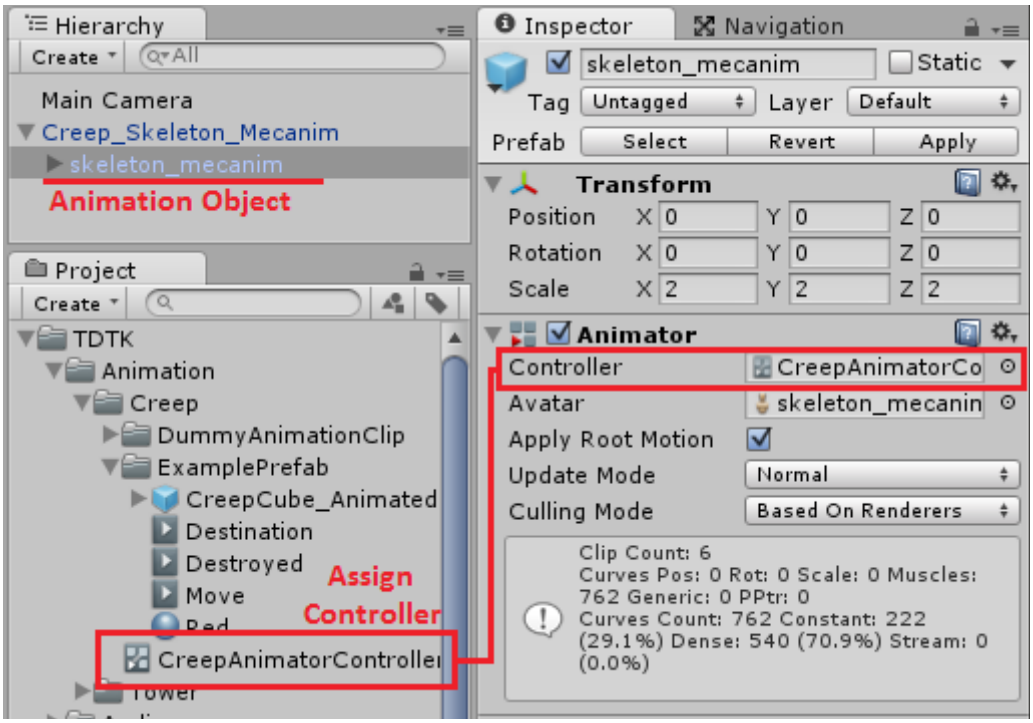
To change the spawn information, you will have to open SpawnEditor. You can use the top panel 'Tools/TDTK/SpawnEditor' or use the 'Open SpawnEditor' in SpawnManager gameObject. Once the editor is opened, it's pretty intuitive from there on.



Add New Animation To The Tower And Creep

TDTK uses mecanim animation system. The animation 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 *UnitAnimator* animation controller 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. You will find the animation controller for towers in '*TDTK/Animation/Tower*' Once done, you should be good to assign the animation clip you want to use in the editors.

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



HOW THINGS WORK:

Prefab Management

TDTK uses a centralized database to store all the information of the prefabs (towers, creep, fps-weapon) 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 BuildManager 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.

When 'UseTowerWeapon' flag is not checked on FPS Control, the player will be able to cycle through all weapon enabled in FPSControl when in fps-mode. These weapon can be individually disabled in FPSControl inspector so that player can access them in runtime. However a disable weapon can be added to the scene during runtime via PerkSystem.

Ability can be individually enabled/disabled in AbilityManager Inspector . Disable ability won't be available in the scene. However a disabled ability can be added to the scene during runtime via PerkSystem.

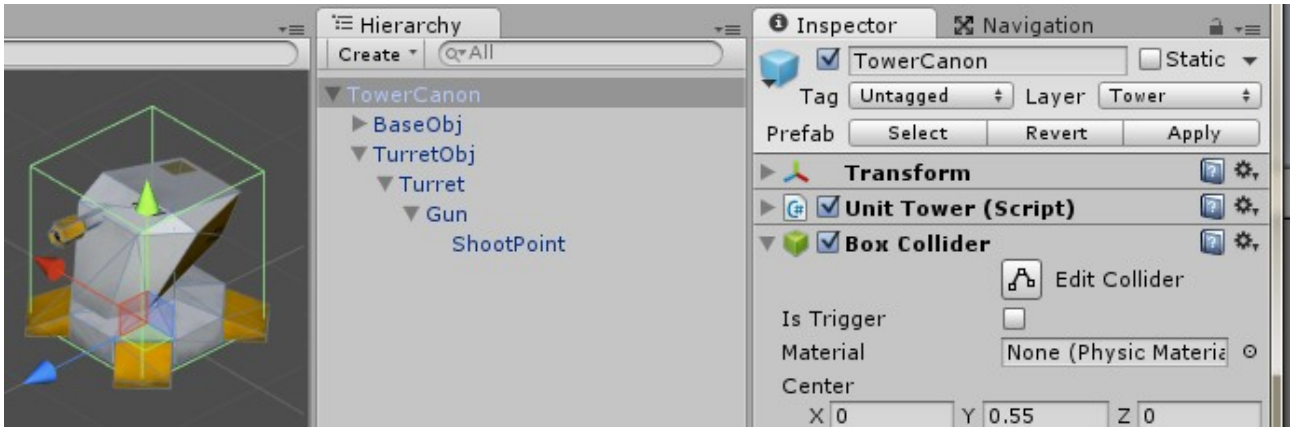
Perk can be individually enabled/disabled in PerkManager via Inspector. Disabled perk won't be available in the scene.

Tower

A working Tower prefab needs to full-fill following criteria:

- script – UnitTower.cs
- collider
- layer assigned to Tower (29 by default)

A primitive tower prefab can be an empty gameObject with just the script *UnitTower.cs* attached on it. There needs to be a collider on the prefab and the prefab has be got the right layer assigned so that it can detect a input to select the tower in runtime. On that note, the collider can be a any collider as long as it fits the mesh of the tower prefab or at least fits the size of the grid. Mesh is entirely optional (an invisible tower can function just fine). It's recommended that any mesh is place as the child transform of the tower prefab.



A typical tower prefab. Note that the collider is adjusted so the it fits the visible mesh.

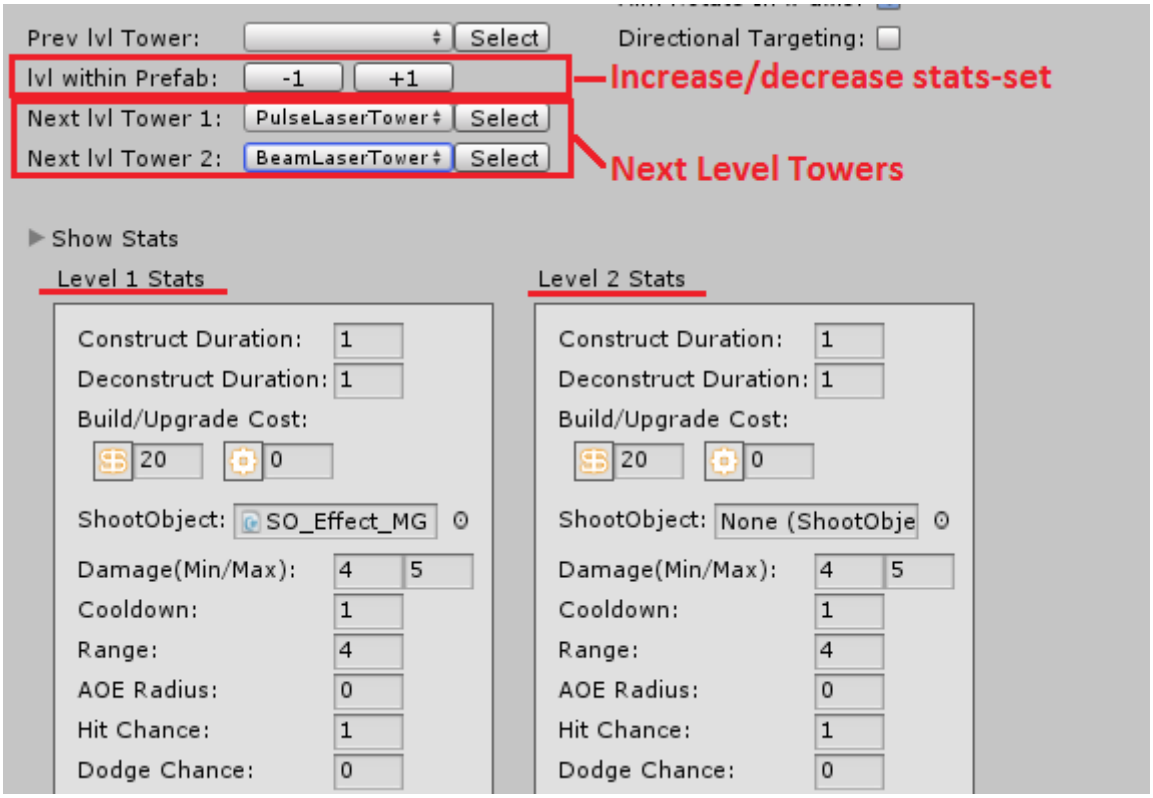
Upgrading System For Tower

There are two ways to upgrade a tower. The first one is a stats upgrade. It simply upgrade the tower stats to the next level. Note that you can has many level of stats as you want setup in TowerEditor for each tower prefab. The first set of stats is always the base stats which the tower used by default before any upgrade. To be eligible for stats upgrade, a tower needs to have at least 2 set of stats specified.

The second way to upgrade a tower is upgrading to a new prefab. This require a '*NextLevelTower*' to be specified in the TowerEditor for the tower prefab. For this upgrade method, the tower will actually destroy itself in the upgrade process and replace itself with the nextLevelTower specified. This is the upgrade method to use if you wish to change the appearance of the tower when upgrading (change of mesh/texture/color)

A tower can only be upgraded to nextLevelTower if it has been upgraded through all the sets of stats. (ie, if a tower has 3 set of stats and a nextLevelTower assigned, it needs to be upgrade 2 time to get pass the stats. Only the thrid upgrade it will be replaced by the nextLevelTower)

Once you assign the first nextLevelTower, an additional option will show up for you to assign a second nextLeveTower. If you assign two nextLevelTowers to a tower prefab, the tower can be upgrade to either of the assigned nextLevelTower (A second upgarde button will shows up in the UI in runtime when the tower is selected). Effectively the tower can branches to different upgrade path.



The tower in the images shown has two set of stats and have both nextLevelTower1 and nextLevelTower2 assigned. When it's first build, it will use the level-1 stats and have 1 upgrade option to level-2 stats. After the first upgrade, it will be using the level-2 stats and has two upgrade option, first to PulseLaserTower and second to BeamLaserTower.

Creep

A working creep prefab needs to full-fill following criteria:

- script – UnitCreep.cs
- collider
- layer assigned to Creep/Creep_Fyling (31/30 by default)

*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.

A primitive creep prefab can be an empty gameObject with the script UnitCreep.cs attached on it. It also needs to have a collider and have right layer assigned so that it can be detected by the towers during runtime. On that note, the collider can be a sphere/square/mesh collider as long as it fits the mesh of the creep prefab. Mesh is entirely optional (an invisible creep can function just fine). It's recommended that any mesh is place as the child transform of the tower prefab.

For creep that attacks, you might want to have a turret that aims at the turret and bullet that fires from the barrel of the turret. The setting required is very much similar to the setting of TurretTower. Please refer to section TurretTower for more info.

FPSWeapon

A FPSWeapon prefab is basically an empty gameObject with script FPSWeapon.cs attached to it. Since it fires shootObject like a TurretTower, shootPoint(s) are required. ShootPoints are empty gameObject placed within the hierarchy of weapon to indicate where the shootObject should be fired from. If a mesh is added to the weapon, the shootPoint should be placed at the tip of the barrel.

DamageArmorTable

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

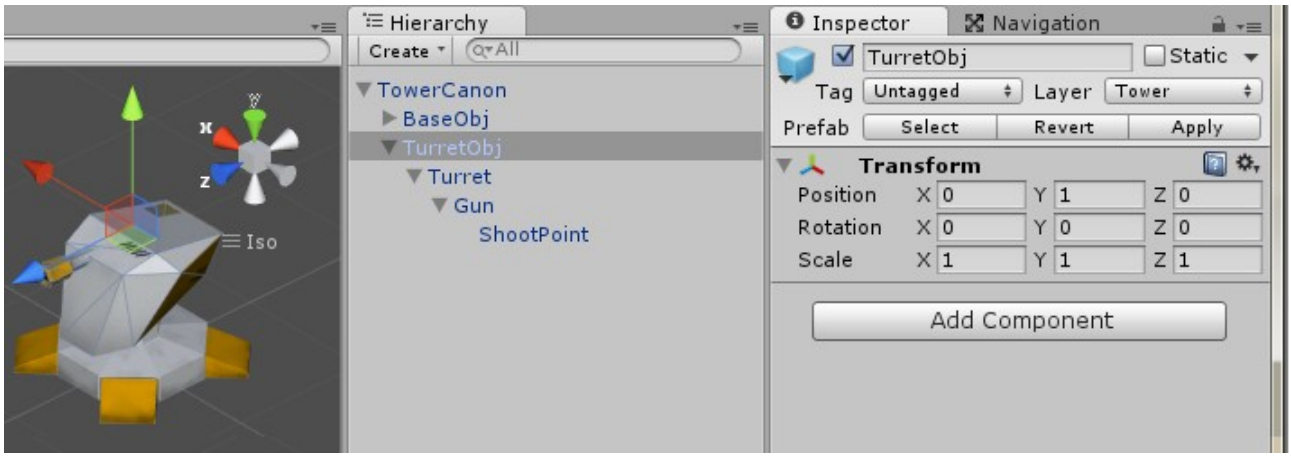
Units (tower/creep) That Fires At Target

If the unit is to fire shootObject at its target (be it tower fires at creep or creep fires at tower), it requires

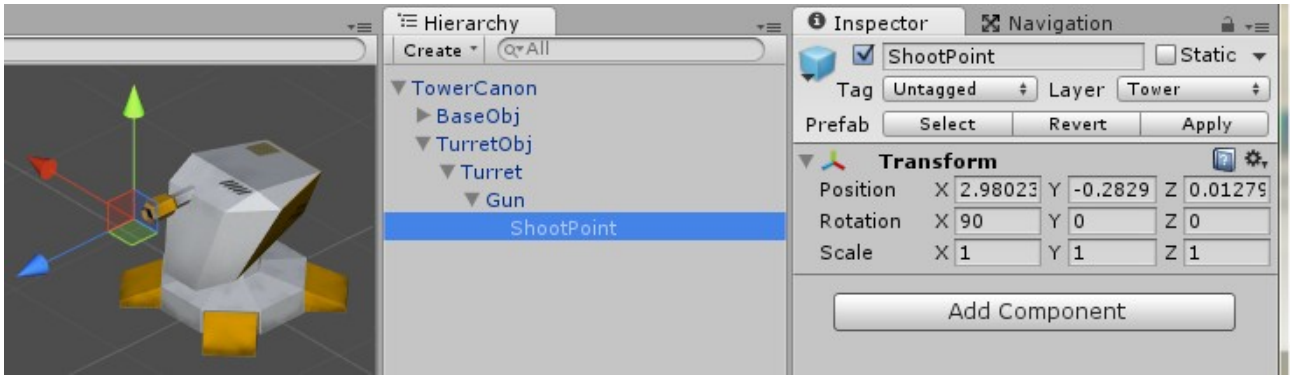
- a shootpoint
- a shootObject
- hitChance value is set to >0

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

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



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



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

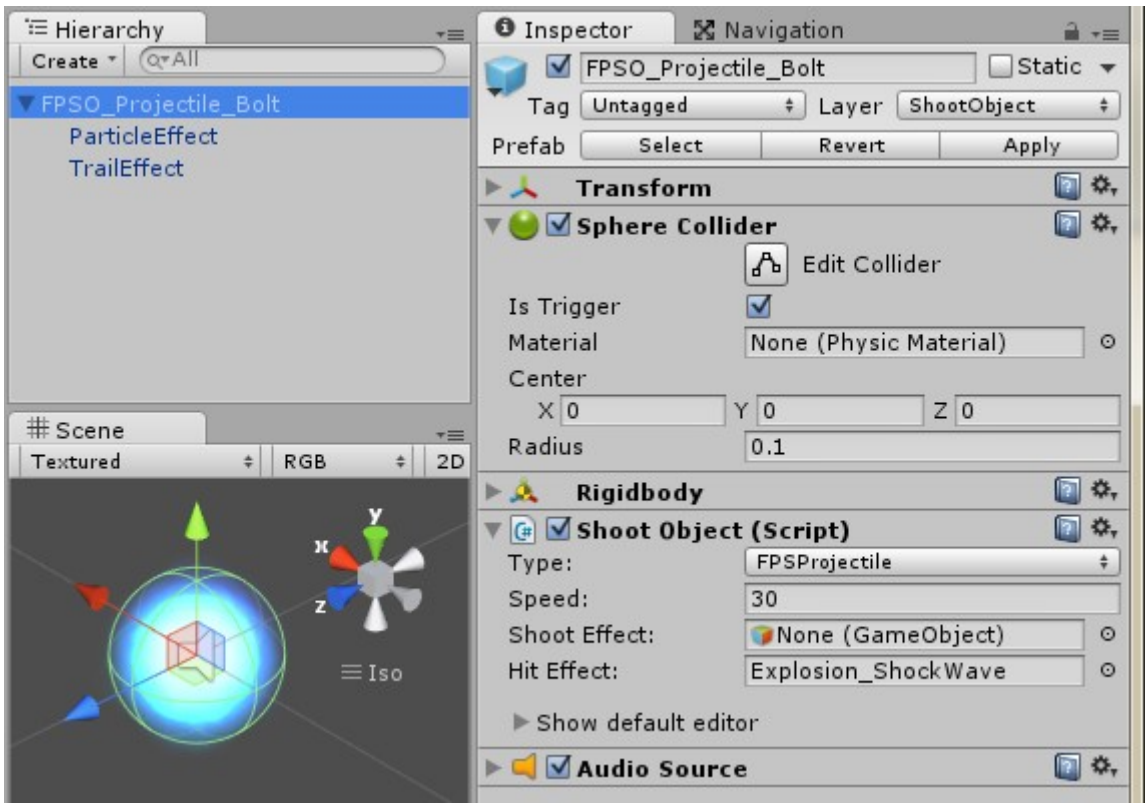
For shootObject, please refer to next section.

ShootObject

ShootObject are the 'bullet' object that is fired from a tower or a creep in an attack. It's also used for weapon in first-person-shooter mode. A shootObject needs to full-filled following criteria:

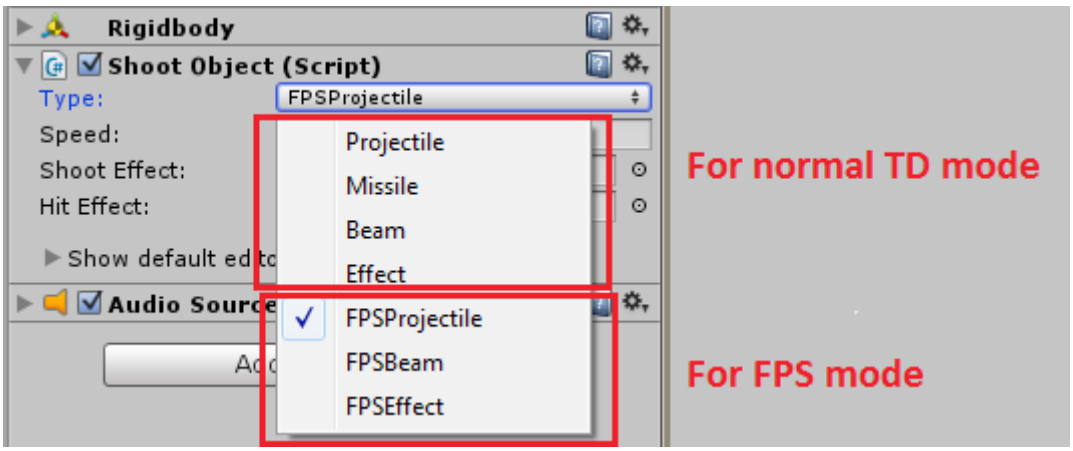
- script – ShootObject.cs
- collider
- a rigidboy with gravity off and all constraint on
- layer assigned to ShootObject (28 by default)

A primitive shootObject can be an empty gameObject with just the script ShootObject.cs attach on it. It also requires a collider and have the correct layer assigned. The collider and layer are there to stop the shootObject from going through terrain object (so it wont get shoot through a mountain or something). In fps mode, the collider and layer assignment are used to detect if a shootObject hits a target.



A shootObject prefab for FPSMode, note that the collider size is adjusted to fit the visible size of the shootObject. Note that like the example prefab, you can add AudioSource to the shootObject to play the shoot sound

There are two primarily two type of ShootObject, one's for FPS-mode and one's for normal TD mode and they need to be set accordingly depends on what they are used for. For FPS-mode, the shootObject type has a 'FPS' prefix in front of it.



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

First Person Shooter Mode

The FPS mode is tied to the tower. Player can only enter FPS mode via a selected tower in which the view point will be anchor on the selected tower. There are two distinct settings for FPS mode, that is with 'UseTowerWeapon' flag in FPSControl turned on/off. When on, player will only be able to use the weapon assigned to the tower and weapon switching is not allowed. When off, player can switch weapon (using q and e key) while in as long as the weapon is enabled in FPSControl in the first place.

FPSControl requires a very sepcific camera setup to work correctly. For that it's highly recommended that you use the default FPSControl prefab (located in 'TDTK/Prefabs/SceneComponent/FPSControl') unless you know what you are doing.

Ability And Perk System

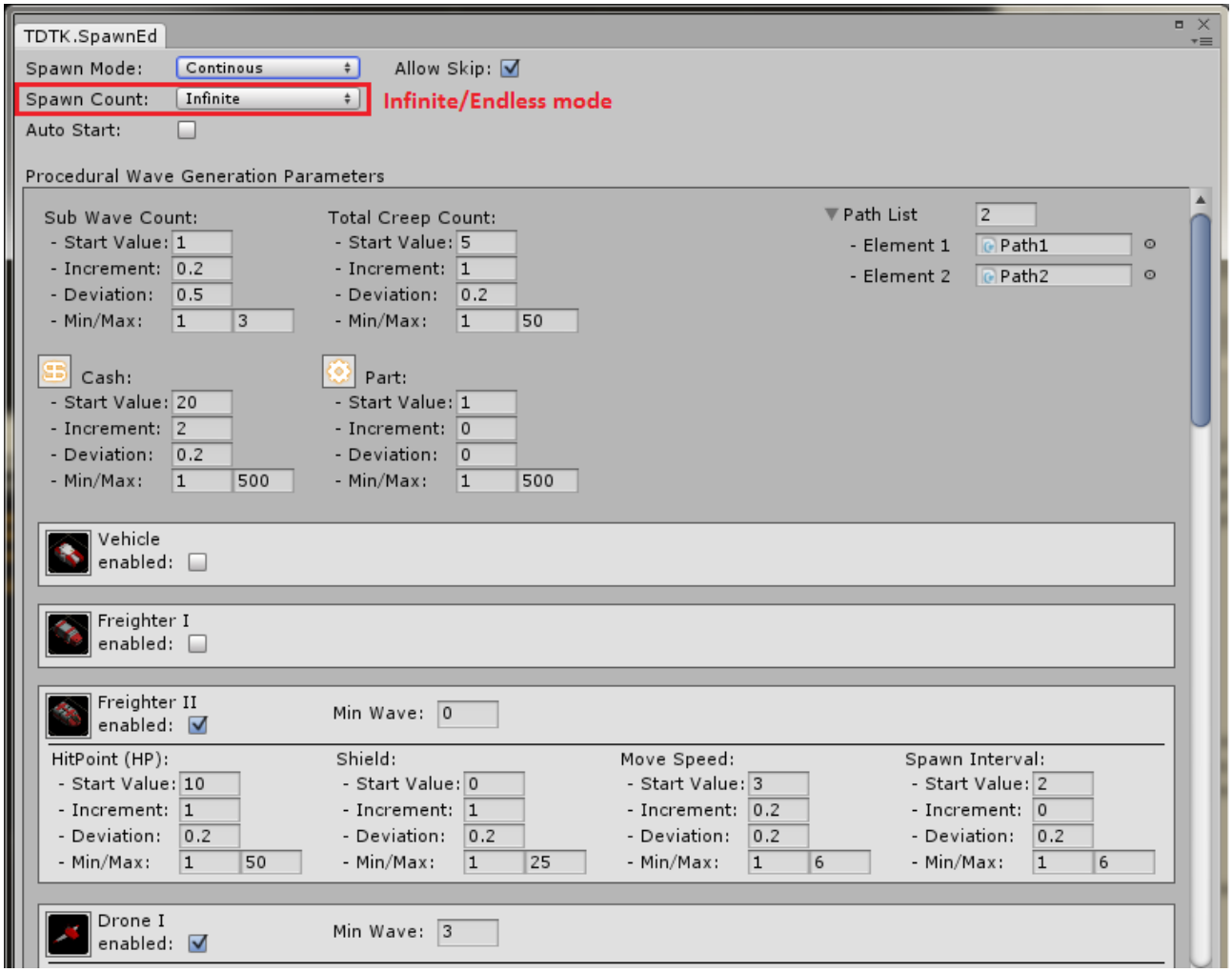
Ability and perk system are optional extra to the framework. Ability is special action that can be performed by player during runtime to achieve various means. Perks are upgrade item that can be purchased during runtime to give player a boost in various means.

Both of them are pretty easy to setup. Ability can be add/edit via AbilityEditor. Perk can be add/edit via PerkEditor. Both AbilityEditor and PerkEditor can be accessed from the top panel. You should be able to find out more about the details of the configuration via the tooltip in the editor. These system are designed to be as flexible as possible to accommodate many design possibilities.

Endless Mode & Procedural Generated Spawn Info

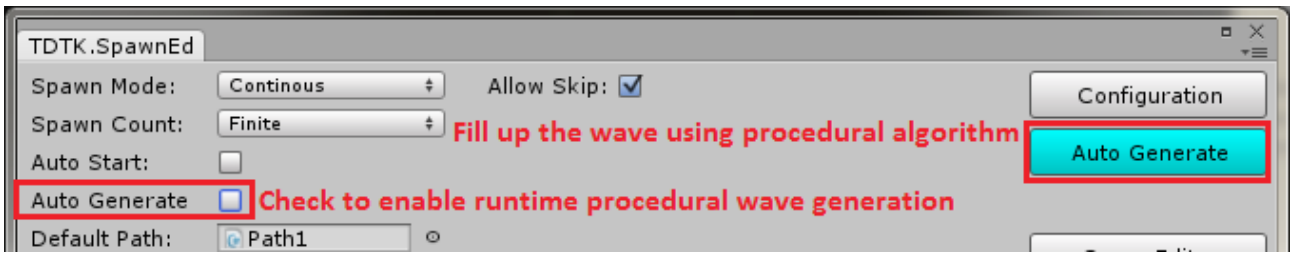
Endless mode is a special mode where there's no wave limit. The SpawnManager simply keeps on spawning until the player is defeated. To enable endless mode, simply set the SpawnCount in SpawnEditor to Infinite.

When set to endless mode, the SpawnManager will no longer use pre-set spawn info. Instead, it will procedurally spawn all the wave as the game progress in runtime. However the wave generation is done according to some preset rules. These can be configured in the SpawnEditor as well.



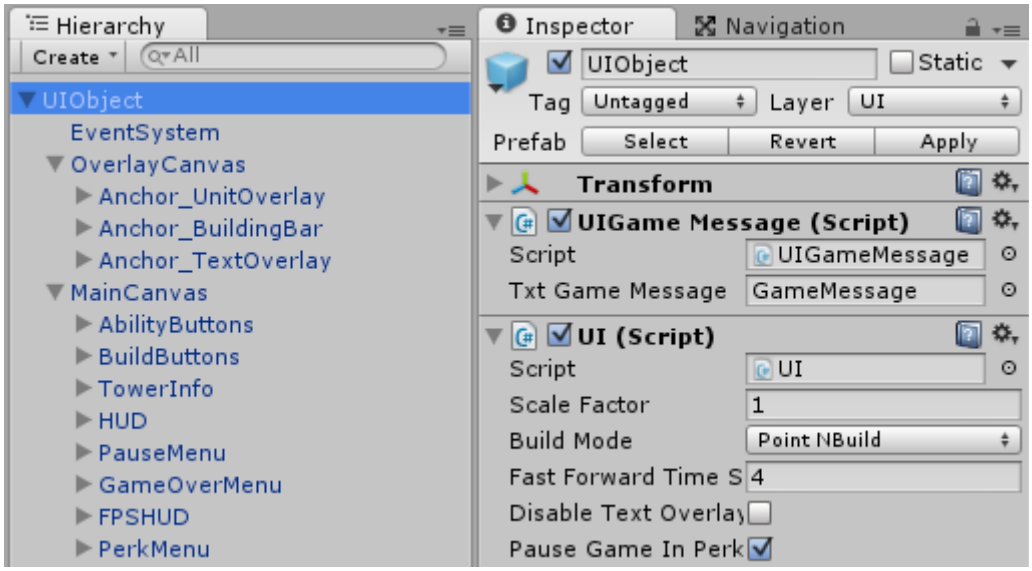
Most of the variable is calculated using a linear equation y=(m*x)+c, where c is the StartValue, m is the Increment and x is the wave number. The calculated value is then randomized by the deviation but the ultimate value is limited within the min/max range.

You can use the procedurally wave generation algorithm to quickly fill up spawn info for level in normal mode (level with finite wave count). You can also enable runtime generation of the spawn info in normal mode to create level with randomized wave.


TDTK.SpawnEd
Spawn Mode:
Continous
Allow Skip:
Configuration
Spawn Count:
Finite
Fill up the wave using procedural algorithm
Auto Start:
Auto Generate
Auto Generate
Check to enable runtime procedural wave generation
Default Path:
Path1

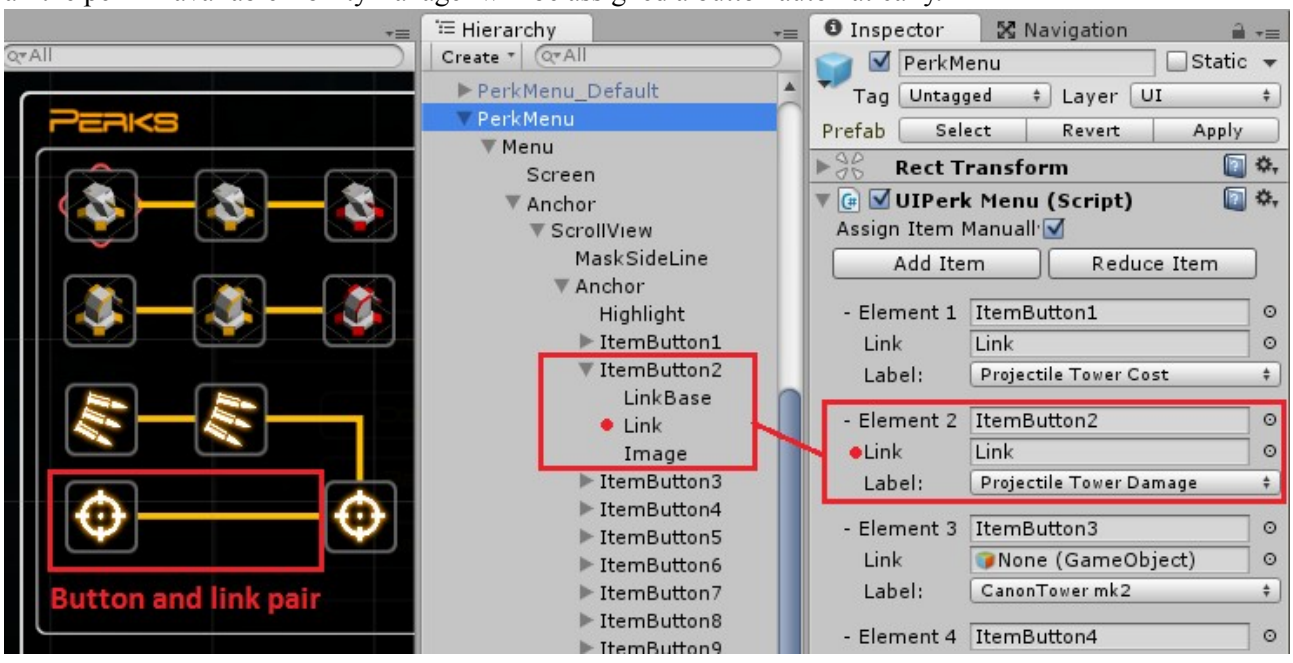
ABOUT USER INTERFACE:

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 'TDTK/Prefabs/UI/UIObject'. Certain parameters in the default UI prefab that are made to be configurable.



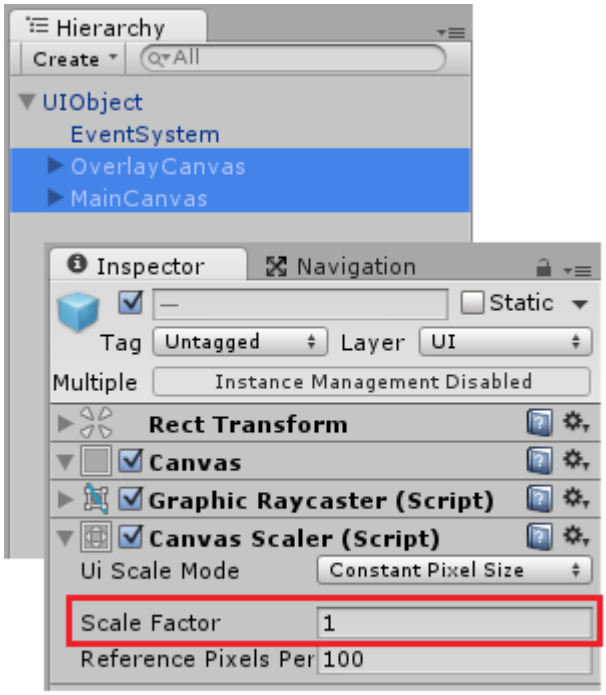
PerkMenu

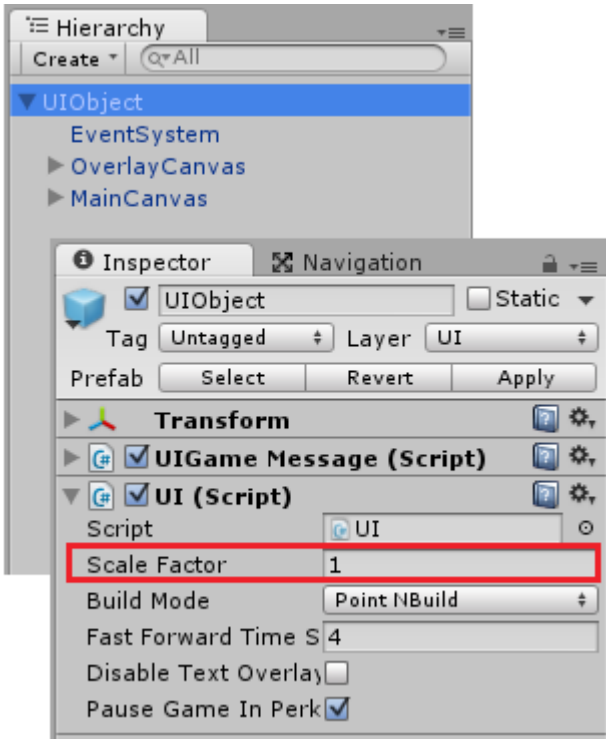
PerkMenu is the special UI object that support customization, mainly to provide a mean to build custom tech-tree in conjunct with the perk system. When the flag '*AssignItemManually*' is checked. You can arrange the button anyway you want, you just need to assign them as item to UIPerkMenu component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



Scalling the UI

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





TIPS:

Almost every configurable item has tooltip that explain what are they for. If you are unsure about what a setting is for, try position your cursor over the label to bring up the tooltip.

You can disable Ability system by removing AbilityManager from the scene

You can disable Perk system by removing PerkManager from the scene

You can disable FPS-mode by removing FPSControl from the scene

You can switch between point-and-click or drag-and-drop mode in UI

You can stop a tower from showing up in BuildManager by checking the disableInBuildManager flag in TowerEditor. This is useful if the tower is only available when unlocked via perk.

You can stop an ability from showing up in AbilityManager by checking the disableInAbilityManager flag in AbilityEditor. This is useful if the ability is only available when unlocked via perk.

You can create your custom ability effect by execute the script it from the effectObject spawn by the ability.

You can create your own persistent PerkSystem. The trick is to save a list<int> of the ID of the purcahsed perks and then use it to override the value of purchasedIDList of PerkManager before PerkManager.Init() is called in GameControl.Awake()

All the indicators for tile select , tower range and abilities target can be replaced. Feel free to create your own prefab and assign them where's appropriate

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 dont hesitate to contact me. You will find all the contact and support information you need via the top 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, thankyou!