TowerDefense-ToolKit

Documentation for Unity3D

Version: 4.1
Author: K.Song Tan
LastUpdated: 1st February 2021

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 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 recommended that you try to familiarise yourself with the basics 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 basics, 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 the Unity Editor. Please leave a review on the Asset Store 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 settings (towers, creeps, damage-table, etc), just uncheck the '*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 settings dictate how the scene will play. For instance, how many waves of creeps there are, how the paths are laid out, how many resources players have and so on.

Then there are 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 towers can be built, which creeps 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 the custom editor window which can be accessed via the drop-down menu on the top of the Unity Editor under '*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* Controls the game state and major game logic.

RscManager* Contains all the code and logic for resources.

SpawnManager* Controls the spawn logic and contains all the spawn information for a scene.

TowerManager* Controls the build logic and contains the information of buildable towers.

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

Path Used to specify the waypoints that form the path(s) and stores the information of the constructed path(s) during runtime.

BuildPlatform Attached to a game object with a collider to form a BuildPlatform object. A BuildPlaform object forms a grid that specifies what tower can be built and where it can be placed. When assigned as a path waypoint, BuildPlatform acts as a walkable grid which the creep has to navigate through via pathfinding.

* There can be only one instance of these particular components in any given scene.

Prefab Components And Class

These are the components attached to their corresponding prefabs to be spawned at runtime.

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

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

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

Optional/Support Components And Class

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

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

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

AStar** Controls logic for pathfinding.

ObjectPoolManager** Controls logic for object-pooling.

CameraControl Controls logic for camera.

AudioManager Contains and plays all sound effects.

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

Layers

TDTK uses Unity layer system to identify various objects in the scene. Unfortunately the layer setting doesn't get imported to your project automatically. The system will still work but to avoid confusion it's recommended that you name the layer accordingly.

The layer used by default are:

- layer 25: ‘Terrain’
- layer 25: ‘Obstacle’
- layer 27: ‘Creep’
- layer 28: ‘Tower’
- layer 29: ‘Platform’
- layer 30: ‘Path’
- layer 31: ‘Terrain’

You can find the definition of these layers in TDTK.cs.

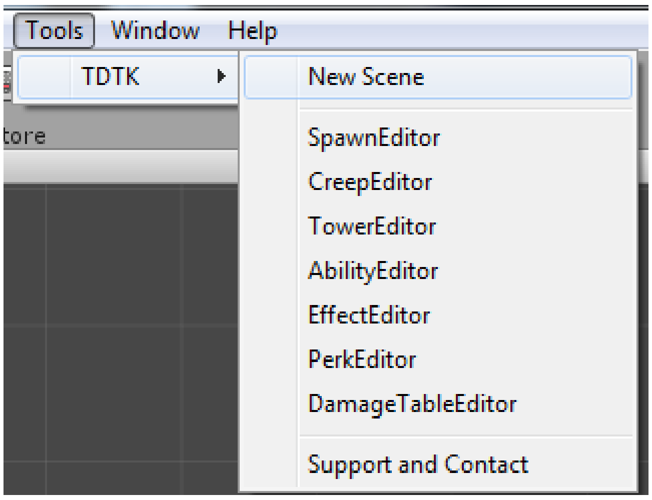
UI

Although the UI is required for a level to be playable, it's a modular extension of the framework. All the base logic can run without it. That means you can delete all placeholder UI components and replace them with your own custom UI.

HOW TO:

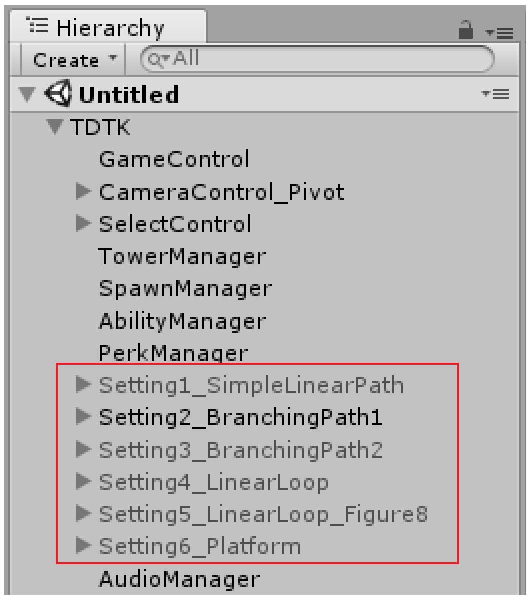
Create A TDTK Scene And Modify It

To create a new scene, simple use the drop-down menu '***Tools/TDTK/New Scene***' and a 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 settings and rules by configuring the active component in the scene via the Inspector. There are also various editor windows that will help you configure more complicated settings such as tower 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 pre-set settings (as highlighted in the image below). You can switch between them by deactivating the active game object's setting then activating the setting you want. Each of them currently provides an example for a specific setting (linear path, branching path, looping path, custom build-platform, etc.). Some have been used in the demo scene.

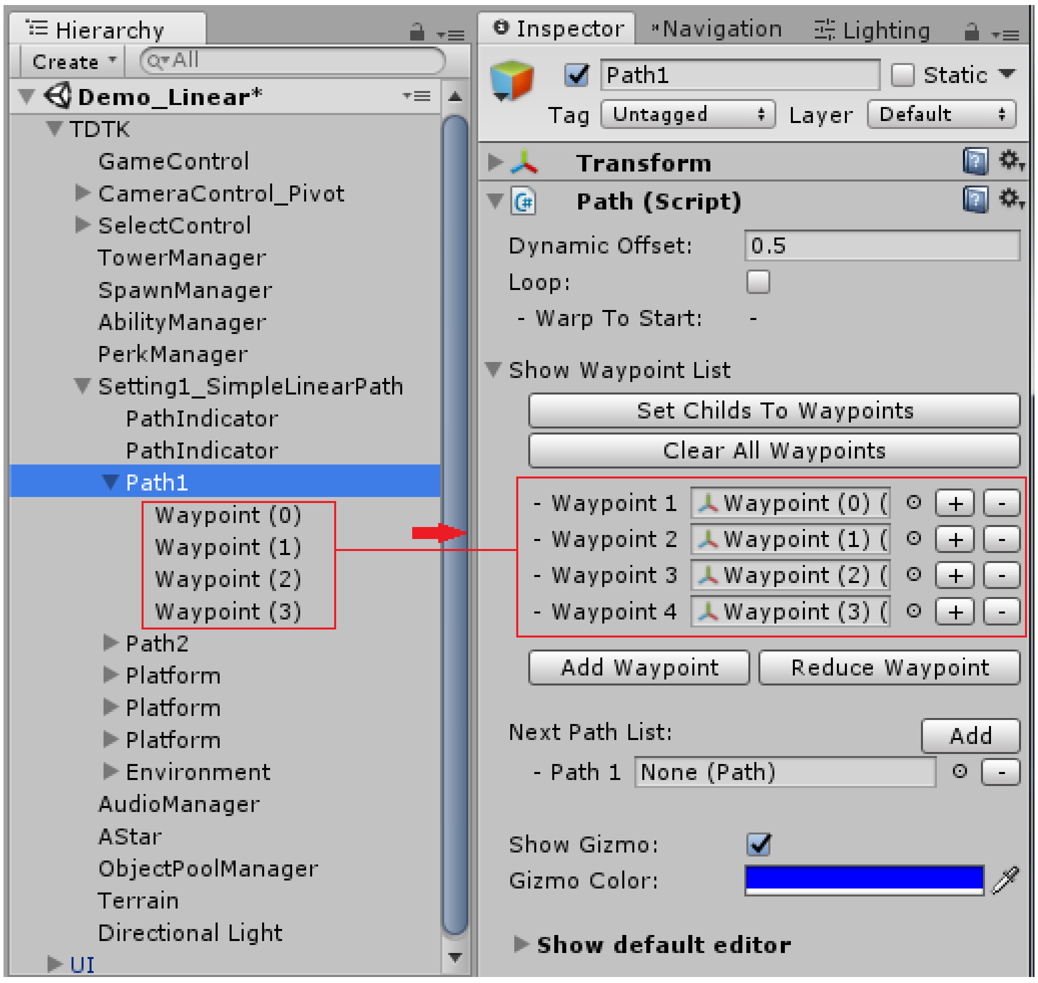


Working With Path

Basic Path Creation and Modification

You can modify existing path routes by changing the position of individual waypoints within the path. To extend the path, you can add a new waypoint in the Hierarchy by creating a new empty game object as a child to the 'Path' game object you want to extend. Rename the empty game object to whichever Waypoint(#) you desire. Then select the 'Path' game object to bring up it's script component in the Inspector, click 'Add Waypoint', then drag the new Waypoint(#) game object to assign it in any order you want it to appear within the path route (reorder the other waypoints as needed).

Remember, when you 'Add Waypoints' from the Inspector, it doesn't create a corresponding game object; the waypoint will only appear within the path once you've assigned a waypoint game object from the Hierarchy. The easiest way to add a new path is to duplicate an existing one from the Hierarchy (select 'Path' game object and press Ctrl-d) and modify the waypoints as needed.



```
Demo_Linear*
  TDTK
    GameControl
    CameraControl_Pivot
    SelectControl
    TowerManager
    SpawnManager
    AbilityManager
    PerkManager
    Setting1_SimpleLinearPath
      PathIndicator
      PathIndicator
      Path1
        Waypoint (0)
        Waypoint (1)
        Waypoint (2)
        Waypoint (3)
      Path2
      Platform
      Platform
      Platform
      Environment
    AudioManager
    AStar
    ObjectPoolManager
    Terrain
    Directional Light
  UI
```

```
Path1
Tag Untagged    Layer Default
Dynamic Offset:   0.5
Loop:             [ ]
 - Warp To Start: -
 - Waypoint 1   Waypoint (0) (
 - Waypoint 2   Waypoint (1) (
 - Waypoint 3   Waypoint (2) (
 - Waypoint 4   Waypoint (3) (
Next Path List:
 - Path 1  None (Path)
Show Gizmo:   [✓]
Gizmo Color:  🟦
```

Paths With a Walkable BuildPlatform

You can create a path that moves through a walkable grid by simply assigning a BuildPlatform object (one with 'BuildPlatform' component attached) as one of the waypoints. The build-platform will automatically form a grid in which the creep needs to navigate through using pathfinding. Note that build-platform cannot be assigned as the first waypoint of the path.

Branching Paths
Within the ‘Path’ component, you can add ‘Next Path’ and assign a different path, which acts as the continuation of the current path once it ends. When multiple paths are assigned as the next path, a branching path is created. When a creep reaches a branching point, it will automatically select the shortest path. A branching path is only useful when the branched-out path contains a walkable ‘BuildPlatform’ object which allows the length of the path to be altered at runtime. Otherwise the creep will always follow the shortest path, making the branching meaningless. An example of this setup can be found in '*TDTK/Scenes_Demo/Demo_BranchingPath*'.

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

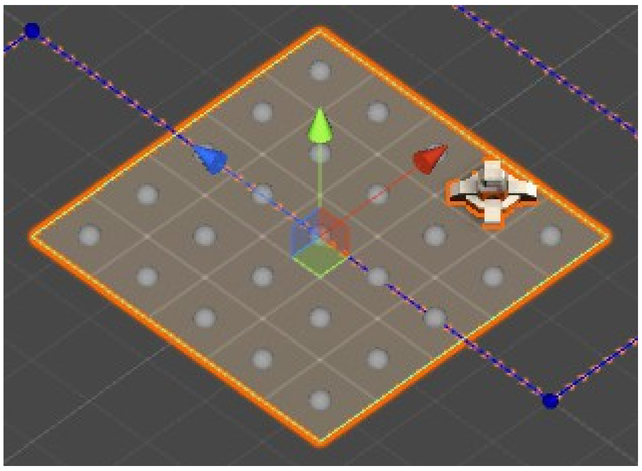
Note on Pathfinding and Branching Paths
There are two things you need to know when configuring paths that involve pathfinding or branching.

- 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. However, if there's more than one path listed under ‘Next Path’, the player can block any path entirely provided there's still another alternative path to allow the creep to reach destination.

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

You can modify existing BuildPlatform objects by simply changing its position or scale (note – a 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). For instance, if your platform has a scale of (4,4) and your grid size is set to 1, you will have 16 (4x4) tower slots; however, if the grid size is set to 2, you will have 4 (2x2) tower slots. To add a new platform, the easiest way would be to duplicate an existing one (select it in the Hierarchy and press Ctrl-d).


A simple walkable BuildPlatform object in runtime in SceneView. Each tile can accommodate one tower and each white sphere can act as a waypoint for creeps.

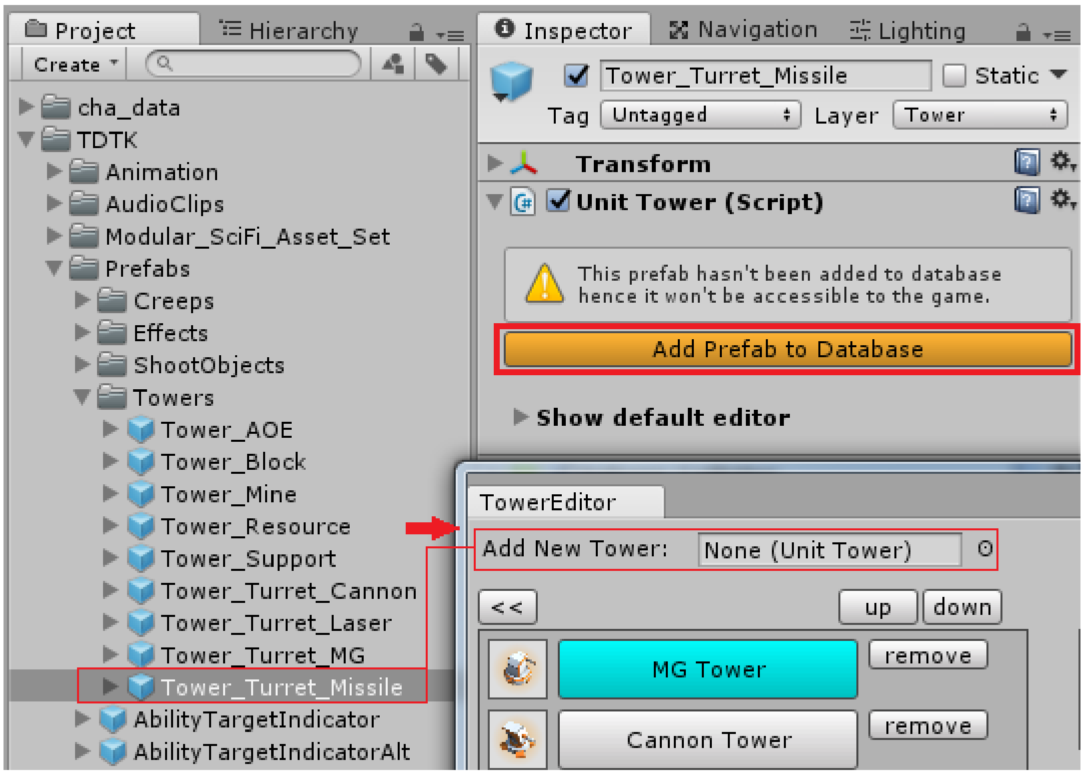
The default BuildPlatform object is created using a default unity's primitive mesh 'Quad' with a box collider. It's recommended you keep the same template as the code will always assume the game object will contain a box collider and a 'quad' mesh. (Quad is the primitve 3D shape that you can create via the drop down menu '*GameObject>3D Object>Quad*') 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 BuildPlatform object 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 visual element.

It's possible to create a BuildPlatform object with your own custom shape and layout. You can place various colliders on layer 26 - ('Obstacle') to block out specific tiles on the grid. Blocked tiles cannot have towers built on them nor can creeps traverse through them. You can also create a 'no build zone' by placing colliders of specific tiles on layer 25 – ('No Build Zone'). The player will not be able to build any towers on a 'no build zone'. However, creeps will still be able to traverse through it. For a BuildPlatform object that is not square or rectanglular, you can simply disable the mesh renderer on the platform itself and add in your own custom mesh. There’s an example for all this in the example scene '*/TDTK/Scenes_Demo/Demo_Platform*'.

Add New Tower To The Game

To add a new tower to the game, first you 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 the TowerEditor. You can either manually drag the prefab to the TowerEditor window (as shown below) or use the '*Add Prefab to Database*' button on the 'UnitTower' component of your tower's 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 disabling it in TowerManager.



Tower_Turret_Missile Static
Tag Untagged Layer Tower

This prefab hasn't been added to database hence it won't be accessible to the game.

Add New Tower: None (Unit Tower)

NOTE: 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 creep 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 you would add new towers to the game. Once a creep prefab is in the Editor, the option to spawn that particular unit should appear in SpawnEditor.

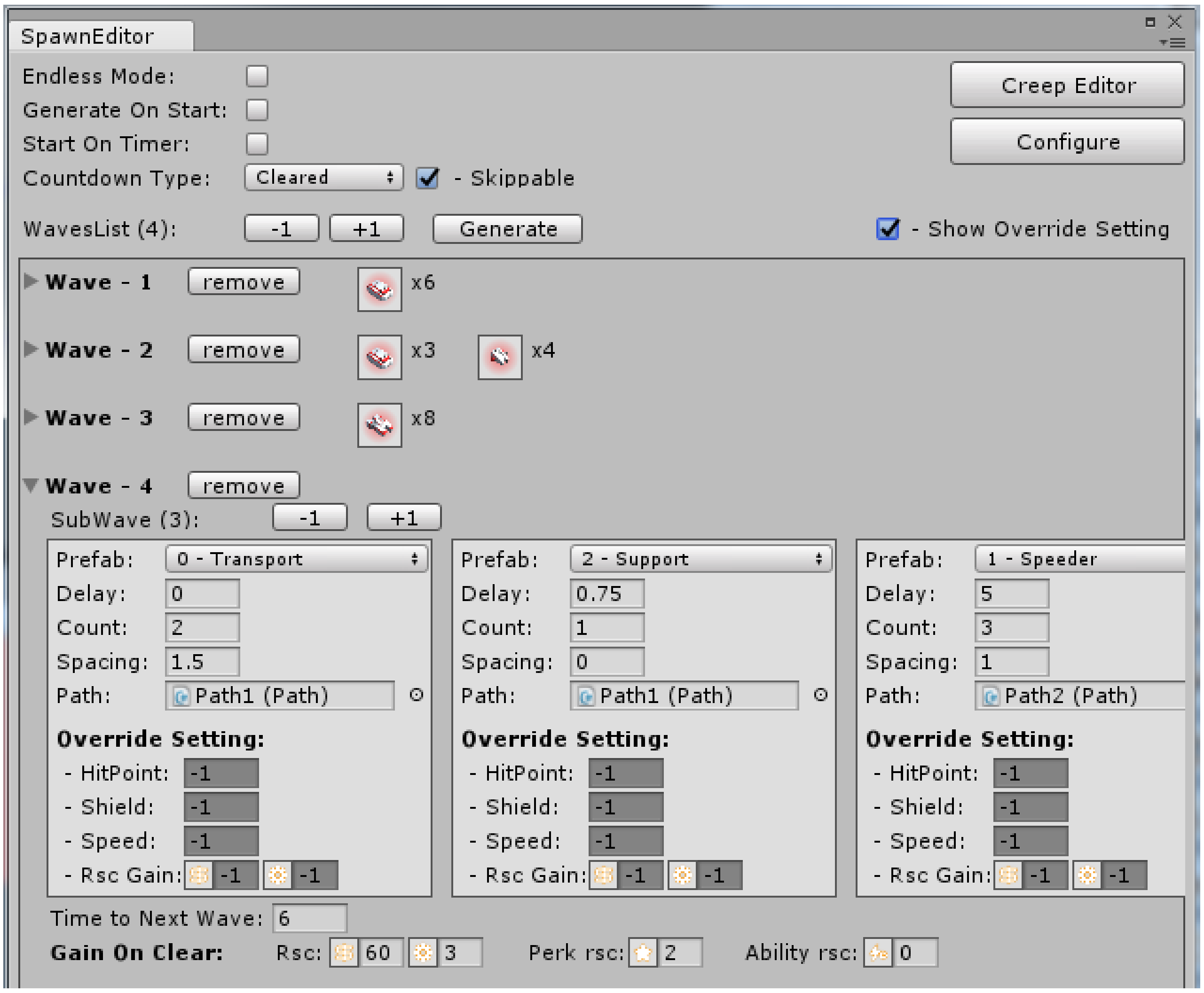
Create A Unit Using Custom Model

Here's a simple step by step to create a new tower or creep prefab using a custom model.

1. Create an empty object, add a Unit component (either UnitTower or UnitCreep) on it.
2. Drag your model into the empty object, set the position to (0, 0, 0). This will make sure the model is positioned at the unit's root object
3. Now rotate the model so that it faces the positive z-axis.
4. Expand the hierarchy, try to find the object which serves as the pivot for the turret. Let's refer to this object as '*TurretObj*'
5. Create an empty object, make sure the rotation is (0, 0, 0) and name it 'TurretPivot'
6. Drag TurretPivot into TurretObj, set the position to (0, 0, 0). Now the TurretPivot should be in a similar position with *TurretObj*.
7. Reverse the child parent relationship between TurretPivot and *TurretObj*. Make TurretPivot the parent and *TurretObj* the child.
8. Now set TurretPivot as the turret pivot of the unit.
9. You can repeat step-3 to step-7 for barrel pivot if you want the barrel to rotate independently of the turret in x-aixs.

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 the SpawnManager game object. Once the editor is opened, it's pretty intuitive from there on. As shown in the image below, you can specify the number of waves to spawn as well as detailed information for each wave.



SpawnEditor

Endless Mode: ☐
Generate On Start: ☐
Start On Timer: ☐
Countdown Type: Cleared ☑ - Skippable

WavesList (4): -1 +1 Generate ☑ - Show Override Setting

Wave - 1 remove x6
Wave - 2 remove x3 x4
Wave - 3 remove x8
Wave - 4 remove
SubWave (3): -1 +1

Prefab: 0 - Transport	Prefab: 2 - Support	Prefab: 1 - Speeder
Delay: 0	Delay: 0.75	Delay: 5
Count: 2	Count: 1	Count: 3
Spacing: 1.5	Spacing: 0	Spacing: 1
Path: Path1 (Path)	Path: Path1 (Path)	Path: Path2 (Path)
Override Setting:	**Override Setting:**	**Override Setting:**
- HitPoint: -1	- HitPoint: -1	- HitPoint: -1
- Shield: -1	- Shield: -1	- Shield: -1
- Speed: -1	- Speed: -1	- Speed: -1
- Rsc Gain: -1 -1	- Rsc Gain: -1 -1	- Rsc Gain: -1 -1

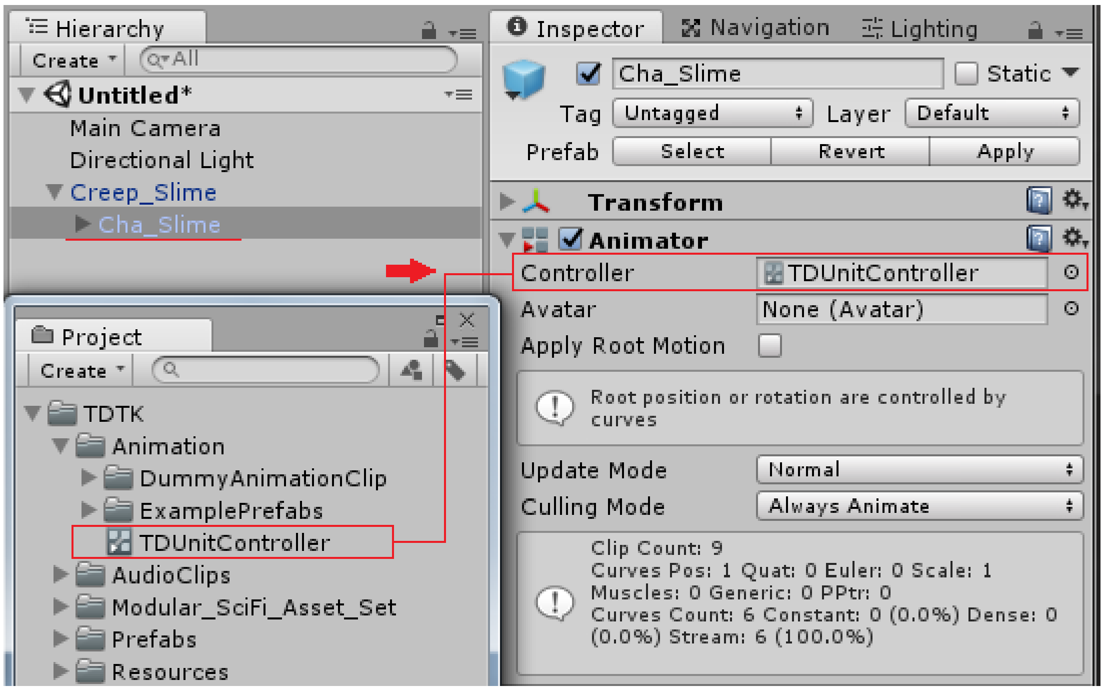
Time to Next Wave: 6
Gain On Clear: Rsc: 60 3 Perk rsc: 2 Ability rsc: 0

There are also options to use procedural generation for the spawn info (by either enabling '*Endless Mode*' or enabling '*Generate On Start*'). When enabled, you configure the input parameters of the entire 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 the mecanim animation system. The animation clip to be used in various events can be assigned in the editor (TowerEditor for towers and CreepEditor for creeps). However, before you can do that you will need to set up the prefab properly. First you will need to make sure there's an Animator component on your tower/creep 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 image shows the controller for the creep animator but the process is similar for towers. Once done, you should be good to assign the animation clip you want to use in the editors.

Note that in the Creep/Tower Editor, > *Animation Setting,* there's a slot for '*Animator Object*'. That should be assigned by the object within the prefab hierarchy that holds the Animator component. 'Cha_Slime' in the case shown in the image below.



Cha_Slime Static
Tag Untagged Layer Default
Prefab Select Revert Apply
Transform
Animator
Controller TDUnitController
Avatar None (Avatar)
Apply Root Motion
Root position or rotation are controlled by curves
Update Mode Normal
Culling Mode Always Animate
Clip Count: 9
Curves Pos: 1 Quat: 0 Euler: 0 Scale: 1
Muscles: 0 Generic: 0 PPtr: 0
Curves Count: 6 Constant: 0 (0.0%) Dense: 0 (0.0%) Stream: 6 (100.0%)

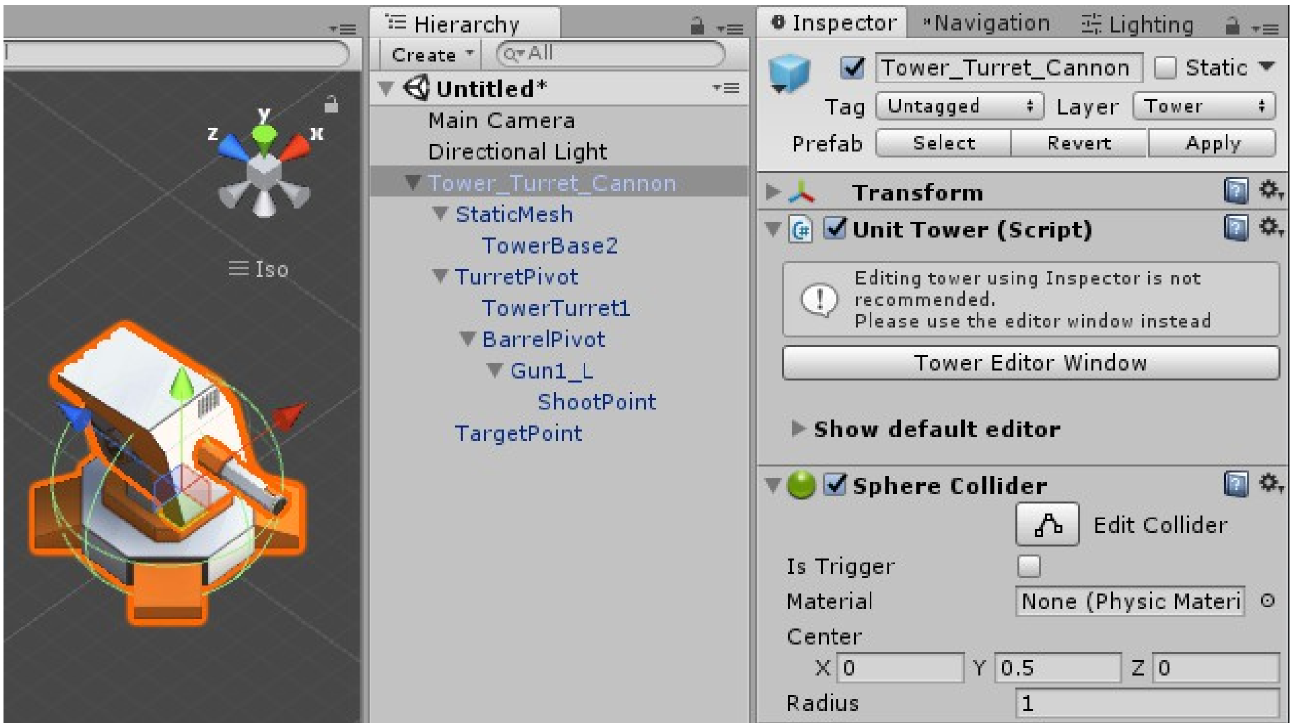
ABOUT PREFABS:

Tower Prefab

A working tower prefab can be as simple as an empty game object with the *UnitTower.cs* script component attached to it. This will enable the tower to be built and function as any regular tower despite it not being visible. Any mesh/model on the tower is optional.

To make the tower targetable in ability targeting phase, it requires a collider. To make sure the collider is not in conflict with other components, 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 encapsulate only the mesh/model of the tower.

Finally, it's recommended you follow the hierarchy structure used by the default tower prefab. Especially when it comes to 'Turret' type towers that involve aiming. All the static meshes are placed in the 'StaticMesh' transform and all the aiming parts are placed under the '*TurretPivot*' transform, 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'.



```
Hierarchy
Create ▾  All
▼ Untitled*
    Main Camera
    Directional Light
  ▼ Tower_Turret_Cannon
    ▼ StaticMesh
        TowerBase2
    ▼ TurretPivot
        TowerTurret1
      ▼ BarrelPivot
        ▼ Gun1_L
            ShootPoint
      TargetPoint
```

```
Inspector   Navigation   Lighting
☑ Tower_Turret_Cannon   ☐ Static
Tag Untagged   Layer Tower
Prefab  Select  Revert  Apply
▶ Transform
▼ ☑ Unit Tower (Script)
  Editing tower using Inspector is not recommended.
  Please use the editor window instead
  Tower Editor Window
  ▶ Show default editor
▼ ☑ Sphere Collider
  Edit Collider
  Is Trigger  ☐
  Material    None (Physic Materi
  Center      X 0   Y 0.5   Z 0
  Radius      1
```

IMPORTANT: When configuring a tower prefab (or a creep prefab for that matter), it's recommended you do so via the TowerEditor; which can be accessed via the drop down menu. The reason for this is because editing units in a scene via the Inspector only alters that *instance* of the prefab; whereas, when you edit via the TowerEditor, it effectively alters the prefab itself and thus all future instances of that prefab, which keeps your cloned units consistent.

Creep Prefab

A working creep prefab can be as simple as an empty game object with the *UnitCreep.cs* script component attached to it. This will enable the creep to be spawned and move along the path as usual despite not being visible. Any mesh/model on the creep is optional.

To make the creep targetable in ability targeting phase, it requires a collider. To make sure the collider is not in conflict with other components, the layer of the creep game object will need to be set to 'Creep' (layer 27 by default). The size of the collider should be set to encapsulate only the mesh/model of the creep.

Also try avoid having rigidbody on the prefab unless you have to. Doing so might interfere with the movement code.

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

**A creep prefab draws many parallels to a tower prefab in terms of hierarchy arrangement. You can always refer to the tower section for an image reference.*

ShootObject Prefab

Shoot-objects are objects that are fired by turret-type towers or creeps during an attack to hit the target. A working shoot-object can be as simple as an empty game object with the *ShootObject.cs* script component attached to it. This will enable the shoot-object to be assigned to any turret-type unit and be fired when the unit is attacking despite the shoot-object not being visible. Any mesh/model/visual effect on the shoot-object is optional.

There are three types of shoot-objects and each serves a different purpose aesthetically.

Projectile A typical object that travels from the 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 is used to determine how long until the target is hit after the shoot-object is fired.

Effect For attacks that don’t require a shoot-object to travel from shoot-point to target. A timer is used to determine how long until the target is hit after the shoot-object is fired.

HOW THINGS WORK:

Item DB (DataBase)

TDTK uses a centralized database to store all information on prefabs (towers, creeps and shoot-objects) and in-game items (resource-types, abilities, perks, damage and armor types). The in-game item can be created with just a simple click of a button in their associated editor. The prefabs however, need to be create manually before they can be added to the database. Once added, they can be accessed and edited via the editor. Prefabs that aren't added to the database will not appear in the game.

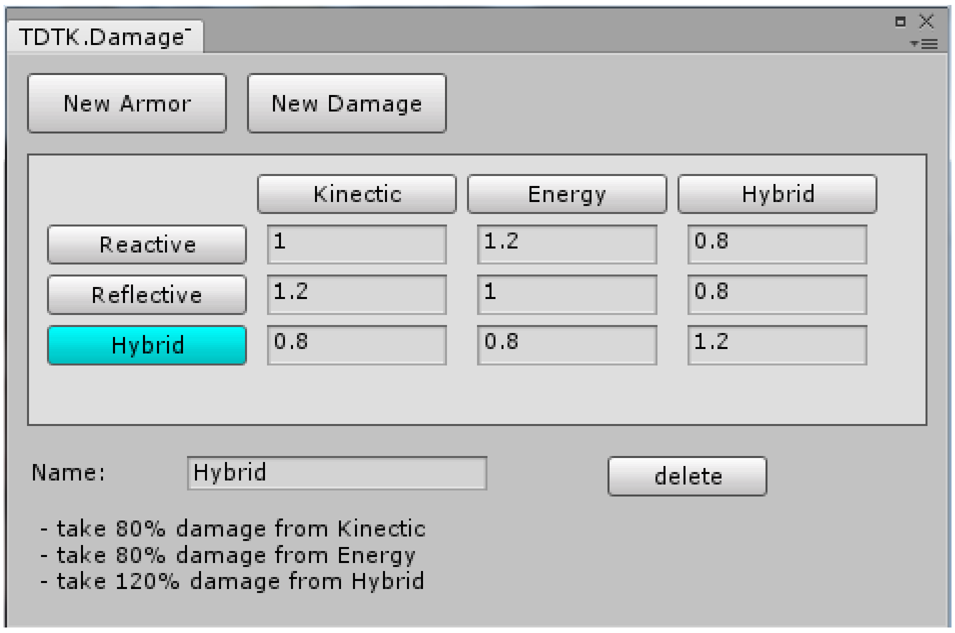
Tower prefabs can be individually enabled/disabled in TowerManager's Inspector. Disabled towers won't be available in the scene. However, a disabled tower can be added to the scene during runtime via PerkSystem. Tower prefabs can also be individually disabled in BuildPlatform. When a tower is disabled on a platform, players won't be able to build 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 the Perks system.

Perks can be individually enabled/disabled in PerkManager's Inspector. Disabled perks won't be available in the scene.

Damage Table

DamageTable is a multiplier table used to create a rock-paper-scissors dynamic between units. You can set up various damage and armor-types using DamageTableEditor (accessed from the drop down menu). Each damage-type can react differently to each armor-type (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.



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

Name: Hybrid

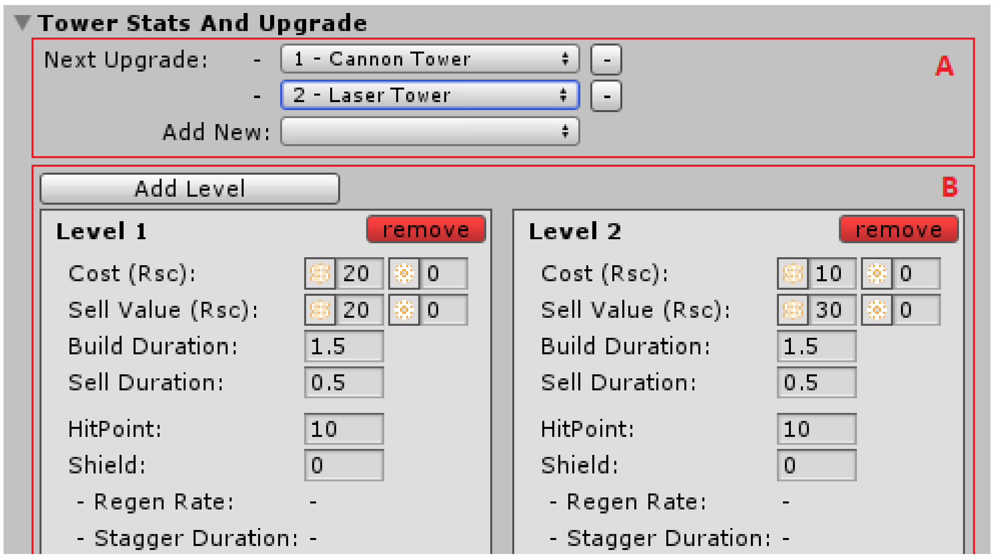
- take 80% damage from Kinectic
- take 80% damage from Energy
- take 120% damage from Hybrid

Upgrading System For Tower

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

In TowerEditor (> *Tower Stats and Upgrade)*, you will find you can add multiple levels 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 levels are for upgrades. When a tower undergoes a stat upgrade, it means they will use the new stat values from the next subsequent level.

You can also assign different tower prefabs as the '*Next Upgrade*' for any tower (as shown in image below, section-A). 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

A

Next Upgrade: - 1 - Cannon Tower -
- 2 - Laser Tower -
Add New:

B

Add Level

Level 1	remove		Level 2	remove	
Cost (Rsc):	20	0	Cost (Rsc):	10	0
Sell Value (Rsc):	20	0	Sell Value (Rsc):	30	0
Build Duration:	1.5		Build Duration:	1.5	
Sell Duration:	0.5		Sell Duration:	0.5	
HitPoint:	10		HitPoint:	10	
Shield:	0		Shield:	0	
- Regen Rate:	-		- Regen Rate:	-	
- Stagger Duration:	-		- Stagger Duration:	-	

The tower upgrade methods are not mutually exclusive. A tower can have both stat upgrades as well as next tower upgrades. However, stat upgrades always come first. That means a tower cannot be upgraded to the next tower unless it has already levelled-up through all stat upgrade levels listed in the Editor.

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

Example:
TowerA has 3 levels 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.

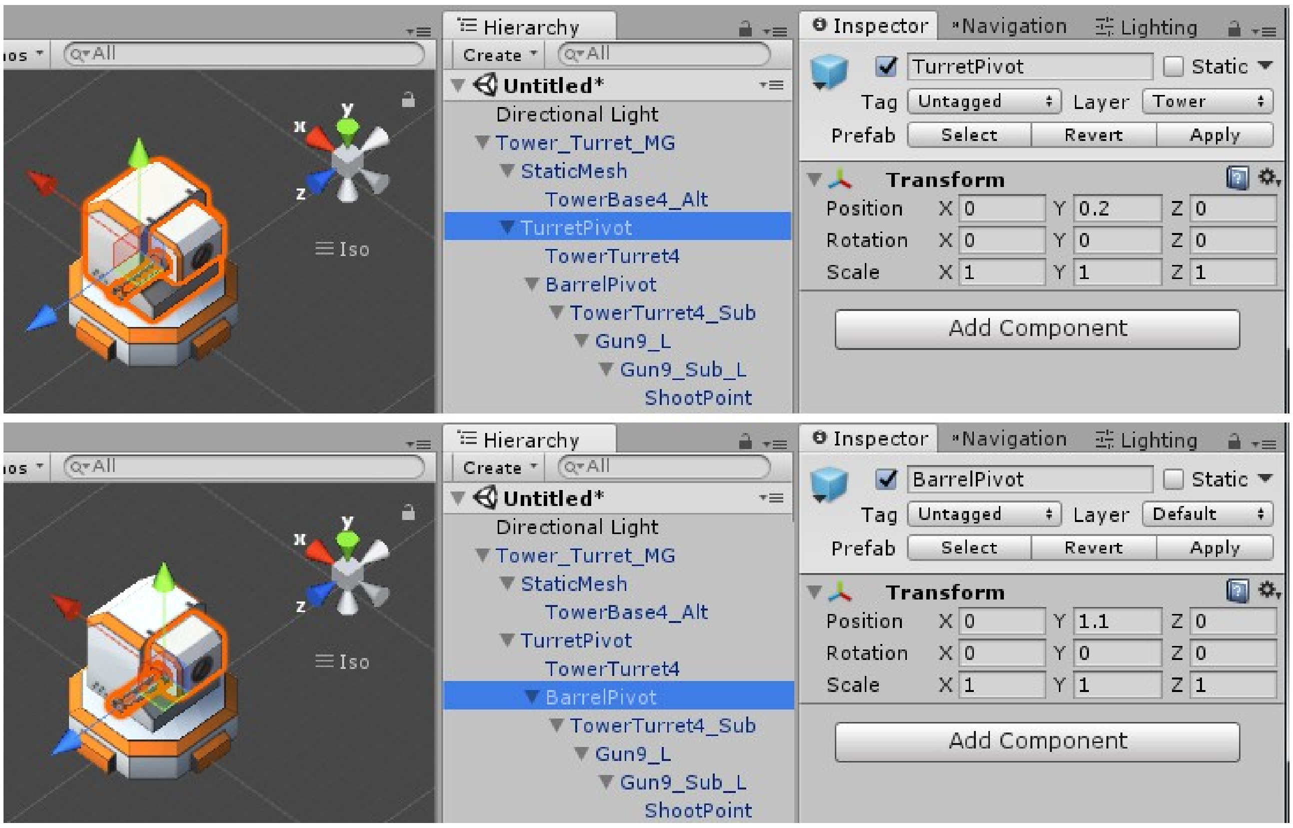
TowerC has 1 level of stats specified. TowerD and TowerE are assigned as 'Next Upgrade' towers. TowerC will never upgrade their base tower's stats but can either be upgraded to TowerD or TowerE.

Units (Turret) That Aim And Fires At Target

Turret-type units attack targets directly by firing shoot-objects at them. In many cases, 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 for 'Turret-Pivot' and (optionally) 'Barrel-Pivot' to face the current active target. These transforms can be assigned in TowerEditor (> *General Tower Setting*). The way the model works, 'Turret-Pivot' is the main pivot that can be rotated 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 toward the positive z-axis as shown in the image below. If you are not sure, please refer to the default tower prefab. You can also follow this step by step instruction at the end of this section. To understand why this is set up this way it's strongly recommended you go through this tutorial video to get the basic understanding of hierarchy and parent-child relationships.



Untitled*
Directional Light
Tower_Turret_MG
StaticMesh
TowerBase4_Alt
TurretPivot
TowerTurret4
BarrelPivot
TowerTurret4_Sub
Gun9_L
Gun9_Sub_L
ShootPoint

TurretPivot — Static
Tag: Untagged — Layer: Tower
Prefab: Select, Revert, Apply
Transform
Position X 0 Y 0.2 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Untitled*
Directional Light
Tower_Turret_MG
StaticMesh
TowerBase4_Alt
TurretPivot
TowerTurret4
BarrelPivot
TowerTurret4_Sub
Gun9_L
Gun9_Sub_L
ShootPoint

BarrelPivot — Static
Tag: Untagged — Layer: Default
Prefab: Select, Revert, Apply
Transform
Position X 0 Y 1.1 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

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

Step by step to set up basic unit with turret that aims in the right direction.

1. Create an empty game-object, add a Unit component (either '*UnitTower,cs*' or '*UnitCreep.cs*') on it. This game-object is the root object of this new prefab.
2. Drag your model into the empty game-object, set the position to (0, 0, 0). This will make sure the model is positioned at the root object center.
3. Now rotate the model's transform so that it faces the z-axis.
4. Expand the hierarchy, try to find the game-object which serves as the pivot for the turret. For reference, we'll call this *Turret*.
5. Create a new empty game-object, make sure the rotation is (0, 0, 0) and name it 'TurretPivot'
6. Drag TurretPivot into '*Turret*', set the position to (0, 0, 0). Now the TurretPivot should be in a similar position with *Turret*.
7. Reverse the child parent relationship between TurretPivot and *Turret*. Make TurretPivot the parent and *Turret* the child.
8. Now TurretPivot is ready to be set as '*Turret-Pivot*' of the unit in the editor.
9. You can repeat step-3 to step-7 to set up the barrel object, if the model has one.

Ability And Perk System

Abilities are actions that can be performed by the player during runtime to achieve various means like damaging creeps or buffing towers.

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

Abilities are entirely optional. You can disable all abilities in a level by simply disabling or deleting the AbilityManager game object.

Perk System

The Perk system is, for all intent and purpose, a customizable upgrade system. You can use it to create various upgrade systems for your game. It can be used as a simple linear upgrade track or a full-on branching tech tree.

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

Perks are entirely optional. You can disable all perks in a level by simple disabling or deleting the PerkManager game object.

Effect

Effects are buffs/debuffs that are applied to individual units. Effects can be applied through a normal unit attack or ability.

To add an attack or ability effect, 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 created, you can assign the effect to a unit/ability/perk using the appropriate editor. Here's a list of where you can assign them:

- *TowerEditor > Tower Stats and Upgrades > Level (X) > Effect On Hit*
- *CreepEditor > Unit Stats > Effect On Hit*
- *AbilityEditor > Ability Stats > Effect On Hit*
- *PerkEditor > Perk Effect Attribute > Effect On Hit*

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

Using Custom Model And Add Background Environment

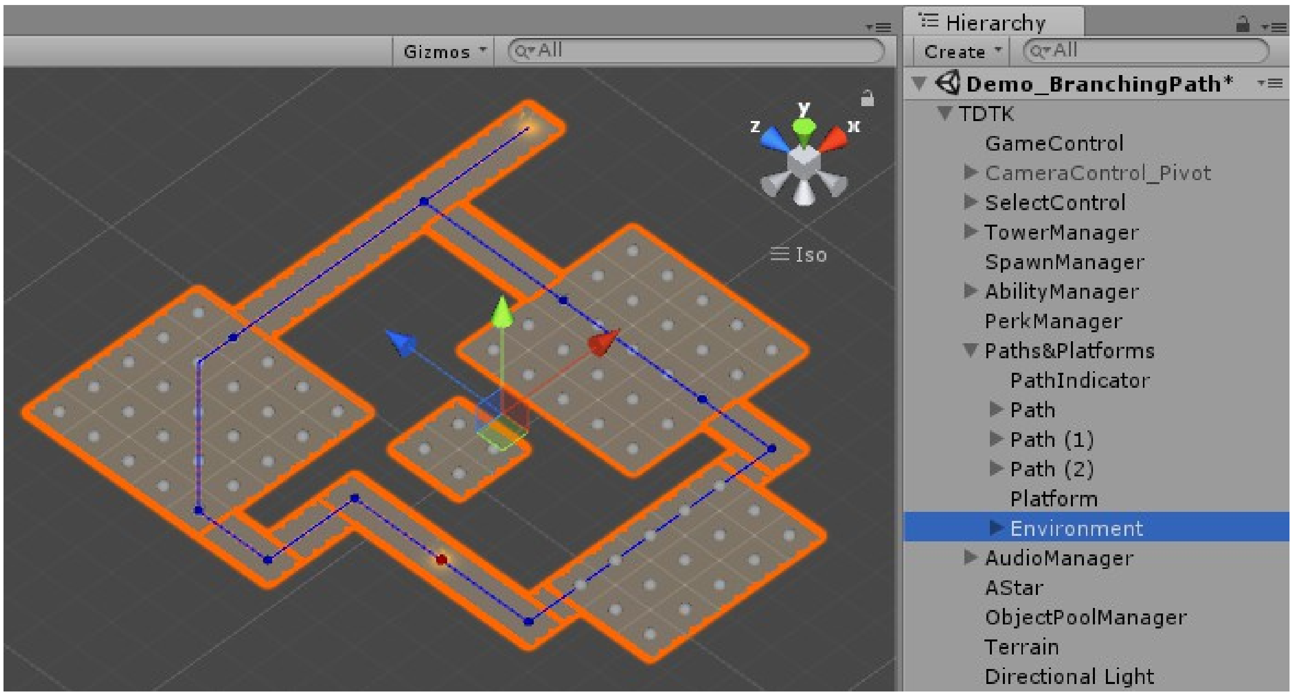
Custom Models

At its minimal form, TDTK doesn't require any visual element at all. All the components can run just fine without any visual elements. They are only necessary because without them the player wouldn't be able to see anything. All default prefabs were designed to be replaced with whatever models you need to use for your own game.

Background Environments

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 components. If you must have collider components in the background game object for whatever reason, you can set those game objects to layer-31 (Terrain). These will allow the code to recognise the collider as a background and not an obstacle for a walkable BuildPlatform.

You can refer to any of the demo scenes to see how this is used in practice. As shown in the image below, all highlighted game objects ('Environment' and its child objects) are there for visual reference. You can disable the 'Environment' game object if you want to remove all visual elements on the level except for the grid.



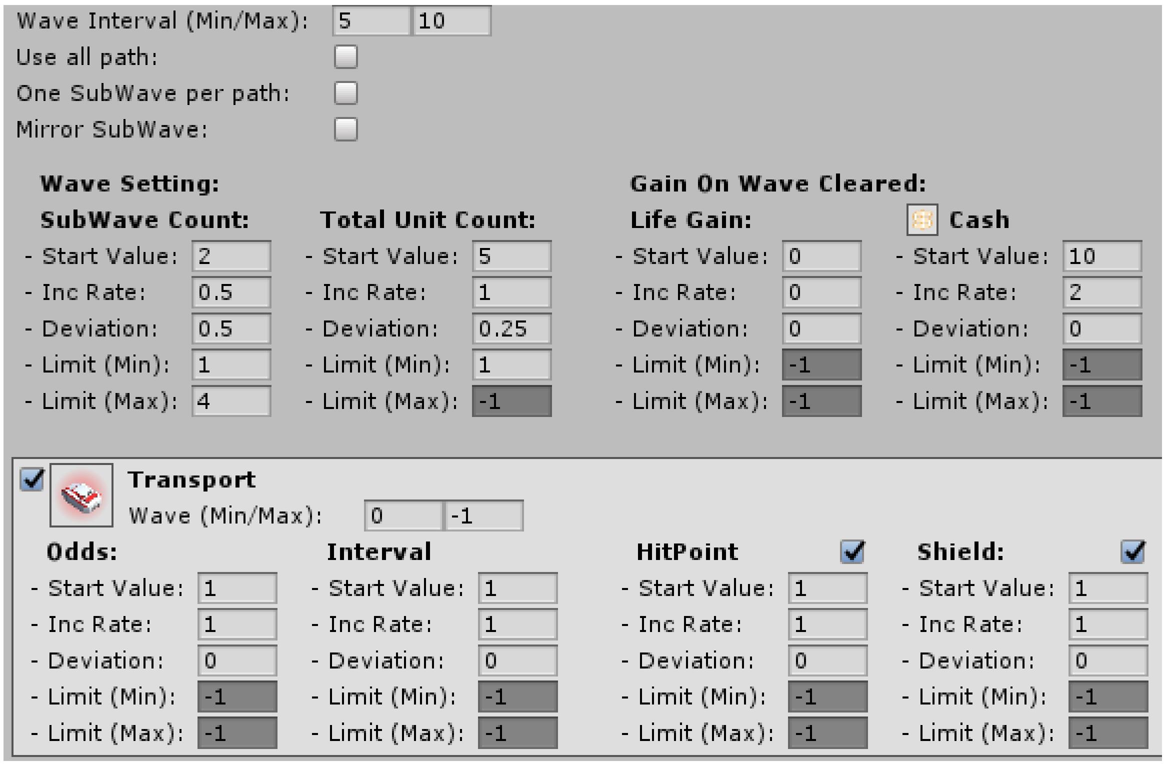
Endless Mode & Procedural Generated Spawn Info

TDTK supports procedural generation for the spawn information. It's mandatory for endless mode since there's no way to manually set up 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 attributes like sub-wave count within each wave, total unit count, how likely a particular prefab will appear, etc.. Each attribute is defined by a linear equation based on a set of values. These values are what you can configure to adjust the output of the generation. The calculation for each attribute goes like this:

*base value on wave-**N** = (**N** x **IncRate** + **StartValue**) * (1+(Random(-**Deviation**, **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:	☐	
One SubWave per path:	☐	
Mirror SubWave:	☐	

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

✓ **Transport**
Wave (Min/Max): 0 -1

Odds:		**Interval**		**HitPoint** ✓		**Shield:** ✓	
- Start Value:	1	- Start Value:	1	- Start Value:	1	- Start Value:	1
- Inc Rate:	1	- Inc Rate:	1	- Inc Rate:	1	- Inc Rate:	1
- Deviation:	0	- Deviation:	0	- Deviation:	0	- Deviation:	0
- Limit (Min):	-1	- Limit (Min):	-1	- Limit (Min):	-1	- Limit (Min):	-1
- Limit (Max):	-1	- Limit (Max):	-1	- Limit (Max):	-1	- Limit (Max):	-1

Persistent Progress

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

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

Similarly, for perk progress, you can check the '*Carry Over*' option on PerkManager. This works the same as the 'Carry Over' option in RscManager. However, with perks, any progress made at all is saved, no matter whether the level is completed successfully or not.

While most demos in this kit feature the Perk menu button enabled during normal gameplay, it's also possible to hide/disable the perk UI button during gameplay and instead run PerkManager in between battles as a perk menu-only scene for tracking persistent perk progress, creating an 'off-gameplay' level upgrade system. You can refer to the sample scene '*TDTK/Scenes_Demo/Example_PerkMenu*' to see how a PerkManager-only scene could work.

Important:
Persistent progress in this context doesn't mean 'saving the game'. The progress only lasts for a single game session. All progress is cleared as soon as you stop play-mode (in Unity Editor) or exit the game (in actual build). You will have to add your own code to actually save the game. That said, you can find some reference code that might be useful in '*TDTK/Scripts/Support_NotInUse/TDSave*'.

Other Things You Should Know

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

You can have pre-built towers in a scene by simply placing a tower prefab directly in the scene view. The tower will automatically be adjusted to the nearest tile on the nearest BuildPlatform when possible.

You can check the '*Hide In Inspector*' option for a tower/ability (in their respective Editors). 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 own custom ability effect and attach it to the '*SpawnOnActivate*' object to create your own custom ability.

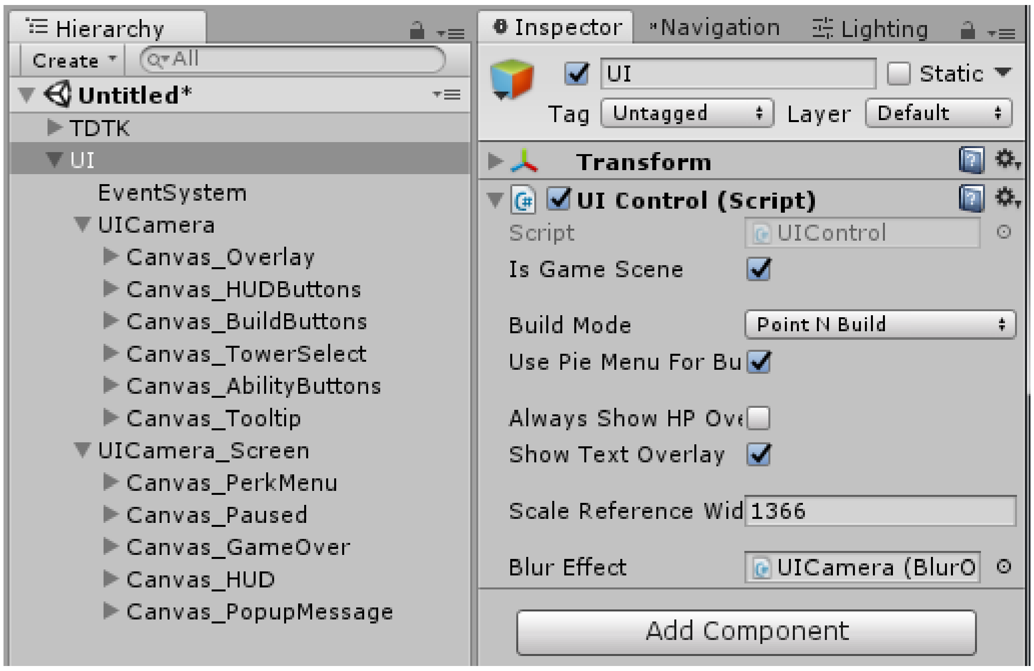
The New Scene from drop-down menu is created using prefabs located in '*TDTK/Resources/NewScenePrefab*'. You can edit those to change the default New Scene settings.

You can check the *'Flying Bypass'* option in AStar to have flying units ignore obstacles/towers on a walkable BuildPlatform.

ABOUT USER INTERFACE:

Configure The Default UI

The default UI is added to the scene every time you create a new scene via the drop-down menu. You will find all UI-related stuff placed under the ‘UI’ game object in the Hierarchy. Although there are quite a lot of scripts governing the UI logic, most of the important settings you’ll need can be accessed via the main control component called ‘UIControl’, attached to the ‘UI’ game object. The only exception being PerkMenu, which is talked about in a later section of this documentation.



Script: UIControl
Is Game Scene: ✓
Build Mode: Point N Build
Use Pie Menu For Bu: ✓
Always Show HP Ove:
Show Text Overlay: ✓
Scale Reference Wid: 1366
Blur Effect: UICamera (BlurO

Of course you can further customize the UI should you wish; however, it's strongly recommended you familiar yourself with the Unity UI system and how it works 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 provides a way to interact with every available mechanic in the framework. This also means it can be replaced with your own solution if you want. You can delete the ‘UI’ game object in any TDTK scene to get rid of the default UI.

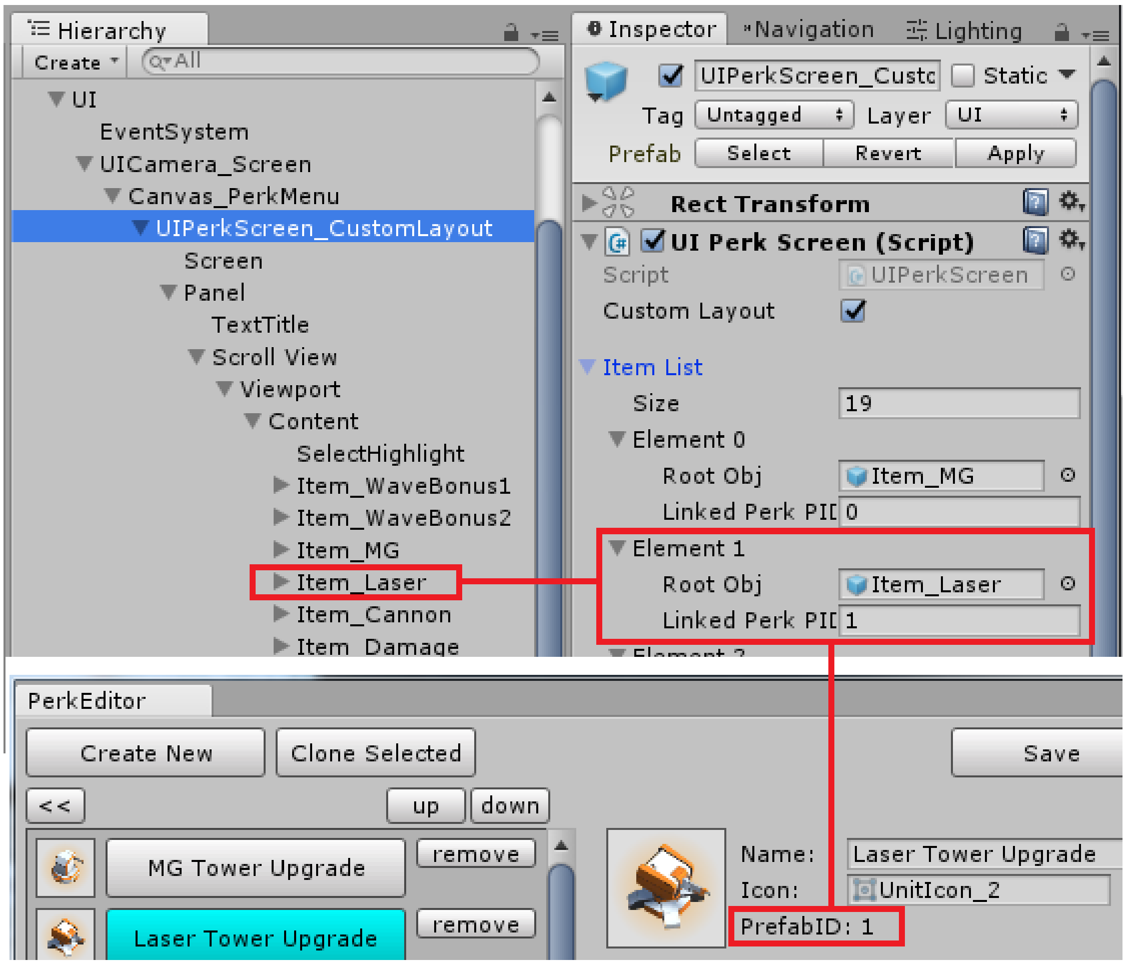
You can refer to the code for an example of interacting with the core component when coding your own UI solution. You will find all the relevant scripts in '*TDTK/Scripts/UI/*'. They are appropriately named so you can tell what each script is responsible for.

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 elements as well.

Perk Menu

By default, the perk menu will simply list all available perks from PerkManager. However, you can override that and arrange your own custom layout, just like the one shown in the demo scene (*TDTK/Scenes_Demo/Example_PerkMenu).*

To do that, first check the '*Custom Layout*' option in UIPerkScreen. Then you can add your own button(s) in the ScrollView and assign them to the '*Item List*' in UIPerkScreen. Note that each item will have a 'Linked Perk PID (Prefab ID)', the PrefabID of the perk the item corresponds to. You will need to manually match that to the PrefabID of the perk in PerkEditor, as shown as the image below. You can automate this process by using the '*GeneratePerkButtons*' button in UIPerkScreen.



Note that under each 'Item' game object, there are 'Connector' and 'ConnectorBase' objects. These are simply stencil line Images indicating the upgrade path from one perk to another (if there is one). The Connector objects must be child objects under the hierarchy of the perk item and be named '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 the demo scene to find out how all this setup comes together. In the demo, the ConnectorBase objects are the (grey) lines indicating the upgrade perk at the end of the path is unavailable, while the Connector objects are the (orange) lines indicating the next connected upgrade(s) is available. When an item is purchased, the ConnectorBase object will be disabled, revealing the Connector object behind it. For the player, it will seem like the line has turned from grey to orange, indicating that the link between two items has been activated and is thus available for purchase.

Perk Menu-Only Scene
As mentioned before, it's possible to have a perk menu-only scene for an 'off-gameplay' level upgrade system. To do that you will need to uncheck the 'Game Scene' option in the ‘UIControl’ component (attached to ‘UI’ game object) and remove any other UI elements that are not required. This will make sure the perk menu always stays on screen in that scene. Obviously, you will still need a PerkManager in the scene and possibly the RscManager. Please refer to the sample scene '*TDTK/Scenes_Demo/Example_PerkMenu'*. It's set up as a scene with no gameplay for the player to access the perk menu between scene with actual gameplay.

ABOUT URP:

How To Make TDTK URP Compatible

Universal Rendering Pipeline (URP) are not supported by default for two reasons. Materials and Cameras. But with some extra work, you should be able to use TDTK in URP.

The materials in TDTK use standard unity shaders which are not compatible with URP. To fix the shader, you can follow these steps:

1. Use the drop down menu - *Edit > Render Pipeline > Universal Render Pipeline > Upgrade Project Materials to Universal Materials*. This will cover most material in the project.

2. Look through the subfolders of *\Assets\TDTK\Modular_SciFi_Asset_Set* for Materials folders containing unconverted (purple) materials. Switch the shader of all these unconverted materials to "*Shader Graphs/BlendColorsOverlayTextureURP*".

3. For remaining objects that still doesn't render properly, you will need to manually update their material to the standard URP materials which can be found in *\Packages\Universal RP\Runtime\Materials*

Next, TDTK uses multiple cameras to render different things. Unfortunately URP doesn't support multiple cameras by default. Instead, a camera stack order has to be set up manually for this to work. Following are the steps to do that:

1. Find all the cameras in the scene. This can be done by typing "*t:Camera*" into the Hierarchy's search bar.

2. For every camera except the Main Camera, set its *Render Type* property to *Overlay*.

3. On the Main Camera, add all other cameras to its *Stack List* property. The stack order should be as follow:
   - MainCamera (base)
   - OverlayCamera
   - UICamera
   - UICamera_Screen

The last steps for material fix and the camera fix needs to be repeat everytime you create a new scene via the drop down menu. If you want to fix that once and for all, just apply the changes to the 'TDTK' prefab in \Asset\TDTK\Resources\NewScenePrefab\TDTK

Huge thanks to ***Mason Wheeler of Stormhunter Studios* for the informing me of the material fix and providing the *BlendColorsOverlayTextureURP shader.*

THANK-YOU NOTE & CONTACT INFO

Thanks for purchasing and using TDTK. I hope you enjoy your purchase. If you have any feedback 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 the time to leave a review on the AssetStore page. Once again, thank you!