TowerDefenseToolKit

Documentation & API Reference for Unity3D

Version: 2.1
Author: K.Song tan
LastUpdated: 25th July 2012
Forum: http://goo.gl/ZX4OA
AssetStore: http://goo.gl/8vAFU

Thanks for using TDTK. This toolkit is a collection of coding framework in C# for Unity3D. This toolkit is designed to cover most of the common tower defense ("TD") game mechanics. Bear in mind TDTK is not a framework to create a complete game by itself, but rather, the framework tries to cover most (if not all) of the TD gameplay mechanics. The framework does not cover other game aspects such as menu scenes, options, etc.

The toolkit is designed with the integration of custom assets in mind. Although there are some minimal amount of exemplary assets included with the toolkit for demonstration, users are expected to integrate their own audio and art assets.

TDTK should be compatible and performance friendly with all of the platforms supported by Unity3D. Please note that it's not tested on Android.

For a series of quick video tutorial, please visit following link:
http://www.youtube.com/watch?v=xI-z6DCOt0w
http://www.youtube.com/watch?v=tVcfI7Km2tY
http://www.youtube.com/watch?v=39libqKnDvs

For previous version's (1.x) user
TDTK2 is not an update from TDTK1.x. It's a totally new and redesigned framework. Therefore it's not backward compatible. There are some key differences; e.g. the entire build configuration has been placed under one component, "Tag" is no longer used, etc. However you will find that all the features from previous version of TDTK are still available. I'm fairly confident that you'll find that this version is better than any previous version.

TABLE OF CONTENTS

OVERVIEW OF THE FRAMEWORK

Essential Components

There are three main essential components comprised of three different scripts to govern the game logic, every scene should have one and only one instance of these components.

- **GameControl** (GameControl.cs) controls game rules and basic mechanics and contains two sub-components.
    - **LayerManager** (LayerManager.cs) is used to customize the layers used in the game.
    - **ResourceManager** (ResourceManager.cs) is used to configure the resources used in the scene.
- **SpawnManger** (SpawnManager.cs): Responsible to spawn and keeping track of creeps.
- **BuildManager** (BuildManager.cs): Responsible for building and deployment of towers.
- Units are the interactive objects consisting of both creep and tower in game.
    - **UnitTower** (UnitTower.cs) is the controller for a single tower.
    - **UnitCreep** (UnitCreep.cs) is the controller for a single creep.
- **PathTD** (PathTD.cs) are each an instance of the route used by your creeps. There can be multiple paths in a scene.
- **Platforms** (Platform.cs) are objects which the towers can be built upon. There can be multiple platforms in a scene.
- **ShootObject** (ShootObject.cs) are the objects emitted by towers to give visual representation to creep-damaging objects (e.g. missiles, cannon-balls, etc.).

Optional Components

The following components are optional. You can easily replace them with your own implementation without affecting the base gameplay in any way.

- **CameraControl** (CameraControl.cs) provides basic camera control for a top-down perspective. Also works on mobile devices.
- **AudioManager** (AudioManager.cs) governs all of the audio, although not necessary, most essential components used AudioManager. Always include AudioManager.cs for any TDTK2 projects.
- **MiniMap** (MiniMap.cs) shows a tactical mini-map. Use this to track whatever object you need.
- **PathIndicator** (PathIndicator.cs) is used with Path to provide a visual cue to it.
- **UnitCreepAttack** (UnitCreepAttack.cs) is used in adjunct with UnitCreep.cs to give the creep the ability to attack.

Support Utilities

These are the engines that support the game mechanic. They required minimal configuration (if any) in most cases. You can even ignore them entirely without manually adding them to a scene. Bear in mind, however, that they are absolutely vital and should always be included for any

TDTK2 project.

- **NodeGenerator** (NodeGenerator.cs) generates navigational nodes on a platform.
- **PathFinder** (PathFinder.cs) finds a route through walkable platforms where navigational nodes have been generated.
- **ObjectPoolManager** (ObjectPoolManager) is a GameObject recycling engine. It is used to greatly improve performance when many GameObjects are loaded into a scene.

UI & UI Utilities

User interface components are completely standalone components in TDTK2. There are two mutually exclusive default UIs for the example scene. Users can build custom UIs from these examples, given that the entire API is present and documented.

- **UI** (UI.cs) is the default UI for non-mobile platforms; uses OnGUI().
- **UIiOS** (UIiOS.cs) is the default UI for mobile platforms, but can be used anywhere.
- **UIRect** (UIRect.cs) is a utility to prevent a click/touch on the UI element from triggering in-game interaction such as selection of a build point or tower.
- **CustomButtoniOS** (CustomButtoniOS.cs) is a custom button class using GUITexture GameObject. As used in UiiOS.

Other Components

These are the components that need no configuration. Some of them are vital components but you never need to concern yourself with these, except perhaps DebugShowSelf.cs, which is useful when positioning empty object such as a waypoint or shootPoints.

- **Unit** (Unit.cs) is the base class of the creep and tower unit.
- **UnitUtility** (UnitUtility.cs) contains utility methods for units.
- **OverlayManager** (OverlayManager.cs) controls the build/upgrade overlay bar.
- **GameMessage** (GameMessage.cs) is a utility component used to display a message on the screen which is intended for debugging on mobile devices. A GUIText can be assigned to it or it will be assigned automatically.
- **DebugShowSelf** (DebugShowSelf.cs) allows empty GameObjects to appear visible in scene-view and game-view when selected.

Example

There are 4 example scenes in the toolkit to show case how the toolkit can be used to put together different kind of levels

ESSENTIAL COMPONENTS

These are the key components of the TD gameplay. Without either of this the game wont function properly.

GameControl (GameControl.cs)

GameControl is the controlling component of the logic and state of the game. It initiates the game, determines when the game is won or lost, etc. A scene must always contain just one GameControl. GameControl by default will require sub-component LayerManager and ResourceManager.

CONFIGURABLE

Property	Description
Player Resource	The amount of resources the player has at the start of the game (i.e. the currency to build towers).
Player Life	The amount of life the player has at the start of the game.
Sell Tower Refund Ratio	The ratio of the total tower value that the player will receive when they sell a tower. The value takes into account the cost to build the tower as well as the resources spent to upgrade it.
SpawnManager	The SpawnManager component in this game scene. Refer to section X for more info about SpawnManger.
Range Indicator H	The range indicator object to be placed on offensive towers (turret and aoe) when they are selected. This is applicable to tower with targetArea set to AllAround.
Range Indicator Cone H	The range indicator object to be placed on offensive towers (turret and aoe) when they are selected. This is applicable to tower with targetArea set to FOV or StraightLine
Range Indicator F	The range indicator object to be placed on passive towers (support and resource) when they are selected.
Building Bar Width Modifier	The modifier to the width of the build bar when a tower is being built.
Building Bar Height Modifier	The modifier to the Height of the build bar when a tower is being built.
Building Bar Pos Offset	The position offset of the build bar when a tower is being built from the center of the tower.

METHODS

Method Signatures & Descriptions

GameControl.GetPlayerLife

```
public static int GetPlayerLife()
```

Returns current value of player's life.

GameControl.Select

```
public static UnitTower Select(Vector3 pointer)
```

Selects a tower based on a provided screen position and returns said tower if it exists.

GameControl.ClearSelection

```
public static void ClearSelection()
```

Clears the selection of any selected towers.

GameControl.GainResource

```
public static void GainResource(int[] value)
```

*Adds an array of values to the current resources. The **first element in the array** corresponds to resource **type1; the** second element **corresponds** to resource type2, and so on. If the array length exceeds the current available resource types list, the value will be ignored.*

```
public static void GainResource(int value)
```

Adds the** value **to the first or default** resource type. **Negative values will remove resources.

```
public static void GainResource(int id, int value)
```

***Adds the value to** a specific resource type.*

Note: Negative values will remove the value provided from the resource(s).

GameControl.GetResourceVal

```
public static int GetResourceVal()
```

Returns the resource value of the default (first) resource type.

```
public static int GetResourceVal(int id)
```

Returns the resource value of the resource type specified by the provided identifier.

GameControl.GetAllResourceVal

`public static int[] GetAllResourceVal()`

Returns an array of integers containing current value of each defined resource type.

GameControl.SpendResource

`public static int SpendResource(int[] value)`

***Subtracts** an array of values **from** the current resources. The **first element in the array** corresponds to resource **type1; the** second element **corresponds** to resource type2, and so on. If the array length exceeds the current available resource types list, the value will be ignored.*

GameControl.HaveSufficientResource

`public static bool HaveSufficientResource(int[] cost)`

Check if there is sufficient resource for the cost provided. The first element of the cost array will be checked against the first resource; the second element of the cost array will be checked against the second resource, and so on. Returns true if there are sufficient resources; otherwise false.

GameControl.GetResourceList

`public static Resource[] GetResourceList()`

Return an array of all the Resource objects in ResourceManager. See the ResourceManager section for more information about Resource.

GameControl.GetSellTowerRefundRatio

`public static float GetSellTowerRefundRatio()`

Returns the towerRefundRatio property.

EVENTS

Event Signatures & Descriptions

GameControl.onGameOverE

`public static event Action<bool> onGameOverE`

fired when the game over condition is satisfied. True is passed if player have won, else if player has lost.

GameControl.onResourceE

`public static event Action ResourceE`

fired when the resource's value has changed.

GameControl.onLifeE

`public static event Action onLifeE`

fired when the player's life value has changed

ResourceManager (ResourceManager.cs)

Resource Manager is used to configure the resource used in the game. It's by default required by GameControl so manual assignment of the script is not needed.

CONFIGURABLE

Property	Description
Total Resource Type	The number of resource that will be used in the game.
Resource*	
- Icon	The icon of the resource. To be used for UI.
- Name	The name of the resource.
- Value	The starting value of the resource

** Custom class contains all the setting for resource.*

LayerManager (LayerManager.cs)

Layer Manager is just a utility to configure layer that will use for TDTK. You can assign any layer you setup in unity layer editor to work in TDTK. The layer assigned here shouldn't be used in other purpose.

Layer Manager is by default required by GameControl. You don't need to manually assign it

CONFIGURABLE

Property	Description
Creep Layer	The layer used for creep, default layer is 31.
CreepF Layer	The layer used for flying creep, default layer is 30.
Tower Layer	The layer used for tower, default layer is 29.
Platform Layer	The layer used for platform, default layer is 28.
Overlay Layer	The layer used for overlay, default layer is 25.

SpawnManager (SpawnManager.cs)

SpawnManger is the component responsible for spawning and tracking of creep in the game. A scene must have one and only one SpawnManager component. That particular SpawnManager component needs to be assigned to GameControl in order to be initiated and run properly.

SpawnManager can be configured with Inspector. But it's much more effective and easier to configure it using TDTK SpawnEditor. The editor can be open by accessing the top panel TDTK->SpawnEditor.

CONFIGURABLE

Property	Description
SpawnMode	Specify spawn mode to be used in the scene. The SpawnMode supported are listed as follow: • **Continuous**: A new wave is spawn upon every wave duration countdown. • **WaveCleared**: A new wave is spawned when the current wave is cleared. • **ContinuousWithSkip**: Similar to Continuous but user can initiate the new next wave before the timer runs out. • **WaveClearedWithSkip**: Similar to WaveCleared but user can initiate the next wave before clearing current wave. • **RoundBased**: each wave is treated like a round. a new wave can only take place when the previous wave is cleared. Each round require initiation from user.
Default Path	The primary path to be used. This path can be overridden by other alternate path for each individual spawn.
Wave Size	The number of wave in this scene.

Wave

Number of SubWave	A single wave of spawn can consist of minimum one and up to many subWave. Each subWave can have each creep, spawn count, spawn timing and path. This parameter specify how many subWave are there in the corresponding wave.
SubWave	A wave is made up from at least 2 or more subWave. A subWave is responsible for spawning a particular creep prefab. Each subWave has it's own unique spawn count, spawn timing and path that is independent from each other. More details on SubWave setting can be found at the bottom part of this list.
Time Before Next	The duration for this wave until next wave is commenced. The next

Wave	wave will be spawned anyway when this time is out even though not all the subWave has been spawned. This is only valid if continuous spawn mode is selected. The number beside the field is the time required for the all the unit in the wave to spawned.
Resource Gain Upon Wave Clear	Resources gain for the player when all the subWave in this wave is cleared. Support multiple resource, SpawnEditor will auto detect the setting in ResourceManager and change the configuration setting.

SubWave	
Unit Prefab	The prefab of the Creep for the subWave. Any prefab assigned must contains UnitCreep.cs. The subWave will be forfeit if this is left empty.
Number of Unit	Number of creep to be spawned for this subWave.
Spawn Interval	Delay in seconds between the spawning of each creep.
Pre-Spawn Delay	Delay in seconds before the subWave started to spawn.
Alternate Path	The path to take for the creep spawned in this subWave. if left empty, the default path will be taken instead.
Override Shield	The Shield value of the unit when deployed. if the value is larger than 0, the value will override the default shield value set on the prefab. Else the default value will be used. This enable using similar creep prefab for stats.
Override HP	The HP value of the unit when deployed. if the value is larger than 0, the value will override the default HP value set on the prefab. Else the default value will be used. This enable using similar creep prefab for stats.
Override Speed	The moveSpeed of the unit when deployed. if the value is larger than 0, the value will override the default moveSpeed set on the prefab. Else the default value will be used. This enable using similar creep prefab for stats.

METHODS

Method Signatures & Descriptions

SpawnManager.IsClearForSpawning

```
public static bool IsClearForSpawning()
```

called to check if spawning for next wave is ready.

SpawnManager.Spawn

```
public static bool Spawn()
```

called to spawn the next wave. Return true if success, false if else.

SpawnManager.GetCurrentWave

```
public static int GetCurrentWave()
```

return the current wave being spawned.

SpawnManager.GetTotalWave

```
public static int GetTotalWave()
```

return the total number of wave in current scene.

SpawnManager.TimeNextSpawn

```
public static float TimeNextSpawn()
```

return the duration before the spawning of next wave.

SpawnManager.GetSpawnMode

```
public static _SpawnMode GetSpawnMode()
```

return the SpawnMode selected in current scene.

EVENTS

Event Signatures & Descriptions

SpawnManager.onWaveStartSpawnE

```
public static event Action<int> onWaveStartSpawnE
```

fired when a new wave start spawning.

SpawnManager.onWaveSpawnedE

```
public static event Action<int> onWaveSpawnedE
```

fired when a waves has finished spawning.

SpawnManager.onWaveClearedE

```
public static event Action<int> onWaveClearedE
```

fired when a waves has been cleared.

SpawnManager.onClearForSpawningE

```
public static event Action<bool> onClearForSpawningE
```

fired when a new waves can be spawned. This is subject to the spawn rule of the SpawnMode selected. For instance, this event will fire when a wave has finished spawning in skippable mode but not in non-skippable mode

---

BuildManager (BuildManager.cs)

A component used to configure build-able tower and other relavent build information. This component also control all the logic of tower building. Every scene must contain one and only one BuildManager.

CONFIGURABLE

Property	Description
Towers	A list of towerPrefab that is buildable in this scene. These are the tower prefabs that is configurable in TowerEditor when this scene is currently loaded.
Grid Size	The gridSize of the building grid in this scene. Limited to 3.
Platforms	A list of the platforms in which the towers can be built in this scene. For what constitute a platform, please refer to section Platform. Every platform assigned will be automatically resized to fit the gridSize, if the AutoAdjustTexutreToGrid option is checked. Platform must be a unity primitive plane object.
Auto Adjust Texture to Grid	Check to re-arrange the material texture tiles and offset value in all the platform to git the grid. In order for this option to work properly, the material texture has to fit the format of the default texture used in the example platform (A cross that evenly distribute the image into 4 quarter).
Enable Tile Indicator	Check to enable highlighting of the tile being selected or hovered over by cursor.

METHODS

Method Signatures & Descriptions

BuildManager.CheckBuildPoint

```
public static bool CheckBuildPoint(Vector3 screenPosition)
```

Use the passed position as screen position to determine which grid or platform is being point toward to and round the corresponding position to the center of the tile. Return true if a tower can be built on that position, false if otherwise.

```
public static bool CheckBuildPoint(Vector3 screenPosition, _TowerType type)
```

a specific tower type is passed so just the tower type is checked. Return true if that particular tower type can be build on the position. Look in Section Tower for more info about TowerType.

BuildManager.BuildTowerPointNBuild

public `static bool BuildTowerPointNBuild(UnitTower tower)`

call to build a tower based on the tower being passed on the position last determined when BuildManager.CheckBuildPoint() is called. This method will determine if all the criteria such as resource is full-filled before building the tower. Return true if the build operation is successful, false if otherwise. This method is typically being called after BuildManager.CheckBuildPoint() and the tower intend to be build obtained from user input.

BuildManager.BuildTowerDragNDrop

`public static bool BuildTowerDragNDrop(UnitTower tower)`

create a new drag and drop tower object for the tower prefab passed. Similar to BuildManager.BuildTower() but user will be able to drag the tower to a specific spot before finalize the building position. No additional code is required to perform the drag and drop mechanism.

BuildManager.GetTowerList

`public static List<UnitTower> GetTowerList()`

return a .net List contains all the tower prefab being assigned to the BuildManager.

BuildManager.GetBuildInfo

`public static BuildableInfo GetBuildInfo()`

return the current build information based on the last selected position. Return null if there isnt a valid build point.

BuildManager.GetGridSize

`public static float GetGridSize()`

return the gridSize for this scene specified in BuildManager.

BuildManager.GetTowerCount

`public static int GetTowerCount()`

return the total number of towers in the scene.

BuildManager.InitiateSampleTower

`public static void InitiateSampleTower()`

initialize all the prefabs and gameObject needed for tower preview during build phase. This is

typically being call only once, when the scene is being loaded, in Start().

BuildManager.ShowSampleTower

```
public static void ShowSampleTower(int ID)
```

show a preview of the tower intend to be build on current designated build point. The ID passed being the tower ID of the tower to be previewed in the BuildManager towers list. BuildManager.InitiateSampleTower() needs to be called be first in order for this function to work properly.

BuildManager.ClearSampleTower

```
public static void ClearSampleTower()
```

clear the preview tower on screen. If there's any. BuildManager.InitiateSampleTower() needs to be called be first in order for this function to work properly.

Platform (Platform.cs)

Platform is the component which all tower will be built on. To enable tower to be built on a platform, it has to be assigned to BuildManager. All platform must be in the format of a Plane primitive. A Platform can only be rotated in y-axis. Any rotation in other axis will be automatically reset by the BuildManager. Platform.cs may not be necessary need for a platform unless there are special restriction of which tower type cannot be build on that platform. By default, all tower Listed in BuildManager is buildable on any particular tower.

Platform with tiles larger than 1x1 can also be used as a waypoint in which the platform will become a small field for the creep to be navigate through.

CONFIGURABLE

Property	Description
Buildable Type	A list of the tower type. Tower type which are specified on the list are the tower type supported by this platform.
Special Buildable ID	This variable is not in used.
Gizmo Show Nodes	For debug purpose, check to show all nodes on the platform. Only valid if the platform is walkable.
Gizmo Show Path	For debug purpose, check to show the active path through the platform. Only valid if the platform is walk-able.

PathTD (PathTD.cs)

Use to create a path for the creeps. It needs to be assigned to the SpawnManager. There can be multiple Path in one scene. Path can share waypoints.

CONFIGURABLE

Property	Description
Waypoints	An array of transform used to indicate the waypoint in the game environment. These series of objects will form a path for the creep. A tower-platform can be insert into this list. In which case the platform will become walk-able. Navigational node will automatically be generated on the platform and pathFinder will be used to search a path through the platform. *Note: Current version doesn't support multiple node connection between platform. That means when two platform are adjacent to each other in a path. A bridge which based on nearest connection between navigational node between both platform will be assign as the link. There can only be one bridge even though there are multiple nearest connection of equal distance. To ensure the creep moves in a path desired, it's recommended to add a waypoint in between both platform. Also for obvious reason, it's not recommended to have two platform overlapped.*
Height Offset On Platform	Only valid if there a platform in the path. The height Offset of the navigational node when generated on the platform.
Dynamic WP	Short for dynamic waypoint. Whenever the value is larger than 0, creeps that move along this path will not move exactly from waypoint to waypoint but rather, a small deviation is used. The value indicate the range of the deviation from the waypoint. Be advise that if you have platform along the path, this value should not be larger than half of the gridSize. Also please note that dynamic waypoint doesn't work very well with slope. Try minimise the usage of slope or avoid using slope with steep angle.
Generate Path Object	Check to auto generate visual object for the path.
Show Gizmo	For debugging, check to draw a line along the path.

Tower (UnitTower.cs)

Tower are the base building unit in a TD game. They can only be built on a platform. There are 7 general type of towers.

TOWER-TYPE

Type	Description
TurretTower	Typical tower, fire shootObject at creep.
AOETower	This tower doesn't shoot anything at a particular target, it return it applies damage and various effect directly to all the creep within range. It uses cooldown just like turret tower.
DirectionalAOETower	This is a hybrid between TurretTower and AOETower. It targets and aims like a TurretTower. But instead of using shootObject it damage the target instantly. It also affect all other creep unit within a conical area project from the tower to the target. The effective angle of the conical area can be adjusted. An example of the usage of this tower would be flamethrower or tower which damage all target within the line of fire.
SupportTower	This tower applies buff effect to all the friendly tower within range. Either by increasing their damage output or regenerate their HP.
ResourceTower	This tower generate resource upon every cooldown. To avoid exploitation, it can only be built after the creep has start spawning. It too, will only function when the game is in progress.
Mine/Trap	Works just like the name sounded. When build along a walkable platform, it doesn't block the path. But any creep that move pass it will trigger it. Support all effect available to TurretTower.
Block	This tower has no function, just an obstacle or a dummy target for creeps to attack.

A tower prefab can be as simple as a single gameObject or with full tower and animated turret. For a basic turret, simply add the script component UnitTower.cs to a gameObject. To add an animated turret though, the turret object needs to be a child transform of the turret object. The turret will need to be assign to the UnitTower.cs. The object that is assigned to be the turret will automatically aims at the target if the tower is a TurretTower and DirectionAOETower (turret aiming is configurable). Any other object that needs to be animated along the turret should be assign as child transform of the turret transform.

Towers in TDTK support changing of tower model upon upgrade. Should any object in the hierarchy of the tower needs to be replace upon upgrade, they should be assigned as either turretObject or baseObject. The objects in active level will be replaced by the object assigned in next level upon upgrade. This is only

applicable if the next object in the next level is assigned.

After a tower prefab is made. It need to be assigned to the BuildManager in a particular scene to become available in the scene. To configure a tower prefab using TowerEditor, the tower prefab also has to be assigned to the scene in the editor

It's advisable to configure the tower prefab various setting using the TowerEditor instead of default inspector.

TOWER-EDITOR

TowerEditor can be launched from the top panel by selecting TDTK->TowerEditor. Everytime tower editor window is launched, the editor will look for the BuildManager in the scene. All the tower prefab assigned in the BuildManager will appear in the editor to be edit.

Please note that not all property listed below are editable for every tower type. Property that are not related or not applicable to the tower type wont be visible on the editor window.

FindBuildManager Button: Unfortunately the editor doesn't auto update the BuildManager upon a scene change. This button will re-update the current BuildManager in the scene.

Tower: A list of tower prefab assigned to the BuildManger in current scene. Select a tower from this list to configure it.

Property	Description
Level Cap	A list of the tower type. Tower type which are specified on the list are the tower type supported by this platform.
Tower Name	Name of the tower that will appear in the game.
Tower Type	The type of tower prefab. Refer to TowerType.
Targeting Mode	The target type of the tower. Only Applicable for TurretTower, DirectionalAOETower and AOETower: • **Air**: tower attack on air unit. • **Ground**: tower attack only ground unit. • **Hybrid**: tower attack both ground and air unit.
Targeting Area	The active targeting area. Only target wihtin certain area will potentilally be targeted. Only applicable to TurretTower and DirectionalAOETower • **AllAround**: targeting all area are within range from the tower • **DirectionalCone**: target a conical area in targeting direction within range from the tower. • **StraightLine**: targeting based on Line-of-sight in

one straight line using targeting Direction specified. Targeting priority will be disregard.

Targeting Direction

The targeting direction in angle of DirectionalCone and StraightLine targeting Area. Value from 0-360, start from positive x-axis in counter-clockwise direction.

Targeting FOV

The angle of the conical area if the targeting area used is DirectionlCone

Targeting Priority

The targeting priority of the tower. Only Applicable for TurretTower, DirectionalAOETower and AOETower:

- **Nearest**: tower attack on air unit.
- **Touchest**: tower attack only ground unit.
- **Weakest**: Select target from all possible targets based on the target current
- **Random**: Randomly select a target from all possible targets.

Turret Animation Mode

How the turret object will be animated. Only applicable for TurretTower and DirectionalAOETower.

- **Full**: full animation, turret will look towards the target as well as taking into account of projectile curve
- **Y-Axis Only**: only rotate the turret to face the target in Y-axis, no elevation.
- **None**: Don't animate the turret.

Turret Rotation Mode

AOE Cone Angle

Valid for DirectionalAoe Tower only. This is the angle of the effective conical area cover by the tower centred from the main target. Any creep object within this conical area will be affected.

Destroy UponTriggered

Valid for mine/trap only. When checked, the mine/trap will be destroy when triggered. If not, it will wait for the specified cooldown duration before become active again.

HP Attributes

- Full HP

The maximum HP of the unit creep.

- Full Shield

The maximum Shield of the unit creep. Shield will regenerate over time at the rate specified in “Shield Recharge” until the shield value is at maximum. However the recharging process will pause for a duration specified in “Shield Stagger” whenever the creep is hit. Shield is optional and can be disabled by setting the Full Shield value to 0.

- Shield Recharge	The recharge value of shield per seconds.
- Shield Stagger	The time in second taken for shield to start recharging after the unit has been hit.
- Overlay HP	The object of the hitpoint overlay. This object has to be a plane primitive.
- Overlay Shield	The object of the shield overlay. This object has to be a plane primitive.
- Overlay Base	The object of the base overlay which will appear as a base layer for both shield and HP overlay. This object has to be a plane primitive.
- Always Show Overlay	Check to show overlay HP all the time. Uncheck this for better performance.

Tower Animation

Build Animation Body	Animation component that play the build animation
Build Animation	Animation clip for building
Fire Animation Body	Animation component that play the fire animation
Fire Animation	Animation clip for firing. This is not applicable on support tower and mine
Fire Animation Base Body	Animation component that play the fire animation for tower base in case when TowerBase has additional animation.
Fire Animation Base	Animation clip for firing for tower base in case when TowerBase has additional animation. This is not applicable on support tower and mine
Tower SFX	A list of sfx corresponded to this tower
Shoot Sound	sfx to play when the tower shoots
Building Sound	sfx to play when tower building starts
Built Sound	sfx to play when tower building is done
Sold Sound	sfx to play when the tower is sold

TowerStat

Cost	The cost to build the tower (for level 1) or upgrade to current level (for subsequent level). Support Multiple resource type. TowerEditor will auto detect the setting in ResourceManager and change the configurable parameters appropriately.
Cooldown	The duration in second between each attack for TurretTower and AOETower. The duration in second between each resource gain for ResourceTower. The duration between each emission of of ShootObj for SupportTower.
Build Duration	The build duration for level 1 or upgrade duration to subsequent level.
Shoot Object	The shootObject which will be emit upon every cooldown. TurretTower must have a shootObject with ShootObject.cs component attached in order to function properly. For the rest of the tower type, this is just for visual effect therefore it can be gameObject.
Turret Object	The turretObject for this tower. This object should be a child transform of the tower prefab. It will be deactivated and replaced by the turretObj of next level when the tower is upgraded. The replacement of towerObj will take place only if the turretObj in next level is not left unassigned. For further information regarding turretObj, read the explanation above.
Base Object	The baseObject for this tower. This object should be a child transform of the tower prefab. It will be deactivated and replaced by the baseObject of next level when the tower is upgraded. The replacement of baseObject will take place only if the baseObject in next level is not left unassigned. BaseObject is mostly static object for aesthetic purpose.

Offensive Tower

Damage	Damage caused upon every attack.
Clip Size	the number of ammunition the tower can shoot before needing to reload. Set this to -1 to give the turret unlimited amount of ammo.
Reload Duration	The reload duration for tower when the tower is out of ammo.
Range	The range of the tower.
Aoe Radius	The aoe radius. unit within this radius of the projectile hit

point will be hit. Set to zero to disable aoe effect .

Stun Duration — The duration which the unit hit will be stunned.

Slow Effect

- **Duration** — The duration of the slow effect
- **Slow Factor** — Modifier factor of how much the affected unit will be slowed, takes value from 0.0 to 1.0.

Damage Over Time

- **Damage** — Damage value per tick
- **Duration** — The duration of the effect
- **Interval** — The duration in section between every tick

Barrel Object — The barrelObject for this tower. This object should be a child transform of the turretObject. It will be deactivated and replaced by the barrelObject of next level when the tower is upgraded. The replacement of towerObj will take place only if the barrelObject in next level is not left unassigned. For further information regarding turretObj, read the explanation above.

Support Tower

Buff Effect

- **Damage Buff** — modifier factor that will be applied to the damage of a buffed tower.
- **Cooldown Buff** — modifier factor that will be applied to the attack cooldown of a buffed tower. Value are limited between -0.8 to 0.8.
- **Range Buff** — modifier factor that will be applied to the range of the buffed tower.
- **Regen HP** — HP regeneration value per second that will be applied to the buffed creep.

Resource Tower

Resource Per CD — Valid for resource tower only. The value of resource generated for every cooldown duration. Support Multiple resource type. TowerEditor will auto detect the setting in ResourceManager and change the configurable parameters appropriately.

TOWERSTAT

TowerStat is a group of stats for tower. Each level of a tower uses a diffferent TowerStat. Each TowerStat to each level can be configured independently. Depend on the tower type. Not all configurable parameter on tower stat are applicable on all TowerType. If they are not relevant to the selected tower's tower type, they wont be showing up in the TowerStat subEditor.

In tower editor, each TowerStat are labelled appropriately accordingly to each level. The subEditor can be minimise within the TowerEditor window by uncheck the little check-box above each subEditor.

SLOW EFFECT

Slow Effect is an effect applied on attack to slow down the target movement.

SlowEffect doesn't stack. When more than one effect with different value have been applied on a unit, The effect which slow the unit more will be applied. Effect which get overridden will not be discard, instead it will be reapply when the overriding effect has due. To sum it out, Say there are two slow effect, slow effect a slow the unit by 50% for 3s and slow effect b slow the unit by 30% for 5s. Effect b is applied at t=0, and effect b is applied at t=1. The unit will first be slowed for 30% at t=0 to t=1 and then 50% form t=1 to t=4 and then back to 30% from t=4 to t=5;.

DAMAGE OVER TIME (DOT)

Damage Over Time or DOT is an effect applied on attack to cause periodic damage to the target over a certain amount of time. DOT effect stacks, so any new effect will simply get add on the existing one.

BUFF EFFECT

BuffEffect is an passive and constant effect applied to all friendly unit in range by support unit (tower/creep).

Buff value is in Percentage/100, ie 1 means 100 and 0.1 being 10. If negative value are assign, it will have the inverse effect. This is applicable to support tower only.

BuffEffect stacks in a cumulative manner. If two support towers in range in which both boost the damage by 10%. The resultant bonus value would be

damage*1.1*1.1=damage*1.21. The same goes for cooldown duration reduction. If two support towers in range in which both reduce the cooldown by 10%, the resultant cooldown duration would be, cooldown*0.9*0.9=cooldown*0.81.

TURRETOBJECT

Should the shootObject needs to be fired from some relative position to the turretObject. This can be done by adding script component TurretObject.cs to the turretObject. Then shootPoint will have to be assigned to the component. ShootPoint can make of an empty gameObject that is placed under the hierarchy of the turretObject as a child transform. A turret can have multiple shootPoint where multiple shootObject will be fired simultaneously. Please note that each shootObject carry as much damage value as it's specified in UnitTower.cs

This applies to all Tower that emits shootObject. As an eactive damaging mechanism or just visual effect.

PUBLIC MEMBER

Property	Type	Description
type	**_TowerType**	The type of the tower. Takes values from either of following: • _TowerType.TurretTower • _TowerType.DirectionalAOETower • _TowerType.AOETower • _TowerType.SupportTower • _TowerType.ResourceTower • _TowerType.Mine • _TowerType.Block
unitName	**String**	The name of the tower
icon	**Texture**	The icon of the tower
description	**String**	The string contains description of the tower. Optional and mainly for UI
targetMode	**_TargetMode**	The target mode, indicating which creep type(flying, ground) the tower is targeting. Only applicable to TurretTower and DirectionalAOETower. Takes value from ether of following: • TargetMode.Hybrid

		• TargetMode.Air • TargetMode.Ground
targetPriority	**_TargetPriority**	For debug purpose, check to show the active path through the • TargetPriority.Nearest • TargetPriority.Weakest • TargetPriority.Toughest • TargetPriority.Random
targetingArea	**_TargetingArea**	For debug purpose, check to show the active path through the • _TargetingArea.AllAround • _TargetingArea.DirectionalCone • _TargetingArea.StraightLine
targetingDirection	**float**	For debug purpose, check to show the active path through the
targetingFOV	**float**	For debug purpose, check to show the active path through the
matchTOwerDir2TargetDir	**bool**	For debug purpose, check to show the active path through the platform. Only valid if the platform is walk-able.

METHODS

Method Signatures & Descriptions

`public int GetLevel()`

return the current level of the tower.

`public float GetDamage()`

return the current active damage of the tower.

`public float GetRange()`

return the current active range of the tower.

`public float GetCooldown()`

return the current active cooldown of the tower.

`public float GetClipSize()`

return the maximum clipSize of the tower.

`public float GetReloadDuration()`

return the reload duration of the tower.

public float GetCurrentClip()

return the current clip count of the tower.

public float GetLastReloadTime()

return time when reload is last called of the tower.

public float GetAoeRadius()

return the current active aoeRadius of the tower.

public Dot GetDot()

return the current active stun effect duration of the tower.

public Slow GetSlow()

return the current active slow effect stats of the tower.

public BuffStat GetBuff()

return the current active buff effect stats of the tower.

public int[] GetIncomes()

return the current active income of the tower.

public bool GetMineOneOff()

return the flag mineOneOff of the tower.

public void SetTargetingArea(int targetingArea)

public void SetTargetingArea(_TargetingArea targetingArea)

set the targeting area of the tower if the tower type is either TurretTower or DirectionalAOETower.

- *_TargetingArea.AllAround or 0*
- *_TargetingArea.DirectionalCone or 1*
- *_TargetingArea.StraightLine or 2*

public void SetTargetingDirection(float angle)

set the targeting direction of the tower if the tower is using DirectionalCone or StraightLine targeting. The passing argument is the angle counter-clockwise from +ve x-axis.

Public bool SetTargetPriority(int priority)

Public bool SetTargetPriority(_TargetPriority priority)

set the targeting priority of the tower if the tower type is either TurretTower or DirectionalAOETower. The input argument can be the following:

- *_TargetPriority.Nearest or 0*
- *_TargetPriority.Toughest or 1*
- *_TargetPriority.Weakest or 2*

- *_TargetPriority.Random or 3*

public bool IsLevelCapped()

return true if tower is level capped, false if otherwise.

public bool IsBuilt()

return true if tower is not being build/upgrade, false if otherwise.

public int[] GetCost()

return the cost to build the tower/upgrade it to the next level.

public int[] GetTowerSellValue()

return the amount of resource gained when tower is sold.

public void Upgrade()

call to upgrade the tower, return true if success, false if otherwise.

public void Sell()

called to sell the tower.

public void GetCurrentBuildDuration()

called to return the total duration needed to build/upgrade tower to next level. This value is constant until the tower reach next level.

public void GetRemainingBuildDuration()

called to return the remaining duration needed to build/upgrade tower to next level.This value will be reduced as the tower is being build

public void CheckForTarget(Vector3 screenPos)

check if any potential target(creep) being pointed at the screen position (typical the cursor position). If yes, use it as the current target.

EVENTS

Event Signatures & Descriptions

UnitTower.onBuildCompleteE

public static event Action<UnitTower> onBuildCompleteE

fired when a tower has finished building or upgrading.

UnitTower.onDestroyE

public static event Action<UnitTower> onDestroyE

fired when a tower has been destroyed.

Creep (UnitCreep.cs)

Creep are the moving unit that act as tower's target in a TD game. Setting up a creep prefab can be as simple as attach UnitCreep.js to a gameObject. The object can then be assign to SpawnManager as the creep to be spawned.

There mustn't be any collider in any child object of the creep object but there must always be one collider component on the creep transform itself. The collider can be a manually pre-defined. If there isnt one, UnitCreep.cs will automatically assign a collider. Please note that auto-assigned collider may not be as accurate as manually assigned counterpart. Although it wont affect any game mechanic, however manual assign target to tower may not be working as well if the collider is significantly bigger or smaller then the creep visible mesh.

An important but optional addition to a creep prefab would be an hitpoint or shield overlay to show how much hit point a creep unit have. The overlay object would need to be a child transform of creep's transform. The overlay also have to be a primitive plane object. Please refer to the example creep prefab to see how the overlay are setup.

Another optional addition is for the creep to have an animated model. In this case, animated model will have be another child transform and the creep transform itself will have to be invisible. Again please refer to the example creep prefab to see how such setting should be done.

CONFIGURABLE

Property	Description
Unit Name	The name of the unit. Just for description and UI purpose.
Icon	The icon of the unit. Just for UI purpose.
HP Attributes*	
- Full HP	The maximum HP of the unit creep.
- Full Shield	The maximum Shield of the unit creep. Shield will regenerate over time at the rate specified in "Shield Recharge" until the shield value is at maximum. However the recharging process will pause for a duration specified in "Shield Stagger" whenever the creep is hit. Shield is optional and can be disabled by setting the Full Shield value to 0.
- Shield Recharge	The recharge value of shield per seconds.
- Shield Stagger	The time in second taken for shield to start recharging after the unit has been hit.
- Overlay HP	The object of the hitpoint overlay. This object has to be a plane primitive.

- Overlay Shield	The object of the shield overlay. This object has to be a plane primitive.
- Overlay Base	The object of the base overlay which will appear as a base layer for both shield and HP overlay. This object has to be a plane primitive.
- Always Show Overlay	Check to show overlay HP all the time. Uncheck this for better performance.
Move Speed	The default move speed of the unit creep.
Immune To Slow	Check to set the unit creep to be immune from all slow effect.
Flying	Check to set the unit creep to be a flying unit. Flying unit has a Height-Offset from the ground and ignore all obstacles when moving pass a field.
Flight Height Offset	Height-Offset from the ground. Only used when flying is checked
Value	An array of value representing the amount of resource player gain for killing this unit creep. Support multiple resource type. Each element in the array is corresponded to the resource specified in ResourceManager. Element that exceeds the available type of resource wont be taken into account.
Spawn Effect	Visual effect to be shown on the unit creep when it's being spawned. Optional and can be left blank.
Dead Effect	Visual effect to be shown on the unit creep when it's killed. Optional and can be left blank.
Score Effect	Visual effect to be shown on the unit creep when it's has reach it's final waypoint. Optional and can be left blank.
Animation Body	The animate object for the unit creep if there's one. Required if any animation is to be assigned to the creep.
Animation Spawn**	A builtin array for spawn animations to be played on the animation body.
Animation Move**	A builtin array for move animations to be played on the animation body.
Animation Hit**	A builtin array for hit animations to be played on the animation body.
Animation Dead**	A builtin array for dead animations to be played on the animation body.
Animation Score**	A builtin array for score animations to be played on the

	animation body.
Move Animation Modifier**	A float value to modify the speed of the creep movement animation speed in accordance to the moving speed.
Audio Spawn	Sound fx to be played when the unit creep is spawned. Optional and can be left blank.
Audio Hit	Sound fx to be played when the unit creep is hit. Optional and can be left blank.
Audio Dead	Sound fx to be played when the unit creep is dead. Optional and can be left blank.
Audio Score	Sound fx to be played when the unit creep have reached it's final waypoint. Optional and can be left blank.
Spawn Upon Destroy	The creep to be spawned when this unitCreep is destroyed. This object has to be a unitCreep object. Optional and can be left blank.
Spawn Number	The number of SpawnUponDestroy creep to be spawned. Only applicable if Spawn Upon Destroy creep is .

** A sub-class to store all the relavent information about HP.*

*** Optional and can be left blank, but an Animation component will needed to be assigned to Animation Body in if this is not left blank.*

EVENTS

Event Signatures & Descriptions

UnitCreep.onScoreE

```
public static event Action<int> onScoreE
```

fired when creep has reached it final destination. The wave count which the creep is belong to is passed.

ShootObject (ShootObject.cs)

All objects emit by towers/creeps upon firing are shootObject, be it a projectile to shoot at creep or just visual effect. For shootObject for TurretTower, ShootObject.cs must be attached in order for it to work properly. It's optional for shootObject of other kind. However for performance purpose, It's recommended that all shootObject, even for visual purpose, are attached with ShootObject.cs. This will bind it to ObjectPoolManager and be recycled through the game. This is especially important for mobile build.

A shootObject can be gameObject of any configuration with various component for visual effect. Please refer to the example prefab for reference. There are 4 type of shootObject's mode, all configurable using ShootObject.cs. The difference and function of each shootObject are explained as follow:

SHOOTOBJECT-TYPE

Type	Description
Projectile	A point to point object that will be shoot from turret to target. Support simulation a shoot trajectory. This can be used with a mesh for a particular type of shootObject like canon, or a particleSystem as a energy bolt, or even a short lineRenderer as a laser.
Missile	A point to point object that will be shoot from turret to target, similar to Projectile but it can swing horizontally.
Beam|Instant	An lineRenderer base shootObject. Instead of shooting from point to point. It instantly shoot a beam from tower to target. It's intend be use used as laser-beam or a muzzle effect.
Effect	A ParticleSystem based shootObject. It's intend to be used for non-Turret type tower. Supports both legacy and shuriken ParticleSystem.

CONFIGURABLE

Property	Description
Type	Choose one of the four type listed above for the shootObject. The rest of the relevant/irrelevant configurable parameters will be displayed/hidden in the inspector accordingly to the shootObject type chosen.
Speed	The travel speed of the projectile.
Max Range*	The maximum range intended for this shootObject. It wont affect the actual range for the ShootObject: but instead it's just to modified the trajectory simulation.

Max Angle*	The maximum trajectory angle for the shootObject in x-axis. For missile type shootObject, this value is also responsible for trajectory angle in y-axis.
Line Renderer	The LineRenderer component intended for Beam|Instant type shootObject. This should be a component on the shootObject itself. By default, the script will auto detect any LineRenderer component on the shootObject.
Beam Length	The maximum length of the beam project by the lineRenderer. Set to infinity to allow a beam type shootObject.
Active Duration	The duration in which the shootObject will stay active when fired. When set to 0, the shootObject will stay active for the duration of one single frame. Note that unlike projectile, Beam|Instant shootObject doesn't require to travel from shootPoint to target.
Continuous Damage	The duration in which the shootObject will stay active when fired. When set to 0, the shootObject will stay active for the duration of one single frame. Note that unlike projectile, Beam|Instant shootObject doesn't require to travel from shootPoint to target.
Effect Type	When checked, the shootObject will apply the damage effect and etc. over the duration when it's active. If left unchecked, the effect will only get applied when the beam| instant shootObject's duration is due.
Shoot Sound	The sound to be played when the shootObject is fired
Hit Sound	The sound to be played when the shootObject hit something
Shoot Effect	The gameObject intend as visual effect to be spawned at the shootObject's position when the shootObject is fired.
Hit Effect	The gameObject intend as visual effect to be spawned at the shootObject's position when the shootObject hit it's target.

** The projectile shoot elevation angle are dependent on the target's distance from the shootPoint. At MaxRange, the projectile will then be shoot at the MaxAngle.*

Note: not all configurable in the list above are applicable for every shootObject type. The editor will show configurable relevant to the shootObject type selected.

OPTIONAL COMPONENTS

These are the optional components of the TD gameplay.

CameraControl (CameraControl.cs)

Script component that allows manipulation of the camera. The camera control in this script works pretty much like a 3rd person character. it pan's around the horizontal plane, zoom in/out along the view direction, and rotate around a centre anchor point. The component supports iOS as well for moving and zooming.

The component is not designed to be used on the camera component itself. Rather, the camera component should be a child transform of the gameObject with the component attached. The parent transform will act as the actor point where the rotation and zooming will be centered around on. When moving, the parent transform is moved instead of the camera transform.

CONFIGURABLE

Property	Description
Pan Speed	The speed of the camera when moving in horizontal plane.
Zoom Speed	The speed of the camera when moving zooming in/out.
IOS Enable Pan	Enable panning for iOS when dragging a finger on the screen.
IOS Enable Zoom	Enable zooming for iOS when pinching the screen.
IOS Enable Rotate	Enable rotation for iOS when dragging two fingers on the screen.
Rotation Speed	Sensitivity for rotation input (for iOS only)
Min Pos X	Minimum x-axis position of the camera's parent transform in world-space.
Max Pos X	Maximum x-axis position of the camera's parent transform in world-space.
Min Pos Z	Minimum z-axis position of the camera's parent transform in world-space.
Max Pos Z	Maximum z-axis position of the camera's parent transform in world-space.
Min Radius	Minimum distance of the camera component from the parent transform.
Max Radius	Maximum distance of the camera component from the parent transform.
Min Rotate Angle	Minimum angle of the camera from horizontal plane. Limited to 10.

Max Rotate Angle	Maximum angle of the camera from horizontal plane. Limited to 89.

AudioManager (AudioManager.cs)

AudioManager is the component which manage all the music and sound fx in a scene. AudioManager is optional. If there isn't one in the scene, GameControl will automatically create one.

CONFIGURABLE

Property	Description
Min Fall Off Range	The minimum fall off range of all the 3D sfx. Higher value allow the sfx to be heard by the audioListener even when it's far away. Has no effect if the sfx used is 2D sound.
Music List	A list of music track that will be used as the background music.
Play Music	Check to enable playing of the background music.
Shuffle	check to enable shuffling of the music list when playing music.
Wave Cleared Sound	Sfx to play when a wave is cleared.
New Wave Sound	Sfx to play when a new wave started to spawn.
Game Won Sound	Sfx to play when the game is won.
Game Lost Sound	Sfx to play when the game is lost.
Tower Building Sound	Sfx to play when a tower is start building.
Tower Built Sound	Sfx to play when a tower has finished building.
Tower Sold Sound	Sfx to play when a tower is built.

Note: all music and sfx are optional, if left blank, nothing will be played.

MiniMap (MiniMap.cs)

Shows a tactical minimap in the game. Please note that it's integrated with ObjectPoolManager. Should you wish to use it for other project, make sure ObjectPoolManager.cs is copied along.

CONFIGURABLE

Property	Description
Update Rate	The update per second for the map.
Minimap Layer	The number ID of the layer to be used exclusively for the minimap. Make sure no other gameObject is using the layer to avoid artifact on the minimap.
Map Rect	The rect information used to describe the position and size of the map on screen. • **x**: The minimap top-left corner's screen coordinate in x-axis. • **y**: The minimap top-left corner's screen coordinate in y-axis. • **width**: The width of the minimap in pixel • **height**: The height of the minimap in pixel.
Map Center	The center point Vector2(x, y) of the map to be shown in world space.
Map Size	he size of the area covered by the map in world space in Vector2(x, y);
Map Texture	The texture to be used as the map background of the minimap. This texture should be the map itself
Panel Alignment	The alignment of the minimap button panel. Choose from one of the following: • **Top**: The panel will appear on top of the map. • **Bottom**: The panel will appear on bottom of the map. • **Left**: The panel will appear on the right side of the map. • **Right**: The panel will appear on the right side of the map.
Track Obj	The Object to track. The minimap will centred around the object when zoomed.
Track Position	Checked to enable the minimap to follow the trackObj's position.

Track Rotation	Checked to enable the minimap to rotate along with the trackObj's rotation.
Trackables*	
- Layer	The layer of the track object.
- Blip Texture	The texture which will be used for the blip of the track object.
- Blip Size Modifier	The blip size.
- Max Num	Maximum number of the object in the scene. This is just to pre-spawn the blip.
- IsStatic	Check if the trackObj is a static object which have fixed position and rotation.

** The track-able object tracked by the minimap component. The objects are tracked via layer so each of them will need to have a dedicated layer.*

PathIndicator (PathIndicator.cs)

This is an optional script to for indicating the active path using a particleSystem. It should be attached along with functional path object in order for it to work properly. The indicator that shows up are configured in the particle system. The script merely emit particle along the active path.

CONFIGURABLE

Property	Description
IndicatorT	The transform which contain the ParticleSystem to be used.
Step Dist	The distance of each indicator step. A particle is emitted with each step.
Update Rate	The duration in seconds between each step.

CreepAttack (UnitCreepAttack.cs) - Optional

This is an optional add-on component for UnitCreep. This component added options to configure creep to perform attacking move.

ADVANCE-CREEP-TYPE

Type	Description
Attack	Typical attacking creep, for causing damage to towers.
Support	Supporting creep, instead of causing damage, this type of creep buff all surounding creeps. By either increase their damage output or regenerate their HP.

CREEPATTACK-EDITOR

CreepAttack can be configured via a custom inspector. Not all configurable parameter listed in the documentation is visible in the inspector. The inspector only shows relevant configurable.

Property	Description
Creep Type	A list of the tower type. Tower type which are specified on the list are the tower type supported by this platform.
Mode	Specify the movement behaviour of the creep. The behaviour supported are listed as follow: • **RunNGun**: The creep will carry on moving regardless of what happen. If there's target in sight, the creep will fire while moving. • **StopNAttack**: The creep will stop to focus on attack as soon as it spotted a target. It will resume moving when it's target is destroyed.
Shoot Point	The transform point in which a shootObject or any effect is spawned. This transform needs to be a child transform of the creep.
Shoot Object	The shootObject which will be emit for every cooldown if there's an target. This is only applicable for range attacking creep or support creep. Range attacking creep must have a shootObject with ShootObject.cs component attached in order to function properly.
Idle Animation*	The animation to play when the target is not mvoing and not attacking. This is optional and mostly visible only for creep with mode set to StopNAttack

Attack Creep

Targeting Area	The target area of the creep. This is similar to the target area of a TurretTower. The option supported are listed as follow: • **AllAround**: The creep will scan in all direction for any possible target • **FrontalCone**: The creep will scan in conical area ahead of itself for any possible target. The angle of the conical area can be specified. • **Obstacle**: The creep will only look for target which blocked it's path.
Frontal Cone Angle	The angle of the conical area where the creep scan for possible target. Only applicable when Targeting Area is set to Frontal Cone.
Attack Method	The attack mechanism used by the creep: • **Range**: The creep will perform range attack. Note that a Shoot Object must be assigned for this to work • **Melee**: The creep will perform melee attack.
CD Tracking	The cooldown tracking mechanism used by the creep: • **Easy**: The attack cooldown will based loosely to what is specified. • **Precise**: The attack cooldown will be tracked precisely accordingly to what is specified.
Attack Range	The maximum attack range of the creep. Creep cannot attack anything beyond this range. This will also act as target range for range attack creep.
Target Range	The maximum target range of the creep. Creep cannot targeting anything beyond this range.
Attack Cooldown	The time required for the creep to perform a subsequent after an attack.
Damage	The damage caused by the creep for every attack
Stun Duration	The stun effect duration on target when attacked by this creep.
Turret Object	Turret Object (transform) of the creep, similar to the turret object of a TurretTower. This turret will always aim at the

	target of the creep. Please note that this should be a child transform of the creep.
Attack Sound	The sound to play when the creep attack
Attack Animation	The animation to play when the creep attack. Please note that this is similar to any animation of UnitCreep, a Animation Body needed to be assigned to UnitCreep.
Support Creep	
Effective Range	The effective range of the support creep, any creep within this range from the unit will received the buff effect.
Damage Buff	Modifier factor that will be applied to the damage of a buffed creep.
Range Buff	Modifier factor that will be applied to the range of the buffed tower.
Cooldown Buff	Modifier factor that will be applied to the attack cooldown of a buffed creep. Value are limited between -0.8 to 0.8.
HP Regen Buff	HP regeneration value per second that will be applied to the buffed creep.
Effect Cooldown	The cooldown between each emission of the Shoot Object.

SUPPORT UTILITIES

These are the support components of the TD gameplay, namely path-finding and object-recycler. They don't affect the gameplay too much but they are the backbone in which many of the mechanic of the game are build on and absolutely vital. It's worth mention that ObjectPoolManager is a completely stand alone component which can be used in other project for similar purpose.

NodeGenerator (NodeGenerator.cs)

Component used to generate node on the walkable platform.

CONFIGURABLE

Property	Description
Connect Diagonal Node	Checked to connect neighbouring diagonal node.

PathFinder (PathFinder.cs)

Component that is responsible for finding path through all walkable platform.

CONFIGURABLE

Property	Description
Path Smoothing	Apply path smoothing after a path is found. • **None**: No path smoothing. • **LOS**: Path smoothing using line-of-sight • **Mean**: Path smoothing by averaging position of neighbouring waypoint.
Scan Node Limit Per Frame	Limit how many node the algorithm will search for in one single frame.

ObjectPoolManager (ObjectPoolManager.cs)

Instantiate and destroy object in run time is expensive on mobile device. To avoid this, we instantiate all the object needed in the scene during loading of the scene. The objects are kept inactive until they are needed. When they are no longer needed, they are made inactive again.

ObjectPoolManager is a static class used to create and keeps track of the pools of objects created. You can use it in any other project. The script doesn't need to be active in the scene. As long as it's in the asset-tab, it should work. An important note is you should call ObjectPoolMnager.Init() in the start of every scene where ObjectPoolManager is used, before you actually create any object pool. The class function are as follow:

METHODS

Method Signatures & Descriptions

ObjectPoolManager.Init

```
public static void Init()
```

this function needs to be called at the start of every scene before any new ObjectPool is created. It will reset and clear all the objects and pools in the manager.

ObjectPoolManager.New

```
public static void New(GameObject object, int number=1, bool stack=true)
public static void New(Transform object, int number=1, bool stack=true)
```

object – Transform or GameObject prefab to be pre-spawned and pooled.

number – the number of gameObject to be spawned. The default value is 1.

stack – flag indicate if the amount of new object should be stack onto the existing pool. The default value is true.

call to create a pool of a particular gameObject. Typically this is called during loading a scene in Start(). If an object of a similar type has existed in the pool, the new object registered into the existing pool. If stack is set to false, the manager will simply fill the pool so the size is no bigger than number. For example, if the passing number is 20 and the stack is false, if the existing pool already have more than 20 objects in the pool, no new object will be spawned. And if there's less than 20 objects in the pool, the function will increase the fill the pool up to 20. On the other hand if the stack is passed as true, no new object will be spawned, else 20 new objects will be spawn regardless.

ObjectPoolManager.Spawn

public static GameObject Spawn(GameObject object, Vector3 pos=Vector3.zero, Quaternion rot=Quaternion.identity)
public static Transform Spawn(Transform object, Vector3 pos=Vector3.zero, Quaternion rot=Quaternion.identity)

object – Transform or GameObject prefab to be pre-spawned and pooled.
pos – the position to be assigned to the newly spawned gameObject. The default value is (0, 0, 0).
stack – the rotation to be assigned to the newly spawned gameObject. The default value is (0, 0, 0).

called to spawn and return an instance of the gameObject, similar to default Instantiate(). The function will attempt to return an inactive object in the pool. If there isn't any object available, the function will then create a new object using Instantiate instead. The newly instantiated object will be added to the pool.

ObjectPoolManager.Unspawn

public static void Unspawn(GameObject object, float delay=0.0f)
public static void Unspawn(Transform object, float delay=0.0f)

object – Transform or GameObject prefab to unspawn.
duration – the delya in second before the object is destroyed. Default value is 0.

called to delete object, similar to default Destroy(). The function will attempt to insert the object back to the pool and deactivate it. If there isn't a pool registered for this object, the object will be delete using Destroy(). Duration is the delay in second before the object is destroyed.

ObjectPoolManager.CearAll

public static void ClearAll()

called to destroy all the objects in the manager.

USER INTERFACE

The UI in this toolkit are separate component from the rest of the toolkit. They only serve as an example of how to put together a function UI using all the API. The toolkit are design to allow user to build custom UI. However there are two default UI which can be used just as it is.

Please note that these two UI are mutually exclusive. Only one is required at any given time. The gameObject "UI" in both example scene contains both UI. You can select between them by disable on and enable another. Both UI are configured to a game resolution of 960x640. The gui-element might not be appear properly placed if the resolution is otherwise.

UI (UI.cs)

This UI is designed for webplayer and PC/Mac. It's not performance friendly with mobile platform as it uses unity default OnGUI() extensively.

CONFIGURABLE

Property	Description
Build Menu Type	The build tower panel placement and arrangement. This is valid in PointNBuild only. • **Fixed**: The build towers panel will be fixated at the bottom left corner. • **Box**: The build towers panel will be floating at the screen where the build-point is. The buttons is arranged in 2 column. • **Pie**: The build towers panel will be floating at the screen where the build-point is. The buttons forms a semi-circle surrounding the build point.
Build Mode	The building scheme that will be used in the scene. • **PointNBuild**: User click on platform and select which tower to build • **DragNDrop**: User select a tower to build and place it on where it meant to be built.
Fast Forward Speed	The time speed up modifier when fast-forward button are pressed.
Enable Target Priority Switch	Shows interface for user to switch TurretTower and DirectionalAOETower targeting priority when the tower

	is selected.
Enable Target Direction Switch	Shows interface for user to switch targeting direction of towers which uses FOV or StraightLine targeting area when the tower is selected.
Always Enable Next Button	Check to enable next level button in end game menu regardless of win/loss
Next Level	Scene's name to be loaded when next button are pressed.
Main Menu	Scene's name to be loaded when menu button are pressed.

UIiOS (UIiOS.cs)

This UI is designed for specifically for mobile device. However it can be use on just about any platform. The UI uses a lots of prearranged GUI gameObject. These gameObject can be arrange in whatever way without affecting the script.

CONFIGURABLE

Property	Description
Fast Forward Speed	The time speed up modifier when fast-forward button are pressed.
Always Enable Next Button	Check to enable next level button in end game menu regardless of win/loss
Next Level	Scene's name to be loaded when next button are pressed.
Main Menu	Scene's name to be loaded when menu button are pressed.
General UIText	Shows interface for user to switch targeting direction of towers which uses FOV or StraightLine targeting area when the tower is selected.
Message UIText	Check to enable next level button in end game menu regardless of win/loss
Spawn Button*	Spawn button related setting.
Ff Button*	Fast-forward button related setting.
Pause Button*	Pause button related setting.

Upgrade Button*	Upgrade button related setting.
Sell Button*	Sell button related setting.
General Box	GUITexture for the general menu panel. (contains Menu, Restart and NextLevel button)
Menu Button*	Menu button related setting.
Restart Button*	Restart button related setting.
Next Lvl Button*	Next level button related setting.
Selected Tower UIbox	GUITexture for the selected tower display panel.
Selected Tower UIText	GUIText for the selected tower display panel.

** This is custom button class, see CustomButtoniOS for more details.*

CursorManager (CursorManager.cs)

Cursor Manager enable support for custom cursor which react accordingly to object player is pointing. This component only support PC and desktop and it is entirely optional.

CONFIGURABLE

Property	Description
Pointer	Default cursor.
Hostile	Cursor to show when mouse pointer is pointing at a creep. Only valid when a tower is selected.
Friendly	Cursor to show when mouse pointer is pointing at a tower.

UIRect (UIRect.cs)

UIRect is a User Interface utility class to define the rectangular area occupy by various UI element on the screen. A method can be called whenever user has click or touch a point to check weather the point is on the UI or not. This is so the user wont be accidentally select a tower or a build point when trying to click on a button.

The class store a list of all the Rect specified and will check against all the rect when IsCursorOnUI is called. These rect can be add or remove during runtime.

METHODS

Method Signatures & Descriptions

UIRect.IsCursorOnUI

```
public static bool IsCursorOnUI()
```

call to check if the pointer is on a UI element. Return true if yes, false if otherwise.

UIRect.AddRect

```
public static void AddRect(Rect area)
```

add a new rectangular area to the list.

UIRect.RemoveRect

```
public static void RemoveRect(Rect area)
```

remove an existing rect area from the list, if a match is found.

CustomButtoniOS (CustomButtoniOS.cs)

This is a component for customise button using GUITexture. It contain several custom button class:

class **GUIButton**
- A typical button. This is the base class.

class **GUIContinousButton** derive from **GUIButton**
- button that will triggered as long as it's pressed.

class **GUIToggleButton** derive from **GUIButton**
- button that toggle between state when pressed.

PUBLIC MEMBER

Property	Type	Description
ID	**int**	A unique integer ID that can be assigned to the button which will be passed to the callback function when the button is pressed. This is optional. Only useful if there are buttons that

		shared callback function.
buttonObj	**GUITexture**	The GUITexture object that will act as the button.
unpressedTex	**Texture**	Button texture when idle.
pressedTex	**Texture**	Button texture when pressed or in alternate state.
triggereOnPressed	**bool**	Check to enabled call-back function to be called upon pressed, else the call-back will only be called upon button released. This is only valid for GUIButton but not for other button type.

PUBLIC DELEGATE

Delegate Signatures & Descriptions

public delegate void ButtonPressedCallBack(int ID)

CustomButtoniOS.callBackFunc

public ButtonPressedCallBack callBackFunc

The callback function for the button instance. This can only be assigned via code.

public delegate void ButtonPressedCallBack(int ID)

CustomButtoniOS.toolTipFunc

public ToolTipCallBack toolTipFunc

The tooltip function call for the button instance which will be called whenever the button is pressed and only valid for GUIButton. This can only be assigned via code.

PUBLIC METHOD

Method Signatures & Descriptions

public Ienumerator Update()

GUIButton class equivalent of MonoBehaviour.Update(). It's a coroutine and there fore has to be started using StartCoroutine().

EXAMPLE CODE

To enabled the button to start and working properly. the class coroutine Update() has to be initiated. Also the callback function has to be assigned otherwise it won't execute anything. An example of how it can be done are shown below:

```
//define the button so it can be configured in inspector
public GUIButton myCustomButton;

void Start(){
        //assign the callback function
        myCustomButton.callBackFunc=this.ButtonPressed;
        //start the button coroutine
        StartCoroutine(myCustomButton.Update());
}

//callback function for myCustomButton
void ButtonPressed(int ID){
        Debug.Log("button with ID:"+ID" is pressed");
}
```

Script UIiOS.cs in the toolkit used the custom button class exclusively for buttons. Refer to it for more example.

OTHERS

These are the components that needs no configuration. Some of them are vital component but you never need to concern yourself with these. Except DebugShowSelf.cs maybe, which is useful when positioning empty object has as waypoint and shootPoint.

Unit (Unit.cs)

The base class of the UnitCreep and UnitTower.

UnitUtility (UnitUtility.cs)

Contains utility method for unit

OverlayManager (OverlayManager.cs)

Component that govern build/upgrade overlay bar

GameMessage (GameMessage.cs)

A utility component used to display message on screen which is intended for debugging on mobile device. A GUIText can be assigned to it or it will be assigned automatically.

METHODS

Method Signatures & Descriptions

GameMessage.DisplayMessage

```
public static void DisplayMessage(string message)
```

display a string of text on screen with the passing argument message being the string of text to be displayed. The message will fade away after a few seconds. If the function is called again before that, the new message will be displayed as a new line after previous message.

DebugShowSelf (DebugShowSelf.cs)

Allow empty gameObject to appear visible in scene-view and game-view when selected.

CONCLUSION & CONTRACT NOTE

Finally thanks for using this toolkit. I hope you enjoy using to create your own TD game. You are welcome to use the components of this tookit for your other game that isn't a TD game.

If you have build your very own TD game using this toolkit. I would appreciate if you give it some credit.
Even better send or link some me some example or demo of the game. I like to know how useful the toolkit has been. I'm happy to help you promote it.

I apologize if there's still anything unclear and missing in the documentation. I also apologize for any limitation of the toolkit. If you have any question or comment, suggestion about this toolkit, or should you come across any bug, Please visit http://songgamedev.blogspot.com/ to leave a comment or email me directly at k.songtan@gmail.com. I'll do my best to provide support on any issue.

I'll try my best to provide support regarding issue within the toolkit. I'll also consider doing feature request. However please understand that I cant possibly accommodate all request. I'll happily do any feature request that is useable for other user or within reason for free. Also you may also learn that I do work as a free-lance developer. For that, I'm more than happy to help you extend the kit to accommodate any custom feature you may like to see. Thanks again!

VERSION HISTORY

Version Change – 2.0.1

- fix “UI” gameObject error (missing child object) in exampleScene1 and exampleScene2
- fix bug where wave clear event wont launch if the last creep unit is kill instantly after spawned
- fix bug where tower cant be upgraded beyond level 2
- fix bug where default platform object with unspecified build-able tower doesn't support mine.
- fix bug on UIiOS where wave info is not being displayed correctly
- fix bug where SpawnEditor cause an error when there's only one resource used in the loaded scene.
- fix bug on ResourceManager editor where information on the editor is not displayed correctly when different scene is loaded
- game logic change, SpawnManager will now stop spawning upon gameOver event.
- added new tower type DirectionalAOETower. A hybrid between TurretTower and AOETower. Actively targeting tower which damage all target within a adjustable conical area.

Version Change – 2.0.2

- fix bug where TurretTower won't shoot when there's no turretObject assigned.
- fix bug where tower will target a creep even when it's destroyed.
- fix bug where GameControl will always auto initiate AudioManager and override user created AudioManager.
- fix bug with editor for GameControl and SpawnManager.
- fix bug for tower editor where TurretAnimateMode doesn't assigned properly.
- fix bug where creep will replay death animation when hit after they are destroyed.
- fix bug where override speed value in SpawnEditor doesn't take effect.
- Added option for dynamic waypoint, randomise creep position along the path in some extent. Make group of creeps movement and placement more natural. The parameter can be turned on/off or adjusted using “Dynamic WP” in Path.cs.
- Added option for creeps to spawn more creep upon destroyed. Configurable in UnitCreep.cs.
- Added camera rotation for iOS, configurable in CameraControl.cs.
- pinch-zooming in iOS has been reworked and now work more consistently.

Version Change – 2.0.2i

- fix bug where wave would not register as cleared correctly.
- fix bug where creep spawned by destroyed creep resetting the path instead of follow it through.
- fix bug where tower only appear partially transparent when dragged in DrapNDrop mode

Version Change – 2.0.3

- added build and shoot animation support for towers
- added manual targeting for selected towers
- added CursorManager, support custom cursor for mouse
- fix bug for creep pathing

- fix bug where slow effect doesn’t apply to creep
- fix bug for UI.cs where level complete/failed message doesn't displayed properly

Version Change – 2.0.4

- added ammo count and reload mechanism to turret and directionAOE towers.
- added range preview for towers in building phase.
- added tower preview for PointNBuild build mode, for both UI.cs and UIiOS.cs.
- added modifier for creep's move animation to match the movement speed.
- added pause menu for UI.cs. Can be triggered by button pressed or 'escape' key.
- tower can now be pre-placed in the scene.
- Small addition to SpawnEditor, time required for each wave to finish spawn in now shown.
- fix bug where modified prefab value using editor is not saved upon unity Editor quit.

Version Change – 2.0.4h

- fix several UI related bug
- fix issue where next level turret and base doesn't appear correctly when upgrading towers.
- fix issue for editor where when adding new level for towers, the new stats are linked across level.

Version Change – 2.1

addition

- Added component which allows UnitCreep to attack tower or buff nearby creeps.
- Various behaviour can be set for attacking creep such as range/melee, stop-while-attack or else, etc.
- new targeting configurable parameters for tower which uses direct targeting.
- The new tower targeting option include area(all-around, fov, straightline) and priority(nearest, toughest, weakest, random). New range indicator support has been added for new added targeting area
- Added shield for both tower and unit, which can be regenerated over time. Overlay support included
- support tower/creep can now 'heal' damaged tower/creep
- added new rotation mechanism to TurretTower where turret turn in y-axis while barrel turn in x-axis
- change camera zoom-limit limitation and y-axis positioning clamp
- add option to enable/disable tile/selection indicator
- UI.cs now no longer use GameMessage.cs, and thus the appearance of game-message can be configure via GUISkin
- tower base now support fire animation.
- Path.cs has not been change to PathTD.cs.
- Added new example scene and prefabs.
- Major overhaul on Documentation

bug fix

- fix bug where tower cant be build on active path on platform when path-smoothing is disable
- fix bug where spawnEditor reset SpawnMode back to continous
- fix bug where projectile trajectory's angle doesn't calculate properly with respect to distance
- fix bug where overlay is not displayed properly when the parent unit scale is not set to 0.5.

- fix bug where platform size will not be correctly auto-adjust when placed as child object where parent scale is not (1, 1, 1)
- fix bug where creep spawned by destroyed creep resetting the path instead of follow it through, again.