TowerDefenseToolkit v2.0.3

for Unity3D by K.SongTan

Thanks for using TDTK. This toolkit is a collection of coding framework in C# for Unity3D. The toolkit is designed to cover most of the common TD game mechanism. Bear in mind TDTK is not a framework to create a complete game. The framework simply try to cover most, if not all the TD gameplay mechanic. The framework does not cover other game aspect such as menu scene, option, etc.

The toolkit is design with integration of custom asset in mind. Although there are some minimal amount of asset included with the toolkit as example. User are expected to integrate their own audio and art asset.

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

For previous version's (1.x) user

TDTK2 is not an update from TDTK1.x. It's a totally new and re-designed framework therefore it's not backward compatible. There might be some rearrangements such as all the build configuration has been placed under one component, Tag is not longer used and etc.. However you will find that all the features from previous version of TDTK is still available. I'm fairly confident that you'll find that this version is better than any previous version.

Overview of the Framework

Essential Component

There are 3 main essential components made up of 3 different script to govern the game logic, every scene should have one and only one instance of these components in the scene. They are:
*<b>GameControl</b> (see pg3): Control game rules and basic mechanic.
SpawnManger (see pg6): Responsible to spawn and keeping track of creeps.
BuildManager (see pg8): Responsible for building and deployment of towers.

*GameControl contains 2 sub-component (see pg5) which is:
LayerManager are used to customized the layer used in the game.
ResourceManager are used to configure the resource use in the scene.

Unit are the interactive object consist of both creep and tower in game.
UnitTower (see pg10): each instance of this govern one single tower.
UnitCreep (see pg14): each instance of this govern one single creep.

Path (see pg9) are instance of route used by the creep, there can be multiple path in a game.
Platform (see pg9) are object which the towers can be built upon, there can be multiple platform in a game.

ShootObject (see pg15) are the object emit by tower, to damaging creep or purely for visual effect.

Optional Component

These are optional, you can easily replaced them with your own implementation without effecting the base gameplay in any way.

CameraControl (see pg17): Control the camera
AudioManager (see pg17): Govern all the audio, although not necessary, most essential components used AudioManager. So at any rate, always include AudioManager.cs for TDTK2 project.
MiniMap (see pg18): Shows a tactical minimap, use this to track whatever object you desired.
PathIndicator (see pg18): an optional component used in adjunct with Path. Provide visual clue to the path.

Support Utility

These are the engines that support the game mechanic. They required minimal to no configuration most of the time. You can even ignore it entirely without manually adding them to a scene. But bear in mind that they are absolutely vital and should always be include for any TDTK2 project.

NodeGenerator (see pg19): Generate navigational node on platform
PathFinder (see pg19): Find a route through walkable platform where navigational node has been generated
ObjectPoolManager (see pg19): Recycling gameObject in game for performance reason

UI & UI Utilities

User Interface are completely standalone component in TDTK2. There are 2 mutually exclusive default UI for the example scene. But user can build a custom UI given all the API are given and documented.

UI (see pg21): The default UI for non mobile platform, uses OnGUI()
UIiOS (see pg21): The default UI for mobile platform, can be used on other platform.
UIRect (see pg22): A utility to prevent a click/touch on the UI element from trigger in game interaction such as selection of a build point or tower.
CustomButtoniOS (see pg22): A custom button class using GUITexture gameObject. As used in UIiOS.

Other (see pg24)

These are the components that needs no configuration. However they are vital component.

ExampleScene

There are 3 example scenes in the toolkit to show case how the toolkit can be used to put together different kind of levels

1. 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 the controling component of the logic and state of the game. It initate the game, and determine when the game is won or lost, etc. A scene must always contain one and only one GameControl. GameControl by default will require sub-component LayerManager and ResourceManager.

Configurable

Player Resource: The resource player have at the start of the game. This will be as currency to build Towers.
Player Life: Player Life at the start of the game.
Sell Tower Refund Ratio: The ratio of the tower value player got back when they sell a tower. The value takes account of the cost to build the tower as well as the resource spent to upgrade the tower.
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.
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 centre of the tower.

GameControl.cs exposes the following methods:

GameControl.GetPlayerLife
public static int GetPlayerLife()
- return current value of player's life

GameControl.Select
public static UnitTower Select(Vector3 pointer)
- called to select a tower based on the pointer on screen, if there's a valid tower being pointed at. Return the tower selected if there's one.

GameControl.ClearSelection
public static void ClearSelection()
- called to clear the selection on selected tower if there's any.

GameControl.GainResource
public static void GainResource(int[] value)
- called to add an array of value to the current resources. The first element in the array is corresponded to resource type1, second element is corresponded to resource type2, so on and so for forth. If array length exceed current available resource type, the value will be ignored.
public static void GainResource(int value)
- called to add/subtract value to/from the first/default resource type.
public static void GainResource(int id, int value)
- called to add/subtract value to/from the a specific resource type.

GameControl.GetResourceVal
public static int GetResourceVal()

public static int GetResourceVal(int id)
- return a specific resource type value. id specify which resource value should be retrieved. By default the value is 0 so the value of the first resource is returned.

GameControl.GetAllResourceVal
public static int[] GetAllResourceVal()
- return a builtin array of int containing current value of each resource type. id specify which resource value should be retrieved. By default the value is 0 so the value of the first resource is returned.

GameControl.SpendResource
public static int GainResource(int[] value)
- called to subtract an array of value to the current resources. The first element in the array is corresponded to resource type1, second element is corresponded to resource type2, so on and so for forth. If array length exceed current available resource type, the value will be ignored.

GameControl.HaveSufficientResource()
public static bool HaveSufficientResource(int[] cost)
- Called to check if there's sufficient resource for the cost passed. 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, so on and so forth. Return true if there's sufficient resource, false if otherwise

GameControl.GetResourceList()
public static Resource[] GetResourceList()
- Return a builtin array of all the Resource object in ResourceManager. See Section ResourceManager for more info about Resource.

GameControl.GetSellTowerRefundRatio
public static float GetSellTowerRefundRatio()
- return the towerRefundRatio

GameControl.GetResourceList
public static float GetSellTowerRefundRatio()
- return the towerRefundRatio

<u>GameControl.cs fires the following events:</u>
public static event Action<bool> GameOverE
- fired when the game over condition is satisfied. True is passed if player have won, else if player has lost.

public static event Action ResourceE
- fired when the resource's value has changed

public static event Action LifeE
- fired when the player's life value has changed

<u>ResourceManager</u> **(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.

<u>Configurable</u>
- **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

ResourceManager.cs contain following class
Resource
Contains information about the resource. The public variables are:
- Texture icon: The icon of the resource
- String name: The name of the resource
- int value: The current value of the resource

<u>LayerManager</u> **(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 dont need to manually assign it

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.

<u>SpawnManager</u> **(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.

<u>Configurable</u>

SpawnMode: specific spawn mode to be used in this scene. The options are:

- Continous: A new wave is spawn upon every wave duration countdown
- WaveCleared: A new wave is spawned when the current wave is cleared
- ContinousWithSkip: Similar to Continous but user can initiate the 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.

DefaultPath: The primary path to be used. This path can be overridden by other alternate path for each individual spawn.
WaveSize: The number of sub wave in this scene.

> ***<u>Wave</u>***
> **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.
>
> > ***<u>SubWave</u>***
> > **Unit Prefab**: The prefab of the Creep for the subWave. 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.
> > **OverrideHP**: The HP value, if the value is larger than 0, the value will override the default HP value for the creep prefab.
> > **OverrideSpeed**: The moveSpeed, if the value is larger than 0, the value will override the default moveSpeed value for the creep prefab.
>
> **Duration Before Next Wave**: The duration for this wave until next wave is commenced. The next 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.
> **Resource Upon Wave Cleared**: 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

<u>SpawnManager.cs exposes the following methods:</u>
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.

<u>SpawnManager.cs fires the following events:</u>
public static event Action<int> WaveStartSpawnE
- fired when a new wave start spawning

public static event Action<int> WaveSpawnedE
- fired when a waves have finish spawning

public static event Action<int> WaveClearedE
- fired when a waves has been cleared

public static event Action<bool> ClearForSpawningE
- fired when a new waves can be spawned

<u>BuildManager</u> (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

<u>Configurable</u>

Towers: A list of towerPrefab that is build-able in this scene. When towerEditor is launch

GridSize: The gridSize of the building grid in this scene.

Platforms: A list of the platforms in which the towers can be built in this scene. For what constitude a platform, please refer to section Platform. Every platform assigned will be automatically resized to fit the gridSize. These platform has to have a Platform(Platform.cs) component attached to them, if not, BuildManager will be automatically assign a platform component to it.

AutoAdjstTextureToGrid: Check to re-arrange the meterial 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).

<u>BuildManager.cs exposes the following methods:</u>

BuildManager.CheckBuildPoint

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

Automatically determine which grid or platform the pointer on the screen is pointing to and round the position to the center of the grid. Return true if a tower can be built on that position, false if otherwise. TowerType supported on the platform will be checked against if a tower type is passed.

- screenPosition: the position of the pointer on screen, usually the mouse cursor or a touch point.
- type: the type of tower to be checked. Look in section Tower for more info.

BuildManager.BuildTowerPointNBuild

public static bool BuildTowerPointNBuild(UnitTower tower)

- Create tower object on the position which specified when BuildManager.CheckBuildPoint is called. The function will automatically determine if there is enough resource or other per-requsite in order for this building operation to take place. Return true if the build operation is successful, false if otherwise

BuildManager.BuildTowerDragNDrop

public static bool BuildTowerDragNDrop(UnitTower tower)

- Create a new drag and drop tower object. 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 the full list of tower available in this scene.

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 BuildableInfo GetGridSize()

- return the gridSize for this scene.

<u>Platform</u> **(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.

Sizable platform can also be used as a waypoint in which the platform will become a small field for the creep to be navigate through. A small open-field section so to speak

<u>Configurable</u>
Buildable Type: A list of the tower type which are buildable on this platform.
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.

<u>Path</u> **(Path.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.

<u>Configurable</u>
***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.
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 doesnt 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.

*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.

<u>Tower</u> **(UnitTower.cs)**

Tower are the base building unit in a TD game. They can only be built on a platform. There are 4 general type of towers. They are:

- ***TurretTower***: Typical tower, fire shootObject at creep
- ***AoeTower***: This tower doesnt 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.
- ***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 doesnt block the path. But any creep that move pass it will trigger it. Support all effect available to TurretTower.

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 (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 explained in next section.

TowerEditor

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.

UpdateButton: Unfortunately the editor doesnt 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.

LevelCap: the maximum level in which the tower can be upgrade to. Minimum value is 1.

TowerName: Name of the tower that will appear in the game.
TowerType: The type of tower prefab

TargetMode: The target type of the tower. Only applicable for as TurretTower or AOETower.

- *Air* - tower attack only air unit
- *Ground* - tower attack only ground unit.
- *Hybird* - tower attacks both ground and air unit.

TurretAnimateMode: How the turret object will be animated. Only applicable for TurretTower.

- *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*: Dont animate the turret.

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

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

Show animation configuration**:** Click to open up animation configuration
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

Show sfx List: Click to open up audio configuration.
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 ends
Sold Sound: sfx to play when the tower is sold

Tower Description: The text to be displayed as tower tooltip.

TowerStat
Each tower type have different TowerStat. Every tower level stats can be configured as a TowerStat. Each of these TowerStat can be edit in a subEditor within a TowerEditor. They 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.

Depend on the tower type. Not all of the following configurable parameter can are applicable. If they are not relevant to the selected tower's tower type, they wont be showing up in the TowerStat subEditor.

Cost: The cost to build the tower (for level 1) or upgarde to current level (for subsequent level). Support Multiple resource type. TowerEditor will auto detect the setting in ResourceManager and change the configurable parameters appropriately.
BuildDuration: The build duration for level 1 or upgrade duration to subsequent level.

Damage: damage caused upon every attack.
Cooldown: The duration in second between each attack for TurretTower and AOETower. The duration in second between each resoruce gain for ResourceTower. The duration between each emission of of ShootObj for SupporTower.
Range: The range of the tower.
AoeRadius: The aoe radius. unit within this radius of the projectile hit point will be hit. Set to zero to disable aoe effect .
StunDuration: The duration which the unit hit will be stunned.

**SlowEffect* – slow the unit movement.

Duration: The duration of the slow effect
SlowFactor: Modifier factor of how much the affected unit will be slowed, takes value from 0.0 to 1.0.

DamageOverTime – apply a certain amount of damage over a fixed duration, DOT effect stacks.
Damage: Damage value per tick
Duration: The duration of the effect
Interval: The duration in section between every tick

**_BuffEffect_ – buff that will be applied to friendly tower within range. 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.
DamageBuff: modifier factor that will be applied to the damage of a buffed tower.
CooldownBuff: modifier factor that will be applied to the attack cooldown of a buffed tower. Value are limited between -0.8 to 0.8.
RangeBuff: modifier factor that will be applied to the range of the buffed 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.

ShootObj: 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.
TurretObj: 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.
BaseObj: 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.

**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;.

** 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 reducation. 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(TurretObject.cs)
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

UnitTower uses following global enum type:

enum _TowerType{TurretTower, AOETower, SupportTower, ResourceTower, Mine}
enum _TargetMode{Hybrid, Air, Ground}

<u>UnitTurret have following public variables:</u>
type: the type of the tower, see enum **_TowerType**
unitName: the name of the tower
icon: the icon of the tower
description: the description of the tower
targetMode: the target mode of the tower, see enum **_TargetMode**

<u>UnitTurret have following public methods:</u>
public int GetLevel() - return current level of the level
public float GetDamage() - return current active damage
public float GetRange() - return current active range
public float GetCooldown() - return current active cooldown
public float GetAoeRadius() - return current active aoeRadius
public float GetStunDuration() - return current active stunDuration
public Dot GetDot() - return current active dot
public Slow GetSlow() - return current active slow
public BuffStat GetBuff() - return current active buff
public int[] GetIncomes() - return current active incomes
public bool GetMineOneOff() - return mineOneOff
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 bool Upgrade() - call to upgrade the tower, return true if success, false if otherwise
public void Sell() - called to sell the tower

<u>UnitTower.cs fires the following events:</u>
public static event Action<UnitTower> BuildCompleteE
- fired when a tower has finished building or upgrading

public static event Action<UnitTower> DestroyE
- 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.

Optional addition to a creep prefab would be an hitpoint 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.

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.

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.

Configurable
Unit Name: The name of the unit. Just for description and UI purpose
Icon: The icon of the unit. Just for UI purpose
HPAttribute: A sub-class to store all the relavent information about HP

- **Full HP**: the maximum HP of the unit creep.
- **HP**: the current HP of the unit creep.
- **Overlay HP**: the object of the hitpoint overlay. This object has to be a plane primitive
- **Overlay Base**: the background object of the hitpoint overlay. This object has to be a plane primitive
- **Always Show Overlay**: Check to show overlay HP all the time. Uncheck 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 heightOffset from the ground and ignore all obstacles when moving pass a field.
Flight Height Offset: heightOffset from the ground. Only used when flying is checked
Value: The amount of resource player gain for killing this unit creep. Support multiple resource type.
Spawn Effect: visual effect to be shown on the unit creep when it's being spawned. This is optional.
Dead Effect: visual effect to be shown on the unit creep when it's killed. This is optional.
Score Effect: visual effect to be shown on the unit creep when it's has reach it's final waypoint. This is optional.

Animation Body: the animate object for the unit creep if there's one.
***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.
Audio Spawn: Sound fx to be played when the unit creep is spawned.
Audio Hit: Sound fx to be played when the unit creep is hit.
Audio Dead: Sound fx to be played when the unit creep is dead.
Audio Score:Sound fx to be played when the unit creep have reached it's final waypoint.

SpawnUponDestroy: The object to be spawned when this unitCreep is destroyed. This object has to be a unitCreep object.
SpawnNumber: The number of SpawnUponDestroy object to be spawned

* Optional and only applicable if there's a AnimationBody assigned. If there's more than one animation is assigned to the array, a random one will be selected when the it's requried to play the animation

ShootObject (ShootObject.cs)

All objects emit by towers 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, they are:

Projectile: A point to point object that will be shoot from turret to target. Support simulation a shoot trajectory.
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 laserbeam 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
(Please note that not all configurable in the list below are applicable for every shootObject type. But you need not worry about that, the editor will sort that out and shows only parameter that are matters)

Type: choose one of the four type for the shootObject

Speed: the travel speed of the projectile
*__MaxRange__: 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.
*__MaxAngle__: 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

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

LineRenderer: 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.
ActiveDuration: 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.
ContinousDamage: 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.

EffectType: For Effect type shootObject only. Select ParticleSystem if default ParticleSystem is used or Select legacyParticleSystem if legacy particle system is used

ShootSound: the sound to be played when the shootObject is fired
HitSound: the sound to be played when the shootObject hit something

shootEffect: The gameObject intend as visual effect to be spawned at the shootObject's position when the shootObject is fired.
hitEffect: The gameObject intend as visual effect to be spawned at the shootObject's position when the shootObject hit it's target.

2. Optional Component

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 centred around on. When moving, the parent transform is moved instead of the camera transform.

Configurable

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.
RotationSpeed: Sensitivity for rotation input (for iOS only)
MinPosX: minimum x-axis position of the camera's parent transform in world-space.
MaxPosX: maximum x-axis position of the camera's parent transform in world-space.
MinPoxZ: minimum z-axis position of the camera's parent transform in world-space.
MaxPosZ: maximum z-axis position of the camera's parent transform in world-space.
MinRadius: minimum distance of the camera component from the parent transform.
MinRadius: maximum distance of the camera component from the parent transform.
MinRotateAngle: Minimum angle of the camera from horizontal plane. Limited to 10.
MaxRotateAngle: Miximum 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

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 finish building
Tower Sold Sound: sfx to play when a tower is built

*all music and sfx are optional, if leave 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

UpdateRate: The update per second for the map.
MinimapLayer: The number ID of the layer to be used exclusively for the minimap. Make sure no other **gameObject** is using the layer to avoid artefact on the minimap.
MapRect: The rect information used to describe the position and size of the map on screen

- **X**: The top-left corner of the minimap's x-coordinate on the screen
- **Y**: The top-left corner of the minimap's y-coordinate on the screen
- **Width**: The width of the minimap in pixel
- **Height**: The height of the minimap in piexl

MapCenter: The center point Vector2(x, y) of the map to be shown in world space.
MapSize: The size of the area covered by the map in world space in Vector2(x, y);
MapTexture: The texture to be used as the map.
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 at bottom of the map.
- *Left*: The Panel will appear on left side of the map.
- *Right*: The Panel will appear on right side of the map.

TrackObj: The Object to track. The minimap will centred around the object when zoomed.
TrackPosition: Checked to enable the minimap to follow the trackObj's position.
TrackRotation: Checked to enable the minimap to rotate along with the trackObj's rotation.
Trackables: 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.

- **Layer**: The layer of the track object.
- **BlipTexture**: The texture which will be used for the blip of the track object.
- **BlipSizeModifier**: The blip size
- **MaxNum**: Maximum number of the object in scene. This is just to pre-spawn the blip
- **IsStatic**: Check if the trackObjt is a static object which have fixed position and rotation.

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

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.

3. Support Utility

These are the support components of the TD gameplay, namely path-finding and object-recycler. They dont 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

Connect Diagonal Neighbour: Checked to connect neighbouring diagonal node

PathFinder **(PathFinder.cs)**

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

Configurable

Path Smoothing: Apply path smoothing after a path is found

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:

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, bool stack)
public static void New(Transform object, int number, bool stack)
public static void New(GameObject object, int number)
public static void New(Transform object, int number)

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.

- object: the object to create
- number: the number to create
- stack: indicate if the amount of new object should be stack onto the existing pool. If 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 void Spawn(GameObject object, Vector3 pos, Quaternion rot)
public static void Spawn(Transform object, Vector3 pos, Quaternion rot)
public static void Spawn(GameObject object)
public static void Spawn(Transform object)

Called to spawn object, 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.

- object: the object to instantiate
- pos: position which the object will be appear
- rot: rotation of the object when spawn

ObjectPoolManager.Unspawn
public static void Unspawn(GameObject object)
public static void Unsapwn(Transform object)
public static void Unspawn(GameObject object, float duration)
public static void Unsapwn(Transform object, float duration)

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().

- object: the object to Destroy
- duration: the delay before the object is actually deactivated

ObjectPoolManager.ClearALL
public static void ClearAll()
Called to destroy all the objects in the manager.

4. UI & UI-utilities

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

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 surounding the build point.

Build Mode: The building scheme.

- *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.

FastForward Speed: The time speed up modifier when fast-forward button are pressed.
Next Level: Scene's name to be loaded when menu button are pressed.
Main Menu: Scene's name to be loaded when menu button are pressed.

UI-iOS (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

FastForward Speed: The time speed up modifier when fast-forward button are pressed.
Next Level: Scene's name to be loaded when menu button are pressed.
Main Menu: Scene's name to be loaded when menu button are pressed.

General UIText: GUIText for displaying general information
****Spawn Button**: Spawn button.
***FF Button**: Fast-forward button
***Upgrade Button**: Upgrade button
***Sell Button**: Sell button
GeneralBox: GUITexture for the general menu dialog. (contains menu, restart and nextLevel button)
***Menu Button**: Menu button, only shows up when game ends.
***Restart Button**: Restart button, only shows up when game ends.
***Next Lvl Button**: Next Level Button, only shows up when game ends.
AlwaysEnableNextButton: Check to always show next level button. If left unchecked, the button will not shows up if the level is failed.
Selected Tower UIbox: GUITexture for the selected tower display dialog
Selected Tower UIText: GUIText for the selected tower display dialog

*this is custom button class, see CustomButtoniOS for more detail

<u>CursorManager</u> **(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.

<u>Configurable</u>
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.

<u>UIRect</u> **(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.

UIRect.IsCursorOnUI
public static bool IsCursorOnUI(Vector3 point)
- 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

<u>CustomButtoniOS</u> **(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**
- button that will triggered as long as it's pressed. Inherited from GUIButton.

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

<u>Common Class Variable</u>
***public int* ID**: 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.
public GUITexture **buttonObj**: The GUITexture object that will act as the button.
public Texture **unpressedTex**: Button texture when idle.
public Texture **pressedTex**: Button texture when pressed or in alternate state
public bool **triggereOnPressed**: 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 void ButtonPressedCallBack(int ID)
public ButtonPressedCallBack **callBackFunc**: The callback function for the button instance. This can only be assigned via code.

<u>Common Class Method</u>
IEnumerator **Update()**
GUIButton class equivalent of MonoBehaviour.Update(). It's a coroutine and there fore has to be started using StartCoroutine().

<u>Example Code:</u>
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.

5. 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.cs
The base class of the creep and tower unit.

UnitUtility.cs
Contains utility method for unit

OverlayManager.cs
Component that govern build/upgrade overlay bar

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.
Following function display a string of text on screen with the passing argument message being the string of text to be display:
- public static void GameMessage.DisplayMessage(string message);

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

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 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 doesnt 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 not follow thought the path taken correctly.
- 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