PathFinder Toolkit version 1.3.3 created by Song Tan

1.0 Introduction & Features

1.1 What is TDTK

TDTK is designed and made as a code base toolkit for easy construction of TD game of any kind. It also meant to be served as a example project for beginner or maybe intermediate user of Unity3D. The TK comes with a bundle of scripts that is flexible and configurable to fit a variety to gameplay scenario. Following version 1.3, the toolkit is now iOS compatible. The toolkit included some example scenes/prefabs with visual special effect to show how the script can be used. In most case these visual effects and some of the component in the toolkit can also be apply on other projects Here's a list of the primary feature in this TDTK. Some features might have been left out.

Configurable Level

- Configurable resource, player's life, number/type of build-able tower for every level
- Configurable number of wave, creep type for each wave, number of creep for each wave, interval for each wave.
- Configurable path and spawn point (multiple spawn point and path supported).
- Options of two play mode (fixed path or open-field play)
- Options of two build mode (towers build on grid or build on designated platform)
- Options of two deploy mode (drag and drop or point and build)

GUI

- Two preset GUI type, scripted OnGUI() function and gameObject-based GUI-elements
- Tooltip showing tower stat and info in building selection
- Selectable tower after being built for stat display, upgrade and selling.
- Range indicator and selection indicator for tower when selected.
- Hitpoint Overlay for creep/unit to display hitpoint

SpawnMode Supported:

- Continous spawn based on time.
- Continous spawn when a wave is cleared.
- Round-based where each round (a wave) require user initiation.
- Option for user to skip a the waiting time for a wave for continuous mode

Creep/Unit:

- Configurable hitpoint, speed, resource gain upon unit destroyed
- Auto determining of waypoints.
- Compatible to use with custom model with custom animation.
- Can be assigned a custom effect (ie. explosion) to display upon reaching end of path or destroyed

Tower:

- Two build mechanism, build on pre-placed tower platform or build on pre determined grid
- Configurable damage, range, cooldown, aoe, slow effect, shootobject, targeting, upgradable level, upgradable stat, resell value.
- Compatible to used with custom model tower base and turret.
- Support model change upon tower upgrade.
- Auto setup component and parameter depend on model used.

Tower firing object:

- Projectile: basic projectile type, with configurable speed, projectile curve
- Missile: a missle type, with configurable speed, curve pattern

- Beam: a straight laser beam
- Shockwave: aoe wave type, with configurable appearance such as colour and fade out range.
- Projectile, missile and beam can be assigned any custom hit effect to display upon hitting the target.

Example and Effects:

- This package included a number of example example prefabs to demonstrate how the base script can be used or configure to its' best effect.
- These example include a number towers, shootobjects, creeps as well as two example scenes with different type of TD game play.
- The package included a number of effects ready to be used in this or other game.
- This effects are used in the example scene and the example prefab. They are mostly use as shootObject hit effects or special tower effects

Minimap:

- Show creeps/units, tower, creeps' path and camera position and field of view(fov).
- Configurable position and size of the minimap on the screen.
- Map scale can be adjusted during runtime. (zoom in/out)
- Configurable blip appearance (texture) and size for creeps, towers and the paths as well as camera position and fov.
- Run as a active rotating map with camera or a static map, configurable during runtime.

Camera(typical mmo character camera):

- Pan on horizontal plane using keyboard or mouse at the edge of screen
- Rotate camera around pivot.
- Zoom in/out using mouse scroll.
- Touch input to zoom and pan

Audio

- Audio Manager enable playing of music as well as different audio clip for different event such as new wave spawn, tower built, tower upgrade, level lost... etc.
- Each unit/creep and shoot object can be assigned individual sound clip for appropriate event

1.2 Important (Please read)

Before start using any of the scripts or running any of the example scene, it's important to do the following step:

Declare following tag, Please note that the they are case sensitive.

- TowerPlatform
- Tower
- OverlayGroup
- Path
- Creep
- HUD
- GRID

The next step is optional but is strongly recommended. it may not affect any of the script execution but do so will be helpful when trying to understand how the code actually works.

Declare following layer:

- User Layer 8: Creep
- User Layer 9: CreepFlying
- User Layer 10: Obstacle
- User Layer 11: Tower
- User Layer 12: Minimap
- User Layer 14: Projectile

- User Layer 15: Missile
- User Layer 16: Beam
- User Layer 17: Shockwave
- User Layer 18: Particle
- User Layer 20: Grid
- User Layer 21: Platform
- User Layer 23: GUI

Also Please note that the GUI elements in this toolkit are configured for the game to run in a screen resolution of at least 1024x768 or higher. They are set to position themselves according to the ratio of the screen so any resolution lower than 1024x768 will result in the UI elements being cramped together.

For iOS scene, the GUI elements are configured to fit the screen of iPhone in landscape. You may need to adjust the GUI-elements if you are building scene for other orientation.

1.3 How to use this ToolKit

Here a general step you could follow to use this toolkit to make your own TD game.

1. Decide what are the type of TD game you want to create
2. Setup up all the towers that are present in your game or only those that are in this level. This means create a tower prefab for each tower type you have.
3. Setup all the creeps that are present in your game or only those that are in this level. This means create a prefab for every type of unit you have, one for every wave of unit that have different stat or appearance. You can create boss creep as well as normal creep
4. Create the gamecontrol element by setting up the script "gamecontrol" and "spawn_manager".
5. Configure the gameplay element to what you wanted, ie spawn mode, timer, resource usage, tower placement.
6. Setup the scene by placing the waypoints, and specific tower placement point if there's any.
7. Assign the tower your setup earlier to the "GameControl".
8. Assign the creeps you setup earlier to the "SpawnManager".
9. Basically at this point the scene is done, what's left is a little fine tuning here and there like balancing the gameplay or adding in minimap and audioclip for the level.

1.4 Example Scene

There are 3 example scene included in this toolkit showing how all the key component can be put together.

- **example_grid**: a very simple level using grid based tower deployment. Creep spawns in fixed duration.
- **example_grid_PnB**: similar to example_grid. Only difference is this one use a point n build scheme.
- **example_platform**: a simple level using pre-placed tower platform deployment. Creep spawns in fixed duration.
- **example_dual_spawn**: a more complicated setup involve two spawn point where creep from each spawn point moves in different path
- **example_open_field_1**: a simple open-field level with one spawn point and one end point.
- **example_open_field_2**: an open-field level with tow spawn points and two end points, as well as other obstacle.
- **example_grid_PnB_iOS**: iOS example of grid mode
- **example_platform_iOS**: iOS example of platform-based
- **example_open_field_iOS**: example of open-field mode.

1.5 Setting up a Scene from sketch

For the toolkit to work on a minimum level, 4 components are required. These are:

- **Control&GUI**
- **SpawnManager**
- **HUD/HUD_iOS** (only if a GUI type is set to HUD)

- **Waypoints**
- **NodeGenerator** (only in a open-field mode)

There are some scene-prefabs located under folder “example_asset/scene_prefabs”. Simple drag one of each into the Hierarchy tab to create a simple level. Components need to be connected in order for everything to work. For instance, the SpawnManager need to be assign to control, Waypoints need to be assigned to SpawnManager and so on.

These components can be further configured to create level of different variety. However these are just basic prefabs, build-able tower-prefab needs to be assigned to GameControl components if there are to be any towers available to be built. So are wave of creeps to be spawned need to be assigned to spawnManager.

Secondary components in the “scene_prefabs” such as Minimap and Camera are optional to build level as seen in the example-scene. However the gameplay elements will not be affected with these

2.0 General GameControl and GUI

GameControl is the component that held everything game elements together. Elements such as towers building and creeps spawning, creeps pathing, resource control and game start/end condition. An example prefab of this component "Control&GUI" of this toolkit can be found at folder named "scene_prefabs".

2.1 Script – "GameControl.js"

A master script use to configure and take control of the game flow by processing all of things such as spawn, resource, player life, build-able towers, GUI. Every scene should have at one “GameControl” script. The configurable parameters are as follow:

- **GUIType:** Select between two GUI mode
  - **DEFAULT**: use scripted OnGUI()
  - **HUD**: use pre-configured GUI-elements (HUD prefab must be in the scene)
- **Play Mode: Select between two play mode**
  - **FixedPath**: creep follow a predetermined waypoint, towers cannot be built on the path.
  - **OpenField**: creep will find their own shortest path to the destination, player may build formation of towers to block the path.
- **Resource 1**: The main resource used to build and upgrade towers
- **Resource 2**: The alternate resource, can be enabled for upgrading towers. Only active and shown in the game GUI if 'Use Resource 2 For Upgrade' is checked.
- **Life**: The player life when game start.
- **Tower**: A flexible sized array used to specify the build-able tower-prefab in this level. (see example scene of how the this is assigned
- **Enable Fast Forward**: Check to enabled time fast forward.
- **Fast Forward Multiplier**: The multiplier to the increase in time scale. Only takes effect when 'Enable Fast Forward' is checked.
- **Use Resource 2 For Upgrade**: enabled resource 2 and use it as a currency for upgrading towers
- Resource1 Name: The resource1 name to be displayed on the UI
- Resource2 Name: The resource2 name to be displayed on the UI
- **Selection Indicator Scale(x, y)**: The scale of the selection indicator in the game scene when a tower is selected. x for horizontal and y for vertical.
- **Selection Indicator Offset(x, y, z)**: The xyz position offset for the selection indicator in the game scene when a tower is selected.
- **Animate Selection Indicator**: Check to animate the the selection indicator.
- **Spawn Manager**: A flexible size list to used to assigned all “SpawnManager” in the scene. (See section 3.1 for further explanation of this)
- **Next Level**: The next level to load after this level is finished. This is the name of the scene to be loaded.
- **No Rsc Building Before Start**: Check to disable building of resource tower before the game start.

This is to prevent player stacking up resource towers before the game begin
- **Enable Game Message**: Check to enable display of general game message such as insufficient fund.

There are two resource available in gameplay. Typical tower building will always use resource 1, resource 2 is an optional extra to be used for upgrading tower. Think it as some sort of experience count. This option is there with TD that utilise experience for upgrading. Regardless of using whatever resource type for upgrade, selling towers will only refund resource1. A fair point consider if tower is upgraded using experience, selling an upgraded tower is not going to regain experience.

There are two preset GUI, dubbed HUD and DEFAULT. HUD is recommended for easy re-position of the layout. HUD is also more performance friendly as OnGUI() function does reduce the frame rate

In order for the script to work properly, there has to be a child object "selection_indicator" assigned to it. The selection_indicator is basically a guiTexture with the tower selection indicator texture. There's a default prefab of it in folder "resources". If there's no such child object attached to the gamecontrol object, the script will attempt to load the default "selection_indicator" prefab in the resource folder.

2.2 Script – "TowerDeploymentSetting.js"

There are two build mode available for tower. Grid based deployment and platform base deployment. The two example scenes are both build to demonstrate each of this build-mode. They are named after the tower build mode they are using.

A virtual grid is laid out on the terrain and the tower is placed according to the grid. If a square of the grid is occupied, either a path is in the way or a tower is already build in the square. The gird size can be customised. Under this build-mode, a tower can be build almost anywhere except on creep's path or a square occupied byt other tower. To Restrict tower bieng build on certain spot. Place a object with a collider there and tag it as "Path".

Second build mode is platform-based placement. The tower can only be placed on certain platforms predefined by user. This platforms needs to be an gameObject with collider tagged "TowerPlatform". An example prefab of tower platform is placed in folder "example_asset/tower".

This is the script attached to gameObject which the “GameControl” script has been attached on. This configuration of this script will determine how all the tower in the scene should be build. Like “GameControl”, only one “TowerDeploymentSetting” script is needed in one scene. It' vital that this script is attached to the "gamecontrol" gameobject otherwise tower deployment wouldn’t work. The configurable parameters are as follow:

- **Build Mode:** Determine weather tower is going to be build on a pre-determined grid or on fixed-predetermined object. There are two build mode:
  - **GRID**: The tower will be placed based on a grid system
  - **PLATFORM**: The tower can only be placed on object with tag “TowerPlatform.
- **Deploy Mode:** Determine user input method to indicate where a tower should be build. There are two deploy mode:
  - **DRAGnDROP**: user select a tower from UI button before dragging the tower to the desire build position.
  - **POINTnBUILD**: User click the a available build position on the map before select the tower to be build from the UI button.
- **Height Offset:** The height at which the tower should be build. For Grid Mode, this is simply the y-axis position. For Platform mode, this value indicate the y-axis offset from the centre of the platform. **Please note** that this is only meant to enable slight adjustment to the tower placement. Try to keep the value in proportion so the tower is still more or less on the “ground”. This is especially true for open-field mode with path-smoothing. Since the path-smoothing is based on line-of-sight, setting this value too far off from the path-height will resulting in the towers not effectively blocking the line-of-sight, thus it wont stop the creep from moving through towers.
- **Grid Size**: the size of gird in current scene. Only applicable if Grid Mode is selected.

2.3 Script - “ControlGUI.js” and Script - “HUD.js”

These are two script responsible for the UI. They need no configuration or during setup of a scene. Each script will be added automatically if required. “ControlGUI” are where all the scripted GUI based on OnGUI() function is written. “HUD” on the other hand control the interaction of all the elements on the GUI-elements based HUD UI.

2.4 HUD prefab

HUD prefab is the default gameObject for a GUI-elements based GUI setup. Within it's hierarchy contains all the GUI-elements object that is required for a “HUD” GUIType setting. This prefab must present in the hierarchy tab if a HUD “GUIType” is set in the “GameControl”. While the layout/position/style of the of components can be adjusted without any real impact. The name of each component is vital to the execution of the script. Unless you change the script “HUD”, don't alter the name of the components.

Following version1.3, iOS designated HUD prefab has been added. It's named HUD_iOS and is placed along with the default HUD prefab in the “/example_asset/scene_prefabs” folder. The major difference between the two HUD is the HUD_iOS used 3D text and plane object as the GUI-elements. The reason being when using same material for this GUI element, unity will dynamically batch them up to reduce drawcall. To show them properly as GUI element, a second camera (the root object of the HUD hierarchy) has been used and these elements are placed relatively with respect to the camera. Furthermore, the entirely hierarchy has been layer “GUI” (layer23) and the main camera's layer mask has been set to ignore this particular layer.

It should be noted that all iOS scene must use this prefab and this setup. The layout/position/style of the GUI can be configured at will. But the changing the type/hierarchy/name of the setup would require modification of the script.

2.5 Script “PointBuildControl.js”

This is the script used to control tower build scheme for POINTnBUILD control. There's no configuration or setup whatsoever required with this script.

3.0 SpawnManager & Waypoint

3.1 Script – "SpawnManager.js"

the script used to control the spawning of the creeps/ units. Ever aspect of the spawn mechanic can be configured using this script. For each spawn point and path in the level, an instance of “SpawnManager” is required. Each the “SpawnManager” components in the scene needs to be assigned to the “GameControl” script in the scene in order to be initiated. This can be done by dragging the gameObject which contains the “SpawnManager” component in to the “Spawn Manager” paramter in “GameControl”.

- **Spawn Mode**: Determine how the wave is spawned. There are five selectable spawn mode
    - **FixedIntervalContinous**: A new wave is spawn upon every wave duration countdown
    - **WaveClearedContinous**: A new wave is spawned when the previous wave is cleared
    - **FixedIntervalContinousWithSkip**: Similar to FixedIntervalContinous but user can initiate the next wave
    - **WaveClearedContinousWithSkip**: Similar to WaveClearedContinous but user can initiate

the next 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.
- **WaveList:** An array of used to configured information of each wave. The size of the array is flexible to allow any number of wave to be spawn in a single level. The waves are spawned in chronological order of this array. The detail of each configurable parameter are as follow:
    - **Units:** The unit information of the wave. Multiple unit type can be assigned to a single wave.
        - **Unit:** The transform of unit to be spawned in corresponding wave.
        - **Count:** The number of unit to be spawned in corresponding wave.
        - **Delay:** The delay for this particular creep to be spawn after this wave is initiated.
        - **Interval:** The time duration in second between the spawning of each unit.
    - **Wave Interval**: Time in seconds until next wave is initiated. Only effective is Use Individual Interval is checked.
    - **Resource 1Gain:** Resource 1 gained when the corresponding wave is cleared.
    - **Resource 2Gain:** Resource 2 gained when the corresponding wave is cleared.
- **Use Individual Interval**: Check to use individual wave interval value in wavelist instead of a Global Wave Interval.
- **Global Wave Interval**: Time interval in seconds between each wave. Parameter makes no effect if a non time based spawn mode is activated.
- **Wave Interval Increment**: Check to increase the value of interval after each wave. (If each wave is getting more unit spawn. More duration might be needed to clear the wave). Only apply to Global Wave Interval.
- **Wave Interval Increment Rate**: number of seconds to increase for 'Wave Interval' if 'Wave Interval Increment' is checked. Only apply to Global Wave Interval.
- **Show Timer**: Check to show a count down timer before next wave is spawned in run time. Only recommended to have this checked if a time-based spawn mode id activated
- **Start Delay**: Time delay in second before the spawning start after player start the game the wave.
- **Creep Path**: The waypoint gameObject that contains all the waypoint for the creeps
- **Generate Path**: Check to Automatically generate gameObject as visual clue to the creep's path.
- **Path Height Offset**: the height offset of the automatic path object generation. Only takes effect if 'Generate Path' is checked.
- **Show Path in OF Mode**: check to enable gizmo to display the path of this spawnManger instance. This only apply in open-field mode. Please note that this is only visible in Unity Editor.

*Note on confugurating spawn manager:
- Some of the variable will be auto-reconfigure if the condition is not compatible with the level setup.
- Only FixedWaveContinous and WaveClearedContinous spawn mode are supported in scene contain two or more “SpawnManager”.
- Show Timer will only apply if a time-based spawn mode is selected. So are Wave Interval and it's relevant parameter.

3.2 Creeps/Unit pathing and waypoints

To create a valid path for the creeps/units. You need to place all the waypoints under one parent common empty gameObject. The child waypoints can be any gameObject. They should be position appropriately to represent the path. Each of the child should be attached a script "Waypoint.js" and assign a "waypoint ID". This waypoint ID should be be in ascending order from the first waypoint in the path to the last waypoint in the path. ie. First waypoint(the spawn point) should be given "waypoint ID" of 0, the next waypoint 1, the waypoint after 2 and so on. Please refer to example prefab "waypoints" placed under folder named "example_asset/scene_prefabs". The parent gameObject should be the waypoint gameObject assign to "Creep Path" of the spawn manager.

The waypoints just the same in an open-field level. However you will only need 2 waypoints for every spawnManager, a spawn point and a destination. The path-finding component will sort out the rest.

4.0 OpenField Play and PathFinder

4.1 Open-field mode

Open-field mode can be set using Play Mode parameter in “GameControl.js”.

As expected open-field play mode would require some sort of path-finding in order for the creep to navigate through any maze of towers the player might have built. To setup the path-finding components, the navigation node needs to be setup. This also give us a chance to define where the “field” area is taking place and where the boundary is. As you would have expected, the creep will only determine a path through where there is a navigation node.

As of version 1.3.1, the player can no longer build towers to block all possible route.

4.2 Script - “NodeGenerator.js”

This is the script that auto-generate the navigation node. It can be attached to any gameObject in the scene for it to work. However the path-finding will only work if this script is actively running in the scene. For that matter, **open-field play will not work without NodeGenerator in the scene**.

The script will do a raycast in every grid point. If there's a collider in the field area defined, the navigation node will be generate on top of the collider. Otherwise it will be generate at point where y-zxis value is zero. For the best result, it's important to define the area to cover both the spawn point and destination.

- **Area**: The area of the field
    - **X**: The starting x coordinate.
    - **Y**: The starting z coordinate.
    - **width**: The length of the boundary in x-axis from x starting coordinate.
    - **Height**: The length of the boundary in z-axis from y starting coordinate.
- **Connect Diagonal Neighbour**: Check to allow the creep to move to a diagonal tile
- **Agent Height**: The height of the unit, this will determined the offset of the node from the ground, which is half the value of this height.
- **Show Gizmo**: Check to enable the node to be drawn by the gizmo, for debug purpose.

4.3 Script - “PathFinder.js”

This is the script component where the path finding algorithm is run. For the most part you can ignore it unless you want to modify the algorithm for whatever reason. There is only one option for this component:

- **Path Smoothing**: Apply Line of sight path smoothing to the path found. Creep will not move from a node to node, rather it takes shorcut, resulting in multiple movment direction other than the normal typical 8 common one.
- **Scan Node Limit Per Frame**: The maximum node that will be scan in a single frame. You can adjust this value to suite the hardware spec of your target platform. Lower it if you are aiming for a platform with lower processing power. Typically a mid-spec PC can take value of 500.

5.0 Towers

5.1 Tower Scripts

There are 2 scripts associated with tower

5.1.1 Script – "Tower.js"

The main operating script for tower. The configurable parameters are as follow:

- **Tower Name**: The name of the tower which will appear on in game GUI and tooltip.
- **Icon**: The tower icon used in the build menu. If left unassigned tower name will be used instead.
- **Tower Type**: The type of the tower. There are three tower type in this toolkit
  - **NORMAL**: Standard tower that aim and shoot at a target
  - **EFFECT**: Special tower which deal damage effect within a certain radius.
  - **RESOURCE**: Special tower that doesn't attack. It generate resources overtime.
- **Anti Target** Type: The target type which this tower will shoot at. There are three:
  - **AIR**: Target air unit/creep only.
  - **GROUND**: Target ground unit/creep only.
  - **HYBRID**: Target both air and ground unit.
- **Cost**: The resource cost to build the tower. Tower building cost will always be resource 1.
- **Damage**: Damage of the tower cause when attacking. Or amount of resource gain for resource tower.
- **Range**: Range of the tower (must be >0, will auto reset to 1 when set to <0)
- **Cooldown**: the cooldown time of the tower in second after each attack.
- **Aoe**: a value indicate how big the Area-of-Effective(aoe) of the tower when inflicting damage. Set to zero if the tower is non-aoe.
- **Slow**: Time in second indicate how long a unit will be slowed if hit by the tower. Set to zero if the tower doesn't slow down its target. The magnitude of slow is depend on the target and can be configure on script "Creep".
- **Shoot Object**: The shoot object to fire when attacking. (this should be a prefab configured using one of the four Shoot Object script). A prefab has to be assigned for a standard tower (non-field tower and a non-resource tower) to function properly.
- **Animated**: Checked to animate the tower. When checked the turret of the tower will actively aiming at it's target. Irrelevant to Field Effect Tower and Resource Tower.
- **Turret Object**: The turret object of the tower. This is the object that will be animated when the Animated flag is checked. This is also the object that will be update with a new model when the tower is upgraded if there's any upgrade model specified.
- **Base Object**: The oject act as the base to the tower. This is the object that will be update with a new
- model when the tower is upgraded if there's any upgrade model specified.
- **Shoot Point**: A dummy object used to mark where the shoot object should be fired from, if it's different from turret object's centre point. Typically this should be placed under the hierarchy of turret object so it moves and rotate according to the turret object rotation when aiming at target. Example tower prefab "TowerCanon" has been setup with such setting. Similar setup can be used for upgrade tower turrent. The script will auto search of child object named "SP" of new turret to assign it as the the shootPoint. If left unassigned, the shoot object will be fired from turret object's centre.
- **Fire Sound Effect**: Sound clip to be played whenever the tower fire a shoot object.
- **Sound Effect Range**: Effective range where the sound effect will be play at full volume. The volume graudally lower as the camera range from the tower increase.
- **Upgrade Cost**: This is an flexible sized array use to store the cost required for upgrade at each tower level. The size of the array can be specify to any length. Each element in the array is indicating the cost required to each corresponding level upgrade. ie. The cost to upgrade from level zero to level one is 10, so Element 0 is 10, he cost to upgrade from level one to level two is 15, so Element 1 is 15 and so on.
- **Upgrade Damage**: This is an flexible sized array use to store the new damage value for upgrade at
- each tower level. It's configured the way Upgrade Cost is configured.
- **Upgrade Range**: This is an flexible sized array use to store the new range value for upgrade at each

tower level. It's configured the way Upgrade Cost is configured.

- **Upgrade Cool Down**: This is an flexible sized array use to store the new cooldown value for upgrade at each tower level. It's configured the way Upgrade Cost is configured.
- **Upgrade Turret**:: This is an flexible sized array use to specifty the new turret model for upgrade at each tower level. It's configured the way Upgrade Cost is configured. (optional)
- **Upgrade Base**: This is an flexible sized array use to specifty the new tower base model for upgrade at each tower level. It's configured the way Upgrade Cost is configured. (optional)
- **Upgrade Effect**: The effect to be shown whenever a tower is upgraded. (optional)
- **Build Effect**: The effect to be shown whenever a tower is first build. (optional)
- **Sell Gain to Cost Ratio**: The resource gained when a tower is sell in a ratio to the cost when it's built.
- **Scale Sell Gain To Level**: Check to scale the selling gain to the tower level. The additional value is calculated based on the cost required to upgrade the tower.
- **Range Indicator Texture**: The tower's range indicator texture. If left unassigned the default will be used.
- **Range Indicator Pos Y**: The height offset of the range indicator when displayed.
- **Range Indicator Color**: The color of the range indicator.
- **Tooltip Damage**: The damage term to be displayed on UI.

- Please note that all the tower prefabs has been updated since there's a slight tweak in how the tower object is setup when built.
- Note when configuring a tower, all the upgrade array (Upgrade Cost, Upgrade Damage, Upgrade Range...) must be same size.
- Since resource tower doesn’t attack, all the attack relevant parameter such as range, cooldown, aoe are ignored
- A resource tower can be configured to deal damage while generating resource by assigning an aoe effect shootobject like shockwave.

5.1.3 "Prebuild.js"

A scirpt used to govern the tower behaviour such as position, color change during the phase where the tower is being deployed. It also initiatlised the tower when it's built. This script is required for every tower along with the script “Tower” (“Prebuild” will be automatically added to the gameObject whenever a “Tower” script is assigned. You dont need to worry about this script for the most part unless you want to know what's going on behind the background or you want to change the way tower is deployed.

There's no need to attach this script components to a tower prefab.

5.2 Building a Tower Prefab

It's recommanded that all the components of the tower are kept under hierarchy of one gameObject. The "tower" scirpt should be attached to the gameObject on the top hierarchy. The models (turret, tower base, etc.) should be positioned in a way appropriate. I,e the turret rested on the tower base and so on. The toolkit does not position these model automatically. If there are any new model to be used after the tower is upgraded, the new base and turret object will be positioned based on current base and turret position. Please refer to the example prefabs.

5.3 Example Prefabs

There are seven example tower prefabs. Despite their difference all the example tower prefabs share the same models, even when upgraded. There are 4 turret models and 4 tower base models placed in folder named "tower_primitive_model_base" and "tower_primitive_model_turret". They are notable in their colour difference.

The tower prefabs themselves are placed in folder named "tower_prefabs". They are name appropriately according to their own function and shootobject. All of them are configured to be upgradable to level 3.

6.0 Creep/Unit

Creep are the moving unit that act as tower's target in a TD game. There are a number of example prefabs showing a variety of complexity level a creep/unit can be setup. The simplest being creep with no overlay display, then there's creep with overlay, then creep with animation and finally creep with blob shadow. All the example creep prefabs uses primitive cube as model. However this can be replaced by any custom animated model.

6.1 Script – "Creep.js"

There is only a creep associated script. It govern all the attribute associate with the creep. All creep gameObject should have this script attached. Everything about the creep can be configured using this script. The configurable attribute of the creeps are as follow:

- **Hp**: Short for hit point. How much the damage can the unit take before it's destroyed.
- **Speed**: The value used to determine how fast the unit will move. 2-15 is just about right. Depend on the scale of the model.
- **Slowed Factor**: the magnitude in percent in which the unit will slow down when slowed by attack.
- **Flying**: Check to make the unit a flying type. A flying unit can only be targeted by tower that has anti-air capabilities.
- **Fly Height**: the flight height of a flying unit. Only take effect if the unit is a flying unit. Otherwise it's ignored.
- **Resource Gain 1**: How much resource type 1 will the player gain when destroy this unit.
- **Resource Gain 2**: How much resource type 2 will the player gain when destroy this unit.
- **Show Resource Gain**: Check to show resource 1 gained on the overlay when unit is killed.
- **Overlay Scale**: The scale to the length of the hp overlay. Increase to make the overlay bigger.
- **Overlay Height**: The modifier for the height of the overlay from center of the creep object;
- **Animated Body**: The custom animated object to be used for this unit if there's any. If no animated object is assigned, none of the animation assigned to the unit will be played. It's recommended that the animated body is placed in the hierarchy as the child of the unit.
- **Animation Move**: the move animation to be played on animated body when the unit is moving.
- **Animation Hit**: the hit animation to be played on animated body when the unit is hit.
- **Animation Dead**: the move animation to be played on animated body when the unit is dead
- **Animation Score**: the move animation to be played on animated body when the unit score against player.
- **Dead Effect**: custom effect to be instantiate when the unit died. If there's a dead animation, the effect only take place after the dead animation is played.
- **Score Effect**: custom effect to be instantiate when the unit score. If there's a score animation, the effect only take place after the score animation is played.
- **Dead Sound Effect**: audio clip to be played whenever the creep/unit is killed/destroyed.
- **Score Sound Effect**: audio clip to be played whenever the creep/unit scores.
- **Sound Effect Range**: Effective range where the sound effect will be play at full volume. The volume gradually lower as the camera range from the unit increases.
- **Death Delay Duration**: The delay in which the creep object will stay in the scene after it is "killed". The value will be override if it is shorter than the play length of "Animation Dead" or "Dead Sound Effect".

* This particular script is very well commented with lots of background explanation. It's my forfeited attempt in making this toolkit a tutorial material. If you find these comment helpful and are interested in having the rest of the script commented as well. Please contact me. I'll consider making a tutorial version.

6.2 Building a Creep/Unit Prefab and Overlay

In this toolkit, all the creep as expected to have an overlay showing the creep current hit point as well as the

text_overlay showing other relevant information such as resource gain when the unit is killed or damage inflicted when the unit is hit. The hp_overlay are basically two guiTexture gameObject with a bar texture and the text_overlay is a guiText gameObject. Please refer to example creep setup in "example_asset/creep" folder to find out how exactly they are setup. This gui elements can be customized in any mean as long as they are named as follow:

- "overlay_text" - the guiText
- "overlay_hp" – the guiTexture used to represent the current hp. (green color in example)
- "overlay_base"- the guiTexture used to represent the full hp. (red colored in example)

The 3 components must be named as stated above. Please note that they are case sensitive. In order to have the overlay element working, all these 3 components must exists.

Please note that the blob shadow projector are not necessary for the creep to work. You can remove it completely or use other technique that is suitable for your game.

6.3 iOS

Fr performance reason, the unit prefab used for iOS uses only one plane object as the hitpoint overlay. There are two iOS unit prefab under folder “example_asset/creep” with _iOS notation at the end of the prefab name.

The other notable difference of the iOS unit prefab is that a plane object is used instead of default blob shadow projector. This is also done for performance reason. Please note the shadow plane is not necessary, its just there for cosmetic. You can remove it completely or use other technique that is suitable for your game.

7.0 ShootObject

ShootObject is the gameObject that shoot from a tower to hit the target. There are generally 4 types of shootObject cover in this toolkit which should cover most, if not all possible scenario to provide visual clue as well as coding means for a tower to attack and hit a target.

7.1 Projectile

The most basic and the most widely use of shootObject in typical tower defence game. Projectile type shootObject is govern by script "projectile". The script basically propel one gameobject toward a designated target, causing damage upon 'hitting' it. "projectile" script can be use to create projectile type like canon, energy bolt, arrow, pulse laser, anything that move from the turret onto the target. To create a projectile type, simply attach the script "projectile" to the gameobject/model/particle/line renderer. Example of projectile type shootObject included in the package are:

- canon: a typical canon with a curved trajectory. A trail-renderer is added enhance the visual effect
- plasma_bolt: example of how a simple and minimal particle system can used instead of a solid gameobject for visual effect. This example can be easily modified to be a fireball or any other energy projectile. Try add a trail renderer like the example 'canon'.
- pulse_laser: example of how a line renderer can be used with the 'projectile' script to create a pulse laser effect.

7.1.1 Script – "Projectile.js"

This script contain all the function require for a projectile based shootObject. These are the attributes used to configure the behaviour:

- **Speed**: The speed which the projectile will travel.
- **Hit Pos Offest**: a modifier to adjust if the shootObject will shoot ahead of the target. Set at >0, the shootObject will aim ahead of the target. Set at <0, it will aim behind the target. The position offset are just graphical and has no effect on the game.
- **Curve Angle**: The cruve trajectory. (can only take value from 0 to 65)
- **Hit Effect**: The effect to instantiate when the projectile hit it's target. If unassigned nothing will

happen.
- **Sound Effect**: Sound clip to be played whenever the projectile object hits it's target.
- **Sound Effect Range:** Effective range where the sound effect will be play at full volume. The volume gradually lower as the camera range from the impact point increases.

7.2 Missile

Missile is basically an extension of projectile type except it has more configurable parameter to allow more complex trajectory in both x and y axis. There's only one example prefab of missile in this package. The example prefab missile object has got a trail-renderer attached as an example of how a trail-renderer can be use to enhance the visual.

7.2.1 Script – "Missile.js"

Like projectile, a missile type shoot object only require this one script. These are the attributes used to configure the behaviour:

- **Speed**: The speed which missile projectile will travel.
- **Hit Pos Offest**: a modifier to adjust if the shootObject will shoot ahead of the target. Set at >0, the shootObject will aim ahead of the target. Set at <0, it will aim behind the target. The position offset are just graphical and has no effect on the game.
- **Hit Effect**: The effect to instantiate when the projectile hit it's target. If unassigned nothing will happen.
- **Sound Effect**: Sound clip to be played whenever the missile object hits it's target.
- **Sound Effect Range:** Effective range where the sound effect will be play at full volume. The volume gradually lower as the camera range from the impact point increases.
- **Fixed Curve X**: Check to disable randomised of the trajectory angle in x-axis. Useful if you want the missile to behave more like a normal projectile. When this is checked, the trajectory of the missile in x-axis will be whatever value you assigned fro "Max Curve X" with a little deviation.
- **Max Curve X**: Maximum possible, trajectory curve in x-axis. The trajectory in x-axis is randomised from 0 to this max value assigned.
- **Max Curve Y**: Maximum possible, trajectory curve in y-axis. The trajectory in y-axis is randomised from negative to positive max value assigned.
- **Rand Coeff X**: randomised the value of coeff X from 0.5 to 1.5. When checked the randomise value will overide the value assigned.
- **Rand Coeff Y**: randomised the value of coeff Y from 0.5 to 1.5. When checked the randomise value will override the value assigned.
- **Coeff X**: A value indicate how many change of direction in x-axis will the missile have from it's source to it's target. Typically range from 0.5 to 2.
- **Coeff Y**: A value indicate how many change of direction in y-axis will the missile have from it's source to it's target. Typically range from 0.5 to 2.

7.3 Beam

Beam type shoot object is for the shoot object type that formed a line from the firing point to the target like a beam laser or energy ray. It's almost certainly need a line renderer. Try to attach the script on an object and use it as beam shoot object without adding a component will cause error. The example "beam_laser" shows how a beam type shoot object can be setup.

7.3.1 Script – "Beam.js"

Only one script require form beam shoot object. However the script will require a LineRenderer component on the gameobject in order to work. These are the attributes used to configure the behaviour:

- **Beam Duration**: The duration which the beam will stay active after being launched.
- **Continuous Damage**: Check to enable continuous damage. The damage to the target will be applied continuously through out the duration of the beam rather than one off when the beam ends.
- **Hit Effect**: The effect to instantiate when the projectile hit it's target. If unassigned nothing will

happen.

- **Sound Effect**: Sound clip to be played whenever the beam shoot object is fired.
- **Sound Effect Range:** Effective range where the sound effect will be play at full volume. The volume gradually lower as the camera range from the shoot object increases.

7.4 Shockwave

Shockwave is designed for large area aoe effect what originated from the source point cover a certain radius. It can be used directly as a shoot object of an tower generating an aoe field around the tower or as a hit effect for other shootObject. "shockwave" script has to used on a plane primitive object. There are two example prefabs for usage of shockwave:

- shockwave: a aoe effect used by example tower prefab "tower_field".
- plasma_aftershock(shockwave): configured to use as an hit effect that slow unit down within a range of 3.

7.4.1 Script – "Shockwave.js"

Similar to the rest of the shoot object script, only this script is needed for a shockwave type object. However it has to be use on a plane. These are the configurable attributes:

- **Damage**: damage cause to each unit within the radius. Only need to be assigned if the object is intend as a hit effect. Otherwise the tower's value will override it.
- **Slow**: duration of slow effect cause to each unit within the radius. Only need to be assigned if the object is intend as a hit effect. Otherwise the tower's value will override it.
- **Speed**: the speed which the shockwave is expanding
- **Radius**: The area of effective from the centre of the shockwave. Only need to be assigned if the object is intend as a hit effect. Otherwise the tower's value will override it.
- **Fade Out Point**: A ratio to Radius indicates point at where the shockwave is starting to fade visually.
- **Fade Out Extension**: A ratio to Radius indicates point at where the shockwave is still visible out of the designated radius even when the shockwave has started to fade.
- **Color Start**: Starting colour
- **Color End**: Ending colour. The starting colour will slowly converge to ending colour over the duration
- of the shockwave.
- **Sound Effect**: Sound clip to be played whenever the shockwave object is created.
- **Sound Effect Range:** Effective range where the sound effect will be play at full volume. The volume gradually lower as the camera range from the shockwave increases.

7.4.1 Script – "Particle.js"

This script is used for particle system in iOS for the particle to work properly with the ObjectPoolManager. There's no configurable parameter. Simply attach it to any particle system prefab intend to be used as shootObject, build-effect, etc.

8.0 MiniMap

The minimap is a self contained component of this toolkit. Including or excluding this component in a level wont affect other aspect of the game. To use the minimap in a level, simply add the "Minimap" prefab into the scene hierarchy. The minimap component can be used on other project. Please note that as of current version the minimap is not optimized for iOS. It should be noted that using minimap in iOS not advisable.

8.1 "Minimap.js" script:

- **Minimap Layer**: The layer which the all the minimap relevant gameObject is given. It should be an integer between 8 to 30. Any layer should be fine as long as it's not been used.
- **Map Texture**: The texture which is used as the minimap background
- **Map Size (x, y)**: The size of the map
- **X**: The length of the map
- **Y**: The width of the map
- **Trackable Tagged Object**: This is a list of the tracked object type that is intended to be tracked using tag. The tracked object need to be assigned a tag. You can specify as many type of object type to be track as you want. Once the size has been set, each trackable object has following parameters:
    - **Tag Name**: The tag of the object type to be tracked
    - **Object Blip Texture**: The texture of the blip appear on the minimap for the object type. If left unassigned, the tracked object type wont be visible on the ninimap
    - **Object Blip Size**: The size of the tracked object type's blip on the minimap. If left unassigned (value=0), it will use the scale value of the tracked object.
    - **Track In Run Time**: Check if this track object type change position during run time. Otherwise the object will remain static on the minimap.
- **Trackable Collider**: This is a list of the tracked object type that is intended to be tracked using collider. To be tracked using this mean, the collider must be assigned a custom layer. You can specify as many type of object type to be track as you want (however there's only a limited number of layer available). Once the size has been set, each track-able object has following parameters:
    - **Collider Layer**: The layer which the the collider-based track object is assigned
    - **Object Blip Texture**: The texture of the blip appear on the minimap for the object type. If left unassigned, the tracked object type wont be visible on the minimap
    - **Object Blip Size**: The size of the tracked object type's blip on the minimap. If left unassigned (value=0), it will use the scale value of the tracked object.
    - **Track In Run Time**: Check if this track object type change position during run time. Otherwise the object will remain static on the minimap.
- **Map Position On Screen (rect)**:
    - **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
- **Enable Map Rotate Option**: Check to enable the the option of rotating the minimap around the camera.
- **Enable Map Click Move**: Check to let user to re-locate their camera position by clicking on the minimap.
- **Enable Track Camera**: Check to change of the minimap centre based on Track Camera Object or the main camera object.
- **Track Camera Object**: The object to based on the minimap centre on. If left unassigned, the default main camera object is tracked instead. Only takes effect if Enable Track Camera is checked.
- **Track Camera Object Texture**: The texture of the blip appear on the minimap for the camera track object. If left unassigned, the Track Camera Object wont be visible on the ninimap
- **Track Camera Object Blip Size**: The size of the Track Camera Object's blip on the minimap. If left unassigned (value=0), it will use the scale value of the Track Camera Object.

9.0 Camera

The camera is another self-contained component. To use the camera, simply add the "Camera" prefab into the scene hierarchy and delete the default "Main Camera". The example camera moves along the x-z plane (parallel to ground) with a pivot base rotation and zoom in/out. This camera can be use on another other unity project where suitable.

9.1 Script – "CameraControl.js"

- **Scroll Mode**: The method which user use to pan the camera around
  - **Mouse**: Pan the camera when mouse cursor reach edge of the screen like a typical RTS.
  - **Keyboard**: a, s, d, w keys is used to move the camera like a typical FPS
  - **Both**: allow use to use both options above (mouse and keyboard).
- **Camera Rotate**: determine if left/right mouse click and drag to rotate the camera.
  - **LeftMouseDrag**: Use left mouse click and drag to rotate camera.
  - **RightMouseDrag**: Use right mouse click and drag to rotate camera.
- **Scroll Speed**: The speed/sensitive at which the camera is moving.
- **Zoom Speed**: The speed/sensitivity at which the camera is zooming.
- **Rot Xupper Limit**: The upper limit of the camera rotation in x-axis in angle, typically value at 90 since it is the top down view.
- **Rot Xlower Limit**: The lower limit of the camera rotation in x-axis in angle, typically take value greater than 0.
- **Vertical Limit Top**: The upper limit of the camera position in vertical(z) axis.
- **Vertical Limit Bottom**: The lower limit of the camera position in vertical(z) axis).
- **Horizontal Limit**: The limit of the camera movement in horizontal plane. It's basically a rectangle boundary stating the starting x and y coordinate and the width and length from the starting point
  - **X**: The starting x coordinate.
  - **Y**: The starting z coordinate.
  - **width**: The length of the boundary in x-axis from x starting coordinate.
  - **Height**: The length of the boundary in z-axis from y starting coordinate.

9.2 iOS

Following iOS porting, the camera script now support touch manipulation. When running the scene on a touch device, drag on the screen will pan the camera along the horizontal (x-z) plane and pinch/unpinch the screen will zoom in/out. The pan and zoom speed of the touch under touch manipulation is governed by the similar pan/zoom speed parameters used in the mouse and keyboard control.

10.0 Audio

Audio implementation are integrated into various components of the game. For instance, the sound effect of a tower firing or the sound effect of a creep being hit or dead have been integrated into the corresponding script.

10.1 Script - "AudioManager.js"

With the exception of general sound effect which is control using "AudioManger" script. The script can be attached to any gameObject in hierarchy tab as long as the gameObject is not position sensitive. It's recommended that an independent gameObject is dedicated as the AudioManager game component like in the example scene. Following the the configurable parameter of the script "AudioManager":

- **Game Music**: The main music that would be played throughout the level
- **Build Tower Sound Effect**: The sound clip to be played whenever a tower is build.
- **Upgrade Tower Sound Effect**: The sound clip to be played whenever a tower is upgraded.
- **Sell Tower Sound Effect**: The sound clip to be played whenever a tower is sold.
- **Game Won Sound Effect**: The sound clip to be played whenever the level is completed..
- **Game Lost Sound Effect**: The sound clip to be played whenever the level is lost.
- **New Wave Sound Effect**: The sound clip to be played whenever a a new wave of creeps is spawned.

All the input are optional. If none are assigned, nothing will be played.

11.0 iOS

iOS is very different from typical webplayer or stand alone build. Lots of optimization has gone in to make sure the toolkit can maintain reasonable(20++) and stable frame per second when running on a mobile device.

A few example scene for iOS has been setup. It should be noted that these scene can only be run without error in the editor if the target platform is set to iOS or on an actual iOS device. The difference between an iOS scene and the normal scene are as follow:

- **HUD**: use optimized HUD prefab (with _iOS notation at the end of the prefab name). Refer to section 2.4 for further detail.
- **Creep**: use optimized creep (with _iOS notation at the end of the prefab name). Refer to section 6.3 for further detail.
- **Minimap**: minimap are disable in iOS for performance reason

Other change include camera, which auto switch to touch manipulation on the device (refer to section 9.2).

12.0 ObjectPoolManager (ObjectPoolManager.js)

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.

This manager create and keeps track the pool of objects. You can use it in any other project. The script doesnt need to be active in the scene,. As long as it's in the asset-tab, it should work. The basic function call are:

ObjectPoolManager.New(object:GameObject, number:int)
call to create a pool of gameObject. Typically this is called during loading a scene.
-object: the object to create
-number: the number to create

ObjectPoolManager.Spawn(object:GameObject, pos:Vector3, rot:Quaternion)
called to spawn object, similar to default Instantiate();
- object: the object to instantiate
- pos: position which the object will be appear
- rot: rotation of the object when spawn

ObjectPoolManager.Unspawn(object:GameObject)
called to delete object, similar to default Destroy();
- object: the object to Destroy

ObjectPoolManager.Clear()
Destroy all the objects in the manager. This need to be call in the start of every scene before populating the manager.

13.0 Example and Prefabs

There are many example of visual effect setup using combination of particle system, line renderer and trail renderer along with simple custom texture or standard asset texture. This effect range from explosion, energy bolt to laser beam.

These example are stored in folder named "particle_effects" and "shoot_objects". Since example of shoot object has been mentioned and explained, this section explain only those under folder "particle_effects".

energy_bolt
an example of a particle system configured to appear like an energy bolt.

energy_cloud
an example of a particle system configured to appear like an coloured cloud

energy_cloud_burst
an example of a particle system configured to appear like a burst of gas

explosion_big
an explosion with spark. To achieve this two particle systems were use. One with material using shader "particles/additive" coloured with yellow and orange to looked like a fireball. The other uses shader "particles/alpha Blended" and dark colour to resemble a smokes. These materials can be found in folder "example_asset/materials". A third particle system spray small particle in all direction to resemble sparks. The materials used for this particle systems can be found in folder "example_asset/materials". All this materials uses the same textures featured in unity standard assets.

explosion_big_upward
an mushroom cloud explosion with shockwave. To achieve this four particle systems were use. Two with material using shader "particles/additive" coloured with yellow and orange to looked like a fireball. One of the fireball particle system were configured to burst upward the other were stretch over the vertical axis. The third particle system uses shader "particles/alpha Blended" and dark colour to resemble a smokes. The fourth and final particle system uses horizontal billboard and made to expand from the centre of the explosion to resemble a shockwave. The materials used for this particle systems can be found in folder "example_asset/materials". All this materials uses the same textures featured in unity standard assets.

explosion_small
A small explosion consist of just one particle system

fire_ball
a basic energy bolt particle system coloured to looked like a fireball. A trail-renderer is added for the fiery trail.

plasma_charge_disperse
Another particle system that can be used as hit effect for plasma bolt
.
sparks
spray of sparks

upgrade_effect_blue
Just another particle system effect made for visual effect when a tower is build/upgraded.

upgrade_effect_red
Just another particle system effect made for visual effect when a tower is build/upgraded.

14.0 Personal Note & Contact Information

Finally thanks for purchasing this toolkit. I hope you enjoy using to create your own game. You are welcome ot use the components of this tookit for your other game that isn't a TD game. Most of all, I hope you manage to learn a trick or two from all the examples.

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 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

p/s: Special thanks to Bob a.k.a wheelbarrow and Tom from Litobyte Softworks for giving me lots of feedback which leads to version 1.3.2

Version Change:

1.01

- update minimap script
- update minimap component in all example scene

1.1

- added GUI-elements based UI
- added new tower deployment scheme
- added scene transition facility
- revamp code logic flow to be more organised. Require less input from user when putting a scene together.
- revamp some of the script configurable parameters to make configuration more intuitive and less error-prone. Notable script modified are: SpawnManager, GameControl, TowerDeploymentSetting and others.
- lots of optimization to improve performance and reduce drawcall where-ever possible.
- bug fix of grid position where it's deviated if x or y value is <0
- added comment on some of the script explaining the vital logic flow
- added untested code block for touch input

1.2

- added path-finding components
- added open-field game play mode
- tweak the tower script and example tower-prefab
- fixed bug where tower-platform is still selectable once a tower has been built on them

1.2.3

- Tweak path-finding algorithm. Using a common path finding component with queued path request. Less lag spike and memory consumption
- Creep now find their individual path rather than follow the general path passed by the spawner. Subsequently fix bug in previous version where they sometime take shotcut through obstacles.
- Change coding for creep movement so they no longer strayed off path when they missed the next waypoint.

1.3

- Tested for iOS compatibility
- Added object pool manager to recycle scene object
- Added example scene for iOS
- Added alternate iOS creeps
- Added alternate iOS GUI
- Added touch camera manipulation (zoom and pan)
- Optimization of code here and there.
- Added damage over time effect for tower

1.3.1

- Improved path-smoothing, the creep no longer clip through tower when moving diagonally and will try to cut the shortest path in open-space.
- Path-validation check before tower building in open-field mode are now added. Player can no longer build on spot where it will effectively block all possible route.
- Further optimize component interaction
- Fix bug on iOS where restart and next button doesn't work after level end
- Slight adjustment to the shader used in iOS scene for more consistent dynamic batching

1.3.2

- fix bug where tower is unselectable in platform mode when gridSize is set to 0.
- fix bug where tower build heightOffset is not considered when in iOS platform mode.
- fix graphic glitch where creep overlay flip for a split seconds when creep reach a waypoint.
- fix bug where creep object is deactivate prematurely before deadAnimation or deadSoundEffect has finished playing.
- added deathDelayDuration variable so the dealy before creep deactivation upon death can be configured.
- added shootPoint for tower to “fire” shootObject from a designated position instead of turret object's centre point. Feature has been extend to upgraded tower turret.
- Tweak projectile and missile shootObject so it aims a little ahead of target.
- added vertical limit to camera so the camera cannot be zoom indefinitely.

1.3.3

- Fix bug where creep are immune to slow after they have been slowed once.
- Fix bug where creep and projectile sometime overshoot from their designated target position.
- Added configurable parameter to projectile and missile to aim ahead of the target for visual correction.
- Added overlay height adjust for creep
- Added option to restrict building of resource tower before the game start
- Fix bug where tower's renderer object is not set to transparent properly in build mode
- SpawnManager now support multiple creep-type in a single wave
- SpawnManager now support individual wave interval