2D INFINITE RUNNER TOOLKIT

Documentation

Table of contents

Note: Thank you for purchasing 2D Infinite Runner Toolkit!
If you have any feedback or questions just let us know and we will try to answer them as soon as possible!

E-Mail: mako752@gmail.com

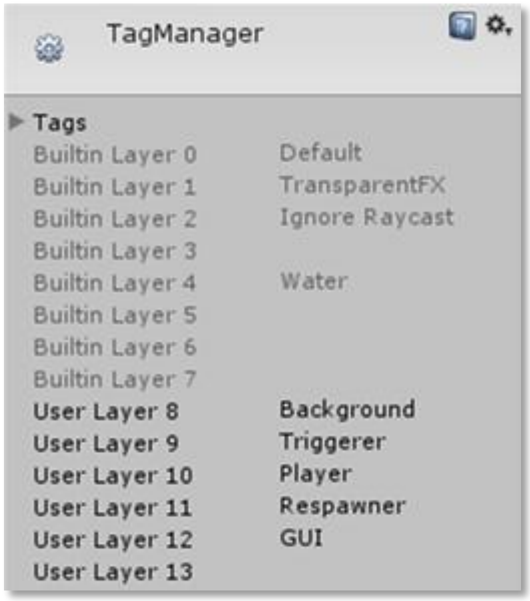
1. OVERVIEW

2D Infinite Runner Toolkit is a feature complete package giving you the chance to create endless runner games of any kind. We invested many hours into this kit to make it as logical and easy to setup and use as possible. It works with Unity3D **built-in features** and does not need any 3rd Party Tools to get it up and running – of course you can easily expand the kit with any Unity3D supported toolset that you might need.

2. PROJECT SETUP

Once you have imported **2D Infinite Runner Toolkit** to an empty/non empty project, you have to set up the layers. Go to **Edit** → **Project Setting** → **Tags**. You need to add 5 layer definitions between user layer 8 and user layer 12. If you imported the toolkit on a non empty project and some of the layers are in use, you have to move your own layers to a different layer. Add the following definitions to the following layers:

- User Layer 8: Background
- User Layer 9: Triggerer
- User Layer 10: Player
- User Layer 11: Respawner
- User Layer 12: GUI



Builtin Layer 0	Default
Builtin Layer 1	TransparentFX
Builtin Layer 2	Ignore Raycast
Builtin Layer 3	
Builtin Layer 4	Water
Builtin Layer 5	
Builtin Layer 6	
Builtin Layer 7	
User Layer 8	Background
User Layer 9	Triggerer
User Layer 10	Player
User Layer 11	Respawner
User Layer 12	GUI
User Layer 13	

If you are using **Unity 3.5**, you can find the game scene in **Scenes → 3_5**. If you are using **Unity 4**, you can find the game scene in **Scenes → 4_X**.

If you are using **Unity 3**, and you would like to deploy to mobile devices, you need to change the default font to a bigger font, because Unity 3 does not support dynamic fonts on mobile devices.

If you open the Unity 4 scene the first time with Unity 4, you will receive some warnings. You can safely ignore them. If you save the scene, and open it next time, you will not receive them again.

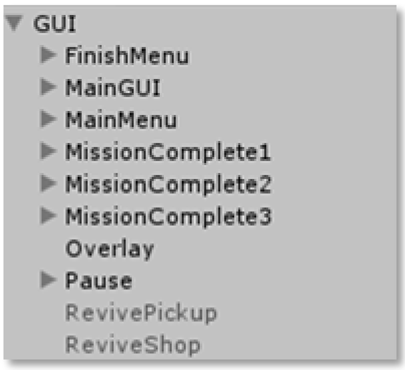
3. Level Setup

The level is divided into 4 main categories, the GUI, the level generator, the player, and the overlord. The first 3 are managed by their own manager scripts, while the overlord has 4 different managers (level manager, input manager, mission manager, and resolution manager).

The main manager is the level manager. It controls level wide events, such as level starting, level restarting, pausing, and so on, and issue commands to the other managers.

The managers are singleton classes, so they can be accessed without a reference variable.

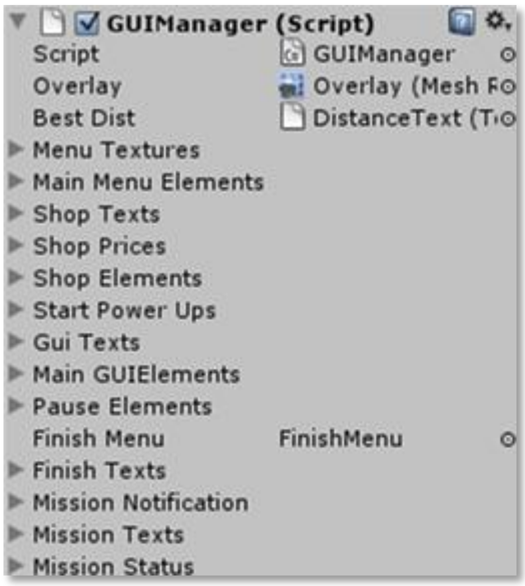
4. GUI


GUI
FinishMenu
MainGUI
MainMenu
MissionComplete1
MissionComplete2
MissionComplete3
Overlay
Pause
RevivePickup
ReviveShop

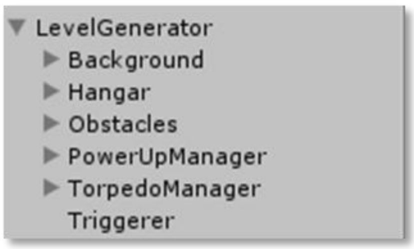
The main GUI object is called "GUI" and contains the GUI Manager script, accordingly. The different menu elements (such as main menu, pause menu, etc.) are stored as child elements. At start, the MainGUI is disabled and the MainMenu is in view. The finish menu, pause menu, and mission complete notifications are out of the camera view and are enabled. During gameplay, the different menu and GUI elements are moved and enabled/disabled by the **GUI Manager**.

The **GUI Manager** contains links to the different GUI elements which are assigned in the inspector. The GUI Manager usually receives input from the Input Manager (if the player is interacting with a button), Level Manager (for example, if the level is paused), or from the Player Manager (if the submarine collides with a hazard).

You can check the manager's variables and methods in the script reference.


GUIManager (Script)
Script GUIManager
Overlay Overlay (Mesh Fo
Best Dist DistanceText (T
Menu Textures
Main Menu Elements
Shop Texts
Shop Prices
Shop Elements
Start Power Ups
Gui Texts
Main GUIElements
Pause Elements
Finish Menu FinishMenu
Finish Texts
Mission Notification
Mission Texts
Mission Status

5. Level Generation Setup



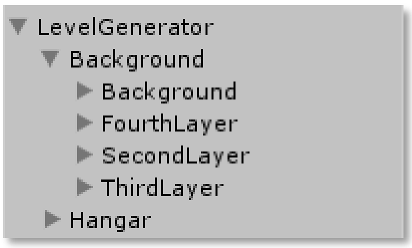
```
▼ LevelGenerator
    ▶ Background
    ▶ Hangar
    ▶ Obstacles
    ▶ PowerUpManager
    ▶ TorpedoManager
      Triggerer
```

The level generation is managed by the Level Generator script, which is attached to the "LevelGenerator" object. This holds several child objects (background, hangar, obstacles, power-up manager, torpedo manager, and triggerer).

The first 5 child objects contain the elements represented by their name (e.g. **Obstacles** contains the obstacle groups), but the Triggerer is special. We will learn about that in a later paragraph.

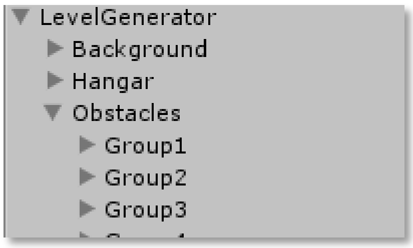
The **level background** is divided into 4 layers:

- background (blue background and the sand)
- second layer (mountains, and big shipwreck)
- third layer (blue plants)
- fourth layer (rocks, vegetation, treasure chest)



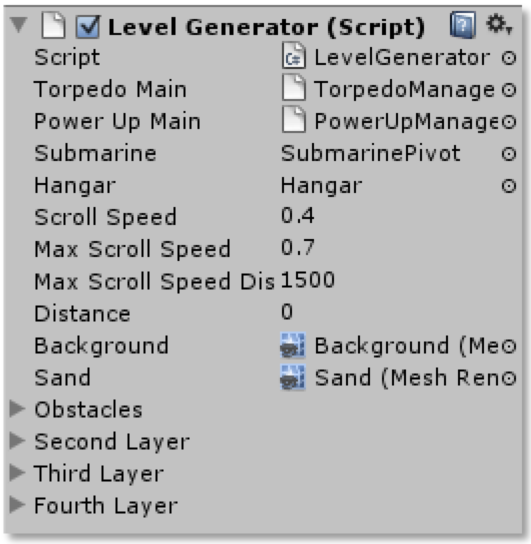
```
▼ LevelGenerator
    ▼ Background
        ▶ Background
        ▶ FourthLayer
        ▶ SecondLayer
        ▶ ThirdLayer
    ▶ Hangar
```

Each layer has premade groups. These groups contain layer specific elements, a spawn trigger, and a reset trigger.



```
▼ LevelGenerator
    ▶ Background
    ▶ Hangar
    ▼ Obstacles
        ▶ Group1
        ▶ Group2
        ▶ Group3
```

The obstacles object contains groups of obstacles. These groups are prearranged - by default there are 20 of them. They can contain level hazards, coins, and they contain a spawn trigger, and a reset trigger. Of course you can easily create more groups.



Level Generator (Script)

Script	LevelGenerator
Torpedo Main	TorpedoManage
Power Up Main	PowerUpManage
Submarine	SubmarinePivot
Hangar	Hangar
Scroll Speed	0.4
Max Scroll Speed	0.7
Max Scroll Speed Dis	1500
Distance	0
Background	Background (Me
Sand	Sand (Mesh Ren
▶ Obstacles	
▶ Second Layer	
▶ Third Layer	
▶ Fourth Layer	

The **Level Generator script** holds references to the obstacle and background groups. You can also configure the starting scroll speed, the maximum scroll speed, and the maximum scroll speed distance. By default, the maximum scrolling speed is 0.7 which is reached at 1500m distance.

The obstacle and background groups are positioned to the right of the camera, out of view. The hangar is positioned in the middle - it also serves as the background for the main menu.

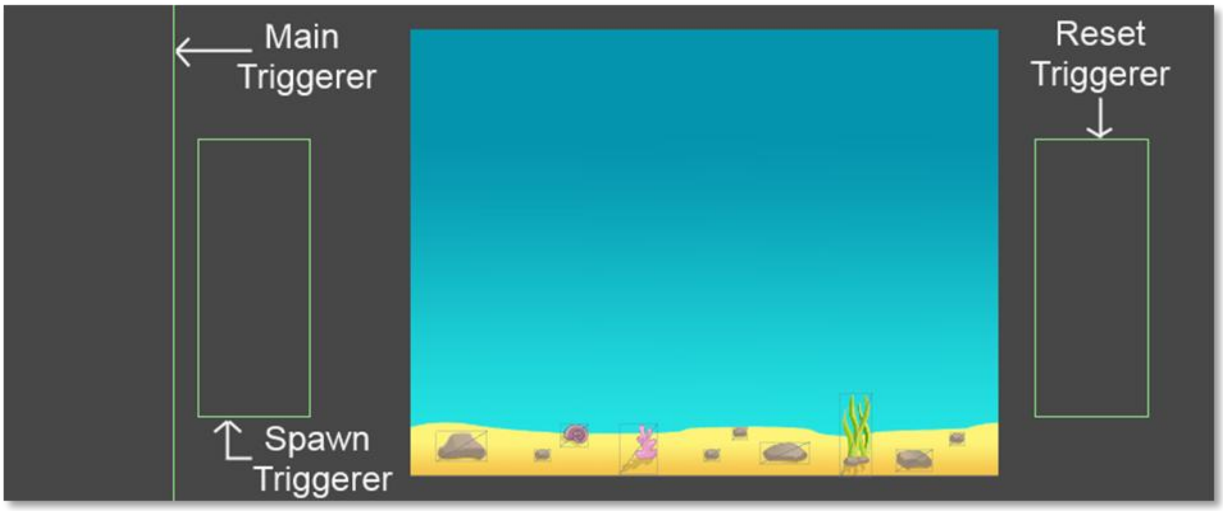
When the game loads, a second and third layer group is positioned in the middle, behind the hangar by the level generator.

When the game is started, the **Level Manager** notifies the **Level Generator** to start moving/generating the level. The level movement looks like the following: The layer groups are moving from the right to the left and their speed is calculated by the current scroll speed and their layer type (e.g. the second layer groups move slower than the fourth layer groups).

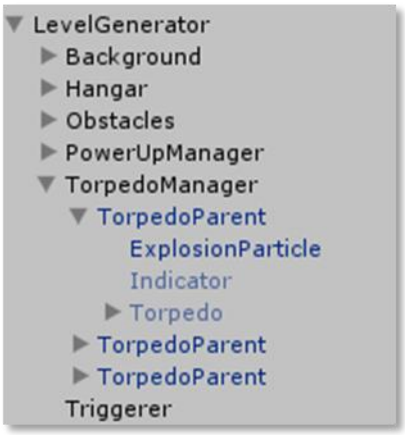
The moving "active" objects are **randomly** selected from the layer groups, and stored in a private list, called activeElements. In other words, the Level Generator moves the contents of the activeElements list.
To the left of the camera, out of the view, the main trigger object can be found. This object is responsible for spawning new layer elements and resetting the current ones. Each group has a spawn and a reset triggerer. As the group is moving to the left, these triggerers will eventually collide with the main triggerer.

If the **spawn triggerer** collides with the main triggerer, a new group from the same layer will be added to the active elements. If the **reset triggerer** collides with the main triggerer, the current group is removed from the active elements and moved to the starting position to the right of the camera.

Here is an example with a fourth layer group.



TORPEDO GENERATION



Torpedoes are generated by the Torpedo Manager, which is attached to the LevelGenerator's child object "TorpedoManager". By default, it holds 3 torpedoes as child objects.

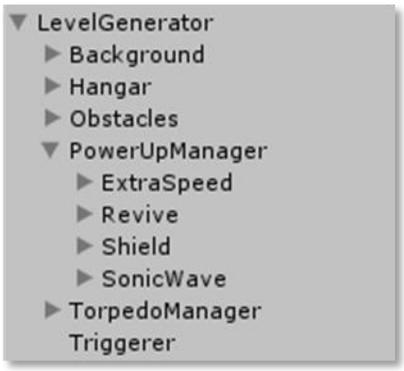
Each torpedo contains an explosion particle, the torpedo indicator, and the torpedo image with its animated flame effect. A Torpedo script is also attached to each TorpedoParent.

The Level Generator commands the Torpedo Manager to **launch torpedoes** (between 2 and 3 based on the distance). Torpedoes are launched randomly between every 15 and 35 seconds.

When the Torpedo Manager received the launch command, it launches the first inactive torpedo. Then the launched torpedo's script calculates a random y position, and places the torpedo indicator. After 3 seconds, the indicator is deactivated, the torpedoes speed is calculated, and the torpedo is launched.

POWER-UP GENERATION

Power-ups are generated by the Power-up Manager, which is attached to the LevelGenerator's child object "PowerUpManager". The object holds the 4 Power-ups ExtraSpeed, Revive, Shield and SonicWave



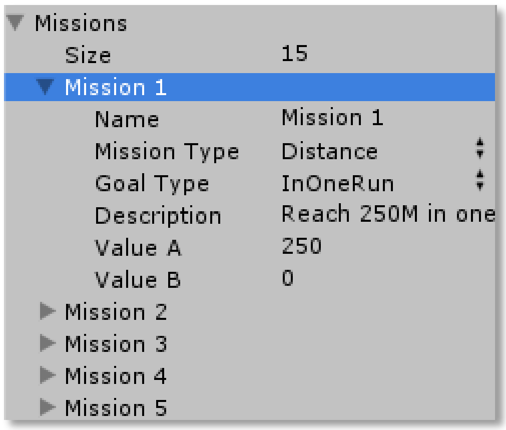
Each power-up contains a trail renderer. A PowerUp script is also attached to each power-up.

The Level Generator commands the Power-up Manager to **launch a power-up** randomly between every 15 and 30 seconds.

When the Power-up Manager received the launch command, it searches for the first compatible power-up, and the selected power-up's script calculates a random y position and launches itself.

MISSION SYSTEM

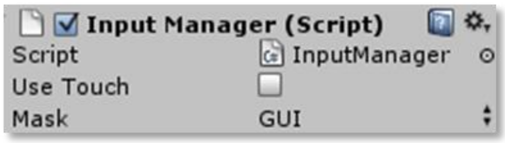
The mission system uses two scripts, **Mission Template**, and **Mission Manager**. The Mission Template script is serializable script, and the Mission Manager holds an array of the Mission Template. This way, adding new mission or editing existing ones is really easy.



```
▼ Missions
    Size               15
  ▼ Mission 1
      Name             Mission 1
      Mission Type     Distance
      Goal Type        InOneRun
      Description      Reach 250M in one
      Value A          250
      Value B          0
  ▶ Mission 2
  ▶ Mission 3
  ▶ Mission 4
  ▶ Mission 5
```

On the technical side, the Mission Manager receives information from the other managers when certain **events** occur (e.g. the player has collected a coin or bought a power-up). The Mission Manager checks the active missions and if the event is related to a mission, it checks the status of the mission based on the new information.

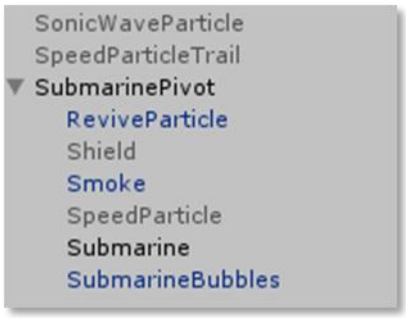
INPUT



```
Input Manager (Script)
Script        InputManager
Use Touch     ☐
Mask          GUI
```

The input between the player and the game is monitored by the **Input Manager**. If a mouse click or a touch has happened, it checks the location of the event. If the event was on a GUI Button, then the Input Manager notifies the GUI Manager. If not, the Player Manager is notified.

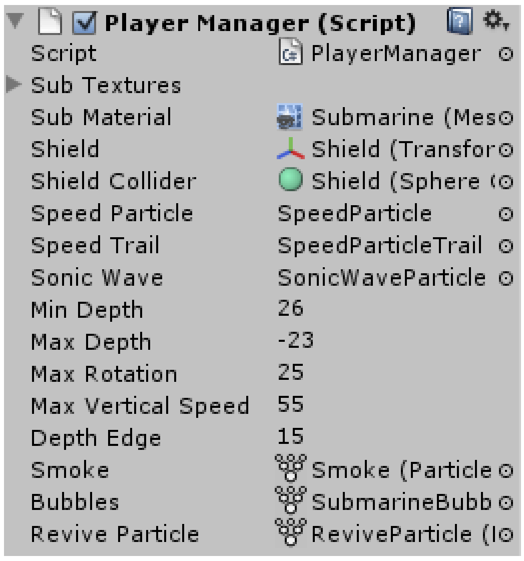
PLAYER SETUP



```
SonicWaveParticle
SpeedParticleTrail
▼ SubmarinePivot
    ReviveParticle
    Shield
    Smoke
    SpeedParticle
    Submarine
    SubmarineBubbles
```

The **Player Manager** is attached to the “SubmarinePivot” object. It contains several child objects, the submarine image, related particles (revive, smoke, bubbles), and other power-up related objects (shield, speed particle). There are two objects, which are related to the player setup, but not a child of the pivot. These are the sonic wave particle and the speed particle trail.

The **Player Manager** holds a link to these objects, and you can also set the min/max depth and max rotation for the submarine. The manager calculates movement direction based on the last input received from the Input Manager. The movement speed calculation is based on the current speed and distance to the min/max depth. Then the rotation is calculated and applied based on the speed. The collision detection is also managed by this manager.



Player Manager (Script)

Script	PlayerManager
Sub Textures	
Sub Material	Submarine (Mes
Shield	Shield (Transfor
Shield Collider	Shield (Sphere (
Speed Particle	SpeedParticle
Speed Trail	SpeedParticleTrail
Sonic Wave	SonicWaveParticle
Min Depth	26
Max Depth	-23
Max Rotation	25
Max Vertical Speed	55
Depth Edge	15
Smoke	Smoke (Particle
Bubbles	SubmarineBubb
Revive Particle	ReviveParticle (I

Save System

Saving and loading the game is managed by the static Save Manager (it is not attached to any object). The coin amount, best distance, owned power-ups and mission data is stored by using Unity's built-in **PlayerPrefs**.

Resolution Manager

The **Resolution Manager** is attached to the Overlord object. The manager is aspect ratio based and supports 3:2, 4:3, 5:3 and 16:9 aspect ratios. Adding support for additional aspect ratios can be done within the manager's SetResolutionSetting function, by expanding the switch statement and setting the variable values.