Pixel Runner Template

Pixel Runner includes everything you need for a complete 2D infinite runner game, including UI, sound fx, background music and all graphics. It is easy to extend, you can add or replace level sections directly in the editor and you can even add new parallax layers without touching any code.

Installation

Simply create a new project, and underneath the heading “Import the following packages” tick PixelRunner.unitypackage. Select “Setup defaults for 2D” and then click ‘Create’. The game is contained in a single scene called PixelRunner.unity, found in the scenes directory.

Layers and Tags

Before looking at the scene itself, let’s just take a moment to see the various layers and tags that have been set up. At the very top right of the Unity window select Layers->Edit Layers… to see all the layer and tags.

Under the ‘Tags’ heading you’ll find that the only custom tag that has been defined is one named ‘Spikes’. This tag is given to any objects that kill the player on contact like the - you guessed it - spikes, and the circular saws. If you create any new types of hazard for the player to avoid and you want them to kill the player if they touch them, then remember to set their tag to ‘Spikes’.

Under ‘Sorting Layers’ you’ll find Background, Midground, Main, Foreground and GUI. These should be fairly self-explanatory, the player and all the level blocks are in the Main sorting layer, the Midground and Background display behind them, the Foreground displays in front, and anything on the GUI layer displays in front of everything.

Under ‘Layers’ you will find two custom layers set up. The Player layer is just used for the player, so we can do collision tests that don’t include the player. The GUI layer is used for all the GUI elements, this is so we can do raycasts against only the GUI items to check what lies under the point the user tapped.

Checking out the Scene

With that out of the way, lets move on the scene itself. When you first load up the Pixel Runner scene, the hierarchy will show 3 items: Game, GUI and MainCamera.

Camera

The camera remains stationary throughout the game (it is the world that scrolls leftwards to give the illusion of motion), so there is no special logic attached to it. Since this is a 2D game, the camera is set to orthographic projection, with a size of 4.

GUI

Underneath the GUI object is all the UI elements for the game. Pixel Runner includes a simple menu system with support for animations, sound effects, and a simple callback system for keeping your menu scripts clear and straightforward. This will be explained in more detail below.

Game

The Game object is the parent for all the in-game related elements. It contains the following child items:

- Game Manager - this handles all the input, the global state of the game and the player score.
- Level Manager - this handles the generation of the level elements. These are split into multiple layers that can scroll at different parallax speeds. We’ll go into some detail about this system below.
- Player - this is the player sprite, and it handles all the logic to do with jumping, updating the player animations and detecting collisions and so on.
- Sound Manager - a very simple class that handles playing of sound effects on request. If your game has complex audio requirements then you’ll probably need something more full-featured, but for a simple game this suffices perfectly well.

Sprite Setup

Before we get on to discussing the game systems in detail, just a quick word on import settings for the sprites. If you take a look inside the textures directory you’ll find the all the images for the game. The important settings to note are:

- Texture type: Sprite - this is the default for 2D games

- Pixels to Units: 32 - this is different from the default value of 100, and means that our scenery blocks (which are in multiples of 32 pixels) fit neatly into Unity units, making snapping much easier.
- Filter mode: point - we want nice crisp edges and visible blocky pixels for that retro look
- Format: Truecolor - don't use any compression on the textures, they're tiny anyway so memory is not going to be a problem.

When replacing any of the textures with your own, if you follow these settings then the replacements should fit in without any problem. If you change the resolution of the textures then remember that you'll also need to change the pixels to units amount by the same factor - for example if you double the size of one of the textures, then set the pixels to units value to 64 (2*32) as well, and everything will remain the same size on screen.

Level Manager - Building infinite scenery

The Level Manager object is responsible for generating all the visible scenery and scrolling it past the camera. Underneath the LevelManager object you will find a number of child objects with names that all end in 'Layer'. The scenery is divided into separate layers that are all built up individually, and may scroll at different speeds.
At the lowest level the scenery is made up from discrete chunks called blocks. Each block is just a short section of the level or background. Each layer has an instance of the SceneryLayer script attached to it, and its child objects are the blocks that it randomly chooses from to generate the level. Every frame the LayerManager asks each SceneryLayer to scroll its blocks to the left. Any blocks that fall off the left side of the screen are removed and new blocks are generated on the right to make sure the screen is fully covered.

Scenery Layer Objects

If you select one of the Layer objects underneath the LevelManager you can see a couple of options under the Scenery Layer script component in the inspector. Let's start by selecting the BackgroundLayer.
Parallax Speed controls the speed that the layer will scroll at, relative to the main layer that the player sits on. A value of 0 is stationary meaning the layer won't scroll at all, whilst a value of 1 means it will scroll at full speed. Values greater than 1 can be used to create foreground layers that scroll faster than the main layer.
The second option 'Is Infinite' controls whether blocks from this layer will keep generating indefinitely. For the background and midground layers this is set to true, but we can use this to do cool things with the main level layers as we'll see in a moment.
Select the Level1Layer object, and have a look in the inspector. First we'll see that the parallax speed is set to 1, because this is one of the main layers that the player runs on. Next we'll see that 'Is Infinite' is set to false and there are a couple of new options that will only appear when 'Is Inifinite' is unchecked. 'Max Block Count' determines how many blocks from this layer will be generated. Here it is set to 6, so after 6 blocks from this layer have scrolled past, this layer will

stop generating new blocks, and instead the layer specified in the 'Next Layer' parameter will begin generating blocks. In this way we can chain layers together and create levels that change in predictable ways as the player progresses, whilst still being randomly generated and inifinite in scope.
By looking at the 3 Level layers and the StartLayer we can see how the game is built up. The StartLayer contains a single block - this is where the player will begin. The StartLayer has 'Max Block Count' set to 1, so after that one block has been generated the start layer becomes inactive and the layer pointed to by 'Next Layer' will become active - in this case it's Level1Layer. Level1Layer contains a number of blocks as child elements (you can see them in the scene stacked vertically just to the right of the start elements). These blocks contain only simple jumps, giving the player a gentle start to the game. Once the player has successfully passed 6 blocks from Level1Layer, it too will become inactive and pass the baton to Level2Layer. This layer contains a lot of different blocks and they are starting to become more difficult, with various spikes and circular saws for the player to avoid. If the player successfully passes enough of these blocks then Level3Layer will start generating blocks. This layer uses a different set of tile prefabs with a brownish color applied to the sprites for some visual variation. These blocks are even more difficult. Level3Layer points back to Level1Layer, so assuming the player survives this far, they will start seeing blocks from Level1 again and the sequence will repeat.

Adding new layers

Lets see how you could go about adding another level layer, perhaps with even more fiendish blocks that could appear after the Level3Layer. First of all create an empty game object and add an instance of the SceneryLayer script to it. Make it a child of the LevelManager object - the LevelManager will only know about layers that are direct children of it. Next decide whether you want the layer to continue for ever, or whether the game should loop back to an earlier level after some time. If you want it to loop back, then set 'Is Infinite' to false, 'Max Block Count' to the number of blocks that the player must survive before looping, and then drag one of the other Level layers onto the NextLayer parameter, to set which level it should loop to. Finally you need to create all the blocks for the layer. Each block is just a game object with a SceneryBlock script on it, and all the actual sprites and so on as children. I tend to lay out the blocks for a level vertically so I can see them all, but their position isn't important - they'll be positioned by the layer when they are needed. You could use the prefabs from the directory Prefabs/Scenery, or create your own.

GUI System

The GUI system included with PixelRunner is simple to use and easy to extend. It features a simple procedural animation component that can be used to easily animate elements on and off screen.
The GUI object in the scene is just a container to hold the various menu pages and keep the hierarchy tidy. It’s children are all the GUI pages in the game, such as the start screen, the pause screen and so on. Each page has its own script that controls animating its elements on and off,

as well as any handler functions that it might need in order to respond to taps. To create an item that the user can tap, simply add an object as a child of one of the pages, and add a Collider2D and a MenuItem script component to it. The collider should have ‘Is Trigger’ set to true, and remember to set the layer of the object to ‘GUI’. The MenuItem script has two parameters to set, one is the name of the handler function to call when the item is tapped or clicked, and the other is an optional name of a sound effect to play - leave this blank if you don’t want a sound to play when the item is tapped. The handler function could be defined in a script attached to this object directly, but it’s often cleaner to keep the handler functions for one page together, which is why you’ll find them declared inside the script attached to the parent ‘page’ object for all the default menus in PixelRunner.

Sound Manager

The SoundManager object is a very simple system for playing menu or ingame sound effects and background music. It simply maintains a list of available sound clips (which you can edit in the inspector). These are then referenced by name, either through the ‘Sfx Name’ parameter on a MenuItem, or in code by calling the static function SoundManager.PlaySfx(“soundname”).