GameDev.tv Starter Kit

2D RPG Template Project

About

Thank you for downloading the GameDev.tv '2D RPG Template Project' Starter Kit.

This template starter kit provides you with everything you need to get a head start on making your very own 2D top-down RPG game.

Everything has been designed to be beginner friendly and easy to understand, with all code fully commented.

By purchasing this asset from the Unity Asset Store, you have also gained access to a detailed video tutorial series that explains how to use the template and provides advice on how to expand the project to make your own cool game!

You can check these videos out here:
https://www.gamedev.tv/p/unity-2d-rpg-starter-kit-videos

If you prefer reading to watching videos, this document will guide you through the setup process and explain the various elements included in this pack.

Note: the videos go into much more detail, so they're definitely worth a watch if you want a deeper breakdown of the code!

This asset contains:

- Sprites for the player character, 2x NPCs, and slime enemy
- Player Camera Controller
- Player and Enemy animations
- Combat system, including; sword, boomerang, and bombs
- Health system
- Dialogue system
- Scene transitioning
- 4 starter levels, including; town, shop, woods, and cave.
- Inventory Management
- Collectable coins/rupees
- Breakable pots (because no RPG is complete without it)

Contents

Requirements

- Unity 2022.1 or later
- Input System 1.3 or later
- Tilemap Extras 3.0.0 or later

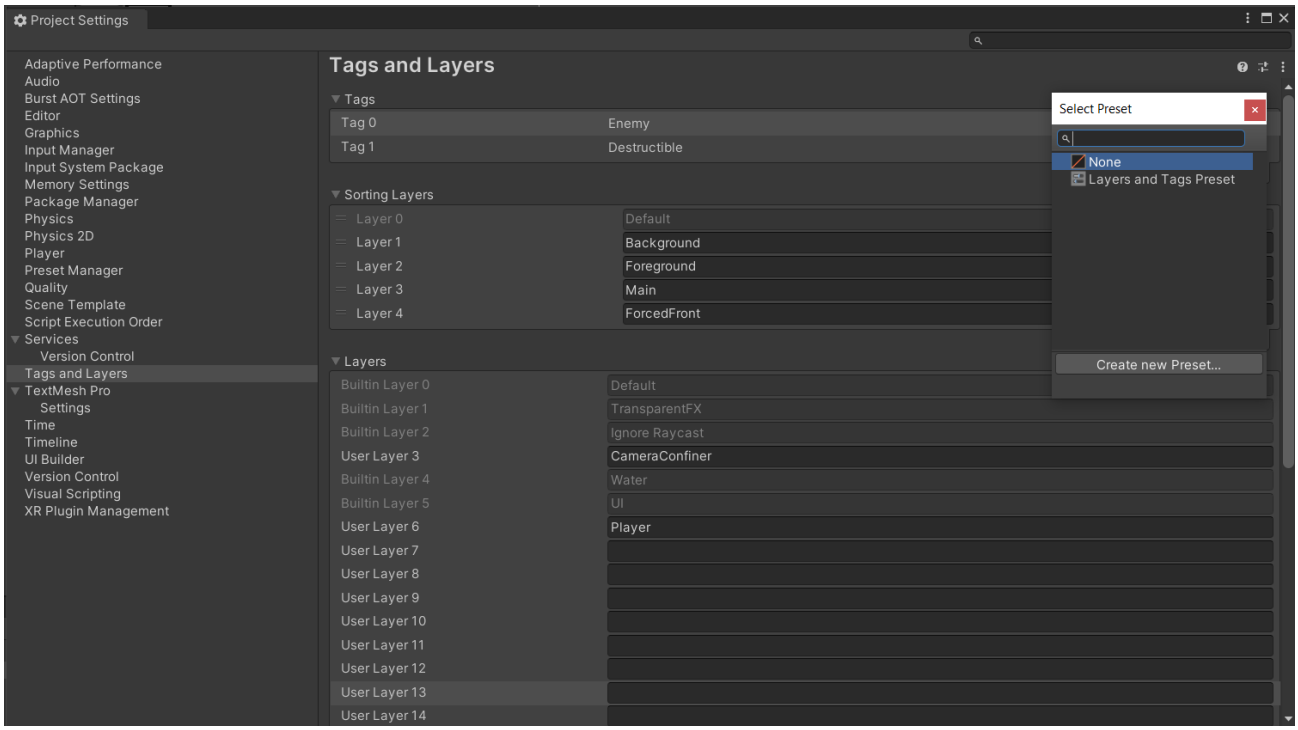
Setup

When you first load up the project, you may notice a few errors in the console and the project on't look quite right.

Just work through the following steps in order and you'll be up and running in no time!

Layers

1. Navigate to; Edit > Project Settings
2. Select "Tags and Layers" from the menu on the left
3. Click on the "Presets" button in the top right (next to the help icon)
4. Select the "Layers and Tags Preset.preset"
5. This should add all the necessary tags, sorting layers and layers.



6. Reload the scene to apply the sorting layers.
7. Check that the layers have been correctly set up and connected

Camera Settings

1. Go to; Edit > Project Settings
2. Select "Graphics" from the menu on the left
3. Change the 'Transparency Sort Mode to "Custom Axis"
4. Change the 'Transparency Sort Axis' to (0,1,0)

Aspect Ratio

1. Go to the Game window
2. Change the aspect ratio to 1920x1080

Build Settings

1. Go to; File > Build Settings
2. Remove the 'Sample Scene' if you have one
3. Add the four scenes from this pack to the build
4. Make sure the Town scene is at index 0

Input Control System

This project is set up to use the new Unity input control system and comes ready with a pre-made player controller, which you can find in Assets/Scripts/Player.

Action maps and key bindings can be found in the `PlayerControls.inputactions` asset.

There is also a generated C# class called `PlayerControls.cs` that the input system requires to read the player input actions. This script is automatically updated whenever changes to the inputactions file are saved.

To use any new input actions you will need to subscribe to them inside your scripts.
Check out the PlayerController.cs script for examples of how this is handled.

Player Movement

The hero character has basic movement controls set up and bound to the WASD keys.
Holding Shift will allow the player to run.

Movement animations can be found in the folder Assets/Animations/Hero. However, you shouldn't need to adjust these.

The associated spritesheet is located in the folder Assets/Sprites/Hero.

On the Hero prefab, you will find exposed parameters to adjust the base move speed and run speed of the player character.

Note: The run speed is added to the move speed when Shift is held, so if `moveSpeed = 4` and `runSpeed = 4`, the character will have a final speed of 8.

Tilemaps & Sprite Sorting

Levels are created using the Unity Tilemap and you can find each tilemap in the hierarchy, as children of the Grid game object.

The world is broadly separated using 3 different tilemaps:

- Ground –All of the general background world tiles, such as; grass, paths, etc.
- Solid Objects – Anything you want the player to collide with, such as; cliffs, water, boulders, etc.
- Cosmetic – Overlays the ground tilemap and include things like; grass, flowers, path edging, etc.

There are also 4 different tilemap palettes set up to make paining new levels easy:

- Cave_1 – Includes; cave walls, ground tiles, and mushrooms
- Inside_1 – Includes; wooden walls, tiled floor, and columns
- Outside_1 – Includes; grass, stone path, and various flowers
- Outside_1 – Includes; grass, cliffs, dirt path, water, and various flowers

Camera Controller

This project uses a custom camera control system.

Note that the Camera in the hierarchy has a 'Pixel Perfect Camera' component attached and all sprites have their 'Filter Mode' property set to "Point".

The camera also has a 'Camera Controller' component that is responsible for finding the player in the scene and following them as they move.
Note that the camera is confined to the bounds of the 'Ground' tilemap and will stop moving when it reaches the edges of the world.

Melee Combat

Inside the Hero prefab you will find 4 hitboxes that will toggle on/off as the player attacks in each direction.

The toggling of these colliders is handled via animation events.

Each hitbox has an 'Attack Damage' component that is responsible for applying damage to the colliding object when triggered.

You can modify the amount of damage delt in the inspector, as well as apply knockback to the colliding object (more on that later).

Player & Enemy Health

When the 'Attack Damage' collider is triggered, the colliding object will receive damage if it has a health component attached.

If this project there are two types of health component; PlayerHealth, and EnemyHealth.

When calling the `TakeDamage(float)` methods you will need to pass in the amount of damage being dealt.

When an object takes damage, it will flash white to show that damage has been dealt.

Once the `currentHealth` value of an enemy reaches 0, a death affect will be played and the object will then be destroyed.

When the `currentHealth` value of the player reaches 0, a death animation will play and control will be taken away from the player. Once complete, the player will then respawn at the designated respawn location (By default, this is the fountain in the town square).

Knockback

The 'KnockBack' component works in conjunction with the 'Attack Damage' component to applies a force away from the attacking objects.

The amount of thrust applied is full configurable via the 'Attack Damage' component in the inspector and the force is applied when the 'Attack Damage' collider is triggered by an object that has the 'KnockBack' component attached.

To knock back and object, call `GetKnockedBack(Transform, float)` passing in the transform of the attacking object and the amount of thrust to be applied.

When the player is knocked back, they temporarily lose control of the character. This `knockbackTime` can be configured via the 'KnockBack' component in the inspector.

Dialogue

Dialogue can be applied to various objects by simply attaching the 'Dialogue Activator' component.

On this component, you will find a list for storing all the dialogue lines.

In addition to conversational dialogue, you can also specify whether a nameplate should appear above the main dialogue window.

To add the nameplate, check the 'IsPerson' property in the inspector.

To specify the text that should appear in that box; add a dialogue line before the conversational line and prefix it with `n-`.

You will also need to ensure that the "DialogueButton" prefab is attached as a child of the game object that contains the 'Dialogue Activator' component. Once attached, connect this to the 'Button UI' property on the 'Dialogue Activator' component.

You can see examples of this system by looking at the town NPCs and town sign.

In addition to the 'Dialogue Activator' component, you will also need to ensure that the 'Dialogue Manager' component is also added to the dialogue box UI component.

By default, activation and progression of dialogue conversations is mapped to the spacebar, but this can be changed by changing the key bindings in the `PlayerControls.inputactions` asset.

Breakable Environment & Rupees

No 2D-RPG would be complete without breakable pots!

To make any object breakable, simply add the 'Breakable' component to your desired game object and add an animation controller with a "Break" trigger.

When the 'Attack Damage' component hits a breakable object, the `BreakObject()` method will be called on the 'Breakable' component. This in turn will trigger the relevant animation and then destroy the game object after a set time.

You can also configure the 'Breakable' component to spawn items when destroyed by specifying the appropriate prefab in the inspector.

By default the dropped item is a blue rupee. When this is collected by the player, this will update their rupee wallet balance and be displayed on the UI.

Scene Transitions

Scene transitions are handed by a combination of linked “Area Entrance” and “Area Exit” portals.

These portals can be placed anywhere in a scene and you can have multiple entrances/exits.

In this project, you will find an “Area Exit” prefab that has been preconfigured for you to use.
Note that the “Area Entrance” is a child of the “Area Exit”.

On the “Area Exit” component, you will find several properties that will need to be adjusted:

The ‘Area To Load’ property should match the name of the scene that you want to load. Be sure to check your build settings and ensure that the relevant scene is present.

The ‘Area Transition Name’ property denotes where the player should spawn when the new scene loads.

To help hide scene transitions, the screen will fade in/out. This is controlled by the “UI Fade” component, which is attached to the “FadeImage” under the UI canvas.

This fade is fully configurable, so you can change the fade duration and fade color to suit your needs.

Inventory Manager

This template comes with several secondary items for the hero character – a boomerang and a bomb.

Switching between these items is handled by the ‘Inventory Manager’ component.

When an item is selected in the “Inventory Container” UI element, the selection is passed through to the ‘Inventory Manager’ and stored for when the player choses to use it.

By default, opening the inventory is mapped to the E key, but this can be changed by changing the key bindings in the `PlayerControls.inputactions` asset.

Bomb

Bombs can be used to damage objects in the game and is the primary weapon for breaking destructible walls.

When a bomb is instantiated into the scene an animation will be played. Once the animation has finished, an animation event will trigger the `Explode()` method in `Bomb.cs`.

This `Explode()` method tells the ‘Attack Damage’ component that damage was dealt by a bomb and this information can then be used for bomb-specific damage actions, such as destroying boulders and opening up hidden entrances.

Boomerang

The boomerang is a ranged weapon that will return to the player after it has travelled a set distance, or if it hits an object.

Parameters such as; throw distance, damage, and speed, are all configurable via the inspector.

Singleton & Inheritance

Several classes within this project are set up as singleton classes. This includes objects like the UI canvas and the Player.

To make the setup of singleton classes easier, we have included a generic class which manages everything for you.

This generic class inherits from Monobehaviour, so you can still attach components that inherit from it to objects in the hierarchy.

It does make use of the `Awake()` and `OnDestroy()` Unity methods, so if you also need to make use of these you will need to declare them in your script with the `protected override` keywords.

To see this generic class in action, check out PlayerContoller.cs for a working example of how this is implemented.

Folder Structure

Once imported, you will find everything organized into several folders:

- Animations
    - Enemies
    - Environment
    - Hero
    - Items
    - Misc
    - NPCs
- Fonts
- Materials
- Prefab
    - Enemies
    - Environment
    - Hero
    - Items
    - Misc
    - NPCs
    - SceneManagement
- Scenes
- Scripts
    - Combat
    - Dialogue
    - Enemies
    - Environment
    - Inventory
    - Items
    - Misc
    - Player
    - SceneManagement
- Sprites
    - Buildings
    - Enemies
    - Environment
    - Hero
    - Inside
    - Items
    - Misc
    - NPCs
    - UI
- TextMesh Pro
- Tilesets
    - Cave_1
    - Inside_1
    - Outdoor_1
    - Outdoor_2
- VFX
    - Explosion
    -

Need Help?

If you're not already part of our amazing community, why not come and say hi!

- Community Forum
- Discord
- Facebook
- Twitter
- Instagram
- YouTube

We also have a ton of great courses to help you develop your skills in; Unity, Blender, Pixel Art, and more.

Swing by GameDev.tv and supercharge your game development skills today!