2D Starter Package Documentation

Digital Worlds Unity 2D Starter Package by Logan Kemper - Version 2

GitHub Repository & Project Setup Guide

Project Assets

Key: Utility, Enemy, Player, Level, Visuals

File	Description
ApplicationController.cs	Provides functionality for quitting or pausing the game.
AudioFader.cs	A simple component for gradually fading audio in or out.
BouncePad2D.cs	A bounce pad that applies an instant force to any GameObject with a Rigidbody2D that collides with it.
ButtonEvents2D.cs	Generic script for adding UnityEvents to button presses.
CameraController2D.cs	Attach to the main camera to control its movement.
ChangeScene.cs	Change the current scene by index or by name.
Checkpoint2D.cs	Used to update the player's respawn position.
Collectable.cs	Used to add collectables to the CollectableManager.
CollectableEvents.cs	Used to add UnityEvents to collectable count thresholds.
CollectableManager.cs	Manages one or more collectable items and displays them on the UI.
CollisionEvents2D.cs	Generic script for adding UnityEvents to 2D collisions.
ConditionalEvents.cs	Used to add UnityEvents to specific count conditions.
DamageOverTime2D.cs	Deals damage to players and enemies continuously over time.
Damager.cs	Add to a GameObject to make it deal damage to other entities.
DisappearingPlatform2D.cs	Attach to a platform and use the UnityEvents to disable its collider or whatever else needed to make the platform disappear.

Project Assets

Key: Utility, Enemy, Player, Level, Visuals

File	Description
EnemyHealth2D.cs	Gives enemies functionality for health, taking damage, and dropping items.
EnemyHealthStages2D.cs	Attach to an enemy to add UnityEvents to stages of their health.
EnemyProjectileAttack2D.cs	Launches projectile attacks towards the player. Can be used for mobile or stationary enemies.
EventRelay.cs	A simple component for invoking a UnityEvent from another UnityEvent. Use to group and organize complex event sequences.
FlashColor2D.cs	Changes a SpriteRenderer to a new color, then back to the original color.
FlickeringLight2D.cs	Add to a Light2D to flicker its intensity up and down.
Floater.cs	Add to a GameObject to make it float up and down and spin.
Inventory.cs	Add to the player to allow them to pick up items and display them on the UI.
Item.cs	Add to a GameObject with a trigger collider to allow the player to add the item to their inventory.
Lock2D.cs	Add to a GameObject with a trigger collider to create a lock that can only be unlocked if the player has the requisite item(s) in their inventory.
Parallax.cs	Add to a GameObject in the background or foreground to create the illusion of depth with parallaxing movement.
PatrolChase2D.cs	A more sophisticated patrol script for dynamic enemies. PatrolChase inherits from PatrolMultiple to extend its functionality and add a chasing state.
PatrolMultiple2D.cs	Moves a GameObject to its assigned waypoints. Can be used for platforms, NPCs, hazards, and more.
PatrolSimple2D.cs	Moves a GameObject back and forth between two positions. Can be used for platforms, NPCs, hazards, and more.

Project Assets

Key: Utility, Enemy, Player, Level, Visuals

File	Description
PlayerAudio2D.cs	Adds sound effects to PlayerMovement.
PlayerHealth2D.cs	Gives the player health, dying, and respawning functionality.
PlayerMeleeAttack2D.cs	Gives the player a melee attack.
PlayerMovement2D.cs	A basic platformer player controller with simple running and jumping behavior.
PlayerMovementAdvanced.cs	A more sophisticated player controller that extends PlayerMovement2D to add dashing and wall jumping functionality.
PlayerMovementBase.cs	A base class for giving shared functionality to player controllers.
PlayerMovementTopDown.cs	A basic top-down player controller with simple movement.
PlayerProjectileAttack2D.cs	Gives the player a projectile attack.
PlayerStealth2D.cs	Adds a stealth mechanic that allows the player to be detected by enemies or be "in cover" to avoid detection.
Projectile2D.cs	Attach to a projectile prefab to give it launching and destroying behavior.
Rotator.cs	Rotates the GameObject by a specified amount on each axis.
Score.cs	Used for keeping track of a score value and displaying it on the UI. Can be used for the number of items collected, number of enemies defeated, etc.
SelfDestruct.cs	SelfDestruct can be used to destroy components and GameObjects via UnityEvents.
Spawner.cs	Generic script for spawning in GameObjects.
StartEvent.cs	Generic script for adding a UnityEvent to the Start method.
StickyPlatform2D.cs	Add to a moving platform to make GameObjects stick to its surface.
Teleporter.cs	Teleports the player to another location.

Project Assets	
Key: Utility, Enemy, Player, Level, Visuals	
File	Description
Timer.cs	Generic script for counting down time. Can be used to display a timer on the UI, or just to delay an action by a specified amount of time.
TriggerEvents2D.cs	Generic script for adding UnityEvents to 2D trigger collisions.

ApplicationController.cs

Attach to any GameObject in the scene. If quitGameOnEscape is enabled, pressing the escape key will close the application when running as a standalone build (not in the Unity Editor). The public method QuitGame() can be called from a UnityEvent, making it useful for UI buttons (e.g., a "Quit" button in the main menu).

AudioFader.cs

Attach to a GameObject in the scene with an audio source component for it to control. Call AudioFader.FadeAudio from a UnityEvent to fade the source's volume from its current volume to the target volume. The fade duration and target volume can also be set from UnityEvents.

BouncePad2D.cs

Attach to an object in the level with a non-trigger 2D collider. When a GameObject that matches the specified tagName touches the bounce pad, this script will try to find a Rigidbody2D on the colliding object. If found, it applies the bounceForce in bounceDirection. Additionally, the onBounce UnityEvent will be invoked any time a valid object bounces on the bounce pad.

ButtonEvents2D.cs

Attach to a GameObject with a 2D collider set to trigger. When an object matching tagName enters the trigger and the keyToPress is pressed, the onButtonActivated event will be invoked. onButtonReleased is invoked once the button has been let go (and the trigger is still activated). Alternatively, requiresTrigger can be set to false to remove the need for a trigger to accept the key input.

CameraController2D.cs

Attach to the main camera in the scene to make it follow a target (likely the player). followSpeed will define how closely the camera should keep up with the target. allowPeeking will allow the camera to be moved up and down with vertical input (up and down arrows or W and S keys by default). Change the offset values to adjust the resting point of the camera from the center of the target transform. The minBounds and maxBounds will prevent the camera from moving outside of the specified box. This box will be visualized in the scene view in yellow when the camera is selected in the hierarchy.

Tips:

- Use the SetPeekingEnabled, SetMaxPeekDistance, and SetFollowSpeed methods via a UnityEvent to change these settings.
- Use the SnapToTarget method to instantly move to the target. Useful when teleporting the player.
- Use the SetTarget method to change the transform that the camera follows.
- Use the PauseFollowing or ToggleFollowing methods to start or stop the camera following the target. ToggleFollowing can be used from the component's context menu.
- Use the HoldPosition method to freeze the camera in its current place for the given number of seconds.
- Use the ViewDestination method to send the camera to look at a position and then return. The viewTime variable controls how long the viewing will last for. Consider using the onViewBegan and onViewEnd events to enable and disable player movement.

ChangeScene.cs

This component houses a few methods for loading a different scene. LoadSceneByName() is looking for the file name of the scene (e.g., "SampleScene") and LoadSceneByIndex() is looking for the integer number associated with the scene in the scene list. LoadNextScene() will load the next scene in the scene list. LoadCurrentScene() will reload the current scene from the beginning. Whichever is chosen, make sure to double-check the scene list by viewing File > Build Profiles and ensuring that all desired scenes are present and enabled.

Checkpoint2D.cs

Attach to a GameObject with a 2D collider set to trigger. When the specified tagName is triggered, the respawn transform will be moved to the position of the GameObject with the checkpoint component. The singleUse checkbox controls whether this respawn point can be activated more than once. The respawn point can also be set from a UnityEvent by calling the SetRespawnPoint() method and passing in the transform of the new respawn point.

Collectable.cs

- Dependency: CollectableManager.cs

Attach to a GameObject and set the collectable name and count. The collectable name must exactly match the respective collectable name in CollectableManager. Call the PickUpCollectable() method from a UnityEvent to add the collectable to the CollectableManager. There is also a PickUpCollectableAmount() method that will override that count value with the given amount. Finally, the SetCollectableTo() method will set the collectable count directly to the given value.

CollectableEvents.cs

This component is used in conjunction with the CollectableManager to add UnityEvents to the collectable system. Attach to a GameObject and set the target count. Choose the condition (e.g., “Equal To”) and set up the onConditionCleared and onConditionFailed events. Then, drag the CollectableEvents component into the onCollectableUpdated UnityEvent on the corresponding CollectableManager collectable. Choose CollectableEvents.CheckCondition (dynamic int). Now whenever the collectable is updated, the condition will be checked. Use this system to unlock a door once five collectables have been picked up, spawn an enemy if a collectable has been depleted, etc.

CollectableManager.cs

- Dependency: TextMesh Pro

Attach to a GameObject and make sure there is no more than one instance of CollectableManager in a given scene at a time. Add a new collectable display for each collectable that should be kept track of. The collectable name is used to identify the collectable, and is case-sensitive. Optionally drag in a TMP text element to display the collectable count to the UI. If persistentCollectable is true, CollectableManager will try to keep collectable counts between scenes using the collectable name to identify them.

CollisionEvents2D.cs

Attach to a GameObject with a non-trigger 2D collider. When a collision is detected with the specified tagName, the onCollision event is invoked. When the collision is exited, the onCollisionExit event is invoked.

ConditionalEvents.cs

ConditionalEvents keeps track of an integer value and can invoke UnityEvents when that value meets a condition. To use, add a new counter condition and set a target count. Then choose how the count will be evaluated (equal to, greater than, etc.) and set whether the condition is single-use or not. The onConditionMet event fires when the count changes and the condition is true. Change the count with the Add, Subtract, or SetCount methods. If evaluateOnStart is checked, all conditions will be evaluated when the game starts. The component’s context menu (three dots in the top right) has helper options to test the count.

DamageOverTime2D.cs

- Dependency: Damager.cs

Add to a GameObject with a trigger collider. Players and enemies inside the trigger collider will be damaged each damageInterval (in seconds).

Damager.cs

Add to a GameObject that should damage other entities such as the player or enemies. Choose the amount of damage it should deal, and the alignment of the damage. Alignment determines who will be damaged by a Damager. The player will be damaged by Enemy and Environment, but not Player. Enemies will be damaged by Player and Environment, but not Enemy. If the instakill setting is on, the damaged entity will instantly die, even if they're in an invincibility state. The healInstead setting will make the Damager heal entities by the set amount instead of damaging them. Make sure the Damager is on a GameObject that will be registering collisions or trigger collisions.

DisappearingPlatform2D.cs

Attach to a GameObject with a non-trigger 2D collider. When the specified tagName collides, the timeUntilDisappear timer begins. At the end of the timer, the onDisappear event is invoked. The disappearing of the platform can be handled in multiple ways, but a simple approach could be to drag in the platform's collider and SpriteRenderer components into the onDisappear event. Set the "enabled" property to false on each, then re-enable them onReappear.

EnemyHealth2D.cs

- Dependency: Damager.cs

Attach to an enemy GameObject to give them functionality for taking damage, dying, and dropping items. When a collision or trigger collision is detected with the specified damageTagName (e.g., "PlayerAttack"), the enemy will be hit and lose one health point. When the enemy's health reaches 0, it will die. If a drop item has been assigned, the enemy will spawn in a copy of that item when it dies. A drop item can be set up as a prefab and then assigned to the enemy from the project window.

Tips:

- If an animator has been assigned to the enemy, it will receive a trigger called "Hit" and "Death" when hit and dying, respectively.
- If a particular position for the drop to spawn is desired, assign it to the dropTransform field. If this position should be relative to the enemy's position, make it a child of the enemy.
- The onEnemyDeath event could be used to add to the player's score, open a door, etc.

- Add a health bar to the enemy by assigning a slider UI component to the healthBar field. This slider could live on a world space canvas above the enemy, or on the normal screen-space canvas.

EnemyHealthStages2D.cs

- Dependency: EnemyHealth2D.cs

Attach to a GameObject with EnemyHealth2D on it. Click the plus button on the component to create a new health stage. Enter a number into the healthThreshold field. When the enemy's health reaches that value, the onStageReached UnityEvent will be invoked.

EnemyProjectileAttack2D.cs

- Dependency: Projectile2D.cs

Attach to a GameObject to make it launch projectiles at the specified target. Set up a projectile prefab with the Projectile.cs component attached to it, then assign it to the projectile field of this component. Assign the target transform to the playerTransform field, likely the player GameObject. The flipToFacePlayer checkbox will flip the GameObject on the x-axis to face the player when a projectile is launched. fireRate controls the number of seconds between launches, and fireRateVariation will add a random offset from the fireRate to make the launches less consistent. If this GameObject is farther from the player than the maxDistanceFromPlayer distance, it will not launch projectiles. This script can be attached to a mobile enemy, a stationary enemy such as a turret, or anything else that fits the game's design.
Use the projectileDirection dropdown to limit the direction projectiles will be fired in.

EventRelay.cs

Attach to a GameObject and subscribe listeners to the event. Invoke the event from another UnityEvent with the InvokeEvent method. This can be used to group and organize complex event sequences. For instance, instead of having many listeners on the player's death event, the death event could call an EventRelay on a child GameObject, removing clutter from the player's inspector. The InvokeEvent method can be used from the component's context menu. The event could also be invoked via an animation event if attached to the same GameObject as an animator component.

FlashColor2D.cs

Attach to a GameObject with a SpriteRenderer to make its color flash to a different color and back. Call the Flash method from a UnityEvent to make it flash. Could be used on the player or enemies to show when they have taken damage.

FlickeringLight2D.cs

Attach to a GameObject with a Light2D component on it. Assign the light, then adjust the intensity and frequency values as desired. Each of these settings can be set from a UnityEvent. This component could be used for a flickering fireplace, a spooky streetlight, or a strobe light effect.

Floater.cs

Attach to a GameObject, then adjust the amplitude, frequency, and degreesOfRotation as desired. Then choose whether its rotation space should be Self (relative to its parent) or World (relative to the axes of the scene).

Inventory.cs

- Dependency: Item.cs

Attach to the player to give them the ability to pick up items. A layout group (VerticalLayoutGroup, HorizontalLayoutGroup, or GridLayoutGroup) can optionally be assigned to display the inventory items on the UI. If persistentInventory is enabled, the inventory will remain intact between scenes. If unchecked, the inventory will be reset on Start.

Item.cs

Attach to a GameObject with a 2D trigger collider to allow it to be picked up and added to the inventory. Fill out the itemData with a name to identify the item, and sprite to display on the UI. Check the isUnique box to only allow one of this item to be added to the inventory.

Lock2D.cs

- Dependencies: CollectableManager.cs, Inventory.cs

Attach to a GameObject with a 2D trigger collider. An unlock is attempted when the specified tagName is triggered, or when triggered and the keyToPress is pressed if requiredButtonPress is checked. An unlock attempt checks if the required counts of the required items are present in the inventory, and invokes onUnlock or onUnlockFailed events depending on the result. If keyLocation is set to CollectableManager, the lock will check the CollectableManager for the keys instead of the player inventory.

Parallax.cs

Attach to a GameObject in the background or foreground and adjust the parallaxStrength value to control the simulated depth of the parallax effect. Use values between 0 and 1 for objects in the background, and values above 1 for objects in the foreground. Add values to continuousScrolling to make the GameObject perpetually move in the given direction, in addition

to the parallax effect. This could be used for clouds slowly passing by, or an endless runner game with the background constantly moving.

The loopMode dropdown selects if the parallaxing object should loop on the horizontal axis, vertical axis, both axes, or neither axes. To set up looping, create a child GameObject with the same sprite as the parent. Then offset the child the exact amount such that it creates a continuous picture. Do this in each direction that the parallax will loop. So one on either side of the parent for horizontal, one above and below the parent for vertical, and 8 to each adjacent and diagonally adjacent position for both horizontal and vertical looping. Holding the V key in the scene view will allow for corner snapping to make this process much easier and more precise.

Patrol Scripts

PatrolSimple2D.cs:
The simplest of the patrol systems. Adequate for an elevator, simple enemy, moving hazard, etc. Attach to a GameObject, then assign a “point B” transform. The GameObject will move back and forth between its starting position and point B. A yellow line showing the patrol path will be drawn in the scene view.

PatrolMultiple2D.cs:
A more sophisticated patrol system. Assign a series of transforms as “waypoints” in the inspector, and the GameObject will move from point to point in order. The patrolType dropdown has 3 options: Loop will make the GameObject loop from the last waypoint back to the first waypoint. Ping Pong will make the GameObject travel through the waypoints to the last one, then back through to the first and repeat. Neither will make the GameObject travel through the waypoints, then stop at the last one. PatrolMultiple has public methods StopPatrolling() and StartPatrolling() that can be called from UnityEvents. Yellow lines showing the patrol path will be drawn in the scene view.

PatrolChase2D.cs:

- Dependencies: PatrolMultiple2D.cs, PlayerStealth2D.cs

PatrolChase extends PatrolMultiple to add a chasing state. PatrolChase extending PatrolMultiple means that PatrolChase retains all of PatrolMultiple’s functionality, and adds new functionality. The GameObject will begin chasing the player once the player is within a specified range. If the player leaves that range, the GameObject will return to its patrol path. It is not recommended to use a rigidbody or any physics interactions on a GameObject with PatrolChase attached.

Tips:

- If any patrol script is combined with EnemyProjectileAttack2D, it may be best to only use one of their flip-to-face options.
- In addition to enemies, PatrolChase2D could be used for a friendly NPC or companion character, or elusive items.

- If the chaseSpeed value on PatrolChase2D is negative, it will run away from the player instead of towards them.
- The distanceThreshold value on all patrol scripts controls the distance at which the GameObject stops moving towards the current waypoint, and moves on to the next one. This value may need to be adjusted depending on the scale and specifics of each game.

PlayerAudio2D.cs

- Dependency: PlayerMovement2D.cs

Attach to a GameObject with PlayerMovement2D or PlayerMovementAdvanced attached to it. Add sound effects for the various actions. Assign an AudioSource for the movement sounds, and an AudioSource for the jumping and landing sounds. The AudioSource for the movement sounds should be set to loop and not play on awake.

PlayerHealth2D.cs

- Dependency: Damager.cs

Attach to the player to give it damaged, dying, and respawning functionality.

healthType determines whether a health bar or health segments should be used. If using a health bar, assign an image with its type set to Filled. The health bar will dynamically fill itself as the player heals and gets damaged. If using segmented health, assign an image for each segment, equal in number to the chosen maxHealth. Each health segment could be discrete icons (such as hearts), or pieces of a larger whole that disappear as the player is damaged.

The UnityEvents for onPlayerDamaged, onPlayerHealed, and onPlayerDeath could be used to play special animations, play sounds, adjust a score, or modify the level.

PlayerMeleeAttack2D.cs

Attach to the player to give them a close-range attack. Create a 2D trigger collider as a child of the player and assign it to the hitbox field. Create a tag for player attacks (e.g., “PlayerAttack”) and tag the hitbox with it. By default, right-clicking the mouse will activate the melee attack. The hitbox will enable itself, wait for the amount of time specified by hitboxTime, then disable. The player’s animator can optionally be assigned to PlayerMeleeAttack to receive a “Melee” trigger. The public method EnableMeleeAttack() can be called from a UnityEvent to enable/disable the ability to melee.

PlayerMovement2D.cs

- Dependency: PlayerMovementBase.cs

Attach to the player to give them movement and animation. Create a child transform on the player and assign it to groundCheck. Then move it just below the player’s feet. Set the groundLayer to “Ground” (to add a new layer, click “Add Layer…” on the layer field of a

GameObject). Make sure to set any platform, elevator, or structure that the player should be able to jump on to the ground layer.

PlayerMovementAdvanced.cs

- Dependency: PlayerMovement2D.cs

PlayerMovementAdvanced contains all the functionality of PlayerMovement2D, but with the addition of dashing, wall jumping, and wall sliding. Do not put both components on the player, just use one of them. Both will work with PlayerAudio2D, PlayerHealth2D, etc. To set up wall jumping, create an empty transform as a child of the player. Place it slightly in front of the player's collider, then assign it to wallCheck. It is recommended to set wallLayer to the same layer as the ground check layer. Make sure the wall check cannot accidentally detect the player's collider.

PlayerMovementBase.cs

PlayerMovementBase is a base class that gives PlayerMovement2D.cs and PlayerMovementTopDown.cs shared functionality. It should be ignored and not directly added to any GameObjects.

PlayerMovementTopDown.cs

- Dependency: PlayerMovementBase.cs

An alternate player controller for games with a top-down perspective. Choose a movement mode to control which axes the player can move on. The "Eight Directions" option will be best for most top-down games. Assign a transform that's a child of the player GameObject to facingTransform. This transform will be automatically placed in front of the player by facingTransformDistance, and it will always rotate such that its x-axis is pointing away from the player. This can be used for a melee or projectile attack position, or anything else that might depend on the direction the player is facing.

PlayerProjectileAttack2D.cs

- Dependency: Projectile2D.cs

Attach to the player to give them a long-range projectile attack. Set up a projectile prefab and assign it. Create a child transform on the player, move it a little bit in front of the player, then assign it to launchTransform. If using PlayerMovementTopDown, it may be best to put the launchTransform as a child of the facingTransform. Assign the playerTransform and make any desired adjustments to the velocity and delay of the projectile attack. By default, left-clicking the mouse will launch a projectile. The launchDirection dropdown will choose between the projectile launching in the direction the launchTransform is facing, or launching directly towards the mouse cursor. The public method EnableProjectileAttack can be called from a UnityEvent to enable/disable the ability to launch projectiles. The methods SetRequireAmmo, SetAmmoCount, AdjustAmmoCount, and SetAmmoCost are available to configure the ammo settings.

PlayerStealth2D.cs

Attach to the player GameObject. When the player enters a trigger collider tagged with coverTag ("Cover" by default), they will be marked as "in cover." This will prevent enemies with the PatrolChase2D component from following them. If the player enters a trigger collider tagged with detectionTag ("Detection" by default), the onDetected UnityEvent will be invoked. If a transform has been assigned to stealthRespawn, the player will respawn there when they are detected. The stealthRespawn position can be changed via a UnityEvent.

Projectile2D.cs

Used for both PlayerProjectileAttack2D and EnemyProjectileAttack2D. Attach to a GameObject with a Rigidbody2D and a collider. Make sure the GameObject is correctly tagged (e.g., "PlayerAttack"). Convert the projectile GameObject into a prefab and assign it where needed. Make sure to create separate projectile prefabs for the player and for enemies.

Rotator.cs

Attach to a GameObject to give it a constant rotation. Add positive or negative values to the rotation axes. Then choose whether its rotation space should be Self (relative to its parent) or World (relative to the axes of the scene).

Score.cs

- Dependency: TextMesh Pro

Attach to any relevant GameObject. The public SetScore() and AdjustScore() methods can be called from UnityEvents to modify the score value. Assign a text element to display the score on the UI.

Tips:

- Use Score to track items collected, NPCs found, enemies defeated, times the player died, PlayerProjectileAttack2D ammunition, or any other quantity.
- Checking on persistentScore will make this score persist between scenes. Important to note is that the name of the GameObject that Score is attached to will be used to track which score value belongs to which Score component. So if the GameObject is named "Score Counter" in one scene, make sure it's named "Score Counter" in the next one as well. If another Score is kept track of, name its GameObject something different.

SelfDestruct.cs

Attach to any GameObject to destroy components and GameObjects from UnityEvents.

Tips:

- DestroyThisGameObject will destroy the GameObject that this script is attached to.

- DestroyTargetGameObject(GameObject target) will destroy the specified GameObject.
- DestroyAfterSeconds(float seconds) will destroy the GameObject that this script is attached to after the given number of seconds.
- DestroyTargetComponent(Component target) will destroy just the specified component (not the entire GameObject).
- DestroyAfterPhysics will destroy the GameObject on the next physics update.
- DestroyNextFrame will destroy the GameObject during the next frame of the game.

Spawner.cs

Attach to a GameObject, then drag that GameObject into a UnityEvent. Fill in the objectToSpawn field with a prefab or a GameObject from the hierarchy. Call SpawnObject() from the UnityEvent to create a copy of objectToSpawn at the position of the GameObject the spawner is attached to.

StartEvent.cs

Attach to any GameObject to add a UnityEvent that invokes on Start.

StickyPlatform2D.cs

Attach to a moving platform with a 2D trigger collider. Triggering GameObjects will stick to the surface of the moving platform.

Teleporter.cs

Attach to a GameObject with a 2D trigger collider. When a trigger is detected with the player's tag and the specified key is pressed, the player will be teleported to the destination transform.

Timer.cs

- Dependency: TextMesh Pro

Attach to any GameObject. Enter the desired number of seconds the timer should last for, and call StartTimer() from a UnityEvent. StopTimer() can be called to cancel an in-progress timer. PauseTimer() and ResumeTimer() methods are also available. When the timer is done, the onTimerFinished UnityEvent will be invoked. While the game is running, you can click the three vertical dots on the top-right of the Timer component in the inspector to test the various methods.

TriggerEvents2D.cs

Attach to a GameObject with a 2D trigger collider. When a trigger is detected with the specified tagName, the onTrigger event is invoked. When the collision is exited, the onTriggerExit event is invoked.