


DIG3713 Starter Package Documentation

Digital Worlds Unity 2D Starter Package by Logan Kemper - Version 1

[Player Controller Setup Guide](#)

Project Assets

Key: Utility, Enemy, Player, Level, Visuals

 File	T Description
ApplicationController.cs	Provides functionality for closing the game.
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.
Checkpoint.cs	Used to update the player's respawn position.
CollisionEvents2D.cs	Generic script for adding UnityEvents to 2D collisions.
DisappearingPlatform2D.cs	Attach to a platform and use the UnityEvents to disable its collider or whatever else needed to make the platform disappear.
EnemyHealth2D.cs	Gives enemies functionality for health, taking damage, and dropping items.
EnemyProjectileAttack2D.cs	Launches projectile attacks towards the player. Can be used for mobile or stationary enemies.
FlashColor2D.cs	Changes a SpriteRenderer to a new color, then back to the original color.
FlickeringLight.cs	Add to a light 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.



Project Assets

Key: Utility, Enemy, Player, Level, Visuals

 File	 Description
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.
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 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.
PlayerProjectileAttack2D.cs	Gives the player a projectile attack.
Projectile.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.

Project Assets

Key: Utility, Enemy, Player, Level, Visuals

 File	 Description
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.
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).

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). Peeking can be toggled on and off from UnityEvents by using the `EnablePeeking()` method. Change the offset vector 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.

ChangeScene.cs

This script 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.

Checkpoint.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.

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.

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 on `Reappear`.

EnemyHealth2D.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.

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.

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.

FlickeringLight.cs

Attach to a GameObject with a light component on it. Assign the light, then adjust the intensity and frequency values as desired.

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

- Dependency: 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 a required count of the required item is present in the inventory, and invokes onUnlock or onUnlockFailed events depending on the result.

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:

- Dependency: PatrolMultiple2D.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 EnemyProjectileAttack, it may be best to only use one of their flip-to-face options.
- In addition to enemies, PatrolChase could be used for a friendly NPC or companion character, or elusive items.
- 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

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, 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 (i.e. "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

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.

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. 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 public method EnableProjectileAttack can be called from a UnityEvent to enable/disable the ability to launch projectiles.

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).

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. When the timer is done, the onTimerFinished UnityEvent will be invoked.

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.

How to Set Up a Basic Player-Controlled Character

Set up movement:

- Right-click in the hierarchy > 2D Object > Sprites > Capsule.
- Rename the GameObject to “Player” or something similar.
- Change the GameObject’s tag to “Player”.
- Add Component > CapsuleCollider2D.
- Add Component > Rigidbody2D.
- On the rigidbody, change Collision Detection to Continuous, Sleeping Mode to Never Sleep, and Interpolate to Interpolate. These settings make the player’s movement smoother and more responsive, but won’t be necessary on most other rigidbodies in the game. Under Constraints, check Freeze Rotation on Z.
- Increasing the Gravity Scale will likely improve the feel of platforming. Try starting with a value of 3.
- Add Component > PlayerMovement2D.
- Right-click the player GameObject in the hierarchy > Create Empty.
- Drag the new child GameObject to be slightly under the capsule (around y = -1.05).
- Rename the new child GameObject to “Ground Check”.
- Drag Ground Check into the Ground Check field in PlayerMovement2D.
- If not already created, make a ground physics layer called by clicking Layer > Add Layer... and naming a blank layer “Ground”.
- On PlayerMovement2D, set Ground Layer to the new Ground layer.
- Create a platform by right-clicking in the hierarchy > 2D Object > Sprites > Square.
- Renaming the square to something like “Platform”.
- On the platform, Add Component > BoxCollider2D.
- Change the platform’s layer to Ground.
- Stretch out the platform horizontally and place it under the player.
- Play the game to test the player. The player should move left and right with the A and D keys and jump with the space bar.

Set up health, damage, and respawning:

- On the player, Add Component > PlayerHealth2D.
- In the hierarchy, right-click > UI > Canvas.
- In the CanvasScaler component, change UI Scale Mode to Scale With Screen Size.
- Change the reference resolution to 1920 by 1080.
- In the hierarchy on the Canvas GameObject, right-click > UI > Image.
- Rename the Image to “Health Segment”.
- On the health bar’s RectTransform component, click on the anchor presets button (red crosshair in a white box, says “middle” and “center” above and on the side).
- Change the health bar’s anchor to the top left setting.
- Double-click the health bar in the hierarchy to zoom in to it. In the scene view, move the health bar to the top left of the UI.
- Duplicate the health segment twice by right-clicking it in the hierarchy and selecting Duplicate.

- Move the duplicates in the scene view such that all three can be seen in the game view.
- Drag the three health segments into the Health Segments field in PlayerHealth2D.
- On PlayerHealth2D, Change Health Type to Segmented.
- Create a new empty GameObject in the hierarchy and rename it "Respawn Point". Change its tag to "Respawn".
- Drag Respawn Point into the Respawn Point field on PlayerHealth2D.
- Create a new square in the hierarchy. Go Tag > Add Tag... and add a tag called "Enemy".
- Add a BoxCollider2D to the new enemy square.
- Now play the game and test if moving the player into the enemy square removes one of the health segments. When the player's health reaches 0, the player should be teleported to the Respawn Point.