

DIG3715 Starter Package Documentation

Digital Worlds Unity 3D Starter Package by Logan Kemper - Version 1

Project Assets

Key: Utility, Enemy, Player, Level, Visuals

 File	 Description
ApplicationController.cs	Provides functionality for quitting or pausing the game.
BouncePad3D.cs	A bounce pad that applies an instant force to any GameObject with a Rigidbody that collides with it.
ButtonEvents3D.cs	Generic script for adding UnityEvents to button presses.
ChangeScene.cs	Change the current scene by index or by name.
Checkpoint3D.cs	Used to update the player's respawn position.
CollisionEvents3D.cs	Generic script for adding UnityEvents to 3D collisions.
Damager.cs	Add to a GameObject to make it deal damage to other entities.
EnemyHealth3D.cs	Gives enemies functionality for health, taking damage, and dropping items.
EnemyProjectileAttack3D.cs	Launches projectile attacks towards the player. Can be used for mobile or stationary enemies.
FaceCamera.cs	Attach to a GameObject to make it always face the main camera in the scene.
FirstPersonController.cs	A first-person character controller that uses physics-based movement.
FlashColor3D.cs	Changes a MeshRenderer's material color 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.
Item.cs	Add to a GameObject with a trigger collider to allow the player



Project Assets

Key: Utility, Enemy, Player, Level, Visuals

 File	 Description
	to add the item to their inventory.
Lock3D.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.
PatrolChase3D.cs	A more sophisticated patrol script for dynamic enemies. PatrolChase3D inherits from PatrolMultiple3D to extend its functionality and add a chasing state.
PatrolMultiple3D.cs	Moves a GameObject to its assigned waypoints. Can be used for platforms, NPCs, hazards, and more.
PatrolSimple3D.cs	Moves a GameObject back and forth between two positions. Can be used for platforms, NPCs, hazards, and more.
PlayerAudio3D.cs	Adds sound effects to FirstPersonController and ThirdPersonController.
PlayerHealth3D.cs	Gives the player health, dying, and respawning functionality.
PlayerMeleeAttack3D.cs	Gives the player a melee attack.
PlayerMovementBase.cs	A base class for giving shared functionality to player controllers.
PlayerProjectileAttack3D.cs	Gives the player a projectile attack.
Projectile3D.cs	Attach to a projectile prefab to give it launching and destroying behavior.
RaycastInteractable.cs	Used in conjunction with RaycastInteractor to add a UnityEvent to interactions.
RaycastInteractor.cs	Add to a player controller to interact with nearby objects. Intended for first-person controllers, but can work with other perspectives.
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.
Teleporter3D.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.
TriggerEvents3D.cs	Generic script for adding UnityEvents to 3D 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).

BouncePad3D.cs

Attach to an object in the level with a non-trigger 3D collider. When a GameObject that matches the specified tagName touches the bounce pad, this script will try to find a rigidbody 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.

ButtonEvents3D.cs

Attach to a GameObject with a 3D 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.

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.

Checkpoint3D.cs

Attach to a GameObject with a 3D 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.

CollisionEvents3D.cs

Attach to a GameObject with a non-trigger 3D 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.

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. Make sure the Damager is on a GameObject that will be registering collisions or trigger collisions.

EnemyHealth3D.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.

EnemyProjectileAttack3D.cs

- Dependency: Projectile3D.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 faceTarget option controls how the enemy should turn to face the target, whether directly, only horizontally on the Y-axis, or not at all. 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.

FaceCamera.cs

Attach to a GameObject to make it always point towards the main camera in the scene on the Z-axis.

FirstPersonController.cs

- Dependency: PlayerMovementBase.cs

Attach to the player GameObject with a Rigidbody and a capsule collider. Assign the main camera and set it as a child of the player. 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. On the Rigidbody, change Interpolate to Interpolate, Collision Detection to Continuous, and freeze rotation on all axes under Constraints. Experiment with the various movement, jump, and gravity settings to find a combination that best suits the game.

FlashColor3D.cs

Attach to a GameObject with a MeshRenderer to make its material 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.

Lock3D.cs

- Dependency: Inventory.cs

Attach to a GameObject with a 3D 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.

Patrol Scripts

PatrolSimple3D.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.

PatrolMultiple3D.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. The faceTarget dropdown controls how the patrolling GameObject should turn to face the next target: directly, horizontally on the Y-axis, or not at all.

PatrolChase3D.cs:

- Dependency: PatrolMultiple3D.cs

PatrolChase3D extends PatrolMultiple3D to add a chasing state. PatrolChase3D extending PatrolMultiple3D means that PatrolChase3D retains all of PatrolMultiple3D'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.

Tips:

- In addition to enemies, PatrolChase3D could be used for a friendly NPC or companion character.
- 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.

PlayerAudio3D.cs

- Dependency: PlayerMovement.cs

Attach to a GameObject with PlayerMovement on 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.

PlayerHealth3D.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, adjust a score, or modify the level.

PlayerMeleeAttack3D.cs

Attach to the player to give them a close-range attack. Create a 3D trigger collider as a child of the player and assign it to the hitbox field. Add a Rigidbody to the GameObject with the collider, and set isKinematic to true. If using the FirstPersonController, consider making the damaging collider GameObject a child of the camera, so that it moves with the first person perspective. 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.

PlayerMovementBase.cs

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

PlayerProjectileAttack3D.cs

- Dependency: Projectile3D.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. If using the FirstPersonController, it may be best to put the launchTransform GameObject as a child of the camera, so that it moves with the first person perspective.

Projectile3D.cs

Used for both PlayerProjectileAttack3D and EnemyProjectileAttack3D. Attach to a GameObject with a rigidbody 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.

RaycastInteractable.cs

Attach to a GameObject with a 3D collider on it. When [RaycastInteractor](#) interacts with it, the onInteraction UnityEvent will be invoked.

RaycastInteractor.cs

- Dependency: RaycastInteractable.cs

Attach to a GameObject on the player and assign the transform that the raycast should come from to `interactorSource`. For a first-person controller, the source should be the camera. For a third-person controller, the source should be a transform pointing straight from the player's head. When the `interactKey` is pressed, a ray is shot out from the source and travels the length of `interactDistance` in a straight line. If the ray intersects with a GameObject with a collider and the [RaycastInteractable](#) component, it will call the `Interaction()` method on it.

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, `PlayerProjectileAttack` 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.

Teleporter3D.cs

Attach to a GameObject with a 3D 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.

TriggerEvents3D.cs

Attach to a GameObject with a 3D 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.