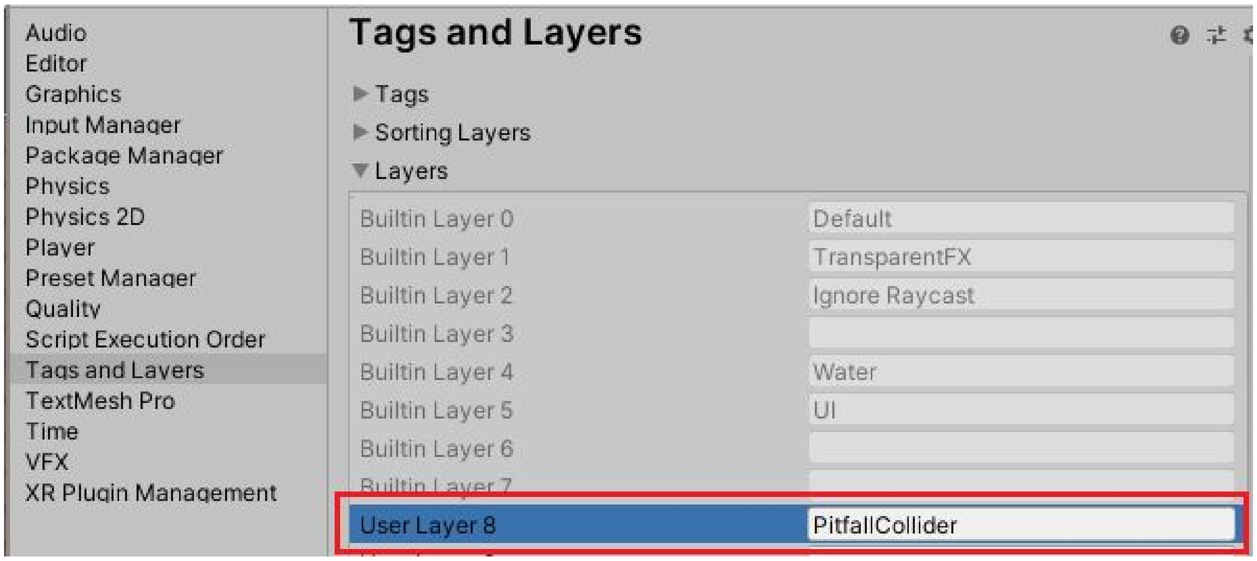
# Topdown2DPitfall

The Topdown2DPitfall asset is an easy-to-implement and highly customizable pitfall solution for top down 2D games. It provides a pitfall animation modifier that makes all game objects in any animation state seem like they're falling. It also lets you customize how gameobjects so that each fall is hand-crafted to their needs.

## Quick Start

In this example, we will get a player with basic movement functionality to start falling into pits. To start using the Topdown2DPitfall System:

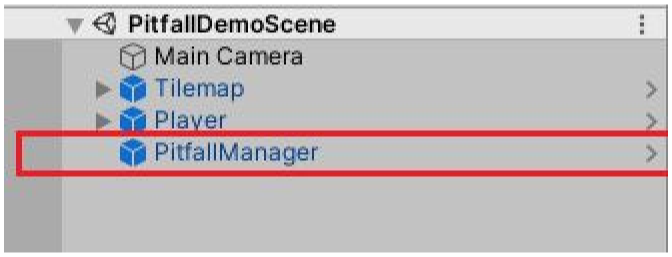
1. Import the package into your project
2. Go to **Edit** > **Project Settings** > **Tags and Layers**
3. Add a new User Layer called: **PitfallCollider**



Tags and Layers

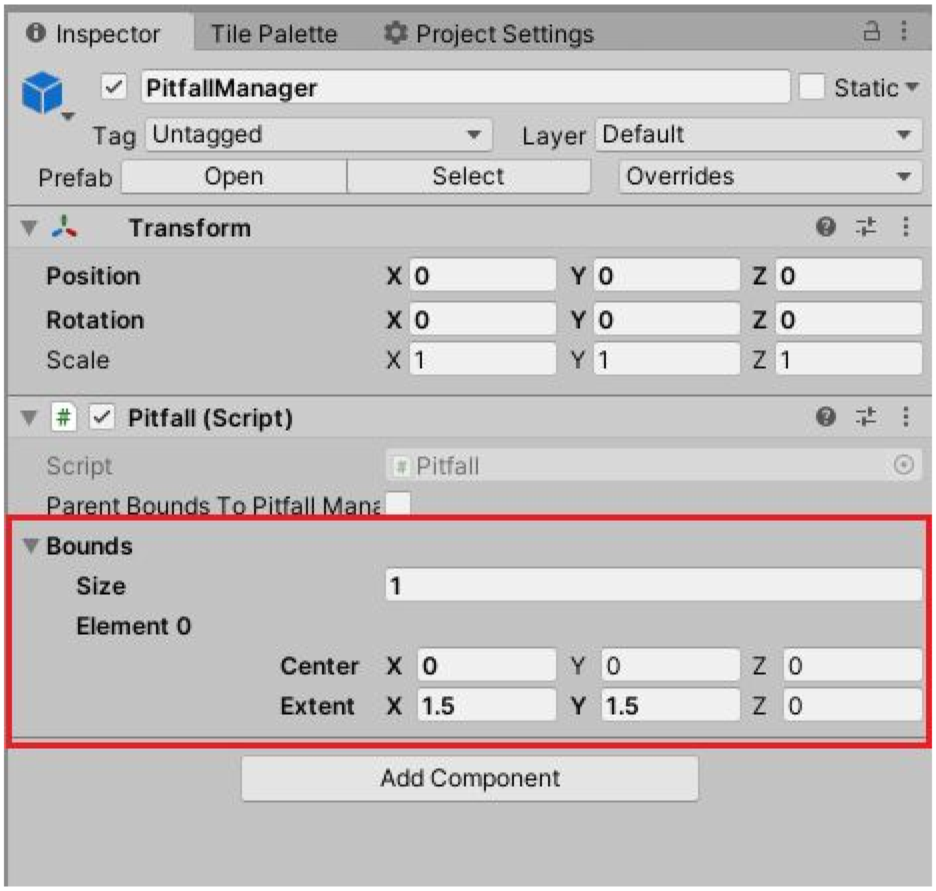
| | |
|---|---|
| Builtin Layer 0 | Default |
| Builtin Layer 1 | TransparentFX |
| Builtin Layer 2 | Ignore Raycast |
| Builtin Layer 3 | |
| Builtin Layer 4 | Water |
| Builtin Layer 5 | UI |
| Builtin Layer 6 | |
| Builtin Layer 7 | |
| User Layer 8 | PitfallCollider |

4. From your **Project** window, go to **Packages** > **Topdown2DPitfall**
5. Drag a **PitfallManager** object into your Scene.



6. In your **Hierarchy** tab, select the **PitfallManager** object.
7. In the **PitfallManager** Inspector tab, set the bounds size to 1.

8. Set the X,Y extents of your bounds. An extent is half the dimension of your pit. For example, if your pit is 4x4. The extent fields should be x:2, y:2, z:0
   **TIP**: Reduce the bounds a little more so it feels like the player is actually stepping off the edge and into the pit.



Inspector | Tile Palette | Project Settings

PitfallManager — Static

Tag: Untagged — Layer: Default

Prefab: Open | Select | Overrides

**Transform**

| | X | Y | Z |
|---|---|---|---|
| Position | 0 | 0 | 0 |
| Rotation | 0 | 0 | 0 |
| Scale | 1 | 1 | 1 |

**Pitfall (Script)**

Script: Pitfall

Parent Bounds To Pitfall Man

Bounds

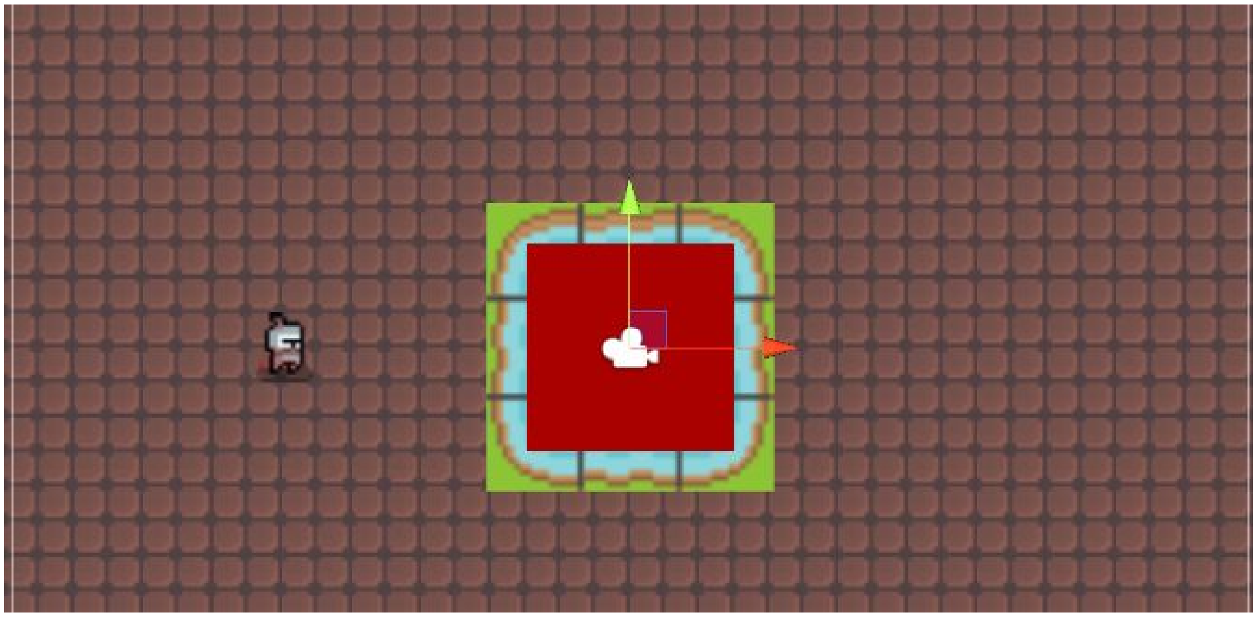
Size: 1

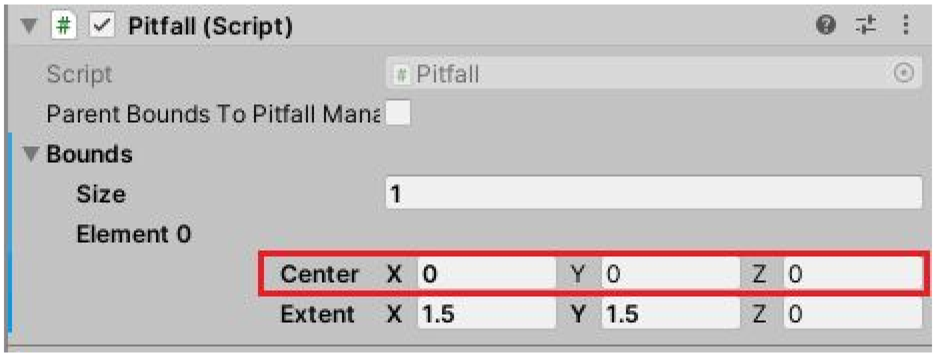
Element 0

| | X | Y | Z |
|---|---|---|---|
| Center | 0 | 0 | 0 |
| Extent | 1.5 | 1.5 | 0 |

Add Component

9. While you have the **PitfallManager** selected in the **Hierarchy**, a red box will display the extents of your pit. This box represents where the pitfall exists in game space. It will not appear ingame. Use the bounds Center X and Y values to set the pit's position in game space. Place it over your pit.

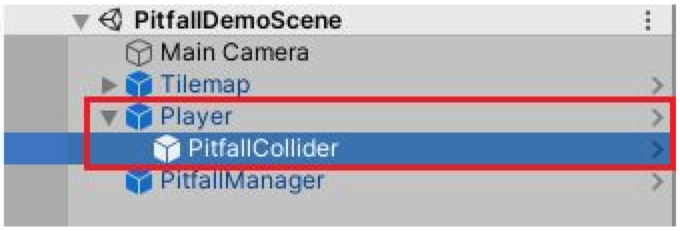




Pitfall (Script)

| | |
|---|---|
| Script | Pitfall |
| Parent Bounds To Pitfall Manɛ | ☐ |
| Bounds | |
| Size | 1 |
| Element 0 | |
| Center | X 0 Y 0 Z 0 |
| Extent | X 1.5 Y 1.5 Z 0 |

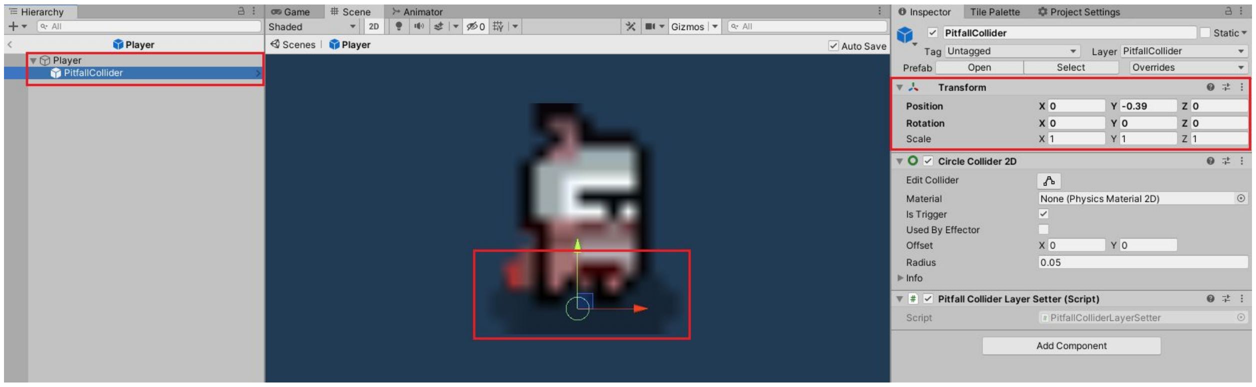
10. Add more bounds and center them over any other pits you may have in your level.
11. From your **Project** window, go to **Packages** > **Topdown2DPitfall** and select the **PitfallCollider** object.
12. Drag the **PitfallCollider** into your **Hierarchy** tab and drop it on the player gameobject.



PitfallDemoScene
- Main Camera
- Tilemap
- Player
  - PitfallCollider
- PitfallManager

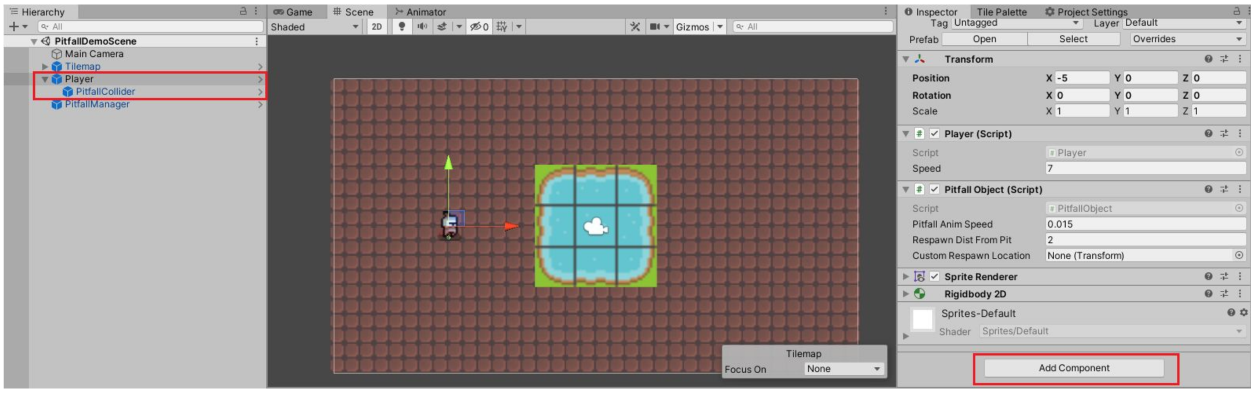
13. Set the PitfallCollider's position to be at the bottom of your gameobject.

    The **pitfallCollider** can be placed on any object you want to be able to fall into the pit. The Pitfall collider is what detects pits and triggers the pitfall. In a top down game with depth, you want objects to detect pits with their feet, or base



```
PitfallCollider                         Static
Tag Untagged              Layer PitfallCollider
Prefab   Open    Select    Overrides
Transform
Position   X 0     Y -0.39   Z 0
Rotation   X 0     Y 0       Z 0
Scale      X 1     Y 1       Z 1
Circle Collider 2D
Edit Collider
Material          None (Physics Material 2D)
Is Trigger        ✓
Used By Effector
Offset            X 0     Y 0
Radius            0.05
Info
Pitfall Collider Layer Setter (Script)
Script            PitfallColliderLayerSetter
```

14. Select the **player** gameobject in your **Hierarchy** window. In your **inspector** window, click **add component**.
15. Add the **Pitfall Object** component.



```
Tag Untagged              Layer Default
Prefab   Open    Select    Overrides
Transform
Position   X -5    Y 0     Z 0
Rotation   X 0     Y 0     Z 0
Scale      X 1     Y 1     Z 1
Player (Script)
Script                    Player
Speed                     7
Pitfall Object (Script)
Script                    PitfallObject
Pitfall Anim Speed        0.015
Respawn Dist From Pit     2
Custom Respawn Location   None (Transform)
Sprite Renderer
Rigidbody 2D
Sprites-Default
Shader   Sprites/Default
```

If you were to run the game, your player would now fall into any pit it came in contact with and respawn at the pit's edge! So far we have not stopped the player from moving or disabled any inputs. These setup actions can be completely different based on **what** is falling into the pit, so we need to configure this on a per-object basis. This is handled in the script that controls this object.

For our player example. These conditions would be coded into the **player.cs** script.

1. Open you player.cs script.
2. Implement the interface: IPitfallObject

   IPitfallObject gives you two methods: PitfallActionsBefore and PitfallActionsAfter. These methods are called right before the pitfall is triggered and directly after. These methods are useful for doing things like:

   - disabling/enabling movement
   - disabling/enabling interaction with other objects

- Take damage from falling
- Triggering and ending animations
- Transitioning to different levels below the pit

Lets use these methods to stop the player from being able while he is falling and enabling his movement once he respawns.

The following is example code from the demo scene Player.cs script. Review the script to see how you can use these methods in your own project:

```
public void PitfallActionsBefore() {
    isMovable = false;
}

public void PitfallResultingAfter() {
    isMovable = true;
}
```

Now when you run your game and walk into the pit, the player will not be able to move once he falls into the pit.  Once the pitfall animation executes tand the player respawns, he will regain control.

## Configuration Options

Topdown2dPitfall has several configuration options.  Use these to customize how pitfall interactions play out.

<u>PitfallManager:</u>
**-ParentBoundsToPitfallManager**: This option makes your pit **bounds** move when you move your PitfallManager.

<u>PitFallObject.cs:</u>
**-PitfallAnimSpeed**: Controls how quickly the falling animation plays.  Default speed is .015.
**RespawnDistFromPit**: If you are respawning this object dynamically beside the pit, this value determines how far away from the pit this object will respawn. Default is 2.

**CustomRespawnLocation**: If you supply a custom respawn location the pitfall object will respawn here every time it falls. This value can be changed at runtime.

## Tips and Tricks

-Implement **IPitfallChecker** in your scripts to set conditions about when to ignore pitfalls.  This is useful for situations like creating a flying enemy, that won't necessarily fall into a pit if it hovers over it, but will drop if it gets stunned.
-Change the **CustomRespawnLocation** at runtime to have players or enemies respawn at different locations on the map.  This is useful if you are making a dungeon game and your player "falls" down to the next level
-Check **ParentBoundsToPitfallManager** and attach the pitfallManager to any level pieces that are used in Procedural Generation.  This will let the bounds position update during runtime so level pieces that spawn in different positions will have their bounds in the proper place.