2020

Drag Me

A 2D AND 3D DRAG AND DROP SOLUTION FOR UNITY

(C) OVER ONE STUDIO LLC 2019-2020

Getting Started

Following these instructions will get your copy of Drag Me up and running in your Unity project.

Note: Code comments have been annotated for use with the VS Code plugin Better Comments. Viewing the source within VS Code with the Better Comments plugin installed will provide a better experience.

Foreword

Thank you for purchasing **Drag Me**, your 2d and 3d drag and drop solution for Unity 3d. **Drag Me** was initially developed for a new tactics card game. It worked so well in our internal development that we thought we should share our work with the Unity community.

The art assets included were created by Kyrise and are only a subset of the overall asset pack. You may download the asset pack in its entirety at Itch.io Kyrise's Free 16x16 RPG Icon Pack. Please consider making a donation!

Installing

Download and import **Drag Me** from the Unity Asset Store.

Note: Make sure all items are selected before clicking the import button

Quick Start

After downloading and importing Drag Me, open the **DragMe 2d Demo** or the **DragMe 3d Demo** scene and click play.

*Note: The demo scenes can be found within the **Assets/DragMe/Demo/Scenes** directory.*

Support

If you require assistance or want to submit a feature request, kindly send an email to dragme_support@over-one.studio.

Demo

Opening and running the **DragMe 2d Demo** or the **DragMe 3d Demo** scene is a great way to visualize the features that **Drag Me** has to offer.

The 3d demo is split into the following scenes:

1. DragMe 3d Demo – Start here.
2. [1] Click & Drag – Demonstrates the basic movement behavior of a *DragMe* component.
3. [2] Drag & Place – Demonstrates the ability to place components.
4. [3] Stack – Demonstrates the ability to create a stack of components.
5. [4] Stack Interaction – Demonstrates the ability to work with an existing stack of components.
6. [5] Playground – Provides the ability to dynamically change the internal state of components and test-out different functionality.

The 2d demo is not a multi-scene example, but demos the use of *DragMe* components to create a simple *inventory system*. You can drag items into the equipment section to place or replace items. If you replace an item or drop an item outside an inventory square, the *InventoryController* in the scene will find an available inventory slot to place the item.

Each scene can be run independently allowing you to jump back to specific scenes, configure component settings and test out different things in isolation.

If while testing, you encounter behavior that doesn't seem quite right, kindly send an email to dragme_support@over-one.studio so we can sort it out and make **Drag Me** a better asset!

Next Steps

Following these instructions will add usable and configured *DragMe* components to a new or existing scene in your project.

After importing **Drag Me** from the Unity Asset Store, new menu items have been added to the *GameObject* menu within the editor and the *Create* menu within the hierarchy window. Examples of where these menu items can be seen below.

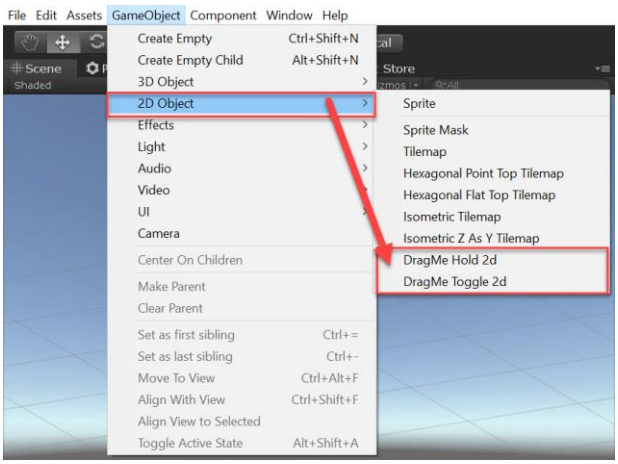

Figure 1: DragMe Hold 2d and DragMe Toggle 2d from the GameObject menu in the Unity editor.

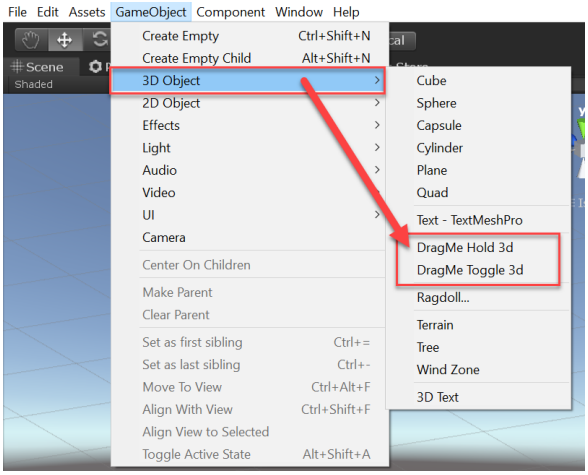

Figure 2: DragMe Hold 3d and DragMe Toggle 3d from the GameObject menu in the Unity editor.

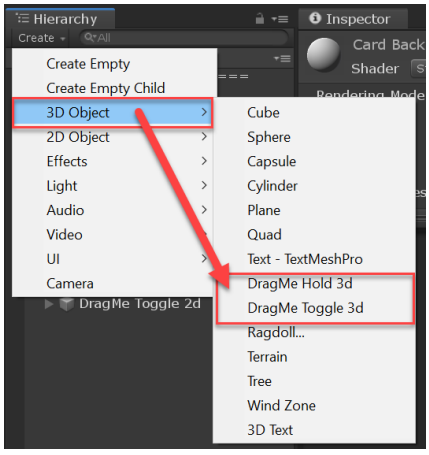

Figure 3: DragMe Hold 3d and DragMe Toggle 3d from the Create menu in the Hierarchy window.

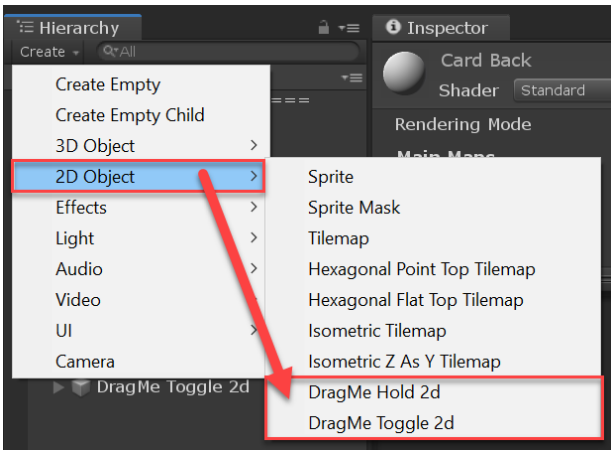

Figure 4: DragMe Hold 2d and DragMe Toggle 2d from the Create menu in the Hierarchy window.

Drag Me supports both 2d and 3d. The *DragMe* component itself is configured by the *DragMeConfig* asset it is assigned. The configuration assets provided by **Drag Me** are:

- **DragMe Hold 2d Config**: Supports drag & drop of 2d sprites by holding the selection button.

- **DragMe Hold 3d Config**: Supports drag & drop of 3d models by holding the selection button.
- **DragMe Toggle 2d Config**: Supports drag & drop of 2d models by toggling the selection button.
- **DragMe Toggle 3d Config**: Support drag & drop of 3d models by toggling the selection button.

*Note: The four pre-configured DragMeConfig assets can be found within the **Assets/DragMe/Resources** directory of the Project window. These can be duplicated and changed to meet the needs of your specific project. Simply drag your new configuration asset into the Config selector of your DragMe component.*

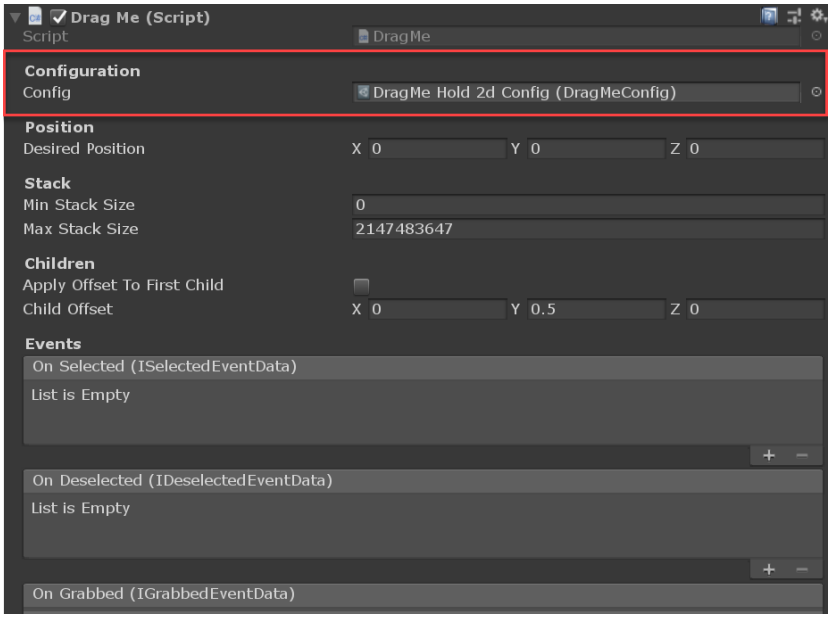

Figure 5: Config selector of a DragMe component in the Inspector window.

To add a *DragMe* component to a new or pre-existing scene you sould first decide if your game is 2d and will use sprites, or 3d and will use models. Once you've decided upon your 2d vs 3d requirements you can use one of the context menus from our previous example to add a new *DragMe* gameObject to your scene.

Note: The differences between a 2d and 3d DragMe gameObject are few but important. Using the context menus will create a new gameObject, wire-up all default event callbacks, and add the correct rigidbody with colliders.

2d *DragMe* components are restricted to the **XY** plane and are affected by their **Z**-position and sprite layering system while 3d *DragMe* components can be used anywhere in world space. Make sure to check out the **DragMe 2d Demo** scene to see how 2d components can be used with the sprite layering system.

Note: Adding a new DragMe component will check your scene for an Input Source*. If an InputSource is not found, a new gameObject will be added to your scene with the UnityInputSource component.*

Note: Repeat these steps or duplicate your existing gameObjects to create more. These new instances can then be 'stacked' and/or removed from one another.

If you are feeling adventurous you can build a *DragMe* gameObject from scratch. You can use the methods found within the **Actions.cs** script as a reference to see how the rigidbody and colliders are configured for either 2d or 3d support. This script can be found within in **Assets/Scripts/Editor** directory.

Scripting

Reading through the following section will inform you about the **Drag Me** Api while providing a clear description of component properties and the data they make available to you, and your code.

The **Scripts** directory of the project is organized in the following manner:

- **Components**: Contains classes that are attached to your Unity gameObjects. These will most likely inherit from the *MonoBehaviour* class or inherit from an abstract class that does.
    - **Abstractions**: Contains interfaces and/or abstract classes implemented or inherited by component classes.
- **Data**: Contains classes that are used by components. These will most likely inherit from *ScriptableObject*, be a POCO *(plain old CLR object)*, or be an enumeration of some kind.
    - **Abstractions**: Contains interfaces and/or abstract classes implemented or inherited by data classes.
    - **Events**: POCO classes that are used by the **Drag Me** event system.
- **Editor**: Contains classes and methods that are integrated into the Unity editor.
- **Events**: Contains classes inheriting from the *UnityEvent* class. These events use the eventData classes and are invoked by the **Drag Me** event system.
- **Extensions**: Contains static classes with extension methods used by **Drag Me**.
- **Integrations**: Contains classes used for an adapter pattern to integrate Unity or other 3rd-party libraries with **Drag Me**.
    - **Abstractions**: Contains interfaces and/or abstract classes implemented or inherited by integration classes.
    - **Unity**: Contains classes that are attached to your Unity gameObject to facility the integration between **Drag Me** and Unity.

Input Source

The *InputSource* is a simple solution for receiving and directing user input to *DragMe* components. **Drag Me** itself provides an integration called *UnityInputSource*. The *UnityInputSource* inherits from the *InputSourceBase* class, implements the *IInputSource* interface and provides the following fields:

- *Grab*: *True when the Left Mouse Button is Pressed, false otherwise.*

- *Hold*: *True when the Left Mouse Button is Held, false otherwise.*

- *Release*: *True when the Left Mouse Button is released, false otherwise.*

If you are using a different input library or framework in your project, you can complete the following to implement a custom *InputSource*.

- Write a class inheriting from the InputSourceBase class.
- Implement the IInputSource interface within that class.
- Create a new GameObject within your Scene and add that class to this GameObject.
- Drag your GameObject with your class attached into your Project window creating a new prefab.
- Finally set the 'InputSource' within a DragMeConfig asset through the Unity Inspector to your InputSource prefab.

Note: If there is an input library or framework you would like to see ***Drag Me*** *integrated with, kindly send an email to dragme_support@over-one.studio.*

DragMeConfig

The DragMeConfig is a class inheriting from the *ScriptableObject* class. It provides the 'system-level' configuration for DragMe and its components.

Note: You can create unique DragMeConfig assets by using the 'Create' menu within the Project window.

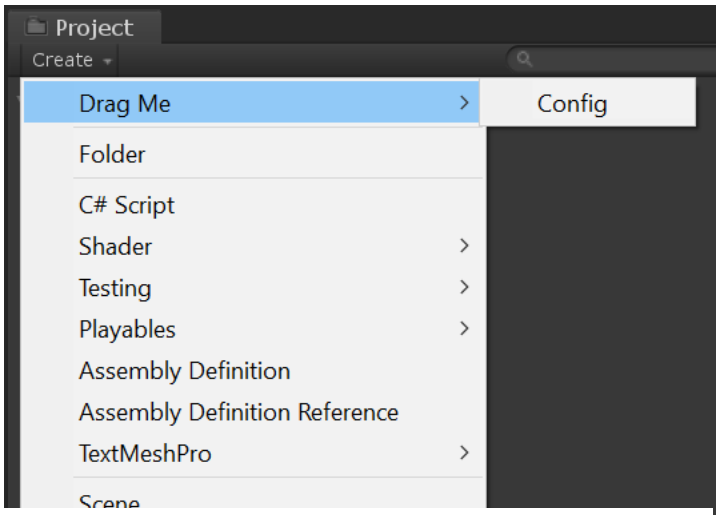

Figure 6: Creating a DragMe Config asset through the Create menu of the Project window.

The *DragMeConfig* implements the *IDragMeConfig* interface and contains the following fields:

- **InputSource**: The implementation of the *IInputSource* interface *DragMe* components will use to query for user input.

- **Camera**: The camera that will be used by *DragMe* components to create a unity *Plane*, allowing components to be clicked and dragged. If no camera is specified **Camera.main** will be used.

- **FixedUpdate**: True if *DragMe* components will update during *FixedUpdate()*, false if *DragMe* components should update during *Update()* instead.

- **CollisionType**: Indicates whether **Drag Me** will use the 2D or 3D physics system.

- **DragType**: Indicates the behavior used when grabbing and dragging an object.

- **Orientation**: The orientation in which the drag plane is created, allowing *DragMe* components to be moved.

- **GrabMask**: The LayerMask containing objects that can be grabbed.

- **PlacementMask**: The LayerMask containing objects that allow a grabbed object to be placed.

- **MaxDistance**: The maximum distance from the **Camera** that an object can be grabbed.

- **TriggerInteraction**: The *QueryTriggerInteraction* used when a 'Trigger' collider is encountered.

- **DragHistoryMaxLength**: The number of *DraggedEventData* entries retained when dragging a *DragMe* component.

- **HideMouseOnDrag**: True if the mouse cursor should be hidden when dragging a *DragMe* component, false otherwie.

- **DebounceTime**: Time in seconds taken for a *DragMe* component to be grabbed after being released.

- **DistanceThreshold**: Time in seconds taken for a *DragMe* component to be grabbed after being released.

[End DragMeConfig]

Events

The *DragMe* component provides events that you can listen to from within your code. These events are also available within the Unity Inspector. Creating a *DragMe* gameObject through the GameObject menu or the Create menu of the Hierarchy window will set the default events for you. If you attempt to build a *DragMe* gameObject from scratch, you will need to set the events on your component explicitly.

Note: The demo scenes use the events within the inspector to enable the drag & drop functionality, including playing soundFx and trigger animations created within the Unity Animator. This is how the 'stutter' animation is played when the mouse selects a card within the demo.

Each event inherits from the *UnityEvent<T>* class, implements its own interface, and provides the following data from their corresponding event data class:

e_Selected: Called when the *DragMe* component is selected.

- **SelectedComponent**: Reference to the *DragMe* component that the mouse is hovering over.
- **MousePosition**: The position the mouse was in when the **SelectedEvent** was fired.

e_Deselected: Called when the *DragMe* component is deselected.

- **DeselectedComponent**: Reference to the *DragMe* component that the mouse was hovering over.
- **MousePosition**: The position the mouse was in when the **DeselectedEvent** was fired.

e_Grabbed: Called when the *DragMe* component is Grabbed.

- **GrabbedComponent**: Reference to the *DragMe* component that was grabbed.
- **MousePosition**: The position the mouse was in when the **GrabbedEvent** was fired.

e_Dragged: Called when the *DragMe* component is Dragged.

- **DraggedComponent**: Reference to the *DragMe* component that was dragged.

- **PreviousMousePosition**: The position the mouse was in when the previous **DraggedEvent** was fired.

- **CurrentMousePosition**: The position the mouse was in when the **DraggedEvent** was fired.

e_Released: Called when the *DragMe* component is released.

- **ReleasedComponent**: Reference to the *DragMe* component that was released.

- **MousePosition**: The position the mouse was in when the **ReleasedEvent** was fired.

e_Placed: Called when the *DragMe* component is placed.

- **PlacedComponent**: Reference to the *DragMe* component that the mouse released.

- **PlacementComponent**: Reference to the *DragMe* component that the **PlacedComponent** has been placed.

- **MousePosition**: The position the mouse was in when the **PlacedEvent** was fired.

e_Blocked: Called when the *DragMe* component is blocked from being placed.

- **BlockedComponent**: Reference to the *DragMe* component that was blocked.

- **PlacementComponent**: Reference to the DragMe component that is blocked.

- **MousePosition**: The position the mouse was in when the **BlockedEvent** was fired.

e_Reset: Called when the *DragMe* component is reset.

- **ResetComponent**: Reference to the *DragMe* component that was reset.

- **MousePosition**: The position the mouse was in when the **ReleasedEvent** was fired.

[End Events]

DragMe Component

DragMe is a partial class inheriting from *MonoBehaviour*. It is this sole component responsible for providing 2D and 3D drag and drop functionality within Unity. The *DragMe* class is split across multiple files, **DragMe.**cs, **DragMeActions.**cs, and **DragMeState.cs**. Doing so allows us to separate the internal fields from the actual implementation providing a cleaner Api. It also allows you to extend the *DragMe* class in the same way we implemented our drag & drop functionality.

*Note: Most of the functions within the **DragMe** class are private and only accessible from within the DragMe class. This approach is intentional and helps keep DragMe components behaving consistently. To add additional functionality to the DragMe component, you can extend the DragMe class by implementing your own DragMe partial class. Doing so will give you access to all internal methods and properties to do whatever you want. See the **DragMe.cs** and **DragMeActions.cs**, and the **DragMeState.cs** script files for an example.*

The *DragMe* class implements the *IDragMe* interface and contains the following fields:

- **Parent**: Returns the parent of the *DragMe* transform.
- **WorldPosition**: Returns the position of the attached *Rigidbody* or *Rigidbody2d*.
- **MyDragMeComponents**: Returns a collection of DragMe components containing myself and my children.
- **MyChildDragMeComponents**: Returns a collection of DragMe components containing my children.
- **MyParentDragMeComponents**: Returns my parent DragMe component, null if I do not have one.
- **MyRootDragMeComponents**: Returns my root DragMe component, null if I do not have one.
- **MySiblingDragMeComponents**: Returns a collection of DragMe components containing my siblings.
- **MySeniorDragMeComponents**: Returns a collection of DragMe components containing my seniors.
- **CanSelect**: Returns true if this DragMe component can be selected, false otherwise.
- **CanDeselect**: Returns true if this DragMe component can be deselected, false otherwise.

- **CanGrab**: Returns true if this DragMe component can be grabbed, false otherwise.
- **CanDrag**: Returns true if this DragMe component can be dragged, false otherwise.
- **Available**: Returns true if this DragMe component is available for placement, false otherwise.
- **Held**: Returns true if this component is held, false otherwise.
- **Placed**: True if this DragMe component has been placed, false otherwise.
- **Selected**: True if this DragMe component is selected, false otherwise.
- **Grabbed**: True if this DragMe component is Grabbed, false otherwise.
- **Config**: Config used to configure the behaviour of this component. If null, a default asset will be loaded.
- **ShowDebug**: True to render the dragPlane and dragPlane normal within the inspector, false otherwise.
- **DebugDrawOffset**: Offset used to move the rendered debug dragPlane away from the grabbed objects transform.
- **GrabTimer**: Used to debounce the selection and grabbing of a previously grabbed component.
- **SelectedTime**: Time in seconds since the DragMe component was selected.
- **DeselectedTime**: Time in seconds since the DragMe component was deselected.
- **GrabbedTime**: Time in seconds since the DragMe component was grabbed.
- **DraggedTime**: Time in seconds since the DragMe component was dragged.
- **ReleasedTime**: Time in seconds since the DragMe component was released.
- **PlacedTime**: Time in seconds since the DragMe component was placed.
- **BlockedTime**: Time in seconds since the DragMe component was blocked.
- **MinStackSize**: Indicates the minimum number of cards that can be on this stack before selection and grabbing is disabled.
- **MaxStackSize**: Indicates the maximum number of cards that can be added to this stack.

- **ApplyOffsetToFirstChild**: True if the first child of this component should have the offset applied, false otherwise.

[End DragMe Component]