Falcon Selection Box

1.2

Documentation

Table of contents

Introduction

What is Falcon Selection Box?

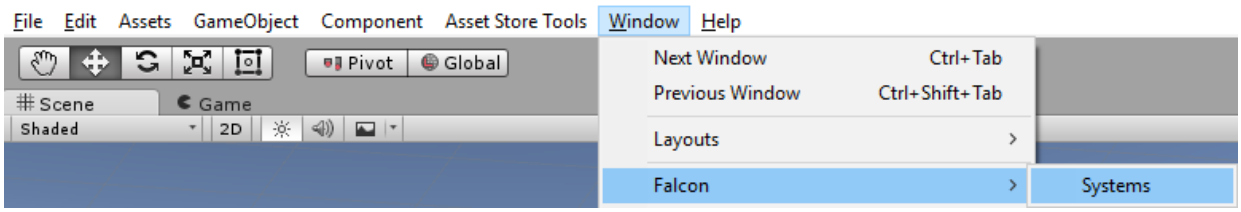
A tool that allows you to quickly and easily add a pointer selection tool like the one in the most popular real time strategy games.

Features

- Works for 2D and 3D game objects.
- Easy to configure as single selection, multi-selection or both.
- Uses Unity's event system for optimal performance.
- Supports click and drag events.
- Handles dragging on any 3D or 2D surface.
- Easy to access currently selected objects from script.
- Easy to set an object as selectable.
- Window editor to setup anything with 1 click.
- 2D and 3D example scenes.
- Modular.

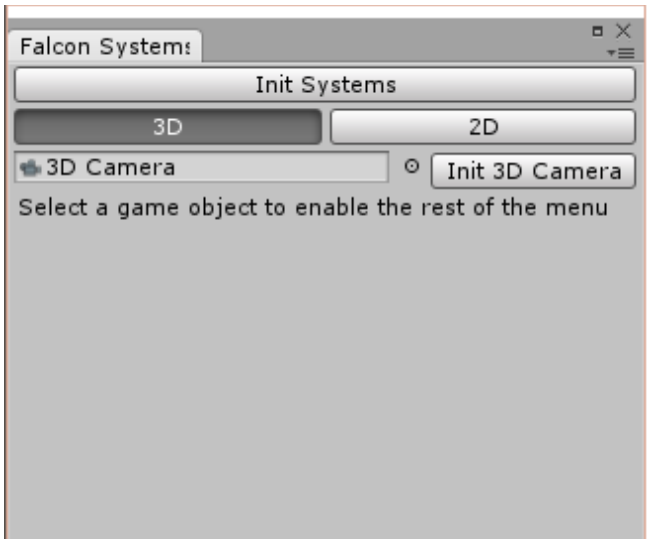
Quick Start Step-by-step: 3D

1. Open a new scene
2. Open the Falcon Systems helper window.
    - Window → Falcon → Systems

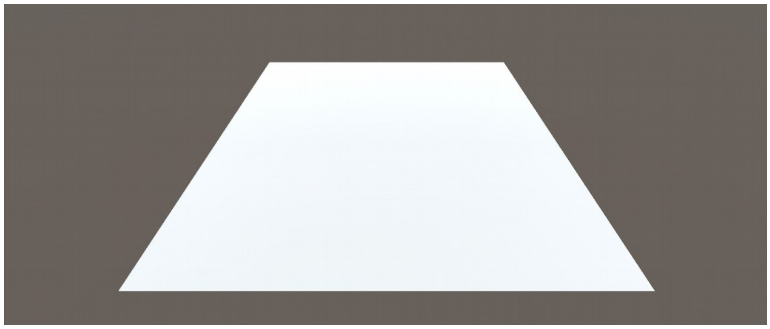


    -

3. Make sure the “3D” button is selected.



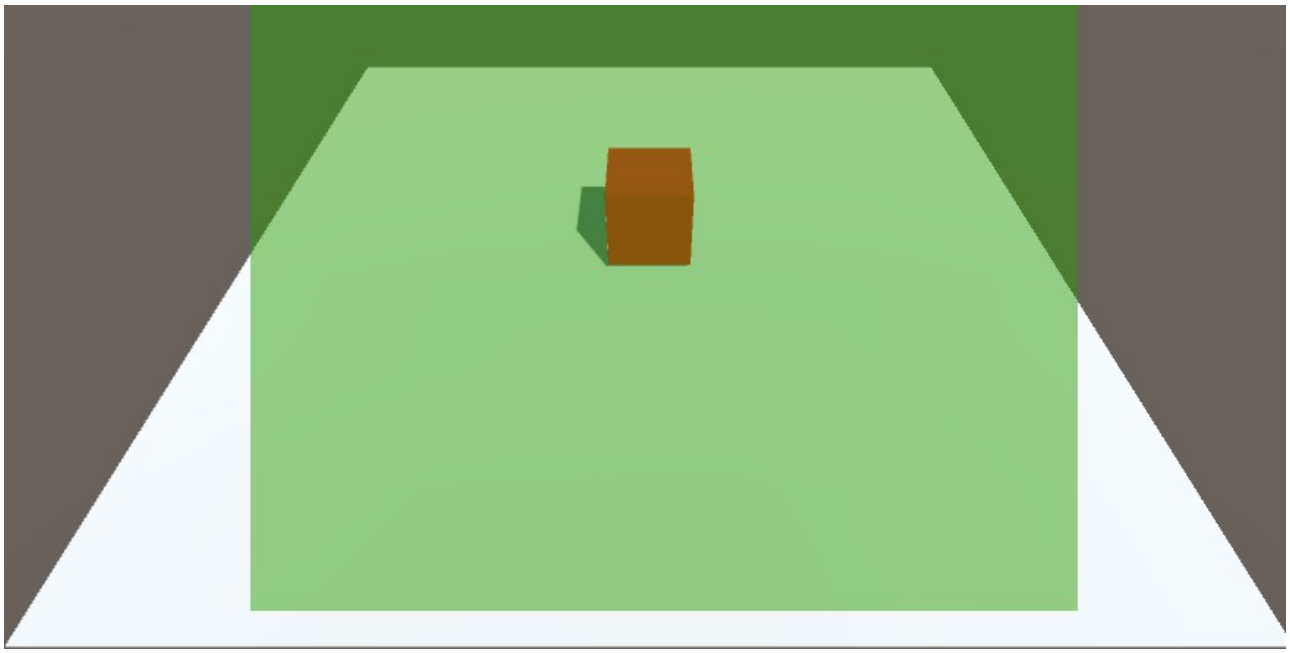
4. Click the “Init Systems” button. This will create a “Falcon Systems” object in your scene with all the required components.
5. Click the “Init 3D Camera” button. This will add the GL box selector and Physics Raycaster scripts to your camera. The box selector is the script in charge of rendering the box when you drag the pointer on the screen.
6. Add a Plane to your scene and set it to a position where you can view it with your camera. With the plane selected, click the “Set 3D Selection Area” button.



7. At this point you can hit play and click anywhere on the plain while dragging your pointer.

8. Now lets add some selectables. Add a cube to your scene and set its position to be above the plane. With your cube selected, click the “Set 3D Selectable Example” button on the 'Falcon Systems' window.

9. Hit 'Play' again and notice how your cube changes color when it gets selected and deselected!

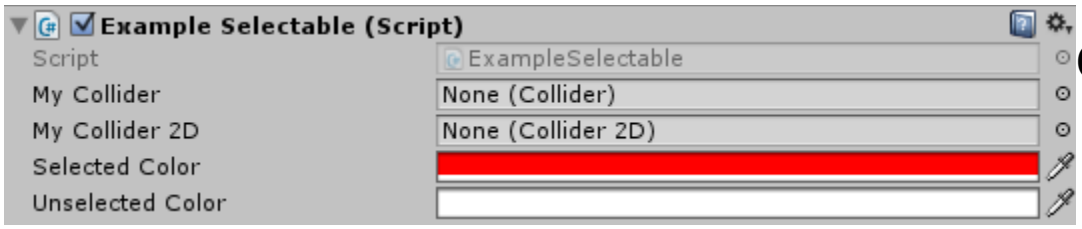


10. You may have noticed that clicking the cube does nothing, this is because it is not listening for click events. Lets make it listen!
11. With your cube selected, click the “Set 3D Click Listener” button. Hit 'Play' and notice how clicking the cube selects it.

Things to experiment with

- *Change the color of the selector box*: find the material called 'selector material.mat', located at 'Assets/Falcon Systems/Art/selector material.mat'. Change the 'tint' property of the material to any color. Hit 'Play' to see the selector box change to that color.
- *Make the selection box unfilled*: click on your main camera. Go to the 'GL Box Selector' script attached to your main camera. Uncheck the 'Filled' property.

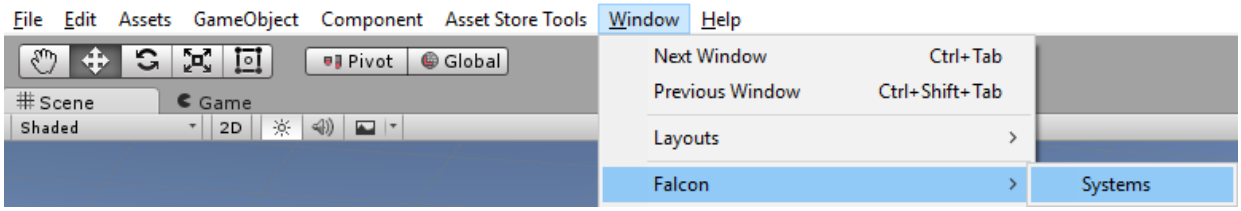
- *Add more selectables to your scene.*
- *Change the color of the selected object*



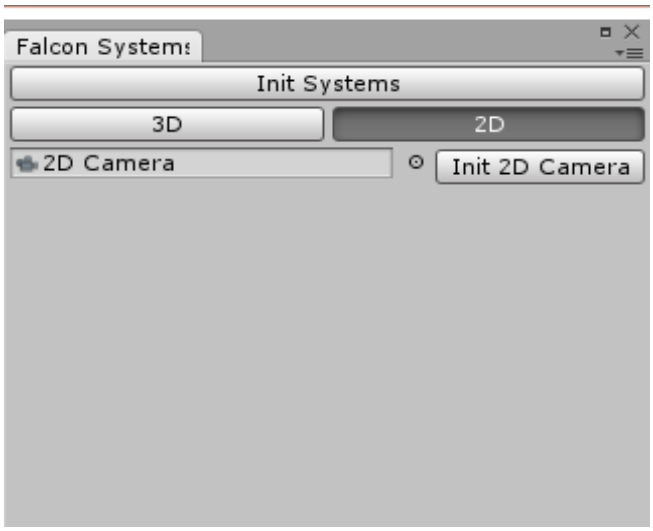
Quick Start

Step-by-step: 2D

1. Open a new scene
2. Open the Falcon Systems helper window.
   - Window → Falcon → Systems

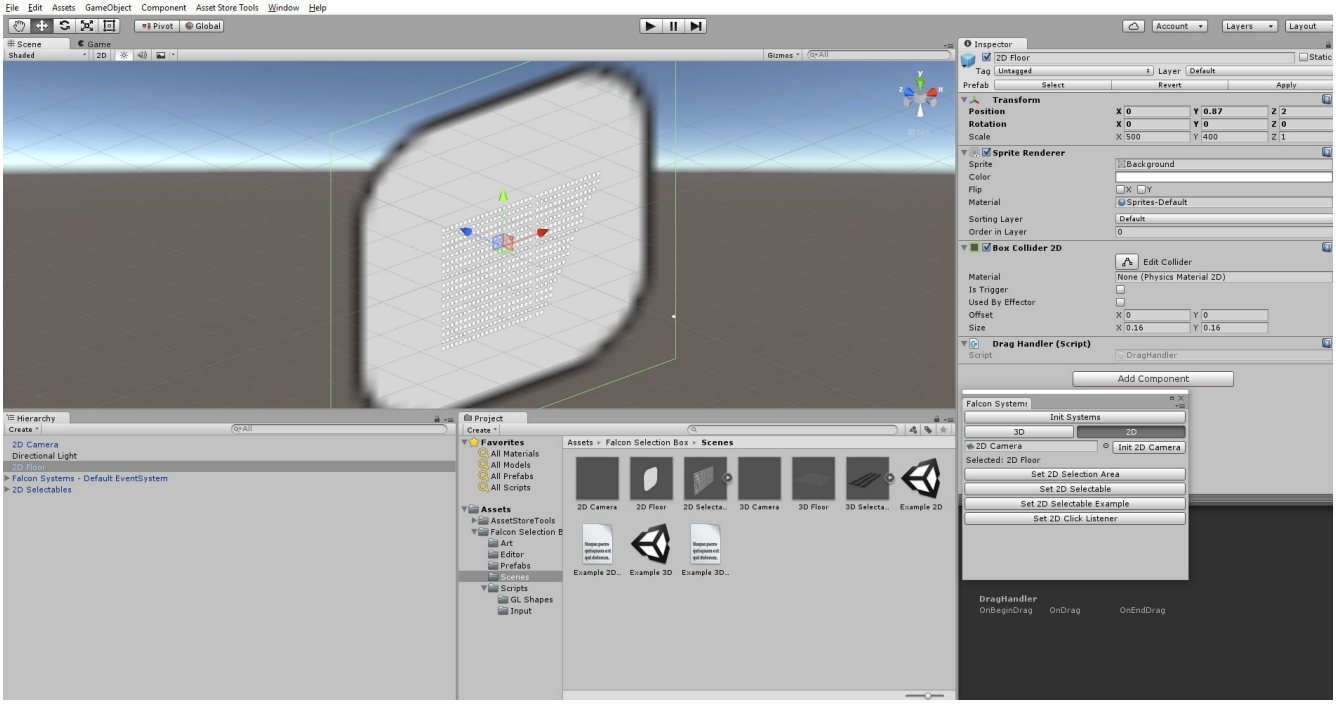


3. Make sure the “2D” button is selected.



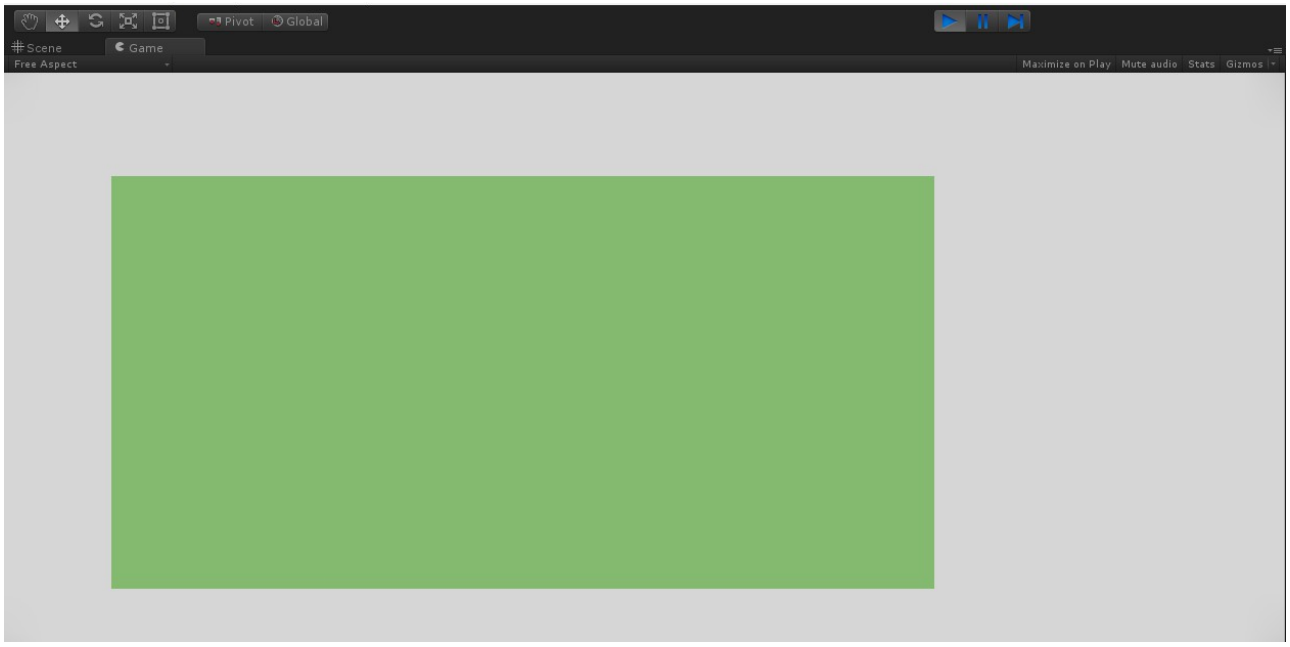
4. Click the “Init Systems” button. This will create a “Falcon Systems” object in your scene.
5. Click the “Init 2D Camera” button. This will add the GL box selector and Physics 2D Raycaster scripts to your camera. The box selector is the script in charge of rendering the box when you drag the pointer on the screen.

6. Add a Sprite to your scene, set it to a position where you can see it with your camera. This will be your floor. With the Sprite selected, click the “Set 2D Selection Area” button.

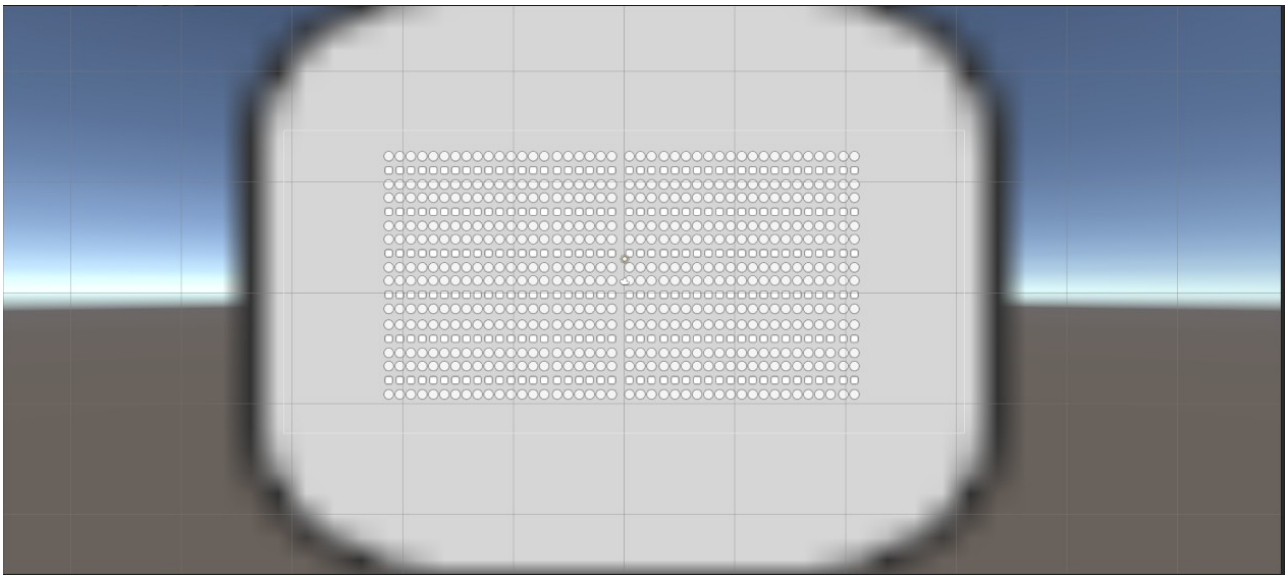


Note: The screenshot is taken from the Example 2D scene.

7. At this point you can hit play and click anywhere on the Sprite while dragging your pointer.



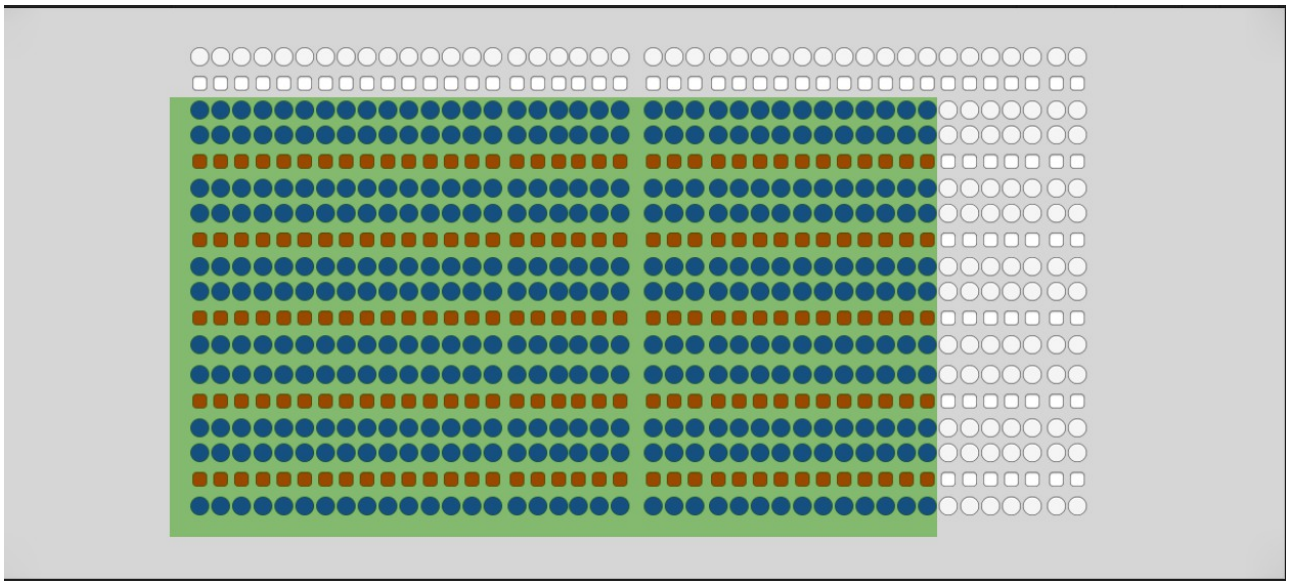
8. Now lets add some selectables. Add a Sprite to your scene and set its position to look like it's above the floor sprite. We will call this sprite: 'Dragon Unit'.



Note: in this screenshot I added several sprites. The screenshot is taken from the Example 2D scene.

9. With your sprite 'Dragon Unit' selected, click the “Set 2D Selectable Example” button on the 'Falcon Systems' window.

10. Hit 'Play' again and notice how your sprite changes color when it gets selected and deselected!



Note: in this screenshot I added several sprites. The screenshot is taken from the Example 2D scene.

11. You may have noticed that clicking the 'Dragon Unit' sprite does nothing, this is because it is not listening for click events. Lets make it listen!
12. With your 'Dragon Unit' sprite selected, click the “Set 2D Click Listener” button. Hit 'Play' and notice how clicking the sprite selects it.

Things to experiment with

- *Change the color of the selector box*: find the material called 'selector material.mat', located at 'Assets/Falcon Selection Box/Art/selector material.mat'. Change the 'tint' property of the material to any color. Hit 'Play' to see the selector box change to that color.
- *Make the selection box unfilled*: click on your main camera. Go to the 'GL Box Selector' script attached to your main camera. Uncheck the 'Filled' property.

- *Add more selectables to your scene.*

How to access selected objects

There are 2 basic ways to access selected objects.

1. You can access the list directly

From your script call

```
Falcon.AInputManager.instance.selector.selectedObjects
Returns: a 'Selectable' array.
```

Example

```
foreach (Falcon.Selectable selectable in
Falcon.AInputManager.instance.selector.selectedObjects)
{
     Debug.Log("Name: " + selectable.gameObject.name);
}
```

2. You have access to several events. The most common one an event that gets called right after the selection has finished.

From your script call

Example

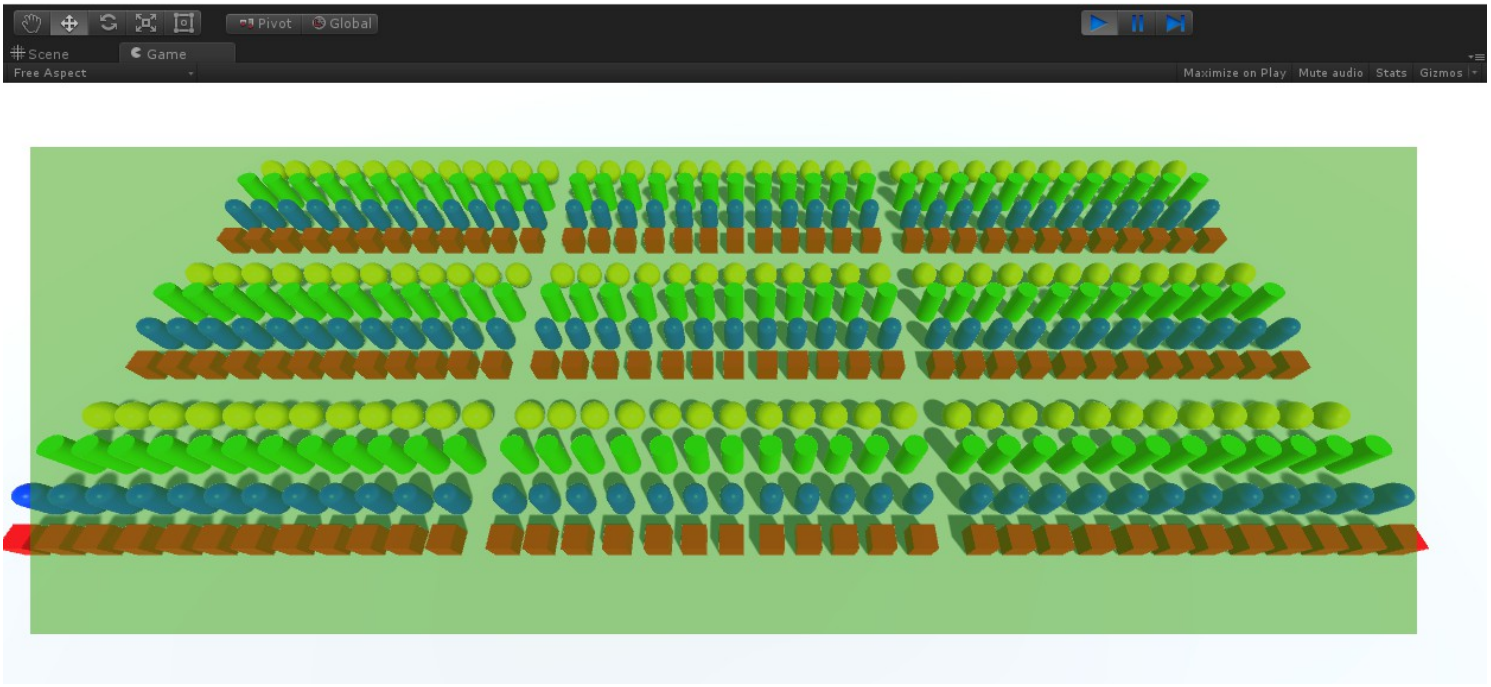
```
void Start()
{
    Falcon.AInputManager.instance.selector.onFinishSelecting += MyFinishSelectingFunction;
}


void OnDisable()
{
    Falcon.AInputManager.instance.selector.onFinishSelecting -= MyFinishSelectingFunction;
}


void MyFinishSelectingFunction(Falcon.Selectable[] selectableArray)
{
    foreach(Falcon.Selectable selectable in selectableArray)
    {
        Debug.Log(“Selected: “ + selectable.gameObject.name);
    }
```

```
}
```

3D Example Scene



Located under the 'Falcon Selection Box → Scenes” folder.

Demonstrates how to setup all the systems in a 3D environment.

Scripts used by the system:

- GLBoxSelector
- Physics Raycaster
- DefaultInputManager
- EventSystem, StandaloneInputModule, TouchInputModule
- SelectionDragHandler
- ClickHandler
- ExampleSelectable

2D Example Scene

Located under the 'Falcon Selection Box → Scenes” folder.

Demonstrates how to setup all the systems in a 2D environment.

Scripts used by the system:

- GLBoxSelector
- Physics 2D Raycaster
- GameManager
- DefaultInputManager
- EventSystem, StandaloneInputModule, TouchInputModule
- SelectionDragHandler
- ClickHandler
- ExampleSelectable

Single Selection Example Scene

Located under the 'Falcon Selection Box → Scenes” folder.

Demonstrates how to setup all the systems to work with single selection and no multi-select.

Only difference from the 2D or 3D examples is the “DragHandler” script is removed.

Scripts used by the system:

- GLBoxSelector
- Physics Raycaster
- DefaultInputManager
- EventSystem, StandaloneInputModule, TouchInputModule
- ClickHandler
- ExampleSelectable

Groups Example 3D

Located under the 'Falcon Selection Box → Scenes” folder.

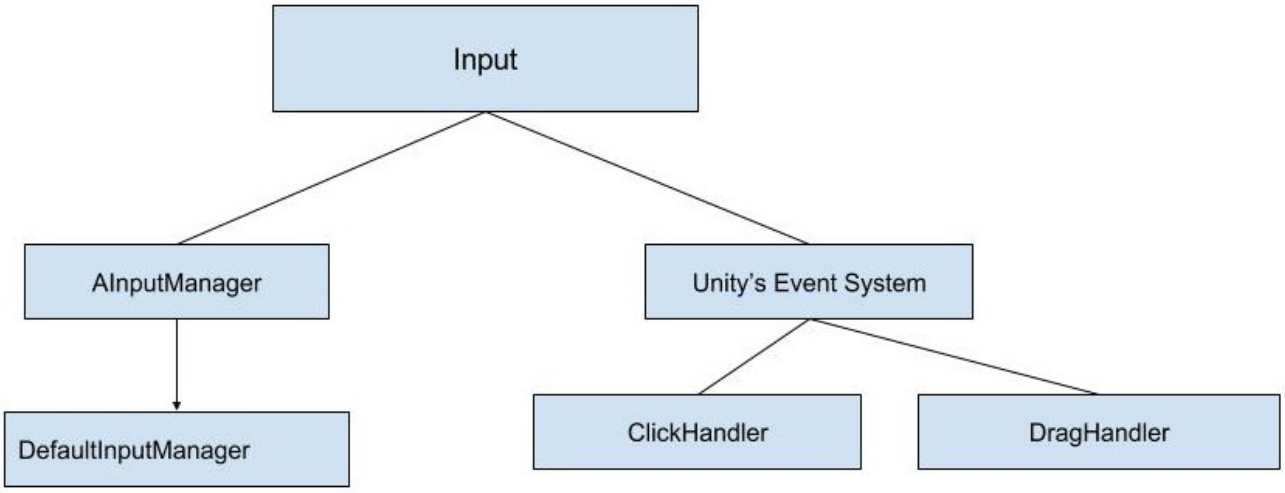
Demonstrates how to use the groups API. It is the same as the 3D example with the addition of 2 more components: SelectableGroupsManager and KeyboardGroupsExample.

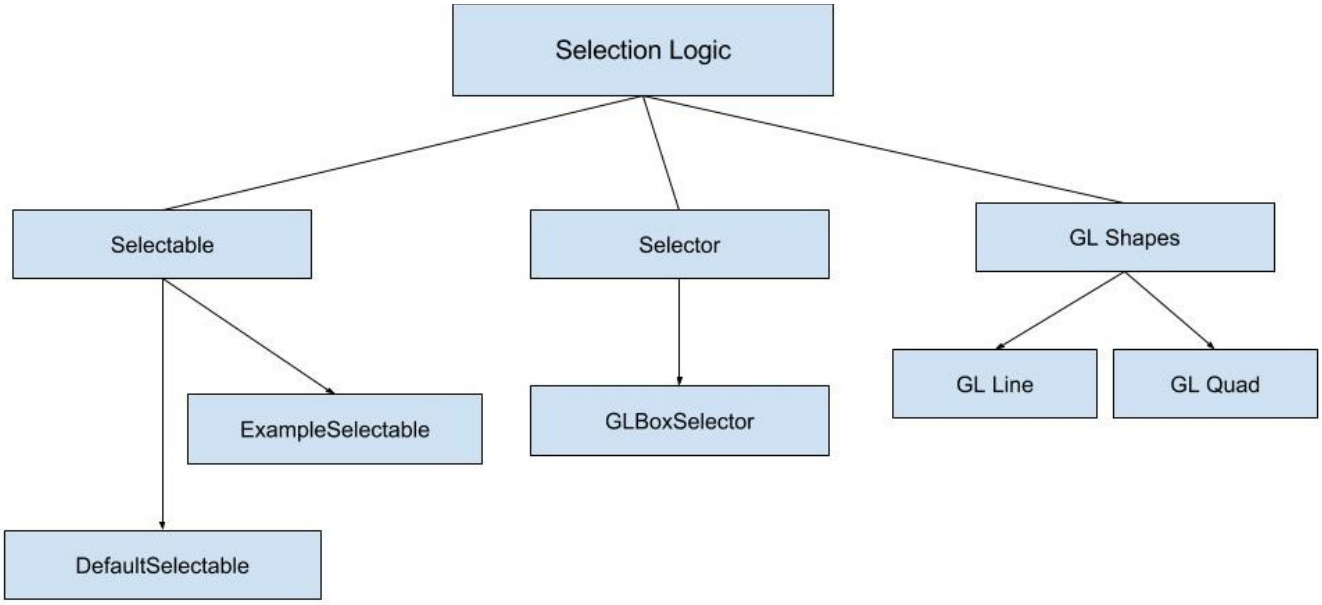
The groups manager is independent from 3D or 2D.

Scripts used by the system:

- GLBoxSelector
- Physics Raycaster
- DefaultInputManager
- EventSystem, StandaloneInputModule, TouchInputModule
- SelectionDragHandler
- ClickHandler
- ExampleSelectable
- SelectableGroupsManager
- KeyboardGroupsExample

Architecture





Input

All input events are handled by Unity's event system. The selection system receives input events by implementing Unity's event system classes.

Unity events caught by the selection system:

- Drag event, caught by SelectionDragHandler; implements IBeginDragHandler, IDragHandler and IEndDragHandler.
- Click event, caught by ClickHandler; implements IPointerClickHandler

The system interprets these events and notifies the selection system.

All input events are sent to the implementation of AInputManager. Any class that wants to listen to a generic input event from the selection system should register it's function to the delegates provided by the implementation of AInputManager. Note that there are more detailed input events defined by other classes. For example, the Selectable class notifies whenever it gets clicked. Use the implementation of AInputManager's events when you need the raw event.

Events Provided

- onPointerBeginMove: Event that gets called when a drag has started.
  - Signature: `onPointerBeginMove(Vector3 startWorldCoordinates, Vector3 currentWorldCoordinates, Vector2 startScreenCoordinates, Vector2 currentScreenCoordinates)`
- onPointerMoved: Event that gets called when the pointer is being dragged.
  - Signature: `onPointerMoved(Vector3 startWorldCoordinates, Vector3 currentWorldCoordinates, Vector2 startScreenCoordinates, Vector2 currentScreenCoordinates)`
- onPointerEndMove: Event that gets called when the pointer has stopped dragging
  - Signature: `onPointerEndMove(Vector3 startWorldCoordinates, Vector3 currentWorldCoordinates, Vector2 startScreenCoordinates, Vector2 currentScreenCoordinates)`
- onPointerClick: Event that gets called when a pointer click event has happened.
  - Signature: `onPointerClick(GameObject clickedObject, PointerEventData eventData)`

Selector

The selector is in charge of remembering all the selected objects and drawing the shape of the selected area.

All selectables should register and unregister themselves to this class by using the RegisterSelectable and UnregisterSelectable methods. It also has delegates to notify when any object got selected or unselected.

To access the selected objects from a script:

```
Falcon.AInputManager.instance.selector.selectedObjects
```

1. Access the Input Manager instance.
2. Access the selector reference.
3. Access the selectedObjects property, which returns an array of type Selectable.

The base class is Selector. Selector is a system class that handles all the selectable objects. This class also handles drawing a representation of the enclosed bounds of the selection. By the default an implementation that uses OpenGL is provided – GLSelector.

Implement this class if you need to use a different type of selector.

Note: Keep in mind that the input manager only handles 1 selector reference in the scene. Remember to delete all other selectors from the scene if you are implementing your own.

Events Provided

- onObjectSelected: called when any object with a Selectable component calls the .Select() function.
    - Signature: `onObjectSelected(Falcon.Selectable selectedObject)`
- onObjectUnselected: called when any object with a Selectable component calls the .Unselect() method.
    - Signature: `onObjectUnselected(Falcon.Selectable selectedObject)`

Selectable

A selectable is the component that enables the system to recognize when a game object gets selected. The base class is called Selectable.

The system provides 2 implementations of this class: DefaultSelectable and ExampleSelectable.

DefaultSelectable notifies it's listeners when it gets selected or unselected. This class should be enough to cover most cases.

ExampleSelectable notifies it's listeners when it gets selected or unselected. It also changes color when a selection or unselection happen.

The functions used to select a unselect a game object are called: Select() and Unselect().

A property of type bool is provided to check if it's selected or not: 'selected'.

Events Provided

- onSelect: called when the .Select() method gets called.
  - Signature: `onSelect(Falcon.Selectable gameObject)`
- onUnselect: called when the .Unselect() method gets called.
  - Signature: `onUnselect(Falcon.Selectable gameObject)`

Groups

The groups API is provided through the SelectableGroupsManager class. It uses a key-value pair to store groups of selections.

To use it just include it in the scene.

The groups manager can be accessed through the Selector component or by directly getting the reference. The Selector class provides convenient methods to manipulate the current selection with groups.

SelectableGroupsManager has methods to easily store and read selectables from a map. Every method will require at least the name of the group to be passed as parameter.

GroupsEditorWindow is an editor window that can be accessed through the Window->Falcon->Groups menu. This window will provide information about the currently stored groups.

Events Provided

- onSetGroup: called when the .SetGroup() method is called.
  - Signature: `onSetGroup(string name, List<Selectable> group)`
- onGroupChange: called everytime a group changes through one of the methods provided.
  - Signature: `onGroupChange(string name, List<Selectable> group)`

How To Listen For Events

There are several classes that provide events for you to listen

You can find out if the class provides events by identifying the following convention:

Any event will have the prefix “on” (note the lowercase) followed by an action. For example, 'onPointerBeginMove'.

For this example I will use the 'onPointerBeginMove' event, defined in the AInputManager class.

1. Identify the method signature. By looking at the AInputManager script I can see the method receives 4 parameters, returns void and is public: (`Vector3 startWorldCoordinates, Vector3 currentWorldCoordinates, Vector2 startScreenCoordinates, Vector2 currentScreenCoordinates`)
2. Define a function that receives 4 parameters, returns void and is public:
   - ```
     public void MyCustomDragFunction(Vector3 startWorldCoordinates, Vector3 currentWorldCoordinates, Vector2 startScreenCoordinates, Vector2 currentScreenCoordinates)
     {
         Debug.Log(“Hello World”);
     }
     ```
3. Register the method to the event provider. Remember to unregister from the event provider whenever your component gets disabled or destroyed. In this case our event provider is the AInputManager class.
4. ```
   Void Start()
   {
       //Register my method to listen for the onPointerBeginMove event
       Falcon.AInputManager.instance.onPointerBeginMove += MyCustomDragFunction;
   }
   void OnDisable()
   {
       //Unregister my method from the onPointerBeginMove event
       Falcon.AinputManager.instance.onPointerBeginMove -= MyCustomDragFunction;
   }
   ```
5. Your method will get called whenever the onPointerBeginMove event gets called.

Coding Standards

- Every class is under the namespace 'Falcon' to avoid clashing with other classes.
- Every class is structured in the following way:
  - Variables are declared first.
  - delegates are declared after variables.
  - Abstract methods are declared after variables and delegates.
  - MonoBehaviour methods are declared after abstract methods.
  - Custom methods are defined after the MonoBehaviour methods.
  - Stub methods are defined after the custom methods.
  - Properties are defined at the end of the class.
- All events have the prefix "on", i.e. onObjectSelected.
- Stub methods have the prefix "Stub".
- Stub methods are provided to avoid empty delegates.