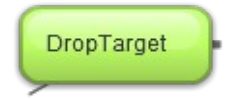
A Drag And Drop Architecture

Touch interfaces are all over the place, and drag and drop is one of the most common interface mechanics used in applications and games.

Main components

In a drag and drop system we need to be able to:

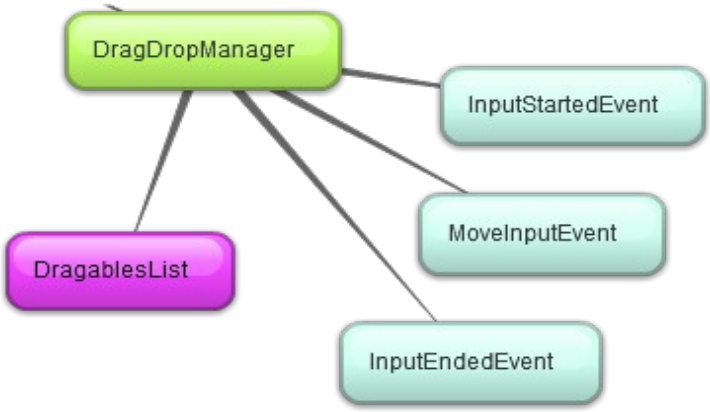


1. Make an object Dragable, So that the user can tap on the object and start moving it around.
2. Specify a DropTarget object to receive any dragables droppen on its surface.
3. Know and access all the dragable objects in the system, and when to BeginDragging, EndDragging, UpdateDragging a Dragable object.

The DragDropManager

The DragDropManager needs two things:

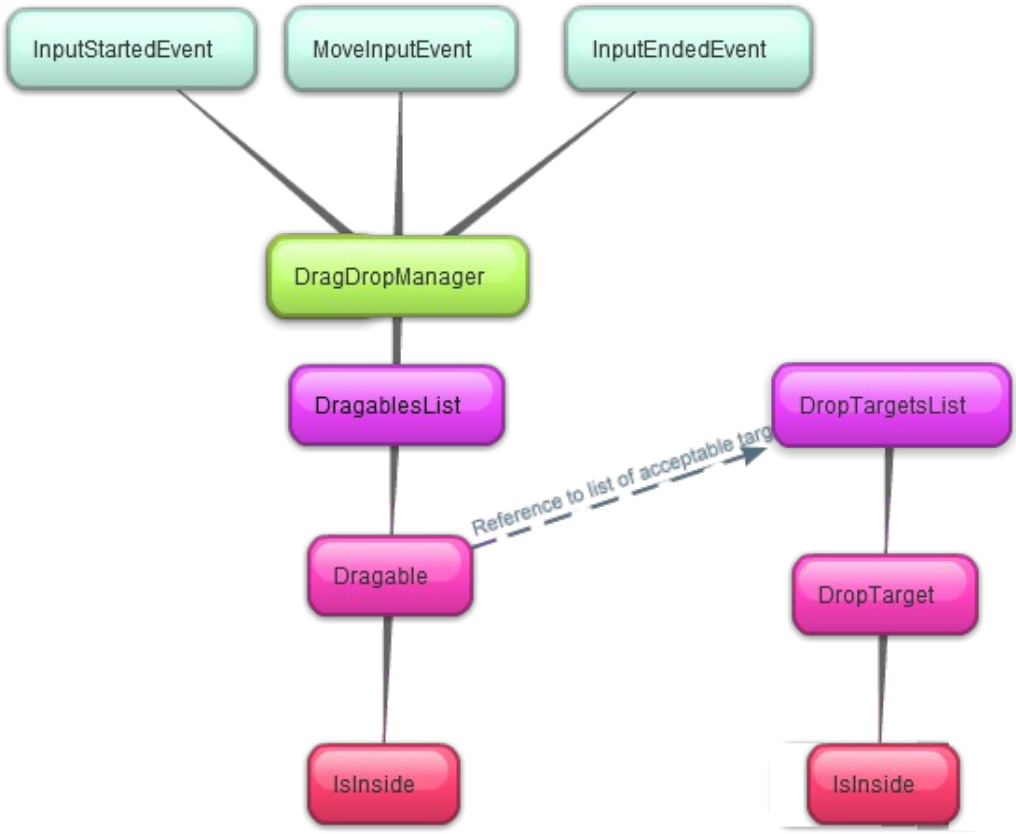
1. A list of all the Dragable objects, so it can check the input against them and decide to Begin/Update/End Dragging for them or not.
2. To be informed of the users tap/swipe actions. This is done by hooking to the 3 event handlers of the DragDropManager.



So at initialization,
InputStartedEvent is subscribed to the MouseDown event of your system or similar.
MoveInputEvent is subscribed to the MouseMove event.
InputEndedEvent is subscribed to the MouseUp event.
Also any Dragable objects needs to be added to the DragablesList of the manager.

A Complete Normalized View Of The Architecture

Below is a complete diagram of all the components of the Drag and Drop system, starting from the moment the input event is triggered in the DragDropManager, until the final stage when the Dragable is droped in a valid DropTarget.



Two more things need to be explained

- **IsInside** is an interface method, that needs to be implemented by your Dragable or DropTarget object. It should return true in case a tap position lies inside your object.

  It has been implemented as an interface because the DragDrop system does not know about the geometry of the object. So it is left to the developer to implement sphere or box collision or some more complex geometry check.

- DropTargetsList is a list of the valid objects the user can drop a Dragable onto. It is left as an external reference to the Dragable and I will explain why.

  If it was a member of the DragDropManager, it would mean that all Dragables can possibly be dropped on all DropTargets.
  If it was a member of the Dragable class it would mean that we would have to assign all valid drop targets to each new Dragable all the time.
  By leaving it as a reference, the programmer can implement his own grouping on valid DropTargets, by creating a DropTargetlist for each group.