Technical Design Document

SCRMG

2017.04.02

Contents

[Script Relations 1](#_Toc478904840)

[Class Descriptions 2](#_Toc478904841)

[Event Manager 2](#_Toc478904842)

[Subscribing and unsubscribing 2](#_Toc478904843)

[Broadcasting 2](#_Toc478904844)

[Creating new events 3](#_Toc478904845)

[Delegates 3](#_Toc478904846)

[Application Manager 3](#_Toc478904847)

[Input Manager 3](#_Toc478904848)

[Game Manager 3](#_Toc478904849)

[UI Manager 4](#_Toc478904850)

[Global Variable Library 4](#_Toc478904851)

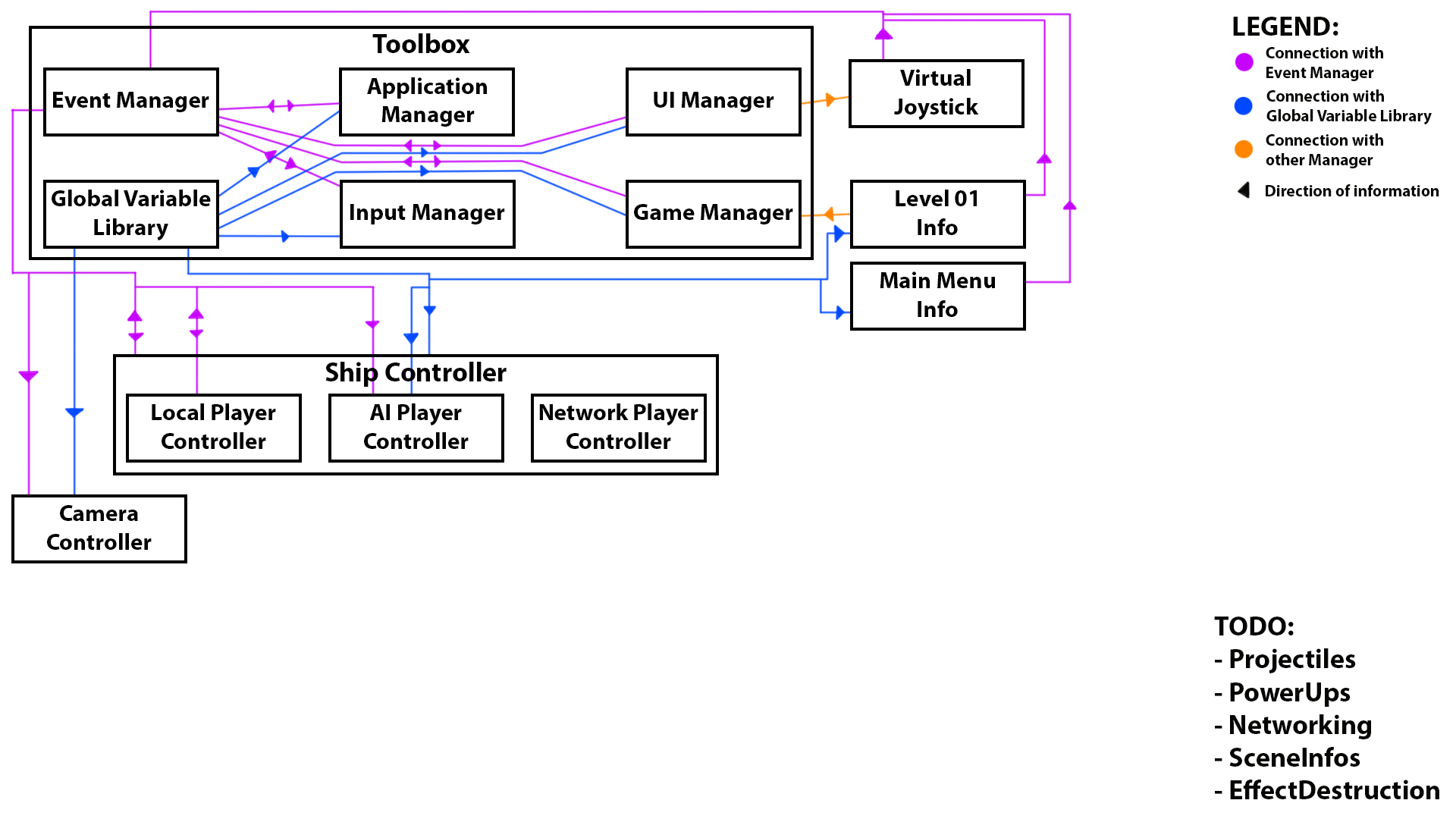
[Ship Controller 4](#_Toc478904852)

[Local Player Controller 4](#_Toc478904853)

[AI Player Controller 4](#_Toc478904854)

[Network Player Controller 4](#_Toc478904855)

# Script Relations



# Class Descriptions

## Event Manager

Main method of communication between scripts. Reference can be found through toolbox. Is a singleton and one always exists. Works on subscription / broadcast basis. Scripts can subscribe their methods through delegates. When an event is broadcasted, all the methods subscribed to the event are called.

### Subscribing and unsubscribing

Any script can subscribe a method to an event in event manager. When an event is broadcasted, all methods that are subscribed to the event in question are called. Subscribe a method to an event with “+=” modifier and unsubscribe with “-=” modifier. The subscribing method must have identical parameter types as the respective event. The subscribing method should be named identical to the event for clarity’s sake.

**Example 01:** Subscribing to MouseButtonLeftDown-event

*em.OnMouseButtonLeftDown += OnMouseButtonLeftDown;*

Where “*em*” is a reference to the EventManager, the former “*OnMouseButtonDown*” is the name of the event in EventManager, and the latter “*OnMouseButtonDown*” is the name of the method being subscribed.

In the case of unsubscribing the “+” would be changed to “-“.

Subscribing is often done in the OnEnable method, to ensure the script is aware of events from the moment it is enabled. However, this is not mandatory. For example, when of pausing the game LocalPlayerController could unsubscribe from all input events and then re-subscribe to them when the game is unpaused, to prevent player inputs to the ship during game pause.

**NB:** Whenever a script is subscribing to an event, ensure that it is also unsubscribed to the respective event in its OnDisable method. This is done to prevent EventManager from trying to call methods in scripts that no longer exist.

### Broadcasting

Any script can broadcast an event. Broadcasting events is the primary way of communicating between scripts in cases when we have no reference to the script we want to communicate with. To broadcast an event, the script first needs a reference to the EventManager (generally abbreviated as “em”). EventManager can always be found as a component of Toolbox. Once we have the reference, simply call the event’s Broadcast method from the EventManager.

**Example 02:** Broadcasting mouse movement

*em.BroadcastMousePosition(keyboardAndMouseIndex, mousePosition);*

Where “*em*” is a reference to the EventManager, “*BroadcastMousePosition*” is the name of the broadcast method of the desired event, ”*keyboardAndMouseIndex*” is the controllerIndex of the keyboard and mouse pair as integer, and “*mousePosition*” is the position of the cursor on screen as Vector2.

### Creating new events

When creating a new event in EventManager, first declare the event. If necessary, create a new delegate type for the event. Then implement a broadcast method directly below the new event. Remember to add a check to prevent broadcasting in the case of zero subscribers. The events always start with “On” and the broadcast methods with “Broadcast”.

**Example 03:** Movement input

*public event IntVector2Void OnMovementInput;*

*public void BroadcastMovementInput(int controllerIndex, Vector2 movementInputVector)*

*{*

*if (OnMovementInput != null)*

*{*

*OnMovementInput(controllerIndex, movementInputVector);*

*}*

*}*

Where “*OnMovementInput*” is the event, “*BroadcastMovementInput*” is the broadcast-method of the event, and the parameters of the broadcast-method (determined by the delegate type of the event) are the parameters given when calling the subscribed methods.

### Delegates

Delegates are used when implementing new events in EventManager. A delegate is a type representing references to methods with particular parameter list and return type. EventManager delegates are named as follows: “Parameter1Parameter2…ParameterNReturnType”.

**Example 04**: IntIntVoid delegate

*public delegate void IntIntVoid(int integer1, int integer2);*

Where “*IntIntVoid*” is the name of the delegate, constructed from the parameter types.

## Application Manager

## Input Manager

## Game Manager

## UI Manager

## Global Variable Library

## Ship Controller

### Local Player Controller

### AI Player Controller

### Network Player Controller