Fibrum SDK v2.0.0

Documentation

Index

Quick start

1. Install the Fibrum SDK (link to the Asset Store)
2. Find prefab VR_Camera located in FibrumSDK/Prefabs and drag it into the scene.
3. Run the scene in the editor. While the scene is running in the editor - you can rotate the camera with the mouse (to simulate the rotation of the head).
4. Compile your app for Android, IOS, Windows Phone, or other platform using the gyro of the device.

Demo-scene (1 -DemoTracking)

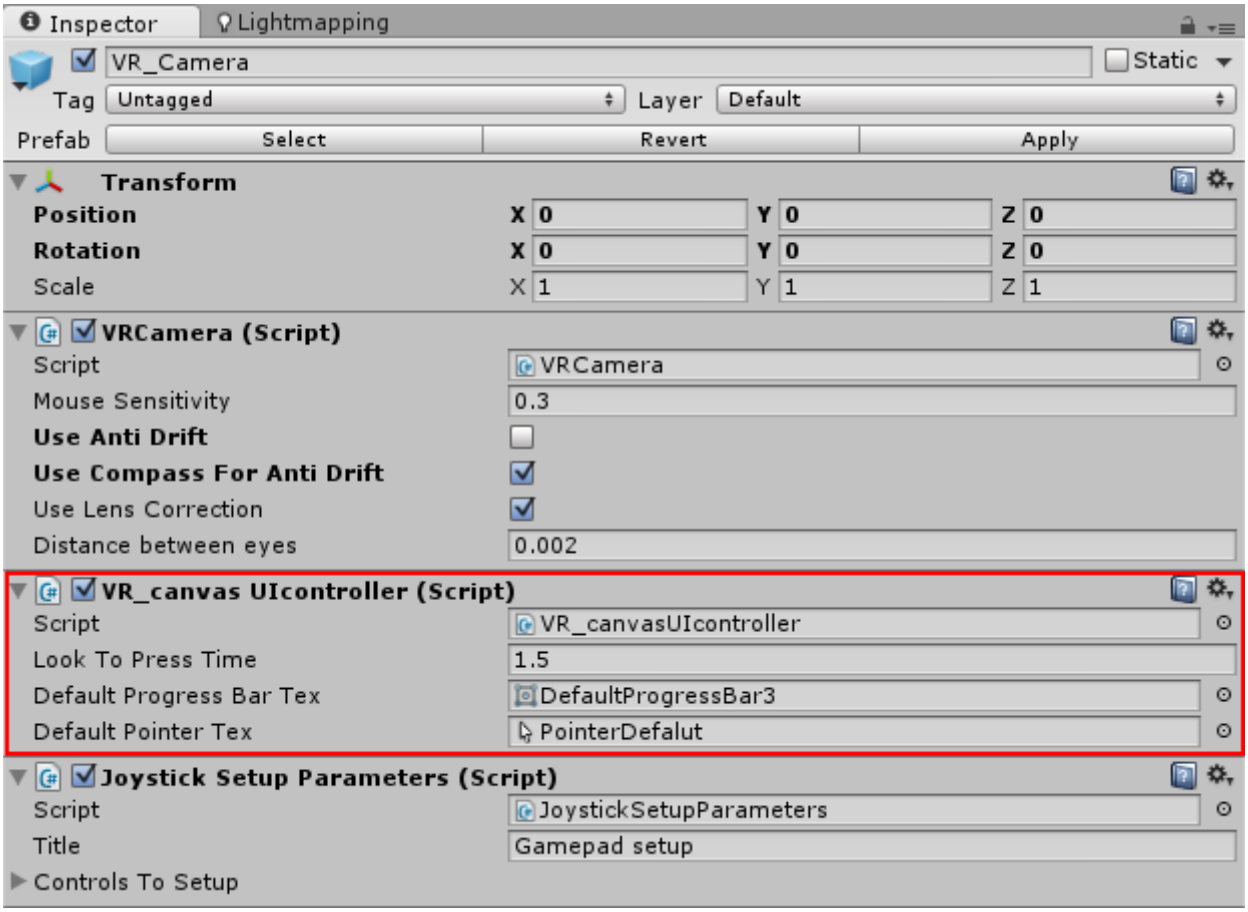
1) Install Fibrum SDK (from the Asset Store)
2) Open the scene FibrumSDK / Scenes /1- DemoTracking.scene
3) Run the scene in the Unity editor.
4) Compile the app for a Smartphone.

Demo-scene of using Canvas-GUI Unity3d 4.6-5.x (2 - NewGUI_demo)

1) Install Fibrum SDK (from the Asset Store)
2) Open the scene FibrumSDK / Scenes / 2-NewGUI_demo.scene
3) Run the scene in the Unity editor.
4) Compile the app for a Smartphone.

Understanding the Demo-scene.

There is a script VR_canvasUIcontroller in the prefabs of virtual camera VR_Camera, and it controls the interaction between virtual camera and Canvas-GUI. If you do not plan to work with the canvas-GUI - you can disable this script.



If the script VR_canvasUIcontroller enabled on the virtual camera, the player will able to interact with buttons, toggles, scrolls, etc., created using standard Unity3d Canvas-UI with his head-motion.

These UI elements can be created as usual, to work with the virtual camera buttons do not need anything extra to customize, but you can.

To set up the script, refer to the documentation VRCamera

In the FibrumSDK/Prefabs you can find prefab CanvasForVR with VR cursor and Progress bar, as an example.

You can create any number of canvases in your scene.

You can instantiate / activate / deactivate / scripts to create any elements of UI - a virtual camera to interact with them.

Using Canvas-UI as a game-menu

If the game uses a canvas-menu, it can be adapted for virtual reality using a script VR_canvas (FibrumSDK / Fibrum / VR_GUI / VR_canvas.cs)

Procedure:

1) Create game menu. It must be of a standard Unity-canvas, with any number of nested interfaces.
2) Add to the object that contains the components Canvas, script VR_canvas.cs (FibrumSDK / Fibrum / VR_GUI / VR_canvas.cs)
3) Select the type of display Bind Type:
    a. Fixed Orientation - menu will follow the player, but will not change the rotation in space (the most infrequent use)
    b. Fixed In Center - the menu is always displayed at exactly the center of the screen (suitable for dialog messages)
    c. Always In View Field - menu will be led by a look to interact with interface elements. When you try to divert look beyond the menu, the menu will rotated, so it will always fall into the field of view of at least half (optimized to display the game menu)
4) Select the zoom scale. If scale = 1 horizontal menu will completely fill Player's field of view.
5) To display the object canvas-screen menu, simply add it to the stage, or activate. If necessary hide the menu - object can be removed, or deactivate it.

Demo-scene of using gamepad (3-joystickSetup_demo)

1) Istall **Fibrum SDK** (from the Asset Store)
2) Open scene **FibrumSDK/Scenes/3 - joystickSetup_demo.scene**
3) Run scene in Unity-editor.

Undestanding of gamepad-demo

Prefab Joystick_Simple_FPS_character has CharacterController-component and 2 child – usual VR_Camera and GUI, which invokes method “SetupJoystick” in Joystick_Simple_FPS_character.

In this scene you can walk and shot with A-button.

To learn more see FibrumInput.

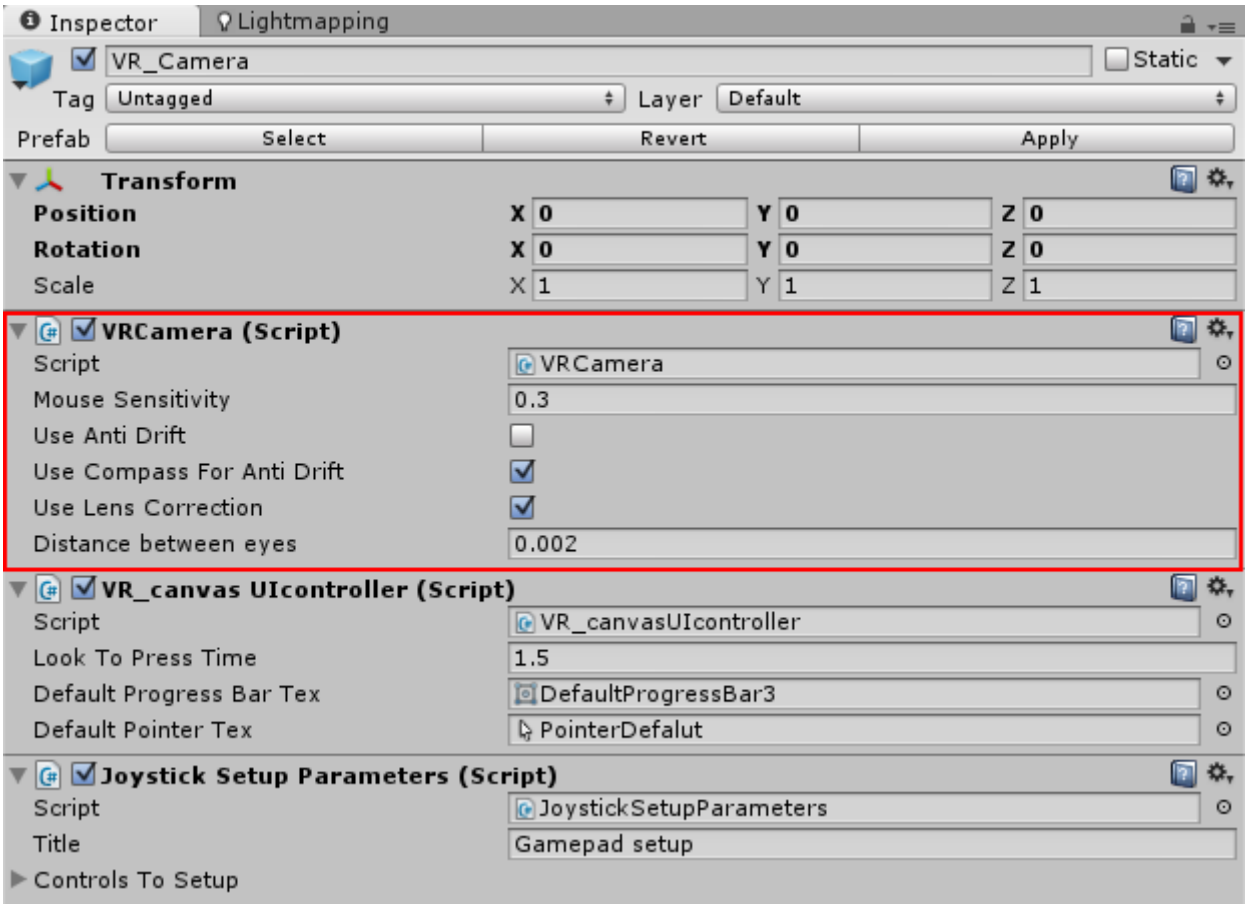
VR_Camera

VR_Camera prefab is a GameObject with a script VRCamera.cs. It has a child “VRCamera”, and the last repeating head-tracking in the scene. So during the head-tracking VR_Camera object can be moving in the scene with scripts or animation, and player could look around during this motion.

Properties of VRCamera

You can easily access VRCamera in scene through static variable FibrumController.vrCamera.

For example: FibrumController.vrCamera.vrCameraHeading – Transform-type variable, returning current rotation in space.



Mouse Sensitivity – mouse sensivity, when emulating head-tracking in Editor.

UseAntiDrift – turning on the anti-drift system, eliminating drift around vertical.

UseCompassForAntiDrift – (not necessary for most apps) if turned on, so rotation aroud vertical will be corrected by compass of the device. It garants that player rotation in real life will remain respect to space.

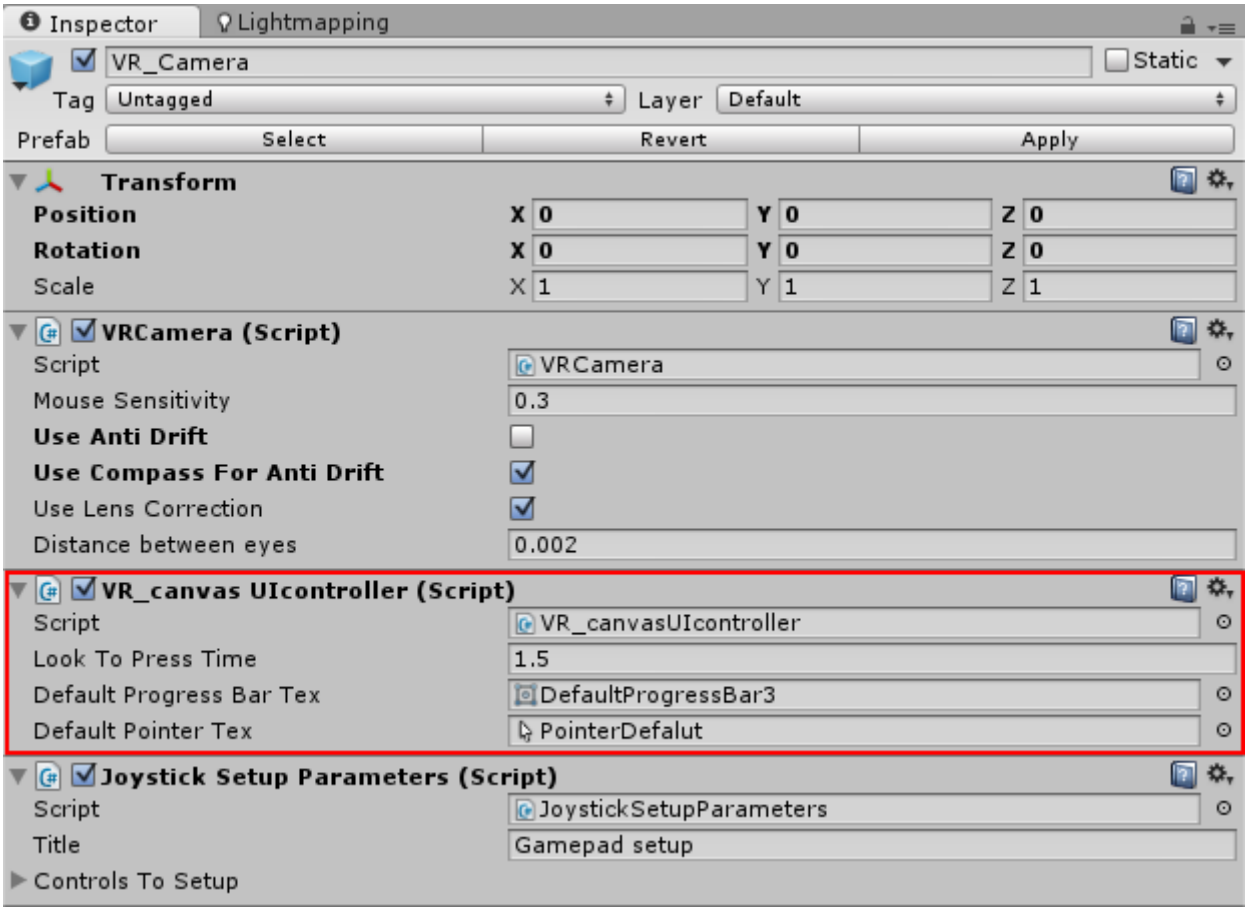
UseLensCorrection – on/off the shader of lens correction. May reduce the productivity slightly.

DistanseBetweenLens – distance between scene cameras, in Unity units. In usual must be equal to 6-7 sm, real distance between eyes.

Auxilary parameters of VRCamera

Transform VRCamera.vrCameraHeading –Transform-type variable, returning head-tracking state of the device.

Свойства контроллера Canvas-GUI VR_CanvasUIcontroller



LookToPressTime – time to element being pressed.

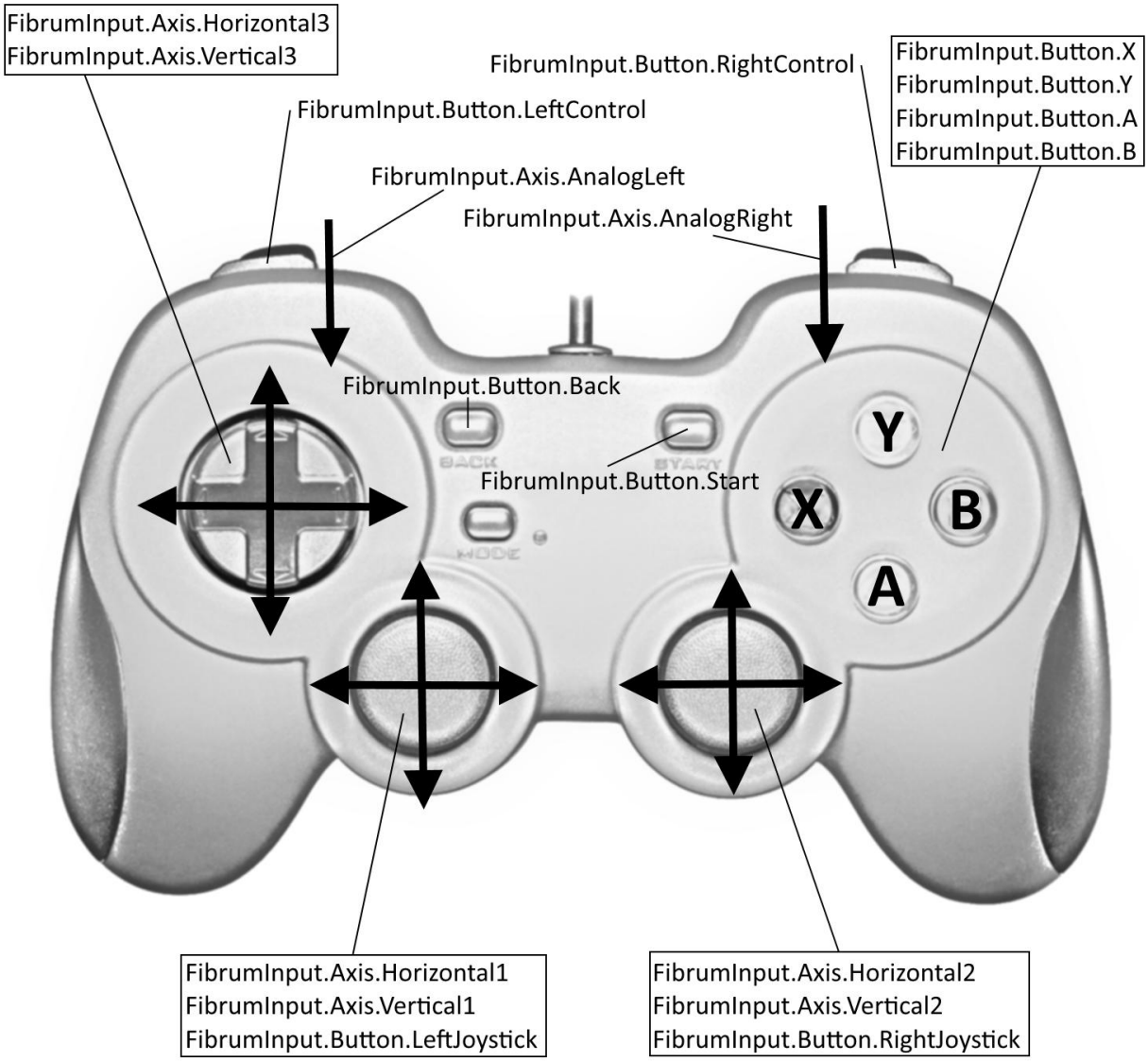
DefaultProgressBarTex – sprite of a progress bar by default. If you need to customize spectial progress bar for special canvas – you can add to this canvas prefab FibrumSDK/Prefabs/VRprogressBar.prefab to it, and change it's sprite. You can see, how it looks like on the 2-NewGUI_demo.scene.

DefaultPointerTex – texture as a default cursor. If it is need to customize cursor on the specific canvas – you can add FibrumSDK/Prefabs/VRcursor.prefab to it, and change it's texture manually.

FibrumInput

FibrumInput – it is a static class, designed to simplify the communication application with axes and buttons for gamepads and joysticks. Realized functions for obtain the status of axes and buttons for gamepads, reassign functions axes and buttons for gamepads "ingame", save and load settings of the gamepad PlayerPrefs. (if gamepad's buttons are reassigned – they will load properly for the other launches).

There are enumerated next axes and buttons in FibrumInput:



float FibrumInput.GetJoystickAxis(FibrumInput.Axis axis) - returns axis value (-1;1), example: FibrumInput.GetJoystickAxis(FibrumInput.Axis.Vertical1)

bool GetJoystickButton(Button buttonNum) – returns a button state, example: FibrumInput.GetJoystickButtonDown(FibrumInput.Button.A)

bool GetJoystickButtonDown(Button buttonNum) – returns true, if the button was pressed during this frame.

bool GetJoystickButtonUp(Button buttonNum) – returns true, if the button was released in this frame.

bool FibrumInput.InitializeJoystickSetup() – Initialization of gamepad setup. Must be followed with **FibrumInput.joystickSetupGO.SetControl** and **FibrumInput.joystickSetupGO.StartSetup**

FibrumInput.joystickSetupGO.SetControl(FibrumInput.Axis axis,string description,Texture tex) – points, what the axis must be setup, and also description and button, will shown to the player.

FibrumInput.joystickSetupGO.SetControl(FibrumInput.Axis axis,bool checkAxisDirection,string description,Texture tex) – the same functionality, but here you can adjust the direction of axis.

FibrumInput.joystickSetupGO.SetControl(FibrumInput.Button button,string description,Texture tex) – points, what button must be setup, description and texture for player.

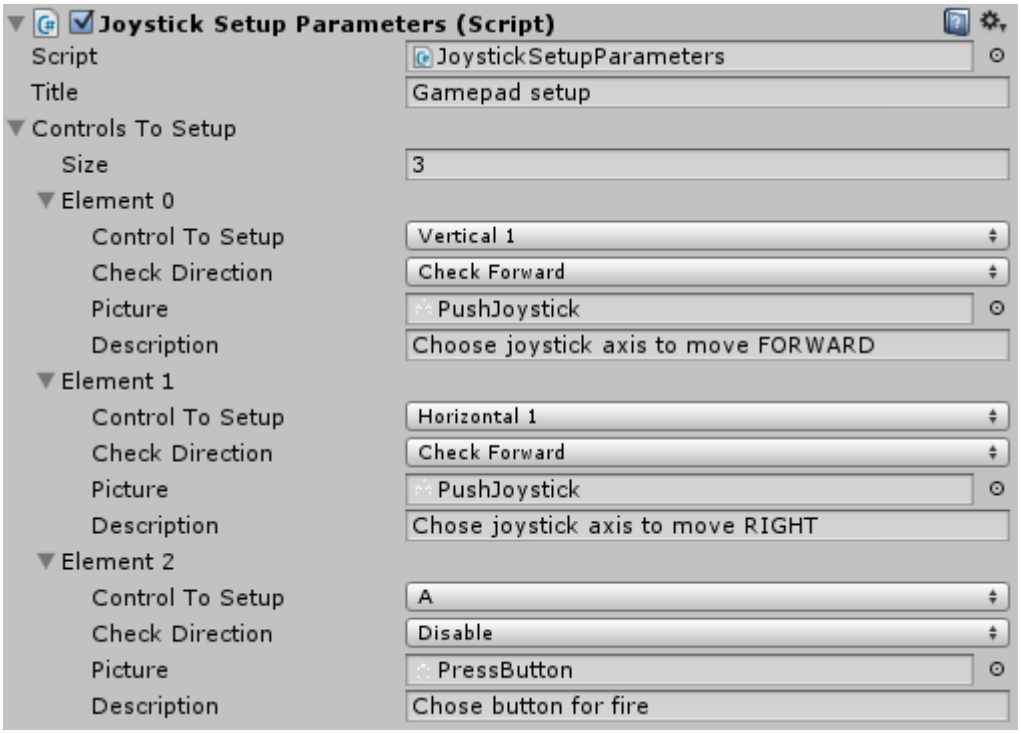
FibrumInput.joystickSetupGO.StartSetup() – starts the gamepad setup.

ClearControls() – reset buttons and axes to default.

Пример запуска настройки джойстика из скрипта

```
public void SetupJoystickScripted()
{
    if( FibrumInput.InitializeJoystickSetup() )
   {
      FibrumInput.joystickSetupGO.ClearControls();
      FibrumInput.joystickSetupGO.SetTitle("Setup joystick input");
      FibrumInput.joystickSetupGO.SetControl(FibrumInput.Axis.Vertical1,true,"FORWARD\nChoose joystick axis to move forward",null);
      FibrumInput.joystickSetupGO.SetControl(FibrumInput.Axis.Horizontal1,false,"RIGHT\nChose joystick axis to move right",null);
      FibrumInput.joystickSetupGO.SetControl(FibrumInput.Button.A,"Chose button for fire",null);
      FibrumInput.joystickSetupGO.StartSetup();
   }
}
```

Setup gamepad buttons in Editor



This setup is the same as from the script (see above). So if you assigned need controls through Editor, you shall just run script like this:

```
public void SetupJoystick()
{
   if( FibrumInput.InitializeJoystickSetup() )
   {
      FibrumInput.joystickSetupGO.StartSetup();
   }
}
```

History

2.0.0 – Much better interfacing with VRCamera prefab, cleaning up the project, visual enhances.

1.0.7 - Fixed setting the virtual camera, increased the stability of the system UI, the ability to disable the antidrift system.

1.0.6 - By default, the camera rotates the picture to fill the screen, you can change the type of display types. In the absence of a gyroscope on a device, the tracking will be emulated by accelerometer and compass. Increased stability antidrift system. Fixed bug with setting the joystick.

1.0.5 - Added support headtracking OS Windows Phone 8 / 8.1; Added ability of limited headtracking if a smartphone does not have a gyroscope (in this case, a warning will be displayed warning the absence of a gyroscope).

1.0.4 - Added support for standard axes and buttons for gamepads, adds the ability to reassign axes and gamepad buttons in the application, with the ability to automatically download and save the settings for each joystick. Added adjusts the size and position of the image under the physical size of the phone (for comfortable viewing in a headset Fibrum). Added information label for smartphones without a built-in gyro.

1.0.3 - Added support for Unity Canvas-element out of the box, using the interface by headtracking.

1.0.2 - Added check the performance of the smartphone, in the case of low-performance lens distortion effects are disabled.

1.0.1 - Added antidrift system as based on accumulating data, and magnetic compass of smartphone.

1.0.0 - SDK for smartphones running Android and IOS, supports headtracking.