

## 02 - Input Handling

### Overview

- Device Types
- Device Abstractions
- Controllers
- Input Handling Packages
- Event Queues
- Input Action Commands  
(making the inputs do things)

2

### Types of Input Devices

- Keyboard
- Mouse
- Joystick (“POV”)
- “POV Hat Switch”
- Gamepad
- Paddle
- Steering Wheel
- Dance Pad
- Guitar
- WiiMote
- Kinect
- others?

3

### Input Handling Goals

#### Keep games device-independent

- Game shouldn't contain hard-coded device details
- Game shouldn't fail when a particular device is absent (allow substitution)

#### Keep engine device-independent

- Engine components should not contain hard-coded device details  
... isolate details in an *Input Manager*

4

### Device Abstractions

#### Two fundamental device types:

- **Button** – returns pressed or not pressed  
Frequently represented as 1.0 or 0.0

- **Axis** – returns a float

#### Two types of Axis:

- **Continuous:** returns a value in a range  
e.g.  $\{-1 \dots 1\}$  or  $\{0 \dots 1\}$
- **Discrete:** returns a value from a set  
e.g.  $\{0, 1\}$  or  $\{-1, 0, 1\}$

Can be absolute or relative

5

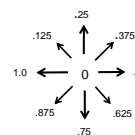
### “D-pad” (Directional-pad) Axes

#### Discrete axis devices

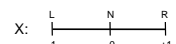
- Can have either *one* or *two* axes



Single-axis form: *one* component;  
returns *one* value:



Dual axis form: *two* components;  
*each* returns a value:



6

## Controllers

Most “devices” are really collections :

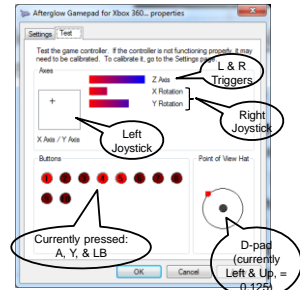
- Keyboard: collection of (e.g. 127) “buttons”
- Mouse: collection of (typically 1 to 3 or more) “buttons”, plus two “axes” (‘X’ and ‘Y’)
- Gamepad: collection of buttons and axes
  - Multiple physical buttons
  - Joystick: two (continuous) “axes” (per joystick)
  - D-pad: one or two (discrete) “axes”
  - “POV hat switch”: one or two (discrete) “axes”

Device collection == “Controller”

## Controller Example: GamePad



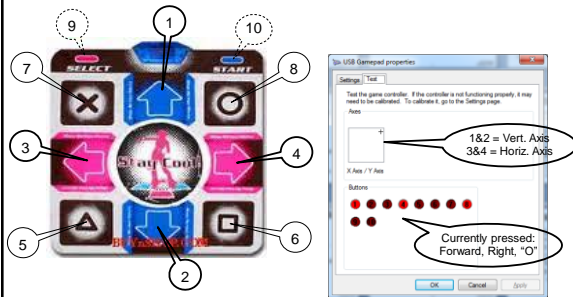
PDP Afterglow Xbox-360 GamePad Controller



16 “devices”: 10 Buttons,  
2 Joysticks (4 continuous axes, each [-1..1]),  
2 “Triggers” (combined as a single Axis [-1..1]),  
1 D-pad (1 discrete axis:  
[0, .125, .25, ... .875, 1])

ControlPanel -> Devices&Printers -> RIGHT-click the controller icon -> Game Controller Settings > Properties

## Controller Example: Dance Pad

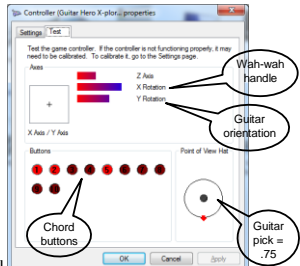


12 “devices”: 10 Buttons,  
2 discrete axes, each [-1, 0, 1]

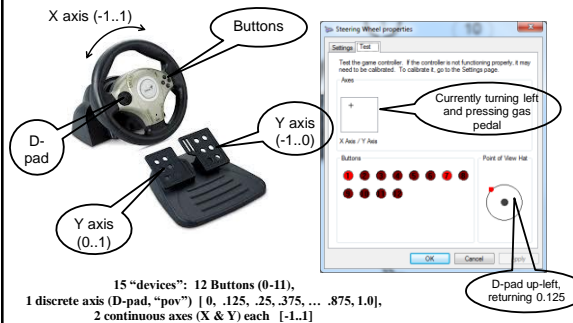
## Controller Example: Guitar



16 “devices”: 10 Buttons,  
5 continuous axes, each [0..1],  
1 discrete axis, [0, .125, .25, .375, ... .875, 1.0]  
(pick Off/Up/Down = [0, .25, .75])



## Controller Example : Steering Wheel



15 “devices”: 12 Buttons (0-11),  
1 discrete axis (D-pad, “pov”) [0, .125, .25, .375, ... .875, 1.0],  
2 continuous axes (X & Y) each [-1..1]

## Accessing Game Controllers

### DirectInput

- Windows-specific (part of Microsoft’s DirectX framework)
- (deprecated)

### XInput

- Microsoft API for Xbox 360 controllers
- No support for keyboards or mice

### OIS (Object-oriented Input System)

- SourceForge (open source) project, mostly C++

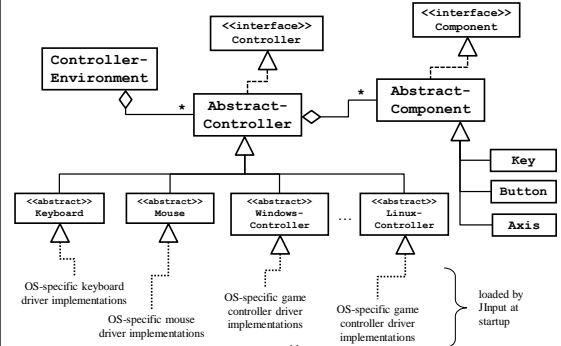
### JInput

- Part of JGI (Java Gaming Initiative) framework (JOGL, etc.)
- Under negotiation with Jogamp
- Supports Windows, Linux, OS-X, AWT, etc.

## Primary JInput Objects

- **ControllerEnvironment**
  - Contains the collection of defined "controllers"
    - Examples: Keyboard, Mouse, Joystick, GamePad...
- **Controller**
  - Contains a collection of "components" (input generators)
    - Examples: button, key, slider, dial, controller
  - Can also contain "rumbler" (output feedback devices)
- **Component**
  - An object with a single "range"
    - Button: on/off
    - Key: pressed/notPressed
    - Axis: a value in some range

## JInput Organization



## Controller Attributes

- Name (human-readable)
- Type
  - Keyboard, Mouse, Fingerstick, GamePad, HeadTracker, Rudder, Stick, Trackball, Trackpad, Wheel, Unknown
- Array of (sub)-controllers
- Array of components
- Array of rumbler
- Event Queue

See code example for accessing controller attributes with JInput

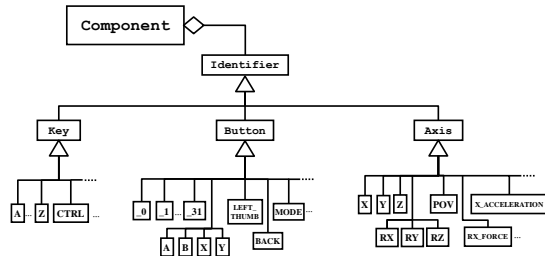
## Component Attributes

- Name (human-readable)
- "Identifier" (type)
  - Axis, Button, or Key
- Return value type
  - Relative*: value is relative to previous return value
  - Absolute*: value is independent of previous return value
- Return value range capability
  - Analog*: allows more than two values
  - Digital*: only two values allowed (e.g. a button)
- Dead zone value
  - Threshold before switching from 0 to non-zero (useful for joysticks: minor movement ignored)

See code example for accessing component attributes with JInput

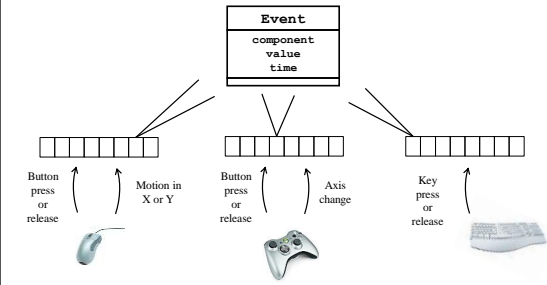
## Component ID

Every component has a *type identifier*  
predefined identifiers in Jinput javadoc for Component.Identifier.\*



## Input Events

Each controller has an *event queue*



## Simplifying Input Handling

### Game goals:

- For each device component event, invoke some (game-specified) **action** associated with that event
- Hide details inside Game Engine

### Examples:

- Gamepad Button 2 pressed → Fire Rocket
- Keyboard 'f' key pressed → Fire Rocket
- Joystick "X" axis moved → Change Camera View
- Guitar "Pick" axis "down" →

```
button = getCurrentChordButton();
if (button == displayedNote) {score++}
```

19

## TAGE *InputManager*\*

### Implements:

```
associateAction(controller,
                component, Action,...)
```

- Registers a user-specified action corresponding to the given controller & component

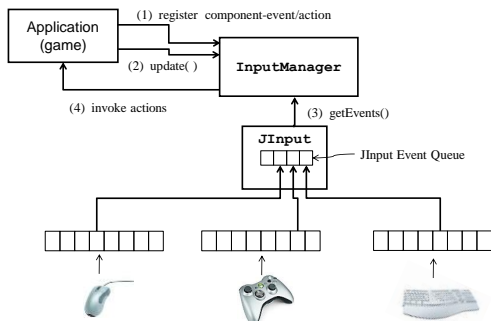
```
update(float time)
```

- Polls the underlying device event queues
- Performs event dispatch (action invocation)

\*adapted from SAGE

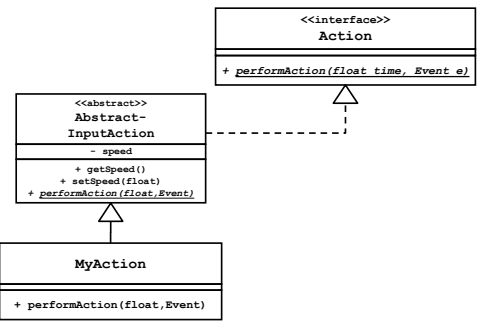
20

## TAGE *InputManager* (continued)



21

## Defining an Action interface



22

## Button/Key Action Types

Not all actions should be invoked on every component state-change

- On Press Only:  
`Fire_Missile_Action, Reset_Camera_Action`
- On Press And Release:  
`Toggle_Running_Action`
- Repeatedly while held down:  
`Move_Forward_Action`

23

## Handling Laptops with Multiple Keyboards

- Many laptops utilize multiple keyboard controllers to allow quick plug-and-play of external keyboards.
- This can cause issues identifying the currently active keyboard to associate with the desired actions.
- A solution is to associate ALL keyboards with a desired action. TAGE has functions to do this:

```
associateActionWithAllKeyboards(
    component, Action, ActionType)
```

24