

USER INPUT

STYLES, HARDWARE, UNITY

TOMÁŠ POLÁŠEK IPOLASEK@FIT.VUT.CZ

BRNO UNIVERSITY OF TECHNOLOGY

FACULTY OF INFORMATION TECHNOLOGY

DCGM, CPhoto@FIT

FACULTY OF FINE ARTS

GAME MEDIA STUDIO



HUMAN INTERFACE DEVICES

PLAYER INTERACTION

- HID = Human Interface Device

- Input & Output Interaction

- Input → Actuators

- Movement
- Sound
- Motion

- Output → Senses

- Sight
- Hearing
- Smell
- Taste
- Proprioception
- Touch

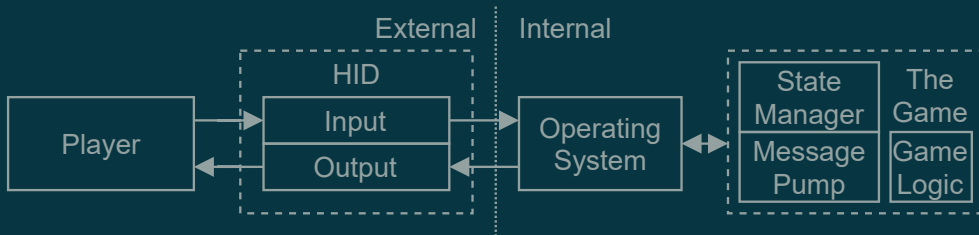
- Control × Immersion

THE PROCESS

■ Player ↔ Game

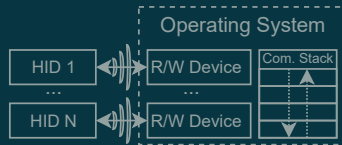
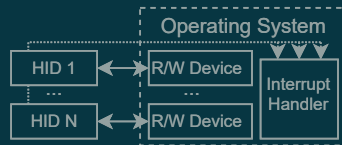
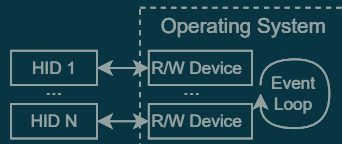
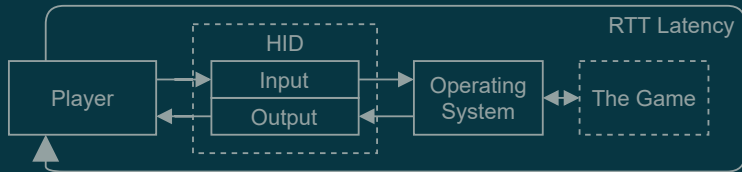
■ Processing Chain

1. Player Interaction
2. HID Communication
3. Game Behavior



HID COMMUNICATION

- $\text{HID} \leftrightarrow \text{OS}$ [2]
- The Channel: Wired $\times$ Wireless
- Communicating State
 - Polling
 - Interrupts
 - Packets
- I/O Latency = Round Trip Time

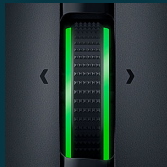
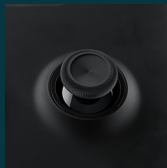


PLAYER INPUTS

Buttons



Positional

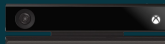


PLAYER INPUTS

Touch



Motion



PLAYER FEEDBACK

- Using Human **Senses**
- Visual (**Sight**) → Rendering
- Sound (**Hearing**) → Audio
- **Smell & Taste**
- Haptics (**Touch**):
 - ▶ Rumble
 - ▶ Force-Feedback
 - ▶ HD Haptics
 - ▶ HD Rumble
 - ▶ Haptic Bass
 - ▶ Resistance



INPUT PROCESSING

■ Messaging **Raw Data** → **Gameplay**

■ Getting **Data**: Polling × Messages

■ Managing **State**

- ▶ Simple Events
- ▶ Composites & Modifiers
- ▶ Sequences & Chords
- ▶ Gestures
- ▶ Value Axes

■ Output API

■ Dynamic Connections

```
void MainPoll()
{ while (mRunning) { InputProcessingPoll(); /* Update, Render, ... */ } }
void InputProcessingPoll()
{
    mNorthPressed = !mNorthDown && mGamePad.buttonNorth.isPressed;
    mNorthReleased = mNorthDown && !mGamePad.buttonNorth.isPressed;
    // ...
}

void MainMessages()
{ while (mRunning) { MessagePump(); /* Update, Render, ... */ } }
void MessagePump() { /* Reset Events, Serve Messages, ... */ }
void OnNorthPressed()
{ mNorthPressed = !mNorthDown; mNorthDown = true;}
void OnNorthReleased()
{ mNorthPressed = mNorthDown; mNorthDown = false;}
```

TAKING ACTION

- Goal: **Inputs** → **Actions**

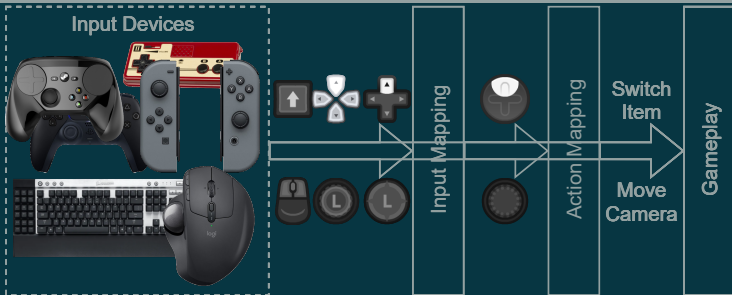
- Action Mapping

- ▶ Simple 1 : 1
- ▶ Input Mapping
- ▶ Action Mapping

- Platform Agnostic & Specifics

- Mapping, Remapping, Binding

- Context-Sensitive Input



```
void DoJump() { /* Make the Player jump... */ }

void ActionSimple()
{ if (mGamePad.crossButton) { DoJump(); } }
void ActionSimpleMultipleDevices()
{ if (mGamePad.crossButton || ...) { DoJump(); } }

void ActionMultiPlatform()
{ if (mGamePad.buttonSouth) { DoJump(); } }

void ActionMapped()
{ if (mPlayerInput.action["Jump"]) { DoJump(); } }
```

INTERFACE DEVICES

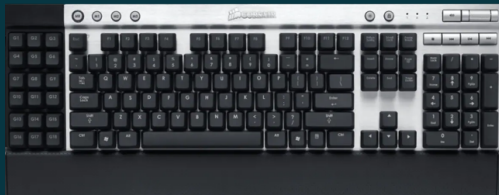
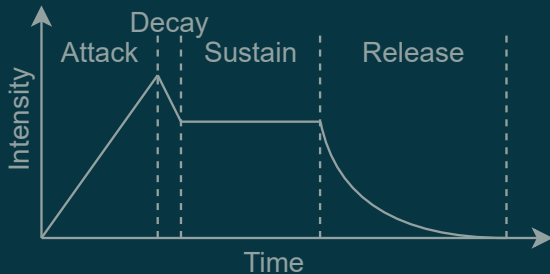
INPUT HARDWARE



CONTROL SCHEME

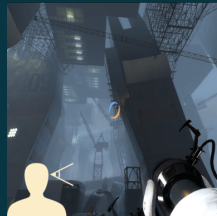
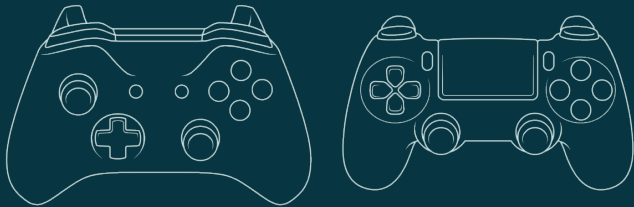
CHARACTER CONTROLS

- **Learning** → Convention
- Conventional Control Schemes
 - Controller
 - Keyboard & Mouse
- Innovation × Tradition
- Humans are **Analog**
- Expectation → Filtering [1]
 - ADSR Curve
 - Control Assist



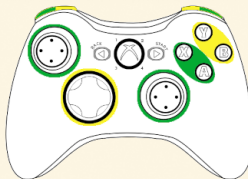
FEELING IN CONTROL

- HID = Means to an End
- Standard → Immersion
- Context Dependent
- Case Study: Camera Styles
 - ▶ Orbit
 - ▶ Top-Down
 - ▶ 1st Person
 - ▶ 3rd Person



DESIGN CONSIDERATIONS

- 10 **Fingers**, 2 **Hands**, 1 **Head**
- Grouping & Neutral Position
- Accessibility Tiers [1]
- Special Circumstances
 - ▶ Casual Games
 - ▶ Disabilities
 - ▶ Virtual / Augmented Reality



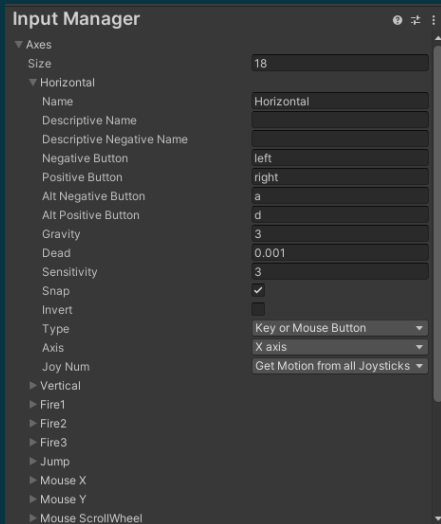
Source: Andrew Dotsenko's Game Design Framework

INPUT IN UNITY

INPUT MODULE

- Default **Input** Method
- “The Old Way”
- Stateless & Axis-Based
- Input Manager

```
void ProcessInput()
{
    if (Input.GetKeyDown(KeyCode.W)) { MoveForward(); }
    if (Input.GetButtonDown("Jump")) { DoJump(); }
    if (Input.GetAxis("Horizontal") > 0.0) { MoveCamera(); }
}
```

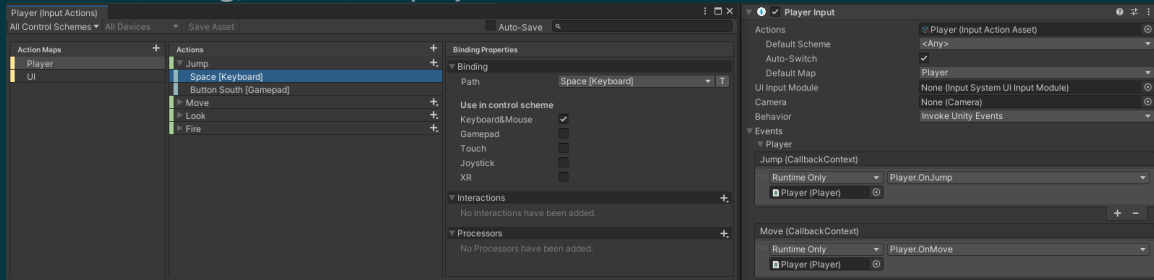


INPUT SYSTEM

- **Input System** Package
- “The New Way”
- Stateless & Action-Based
- Mapping, Bindings, Virtual
- Higher Complexity
- Rebinding, Local Multiplayer

```
public void OnMove(InputAction.CallbackContext ctx)
{ DoMove( ctx.ReadValue<Vector2>()); }
```

```
public void OnJump(InputAction.CallbackContext ctx)
{ if (ctx.started) { DoJump(); } }
```

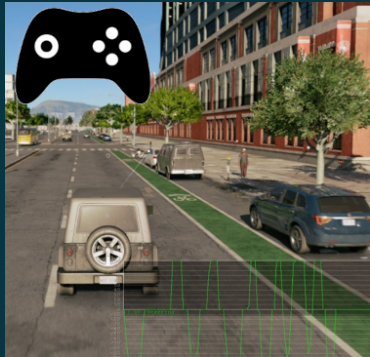


DEBUG AND PROFILING



ADDITIONAL RESOURCES

- [YouTube] Evolution of Video Game Controllers
- [Article] John Harris: 20 Unusual Control Schemes
- [Product] Olorama: Smells in Virtual Reality
- [Article] Andrew Dotsenko: Designing Game Controls



Source: Designing Game Controls

Thanks For
Your Attention!

Baldur's Gate II

MINSC: Minsc and Boo stand ready.

111/111



108/108



95/95



86/86



45/45



49/49



49/49



49/49



REFERENCES I

- [1] ANDREW DOTSENKO. ***DESIGNING GAME CONTROLS***.
<https://gamedesignframework.net/designing-game-controls/>. 2018.
- [2] JASON GREGORY. ***GAME ENGINE ARCHITECTURE, SECOND EDITION***. 3rd. USA: A. K. Peters, Ltd., CRC Press, 2018. ISBN: 1351974288.