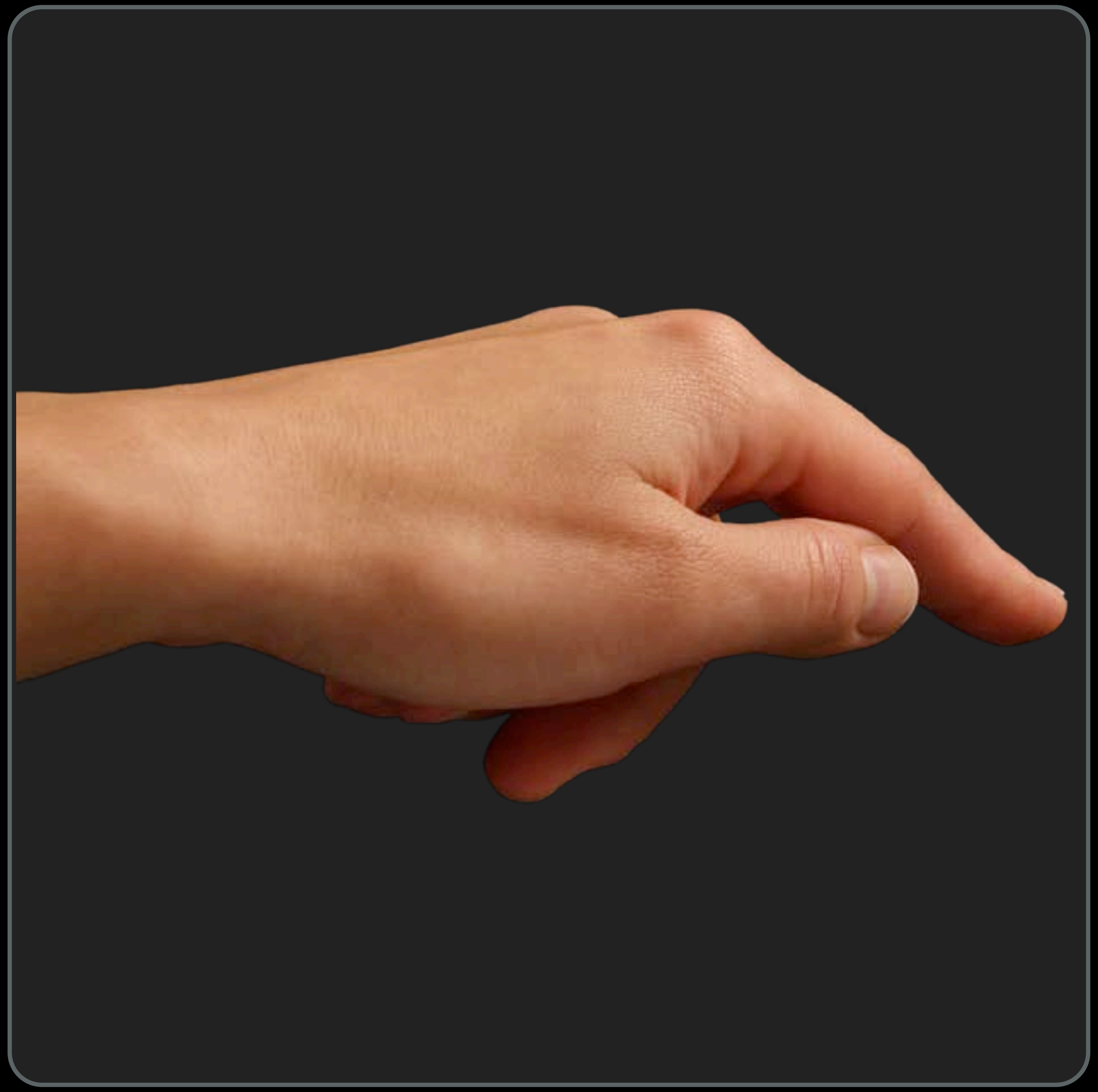
Integrating with Game Controllers

Session 501

Jacques Gasselin de Richebourg
Game Technologies

Game Controllers


Multi-Touch and Motion

Game Controllers

Introduction

Game controller MFi specification

- Physical hardware requirements
- For third-party accessory developers
- MFi Program membership required

Introduction

Game controller framework

- Connect with physical controllers
- Read inputs in-game
- For iOS and OS X game developers

The Controllers

Edwin Foo
iOS Accessories Engineering

The Controllers

Goals

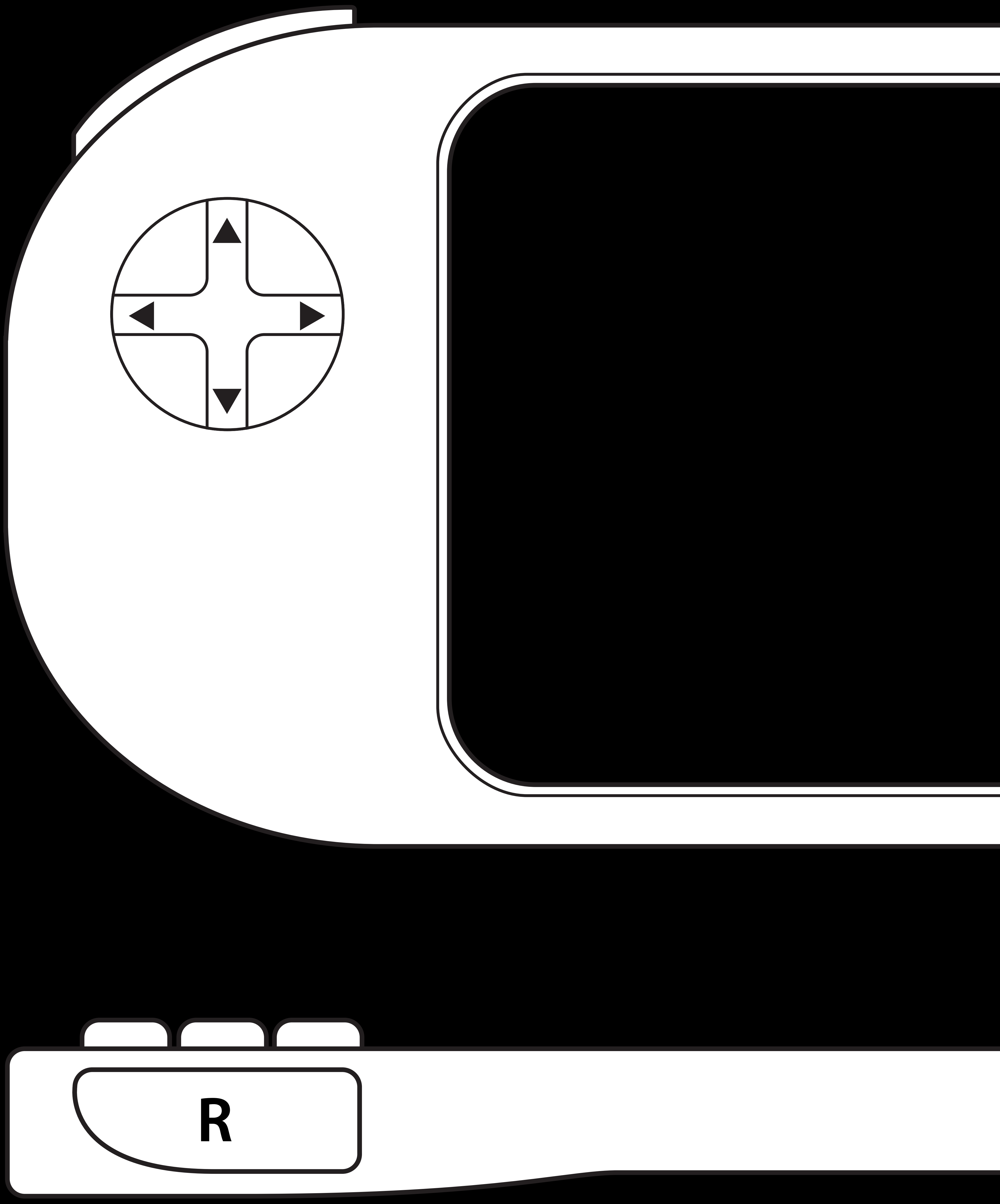
- Consistently high-quality controllers from MFi partners
- Focus on making great games, not dealing with controller differences

Made for
iPod iPhone iPad

The Controllers

Key features

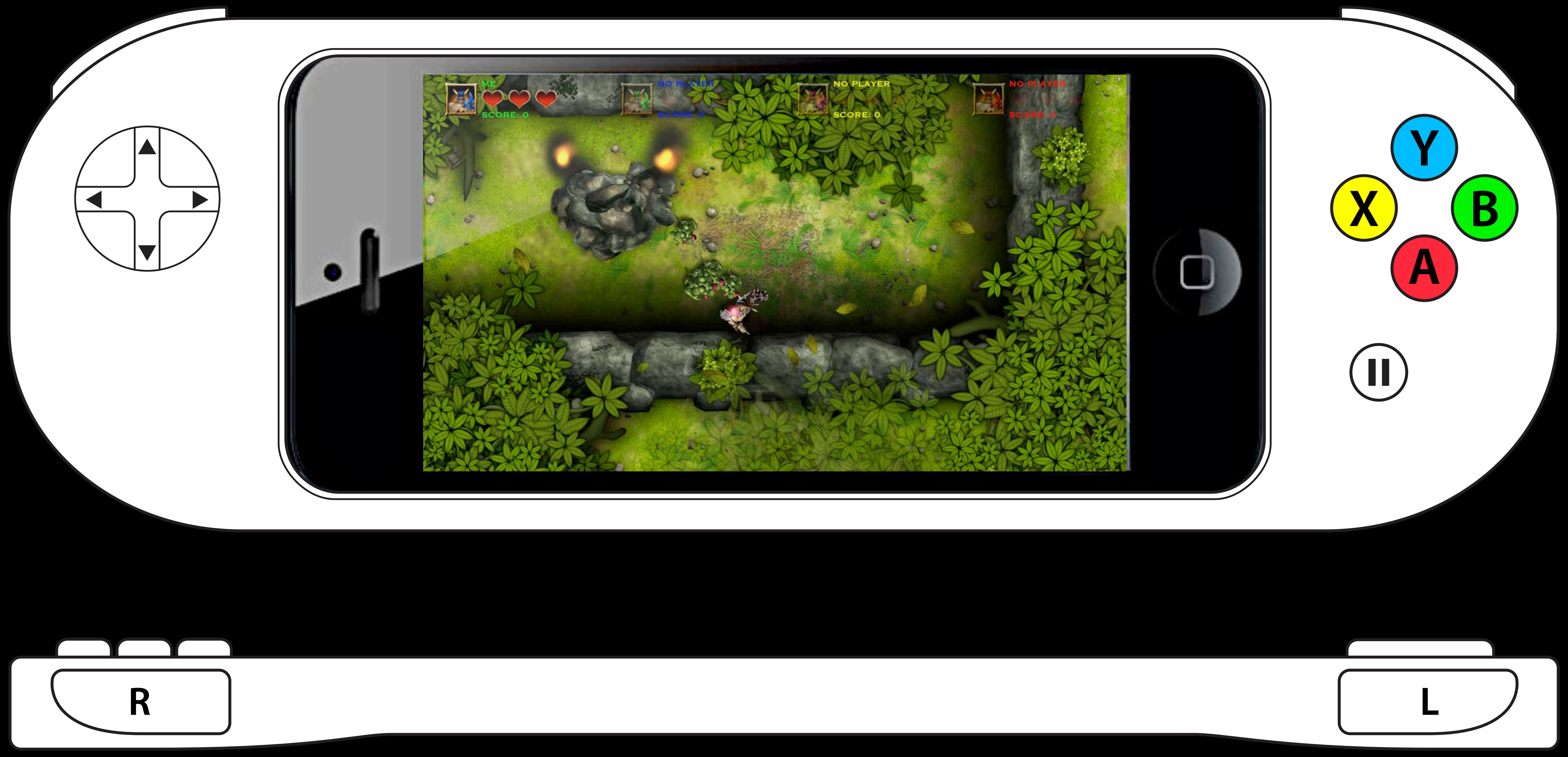
- Consistent control layouts
- Transport agnostic
- Fast report rate
- Buttons
  - Pressure sensitive
  - Consistent feel
- Thumbsticks and D-pads
  - Minimum unit circle coverage
  - No drift
  - No dead zones



The Controllers

Form-fitting standard gamepad

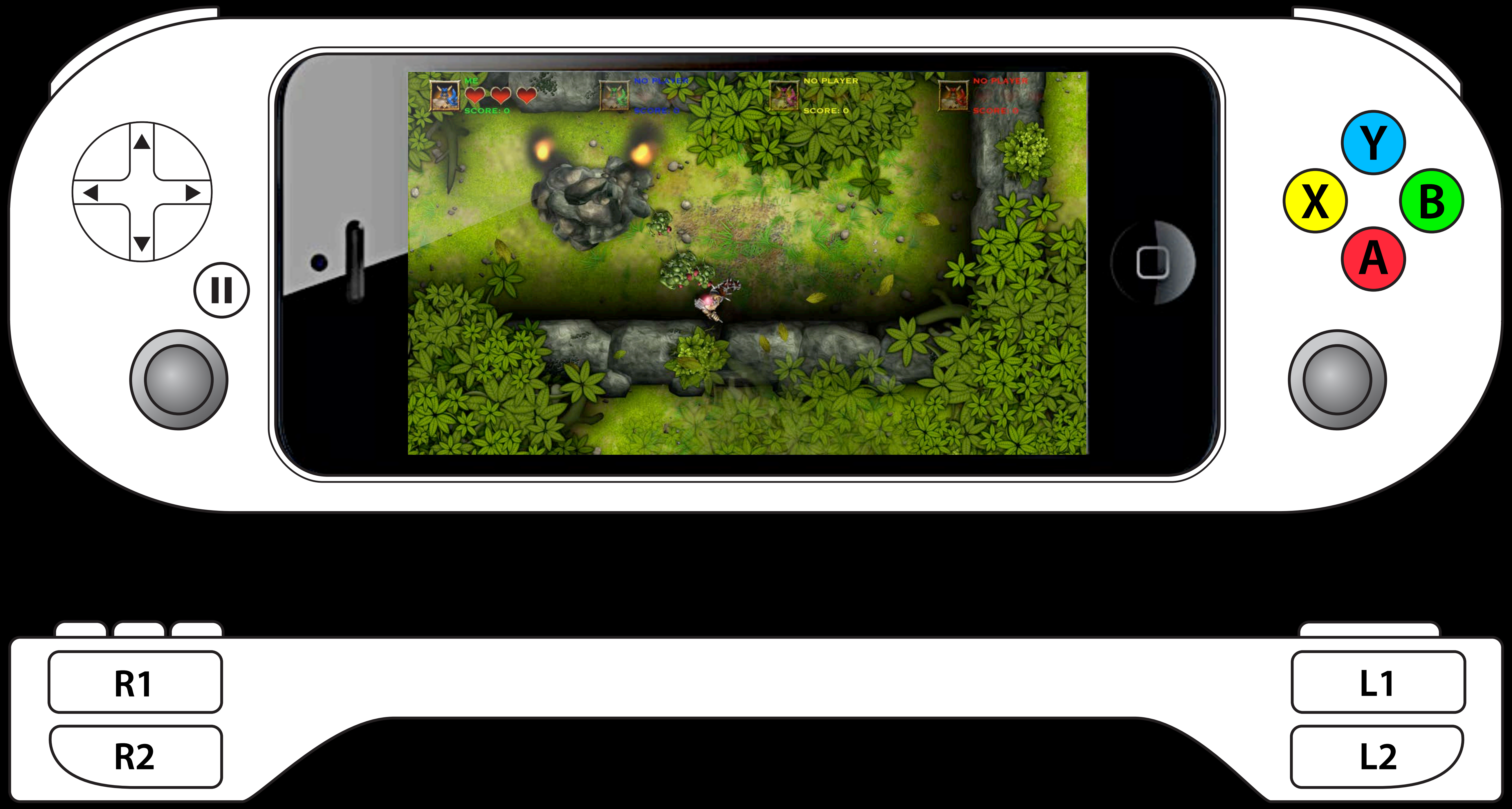
- Form-fitting
  - Physically encases the device
  - User can touch the screen
- Controls
  - D-pad
  - ABXY
  - Shoulders



The Controllers

Form-fitting extended gamepad

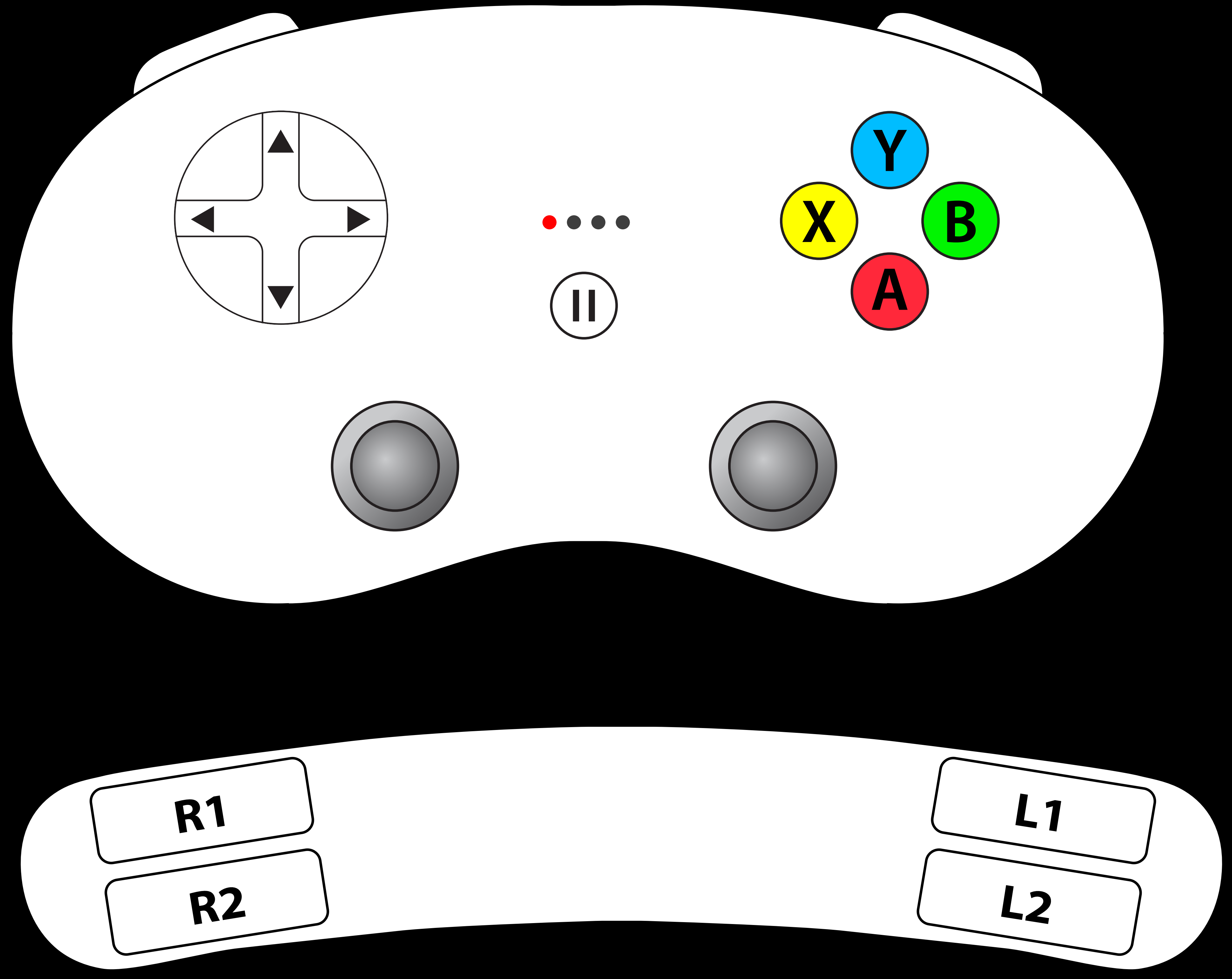
- Form-fitting
  - Physically encases the device
  - User can touch the screen
- Controls
  - D-pad
  - ABXY
  - Shoulders
  - Thumbsticks
  - Triggers



The Controllers

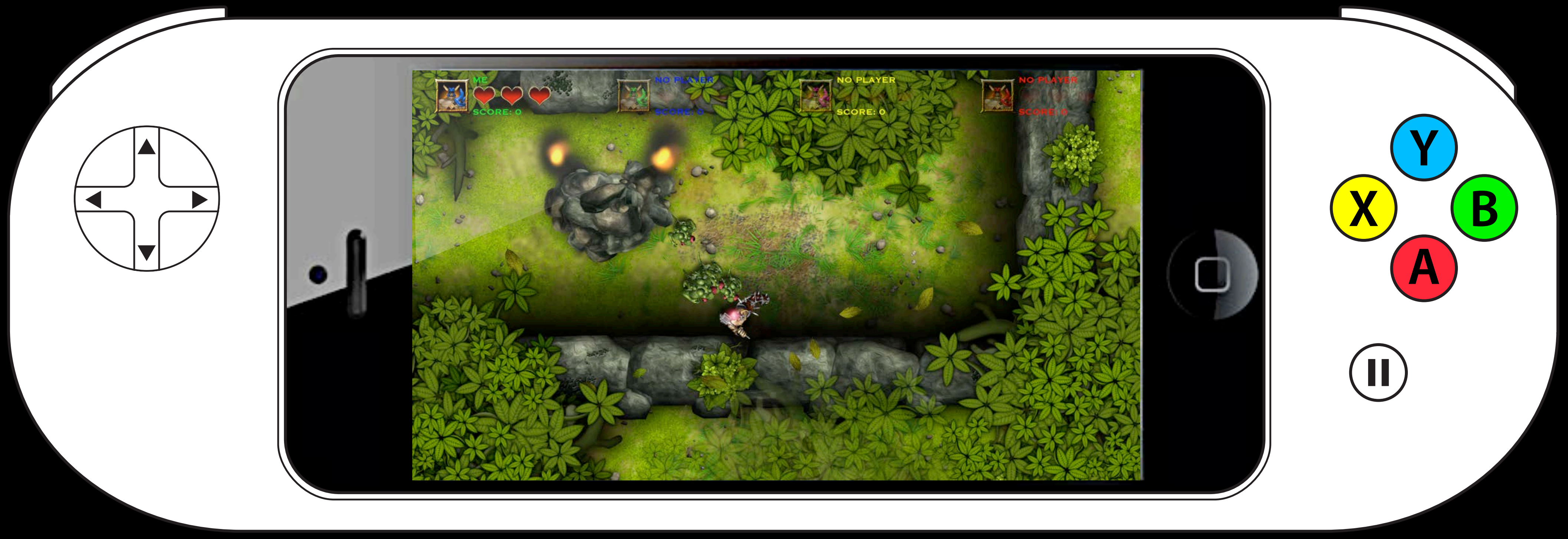
Standalone extended gamepad

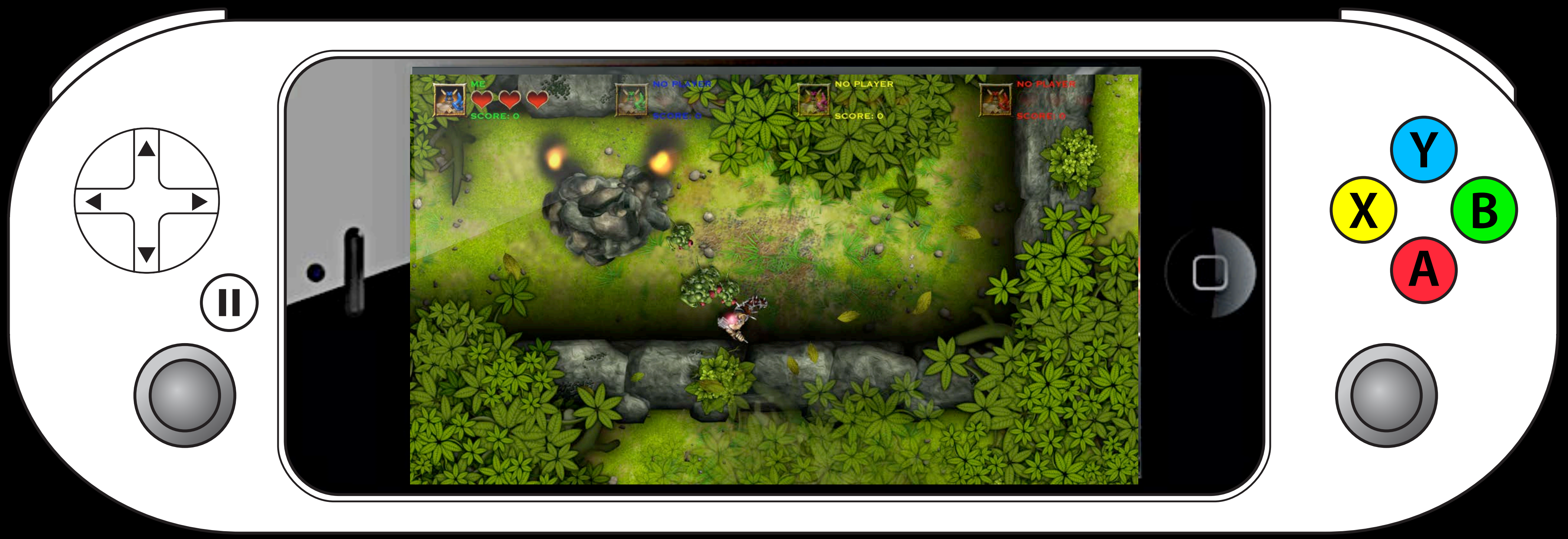
- Standalone
  - Not attached to device
- Controls
  - D-pad
  - ABXY
  - Shoulders
  - Thumbsticks
  - Triggers

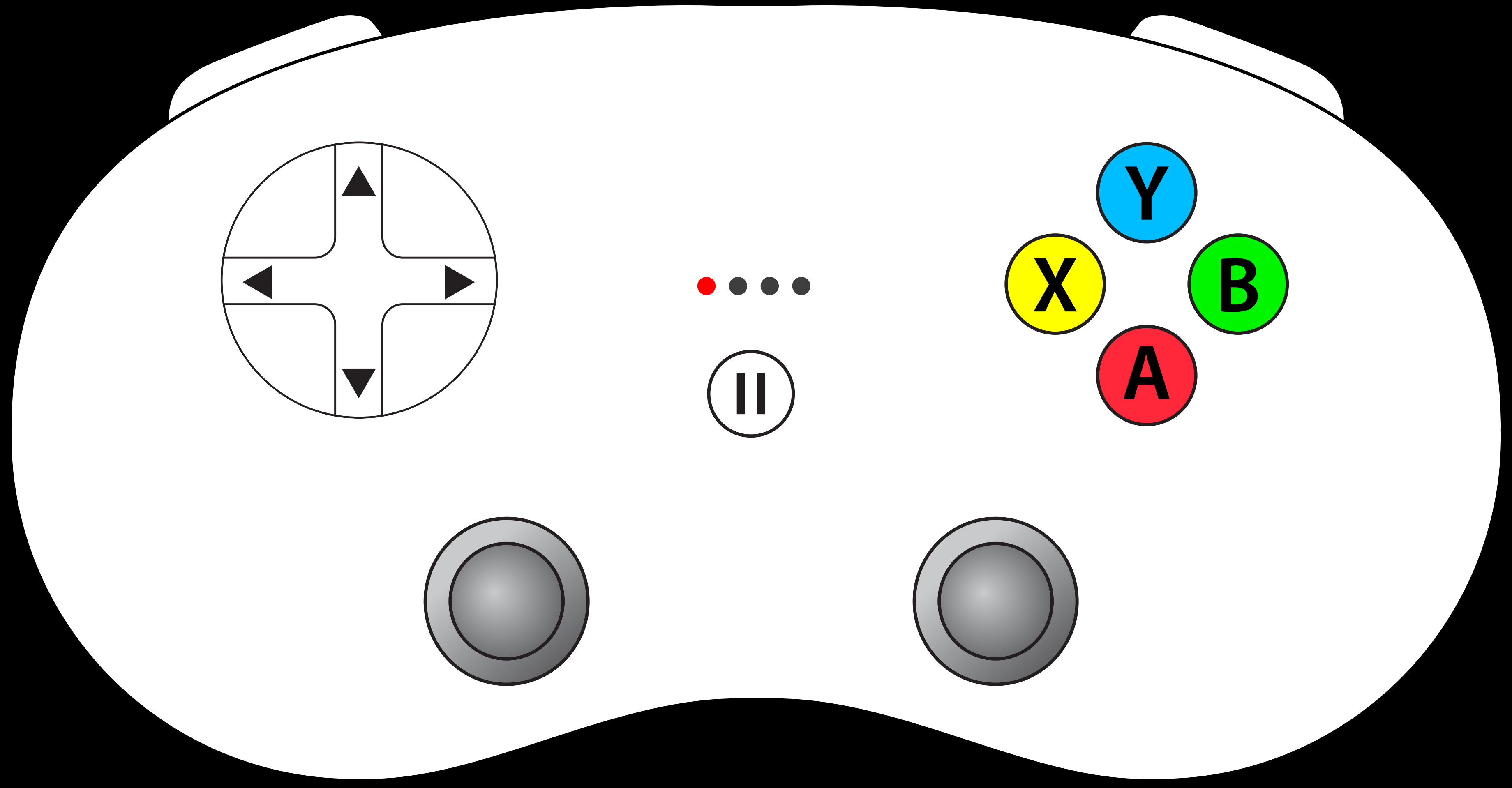


The Controllers

Recap







The Controllers

Apple MFi program

- Technical information
- Hardware components
- Testing tools
- Technical support
- Accessory certification
- Logos and compatibility icons

http://developer.apple.com/mfi/

Made for iPod

Made for iPhone

Made for iPad

The Controllers

Availability

- Working with key partners
- On store shelves later this fall

Logitech

MOGA

Connecting to Controllers

JJ Cwik
Game Technologies

Agenda

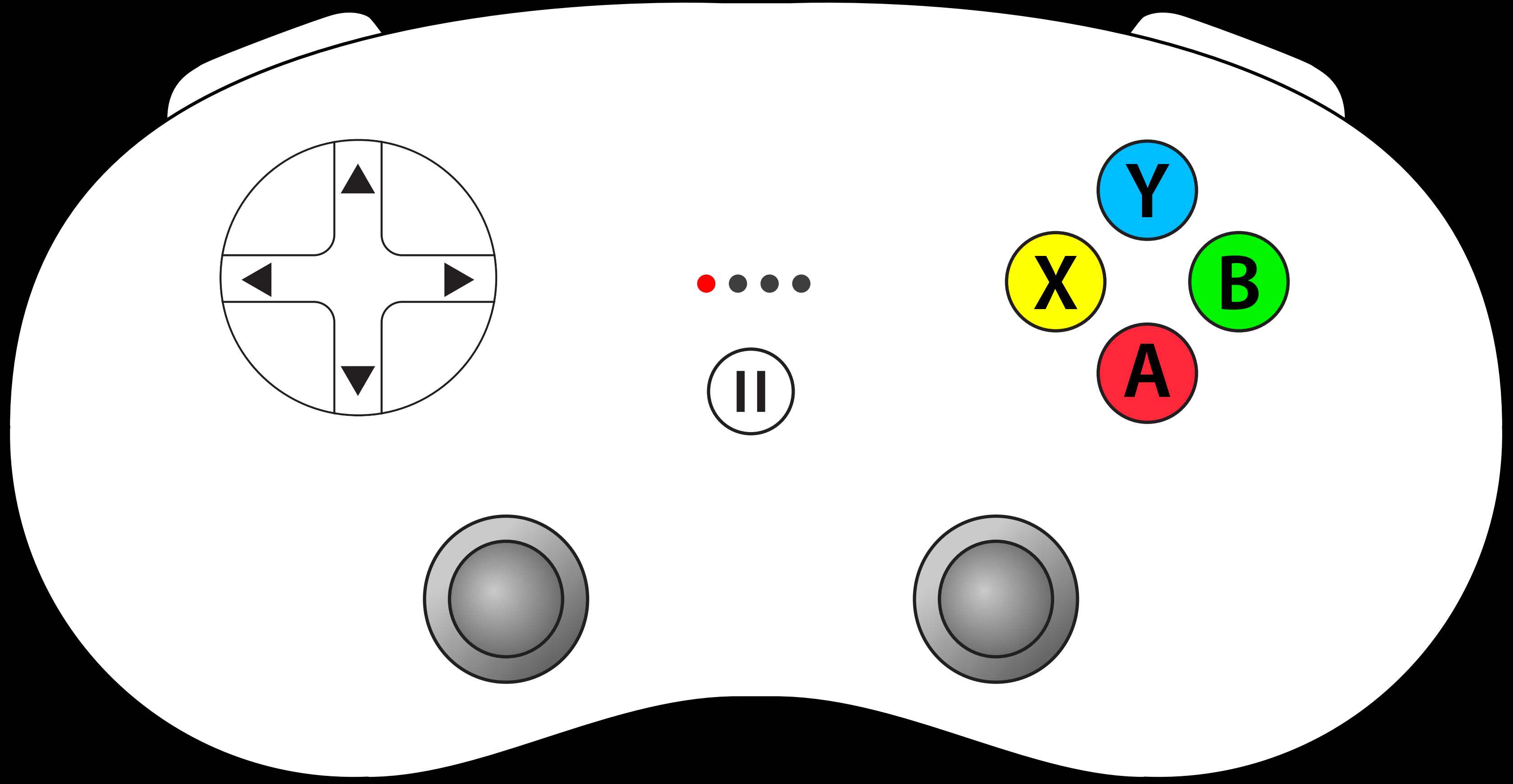
Integrating with game controllers

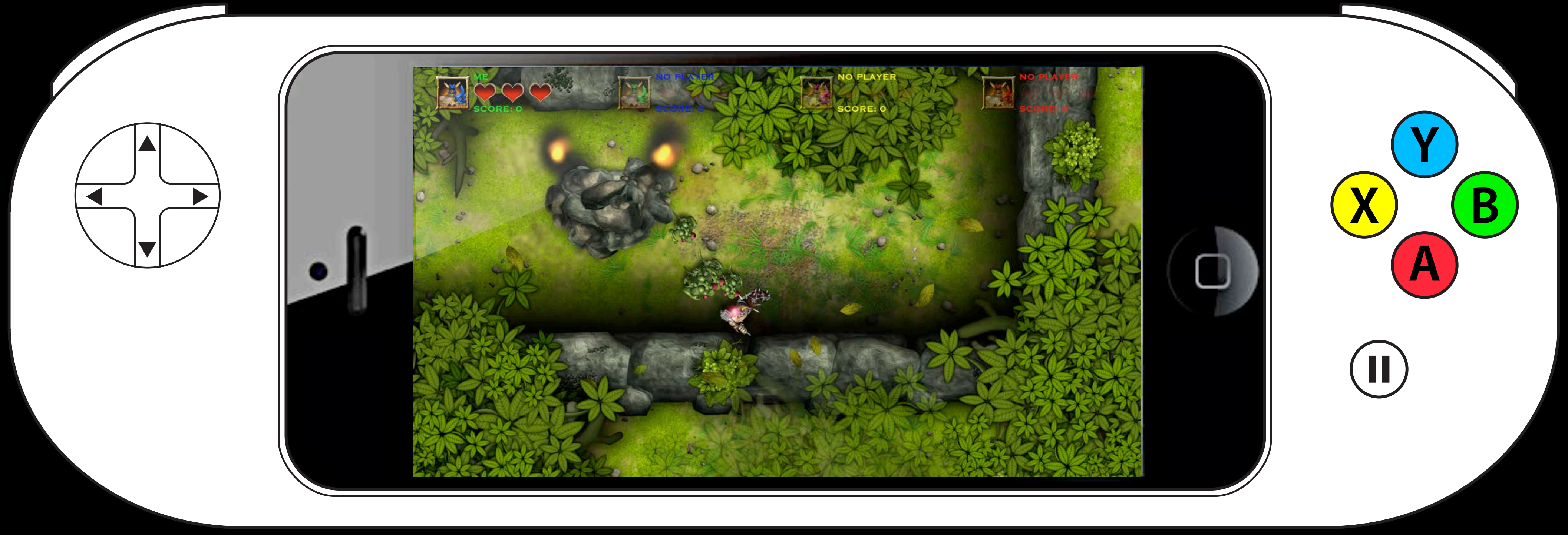
- Overview
- Handling connection and disconnection
- Reading controller inputs
- Using pause button and player indicators
- Best practices

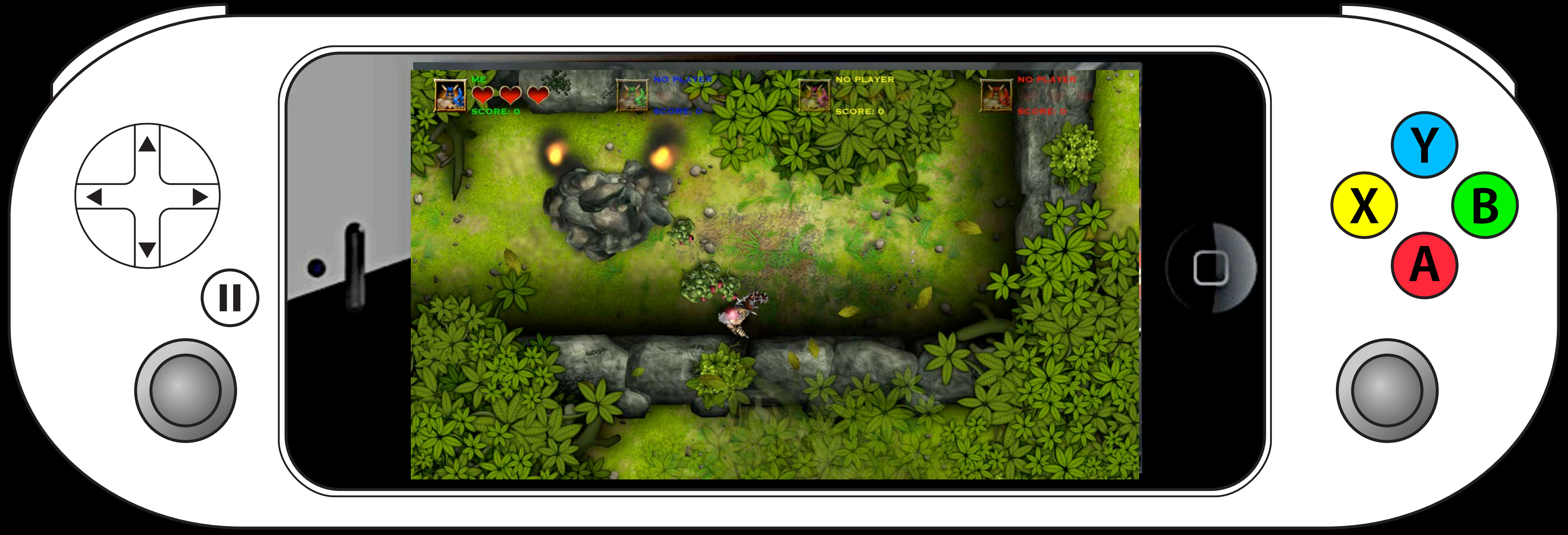
GCController

Overview

- Represents a connected game controller
- Same class for all supported controllers







GCController

What it provides

GCController

What it provides

- Methods for finding controllers
    - Get controllers currently connected
    - Notifications for live connect or disconnect
    - Discover wireless controllers

GCController

What it provides

- Methods for finding controllers
  - Get controllers currently connected
  - Notifications for live connect or disconnect
  - Discover wireless controllers
- Access to physical input data
  - D-pad, buttons, triggers, thumbsticks

GCController

What it provides

- Methods for finding controllers
  - Get controllers currently connected
  - Notifications for live connect or disconnect
  - Discover wireless controllers
- Access to physical input data
  - D-pad, buttons, triggers, thumbsticks
- Information about this controller
  - Type, vendor, player index

Connecting and Disconnecting

Main entry point

```
@interface GCController : NSObject
+ (NSArray *)controllers
...
```

- List of currently connected controllers
- Array of GCController instances (empty if none)
- Updated whenever controllers connect or disconnect

Connecting and Disconnecting

Main entry point

```
@interface GCController : NSObject
+ (NSArray *)controllers
...
```

- List of currently connected controllers
- Array of GCController instances (empty if none)
- Updated whenever controllers connect or disconnect

Connecting and Disconnecting

Example

```
- (void)setupControllers:(NSNotification *)notification
{
    // Get Controllers
    self.controllerArray = [GCController controllers];

    if ([self.controllerArray count] > 0) {
        // Found controllers
    } else {
        // No controllers
    }
}
```

Connecting and Disconnecting

Example

```
- (void)setupControllers:(NSNotification *)notification
{
    // Get Controllers
    self.controllerArray = [GCController controllers];

    if ([self.controllerArray count] > 0) {
        // Found controllers
    } else {
        // No controllers
    }
}
```

Connecting and Disconnecting
Example

```
- (void)setupControllers:(NSNotification *)notification
{
    // Get Controllers
    self.controllerArray = [GCController controllers];

    if ([self.controllerArray count] > 0) {
        // Found controllers
    } else {
        // No controllers
    }
}
```

Connecting and Disconnecting

Notifications

- User may connect or disconnect controller
  - Provides notification of the change

```
NSNotificationCenter* center = [NSNotificationCenter defaultCenter];

// Set up connect notification
[ center addObserver:self selector:@selector(setupControllers:)
                name:GCControllerDidConnectNotification object:nil];

// Set up disconnect notification
[ center addObserver:self selector:@selector(setupControllers:)
                name:GCControllerDidDisconnectNotification object:nil];
```

Connecting and Disconnecting

Notifications

- User may connect or disconnect controller
  - Provides notification of the change

```
NSNotificationCenter* center = [NSNotificationCenter defaultCenter];

// Set up connect notification
[ center addObserver:self selector:@selector(setupControllers:)
                name:GCControllerDidConnectNotification object:nil];

// Set up disconnect notification
[ center addObserver:self selector:@selector(setupControllers:)
                name:GCControllerDidDisconnectNotification object:nil];
```

Discovery

Wireless controllers

- Must be “discovered” before use
  - Generally needed once per device pair
  - Connects automatically thereafter
- App can initiate discovery

  `+startWirelessControllerDiscoveryWithCompletionHandler:`
  - Search runs asynchronously until finished, timeout, or stopped
  - Use in conjunction with Notifications
- To stop early

  `+stopWirelessControllerDiscovery`

Discovery

Example

```
- (void)userStartedDiscovery
{
    [self startMySpinner];
    // Find wireless controllers – triggers notifications
    [GCController startWirelessControllerDiscoveryWithCompletionHandler:^{
        // Discovery ended
        [self stopMySpinner];
    }];
}

- (void)userStoppedDiscovery
{
    // Stop discovery early
    [GCController stopWirelessControllerDiscovery];
}
```

Discovery

Example

```
- (void)userStartedDiscovery
{
    [self startMySpinner];
    // Find wireless controllers - triggers notifications
    [GCController startWirelessControllerDiscoveryWithCompletionHandler:^{
        // Discovery ended
        [self stopMySpinner];
    }];
}

- (void)userStoppedDiscovery
{
    // Stop discovery early
    [GCController stopWirelessControllerDiscovery];
}
```

Discovery

Example

```
- (void)userStartedDiscovery
{
    [self startMySpinner];
    // Find wireless controllers - triggers notifications
    [GCController startWirelessControllerDiscoveryWithCompletionHandler:^{
        // Discovery ended
        [self stopMySpinner];
    }];
}
```

```
- (void)userStoppedDiscovery
{
    // Stop discovery early
    [GCController stopWirelessControllerDiscovery];
}
```

Connecting and Disconnecting

Caveats

Connecting and Disconnecting

Caveats

- `[GCController controllers]` array will always be empty in `-application:didFinishLaunchingWithOptions:`
  - Set up notifications there

```
GCControllerDidConnectNotification
GCControllerDidDisconnectNotification
```

Connecting and Disconnecting

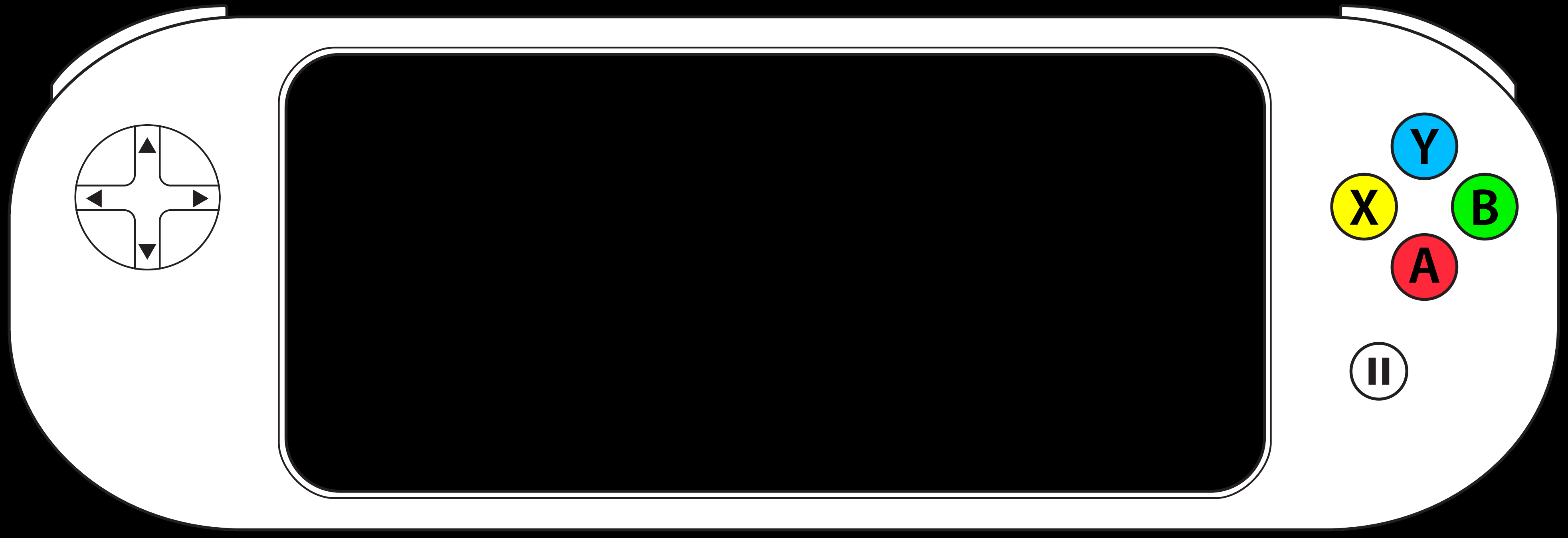
Caveats

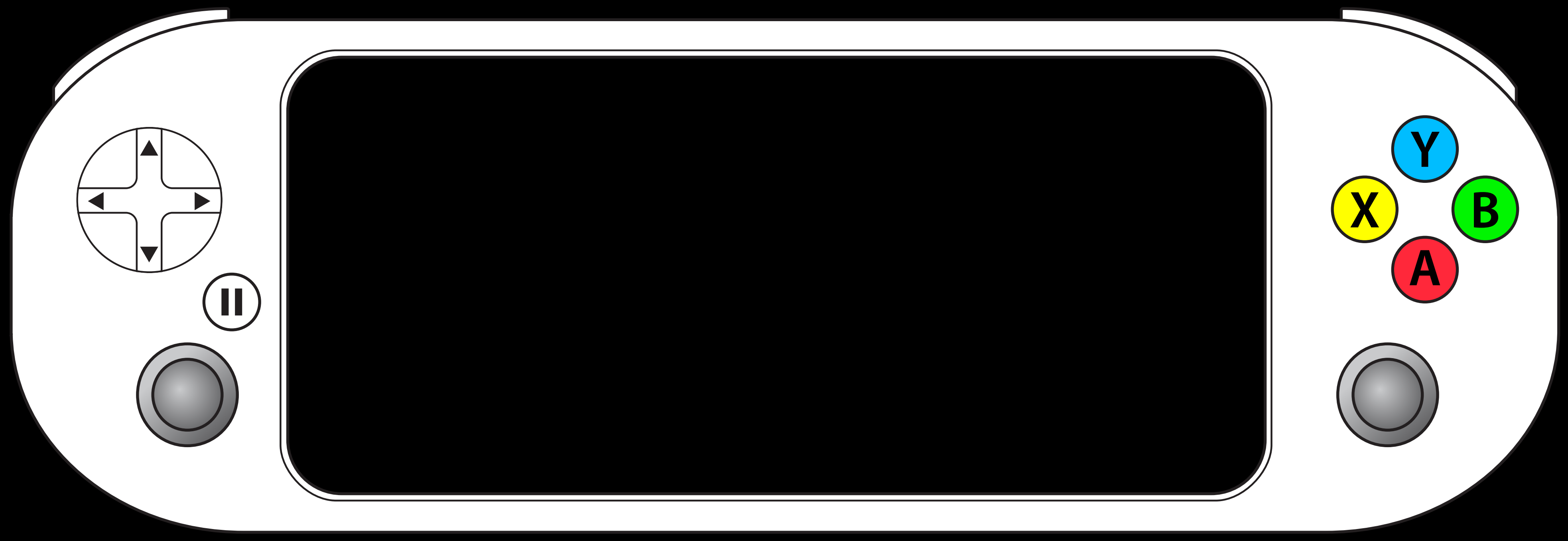
- `[GCController controllers]` array will always be empty in `-application:didFinishLaunchingWithOptions:`
    - Set up notifications there

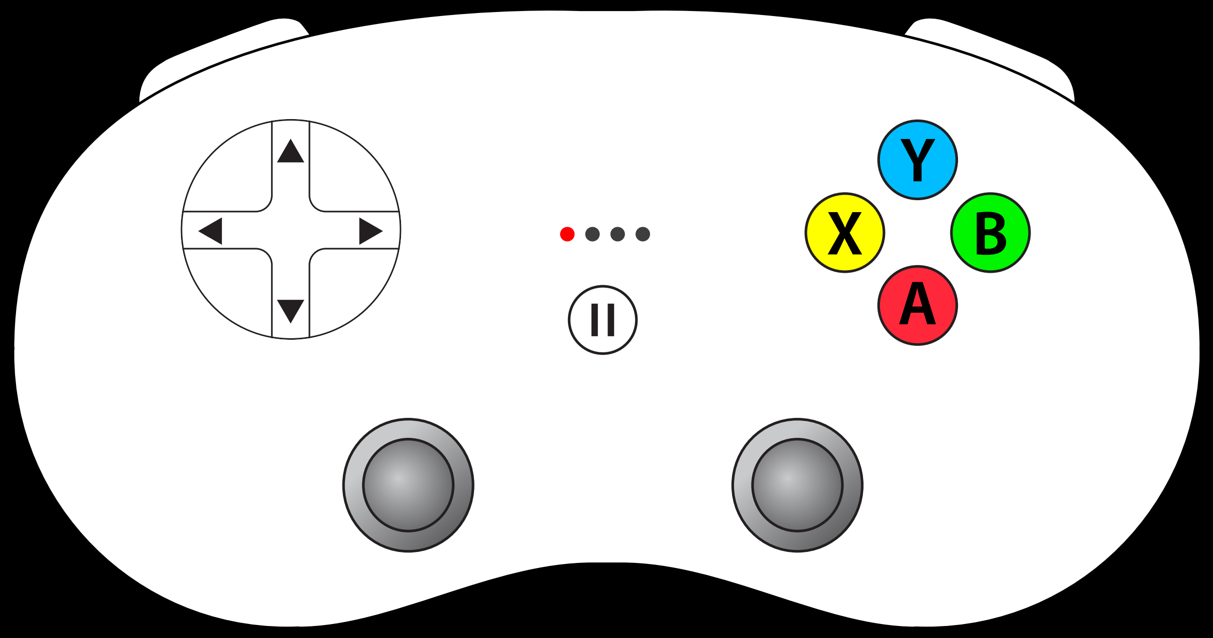
      ```
      GCControllerDidConnectNotification
      GCControllerDidDisconnectNotification
      ```
- Note:
    - If your game waits to set up notifications later
    - Controller array may already be populated by then

Reading Controller Input

Controller Profiles

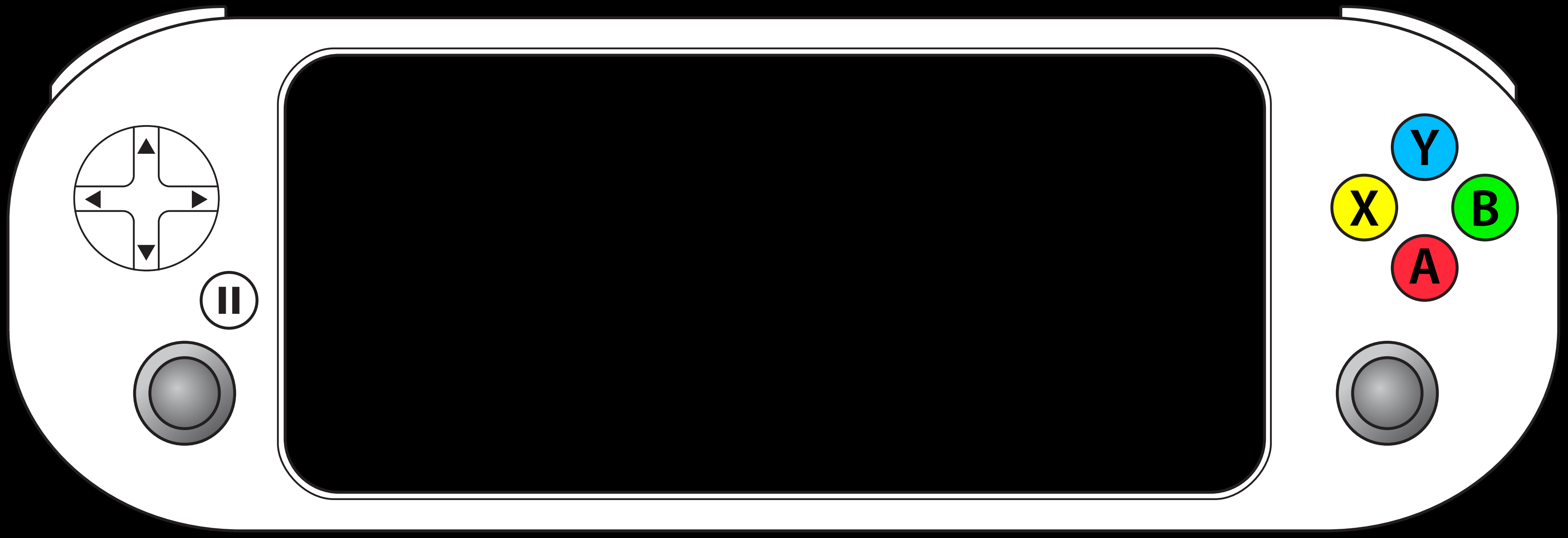


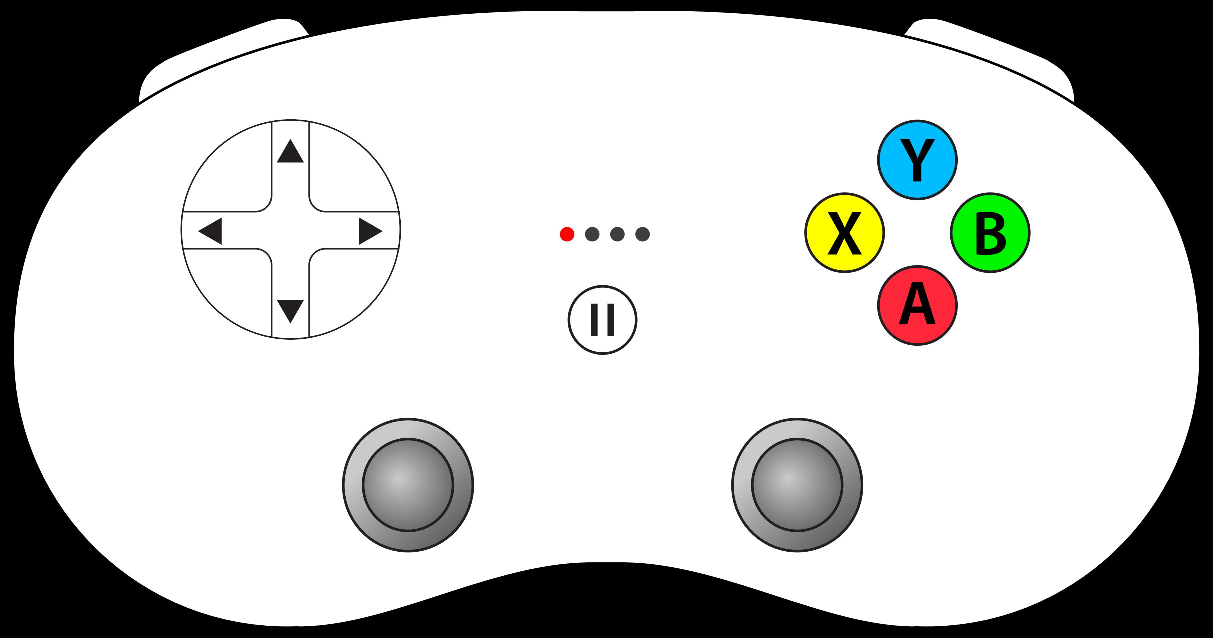




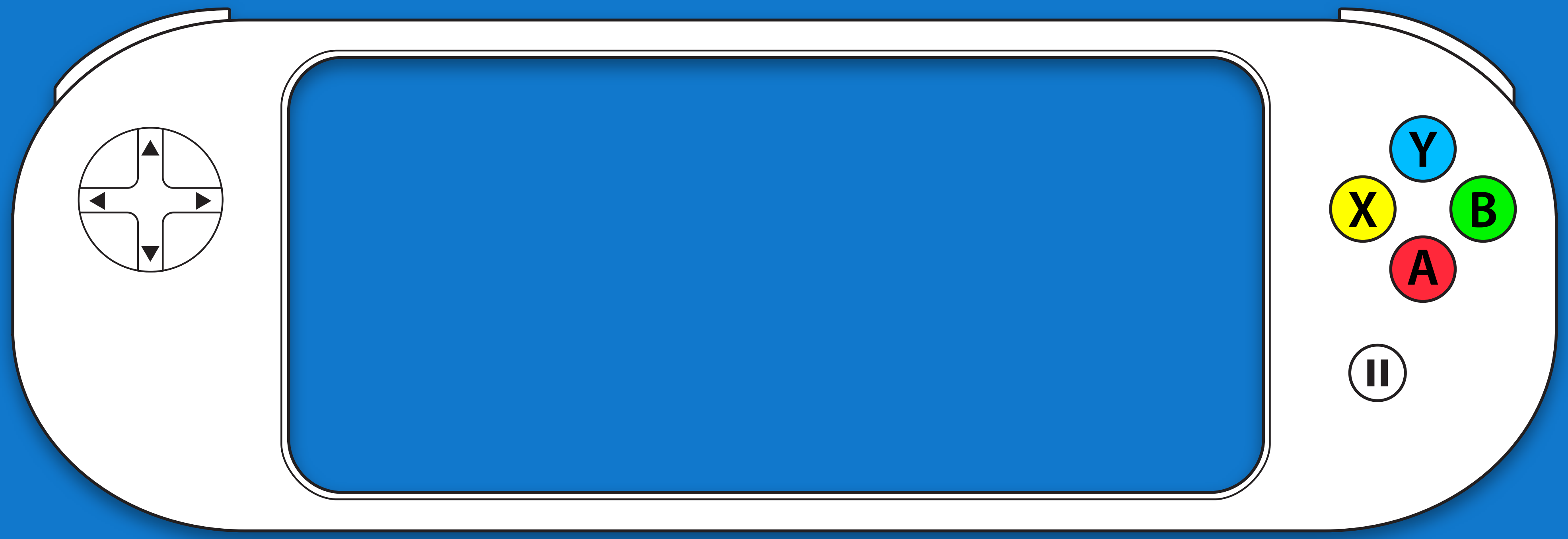
Controller Profiles

Standard Gamepad Profile

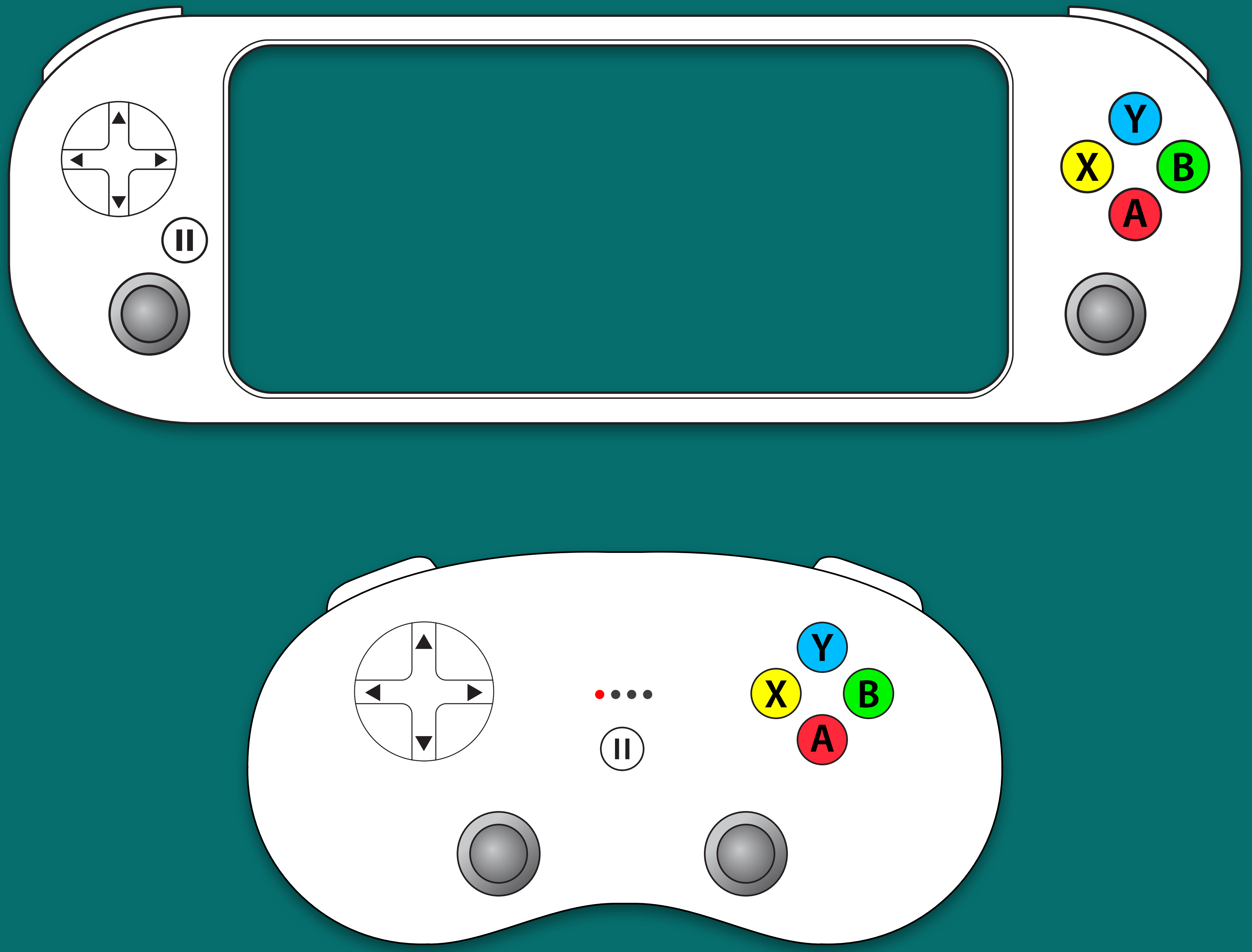




Controller Profiles



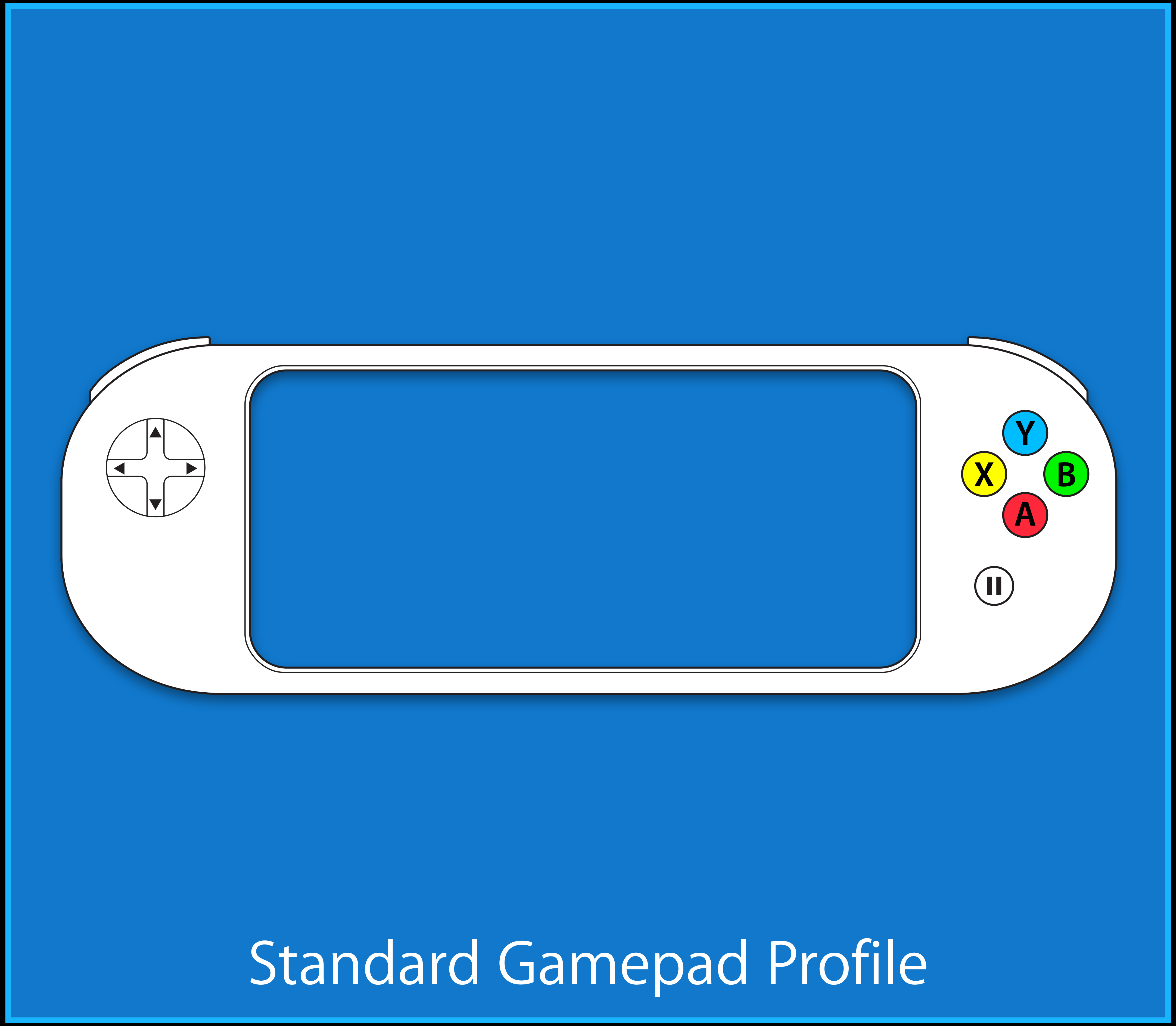
Standard Gamepad Profile



Extended Gamepad Profile

Controller Profiles

Standard gamepad profile



- Four face buttons
  - A, B, X, Y
- Two shoulder buttons
  - L, R
- One D-pad

Standard Gamepad Profile

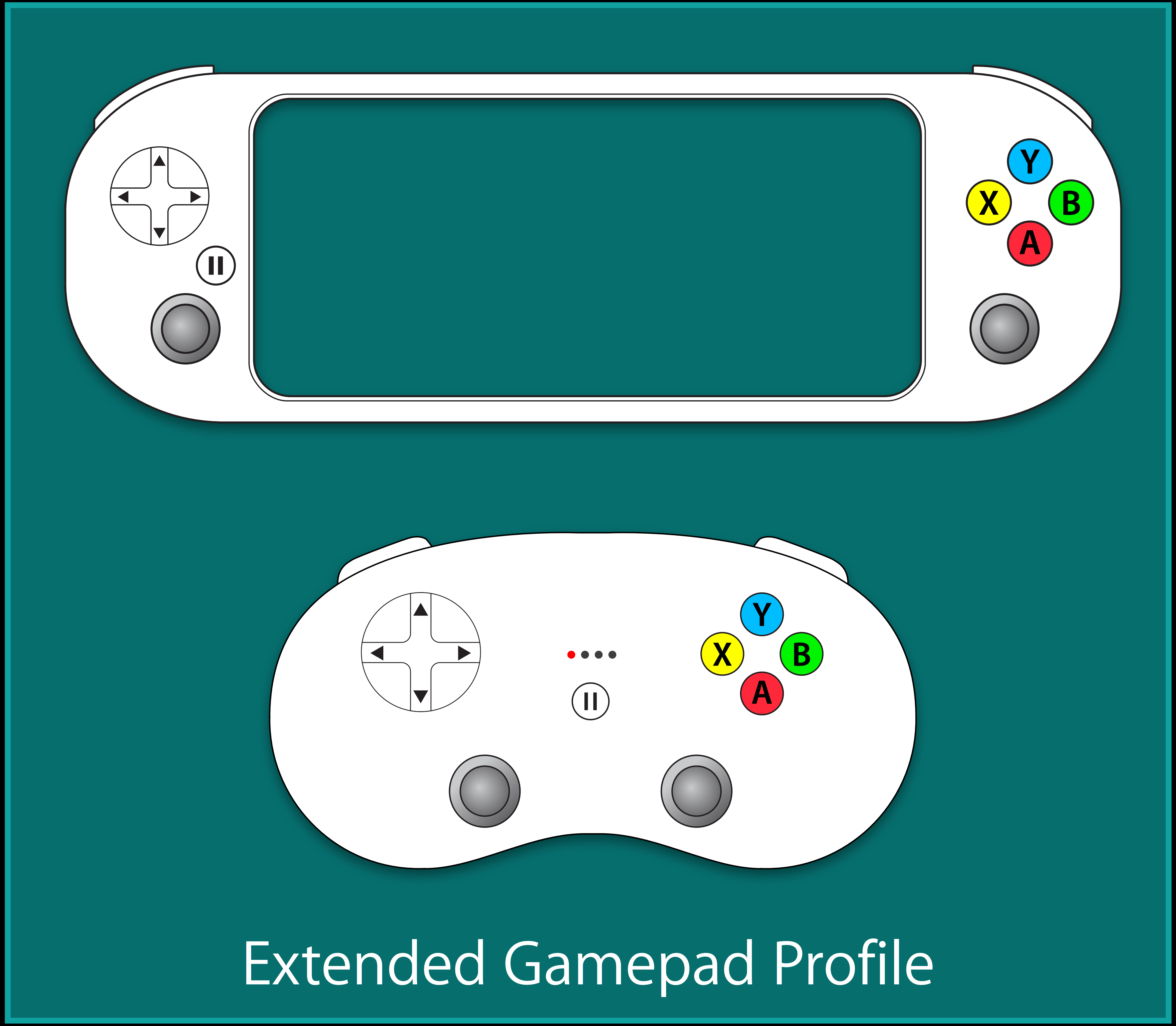
`self.myController.gamepad`

Control	Property Name	Type
Face buttons	`buttonA`	ButtonInput
	`buttonB`	
	`buttonX`	
	`buttonY`	
Shoulder buttons	`leftShoulder`	ButtonInput
	`rightShoulder`	
D-pad	`dpad`	DirectionPad

Controller Profiles

Extended gamepad profile

- Two thumbsticks
  - Left, right
- Two triggers
  - L2, R2
- Four face buttons
  - A, B, X, Y
- Two shoulder buttons
  - L1, R1
- One D-pad



Extended Gamepad Profile

Extended Gamepad Profile

`self.myController.extendedGamepad`

Control	Name	Type
Thumbsticks	`leftThumbstick`	DirectionPad
	`rightThumbstick`	
Triggers	`leftTrigger`	ButtonInput
	`rightTrigger`	
Face buttons	`buttonA`	ButtonInput
	`buttonB`	
	`buttonX`	
	`buttonY`	
Shoulder buttons	`leftShoulder`	ButtonInput
	`rightShoulder`	
D-pad	`dpad`	DirectionPad

Element Types

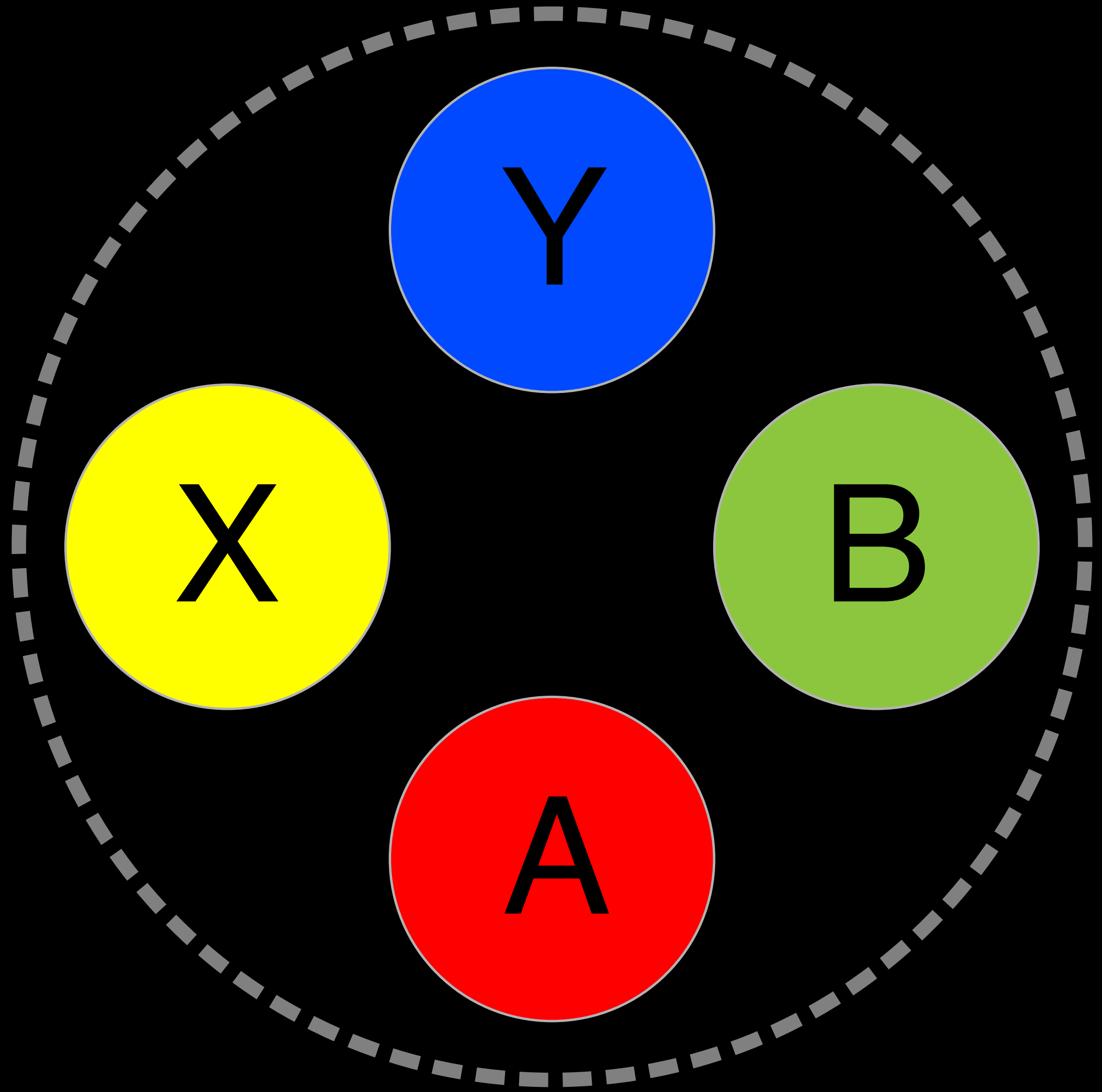
GCControllerButtonInput

- Classic button state

```
BOOL pressed;    // Whether button is pressed
```

- Buttons are also pressure sensitive

```
float value;     // Amount of pressure (analog)
                 // Normalized from 0.0 to 1.0
```



Element Types

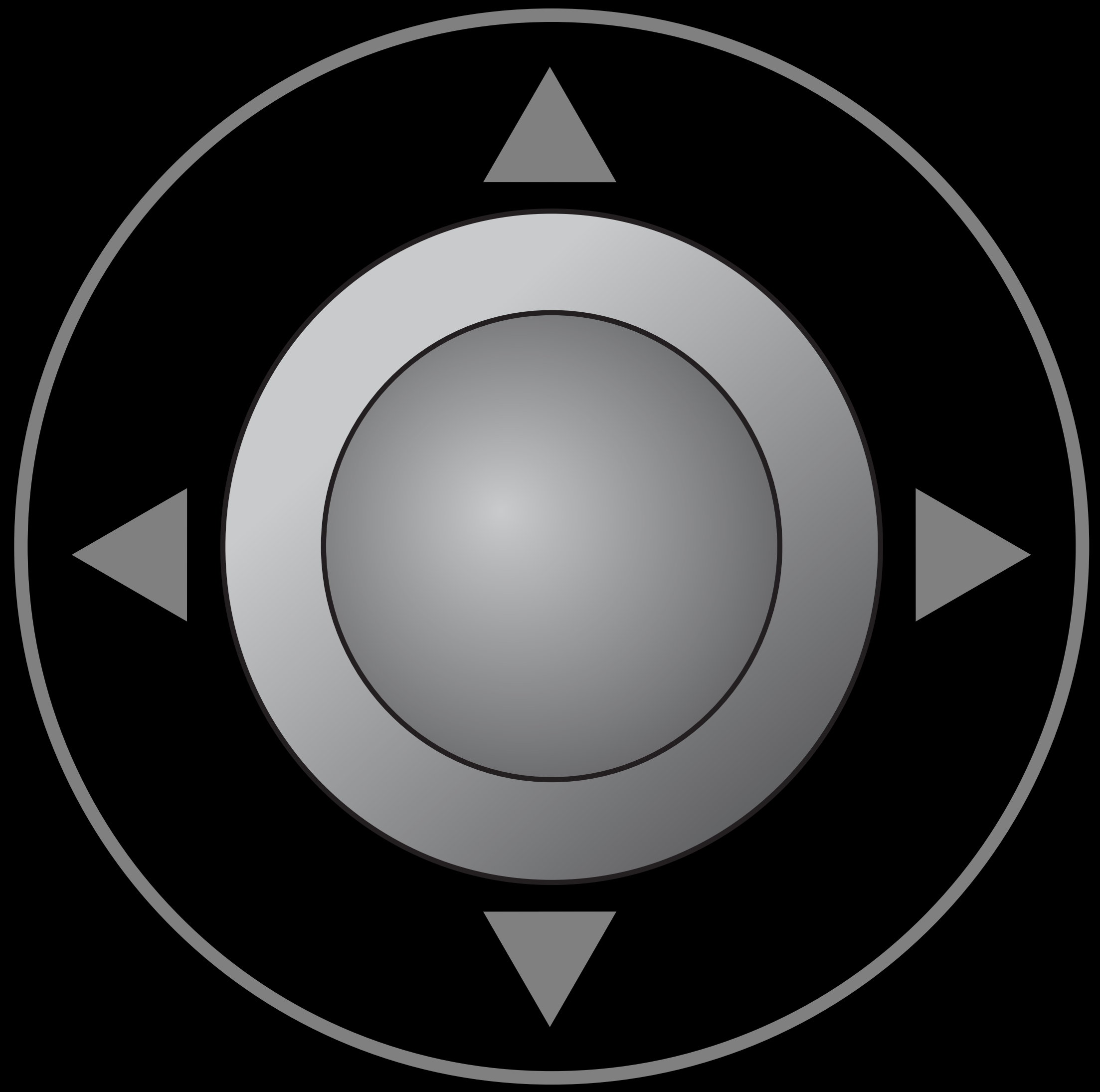
GCControllerDirectionPad

- Treated as four buttons

```
GCControllerButtonInput *up, *down, *left, *right;
```

- Or as two axes

```
GCControllerAxisInput *xAxis, *yAxis;
```



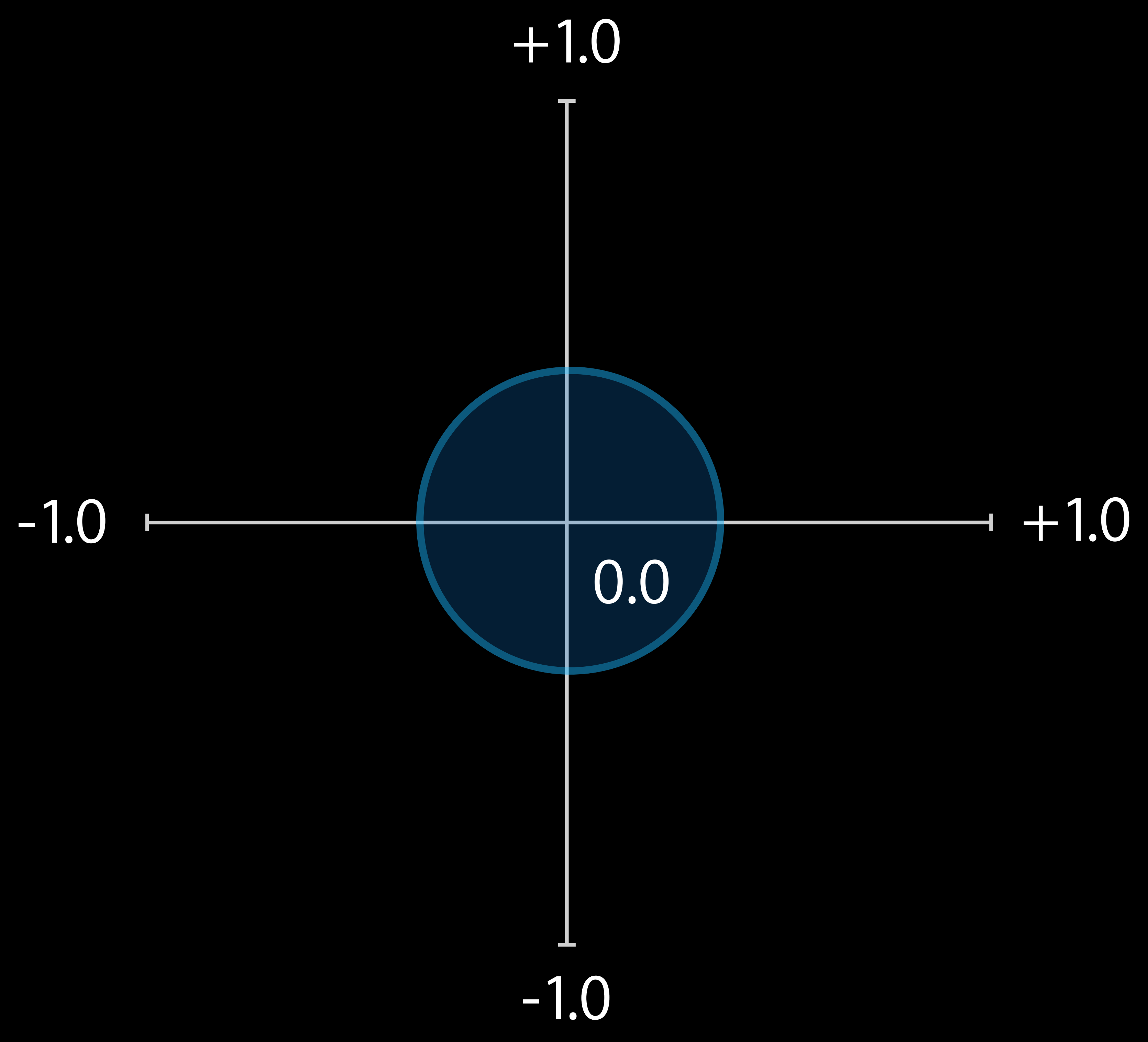
Element Types

GCControllerAxisInput

- Measures movement along a particular axis

```
float value;     // Normalized from −1.0 to 1.0; 0.0 is neutral
```

- Non-zero values indicate movement is outside neutral dead-zone



Reading Element Values

Three techniques

Reading Element Values

Three techniques

- Poll elements directly
    - For querying current value
    - Best for values changing over time (query each game loop)
    - e.g., thumbstick position

Reading Element Values

Three techniques

- Poll elements directly
  - For querying current value
  - Best for values changing over time (query each game loop)
  - e.g., thumbstick position
- Register a value change callback
  - For detecting changes of state
  - e.g., callback for trigger started being pulled

Reading Element Values

Three techniques

- Poll elements directly
  - For querying current value
  - Best for values changing over time (query each game loop)
  - e.g., thumbstick position
- Register a value change callback
  - For detecting changes of state
  - e.g., callback for trigger started being pulled
- Take snapshot of entire controller state
  - For capturing all elements simultaneously
  - e.g., button combos, input recording

Reading Element Values

Polling

- Typically done each game loop iteration
- All elements can be accessed directly via current profile

```
// face button
self.myController.gamepad.buttonY.pressed;
self.myController.gamepad.buttonY.value;


// thumbstick
self.myController.extendedGamepad.leftThumbstick.yAxis.value;
self.myController.extendedGamepad.leftThumbstick.up.value;
self.myController.extendedGamepad.leftThumbstick.up.pressed;
```

Reading Element Values

Polling

- Typically done each game loop iteration
- All elements can be accessed directly via current profile

```
// face button
self.myController.gamepad.buttonY.pressed;
self.myController.gamepad.buttonY.value;
```

```
// thumbstick
self.myController.extendedGamepad.leftThumbstick.yAxis.value;
self.myController.extendedGamepad.leftThumbstick.up.value;
self.myController.extendedGamepad.leftThumbstick.up.pressed;
```

Reading Element Values

Polling

- Typically done each game loop iteration
- All elements can be accessed directly via current profile

```
// face button
self.myController.gamepad.buttonY.pressed;
self.myController.gamepad.buttonY.value;
```

```
// thumbstick
self.myController.extendedGamepad.leftThumbstick.yAxis.value;
self.myController.extendedGamepad.leftThumbstick.up.value;
self.myController.extendedGamepad.leftThumbstick.up.pressed;
```

Reading Element Values

Polling example

```
-(void)update
{
    // Using Extended Gamepad controller
    GCExtendedGamePad *profile = self.myController.extendedGamepad;

    // Take actions for triggers
    if (profile.rightTrigger.isPressed)
        [self fireLasers];
    if (profile.leftTrigger.isPressed)
        [self launchMissiles];

    // Apply thrust based on Y value of thumbstick
    [self applyThrust: profile.leftThumbstick.yAxis.value];
}
```

Reading Element Values

Polling example

```
-(void)update
{
    // Using Extended Gamepad controller
    GCExtendedGamePad *profile = self.myController.extendedGamepad;

    // Take actions for triggers
    if (profile.rightTrigger.isPressed)
        [self fireLasers];
    if (profile.leftTrigger.isPressed)
        [self launchMissiles];

    // Apply thrust based on Y value of thumbstick
    [self applyThrust: profile.leftThumbstick.yAxis.value];
}
```

Reading Element Values

Polling example

```
-(void)update
{
    // Using Extended Gamepad controller
    GCExtendedGamePad *profile = self.myController.extendedGamepad;

    // Take actions for triggers
    if (profile.rightTrigger.isPressed)
        [self fireLasers];
    if (profile.leftTrigger.isPressed)
        [self launchMissiles];

    // Apply thrust based on Y value of thumbstick
    [self applyThrust: profile.leftThumbstick.yAxis.value];
}
```

Reading Element Values

Polling example

```
-(void)update
{
    // Using Extended Gamepad controller
    GCExtendedGamePad *profile = self.myController.extendedGamepad;

    // Take actions for triggers
    if (profile.rightTrigger.isPressed)
        [self fireLasers];
    if (profile.leftTrigger.isPressed)
        [self launchMissiles];

    // Apply thrust based on Y value of thumbstick
    [self applyThrust: profile.leftThumbstick.yAxis.value];
}
```

Reading Element Values

Value change handler

- Register a block as a change handler
- How it works:
  - Framework updates profile(s) on main thread at 60Hz
  - When updating, any handlers will be called
- Handlers can be registered on:
  - Specific element(s)
  - Collections (D-pad, axis)
  - An entire profile

Value Change Handler

Example 1

```
-(void)setupHandlers
{
    // Using extended gamepad profile
    GCExtendedGamepad *profile = self.myController.extendedGamepad;

    // Set up callback for right trigger
    profile.rightTrigger.valueChangedHandler =
        ^(GCControllerButtonInput *button, float value, BOOL pressed)
    {
        // Take action if pressed
        if (pressed)
            [self fireLasers];
    };
}
```

Value Change Handler

Example 1

```
-(void)setupHandlers
{
    // Using extended gamepad profile
    GCExtendedGamepad *profile = self.myController.extendedGamepad;

    // Set up callback for right trigger
    profile.rightTrigger.valueChangedHandler =
        ^(GCControllerButtonInput *button, float value, BOOL pressed)
    {
        // Take action if pressed
        if (pressed)
            [self fireLasers];
    };
}
```

Value Change Handler

Example 1

```
-(void)setupHandlers
{
    // Using extended gamepad profile
    GCExtendedGamepad *profile = self.myController.extendedGamepad;

    // Set up callback for right trigger
    profile.rightTrigger.valueChangedHandler =
        ^(GCControllerButtonInput *button, float value, BOOL pressed)
    {
        // Take action if pressed
        if (pressed)
            [self fireLasers];
    };
}
```

Value Change Handler

Example 2: Shared handlers

```
// Using standard gamepad profile
GCGamepad *profile = self.myController.gamepad;

// Shared handler
void (^myFaceButtonsHandler)(GCControllerButtonInput *, float, BOOL) =
    ^(GCControllerButtonInput *button, float value, BOOL pressed)
{
    // Face button pressed
    [self dismissUI];
};

// “Press any face button to continue”
profile.buttonA.valueChangedHandler = myFaceButtonsHandler;
profile.buttonB.valueChangedHandler = myFaceButtonsHandler;
profile.buttonX.valueChangedHandler = myFaceButtonsHandler;
profile.buttonY.valueChangedHandler = myFaceButtonsHandler;
```

Value Change Handler

Example 2: Shared handlers

```
// Using standard gamepad profile
GCGamepad *profile = self.myController.gamepad;

// Shared handler
void (^myFaceButtonsHandler)(GCControllerButtonInput *, float, BOOL) =
    ^(GCControllerButtonInput *button, float value, BOOL pressed)
{
    // Face button pressed
    [self dismissUI];
};

// “Press any face button to continue”
profile.buttonA.valueChangedHandler = myFaceButtonsHandler;
profile.buttonB.valueChangedHandler = myFaceButtonsHandler;
profile.buttonX.valueChangedHandler = myFaceButtonsHandler;
profile.buttonY.valueChangedHandler = myFaceButtonsHandler;
```

Value Change Handler

Example 2: Shared handlers

```
// Using standard gamepad profile
GCGamepad *profile = self.myController.gamepad;

// Shared handler
void (^myFaceButtonsHandler)(GCControllerButtonInput *, float, BOOL) =
    ^(GCControllerButtonInput *button, float value, BOOL pressed)
{
    // Face button pressed
    [self dismissUI];
};

// “Press any face button to continue”
profile.buttonA.valueChangedHandler = myFaceButtonsHandler;
profile.buttonB.valueChangedHandler = myFaceButtonsHandler;
profile.buttonX.valueChangedHandler = myFaceButtonsHandler;
profile.buttonY.valueChangedHandler = myFaceButtonsHandler;
```

Value Change Handler

Example 2: Shared handlers

```
// Using standard gamepad profile
GCGamepad *profile = self.myController.gamepad;

// Shared handler
void (^myFaceButtonsHandler)(GCControllerButtonInput *, float, BOOL) =
    ^(GCControllerButtonInput *button, float value, BOOL pressed)
{
    // Face button pressed
    [self dismissUI];
};

// “Press any face button to continue”
profile.buttonA.valueChangedHandler = myFaceButtonsHandler;
profile.buttonB.valueChangedHandler = myFaceButtonsHandler;
profile.buttonX.valueChangedHandler = myFaceButtonsHandler;
profile.buttonY.valueChangedHandler = myFaceButtonsHandler;
```

Value Change Handler

Example 3: Element Collections

```
// Using extended gamepad profile
GCExtendedGamepad *profile = self.myController.extendedGamepad;

// “Move right thumbstick to look around”
profile.rightThumbstick.valueChangedHandler =
    ^(GCControllerDirectionPad *dpad, float xValue, float yValue)
{
    // Right thumbstick position changed
    NSLog(@"Right thumbstick value changed: (%f, %f)", xValue, yValue);
};
```

Value Change Handler

Example 3: Element Collections

```
// Using extended gamepad profile
GCExtendedGamepad *profile = self.myController.extendedGamepad;
```

```
// “Move right thumbstick to look around”
profile.rightThumbstick.valueChangedHandler =
    ^(GCControllerDirectionPad *dpad, float xValue, float yValue)
{
    // Right thumbstick position changed
    NSLog(@"Right thumbstick value changed: (%f, %f)", xValue, yValue);
};
```

Value Change Handler

Example 3: Element Collections

```
// Using extended gamepad profile
GCExtendedGamepad *profile = self.myController.extendedGamepad;
```

```
// “Move right thumbstick to look around”
profile.rightThumbstick.valueChangedHandler =
    ^(GCControllerDirectionPad *dpad, float xValue, float yValue)
{
    // Right thumbstick position changed
    NSLog(@"Right thumbstick value changed: (%f, %f)", xValue, yValue);
};
```

Value Change Handler

Hierarchy of precedence

- Value change handlers called in hierarchical order
- From individual elements to profile
- e.g., order of handler callbacks when D-pad pressed:
    - Four D-pad buttons
    - Two D-pad axes
    - D-pad itself
    - Controller profile

Reading Element Values

Snapshots

- Captures all elements in profile simultaneously
- Allows serialization of controller state
    - Pack to and from NSData
    - Mutable
- Use in conjunction with polling or value change handlers
- Example usage:
    - Input recording and replay
    - Send over network
    - Debugging

Snapshot

Example 1: Capture snapshot

```
- (BOOL)writeSnapshotToFile:(NSString *)filePath
{
    // Grab snapshot
    GCGamepadSnapshot *snapshot = [self.myController.gamepad saveSnapshot];

    // NSData representation of snapshot
    NSData *snapshotData = snapshot.snapshotData;

    // Save data to file
    return [snapshotData writeToFile:filePath atomically:YES];
}
```

Snapshot

Example 1: Capture snapshot

```
- (BOOL)writeSnapshotToFile:(NSString *)filePath
{
    // Grab snapshot
    GCGamepadSnapshot *snapshot = [self.myController.gamepad saveSnapshot];

    // NSData representation of snapshot
    NSData *snapshotData = snapshot.snapshotData;

    // Save data to file
    return [snapshotData writeToFile:filePath atomically:YES];
}
```

Snapshot

Example 1: Capture snapshot

```
- (BOOL)writeSnapshotToFile:(NSString *)filePath
{
    // Grab snapshot
    GCGamepadSnapshot *snapshot = [self.myController.gamepad saveSnapshot];

    // NSData representation of snapshot
    NSData *snapshotData = snapshot.snapshotData;

    // Save data to file
    return [snapshotData writeToFile:filePath atomically:YES];
}
```

Snapshot

Example 1: Capture snapshot

```
- (BOOL)writeSnapshotToFile:(NSString *)filePath
{
    // Grab snapshot
    GCGamepadSnapshot *snapshot = [self.myController.gamepad saveSnapshot];

    // NSData representation of snapshot
    NSData *snapshotData = snapshot.snapshotData;

    // Save data to file
    return [snapshotData writeToFile:filePath atomically:YES];
}
```

Snapshot

Example 2: Retrieve snapshot

```
- (GCGamepadSnapshot *)snapshotFromFile:(NSString *)filePath
{
    // Read data from file
    NSData *data = (NSData *)[NSData dataWithContentsOfFile:filePath];

    // Init snapshot
    GCGamepadSnapshot *snapshot =
        [[GCGamepadSnapshot alloc] initWithSnapshotData:data];

    return snapshot;
}
```

Snapshot

Example 2: Retrieve snapshot

```
- (GCGamepadSnapshot *)snapshotFromFile:(NSString *)filePath
{
    // Read data from file
    NSData *data = (NSData *)[NSData dataWithContentsOfFile:filePath];

    // Init snapshot
    GCGamepadSnapshot *snapshot =
        [[GCGamepadSnapshot alloc] initWithSnapshotData:data];

    return snapshot;
}
```

Snapshot

Example 2: Retrieve snapshot

```
- (GCGamepadSnapshot *)snapshotFromFile:(NSString *)filePath
{
    // Read data from file
    NSData *data = (NSData *)[NSData dataWithContentsOfFile:filePath];

    // Init snapshot
    GCGamepadSnapshot *snapshot =
        [[GCGamepadSnapshot alloc] initWithSnapshotData:data];

    return snapshot;
}
```

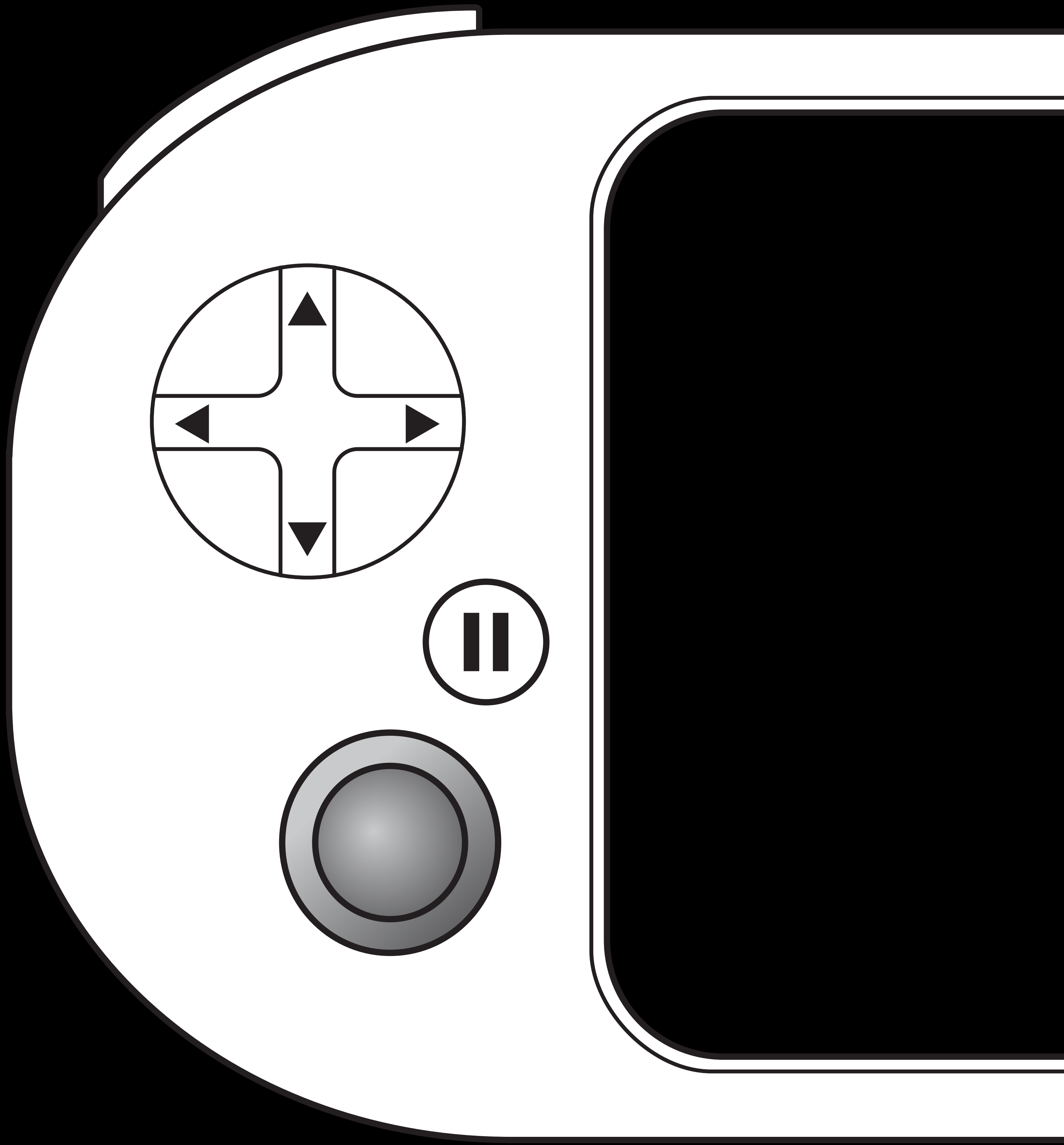
Demo

Additional Controls

Additional Controls

Pause button

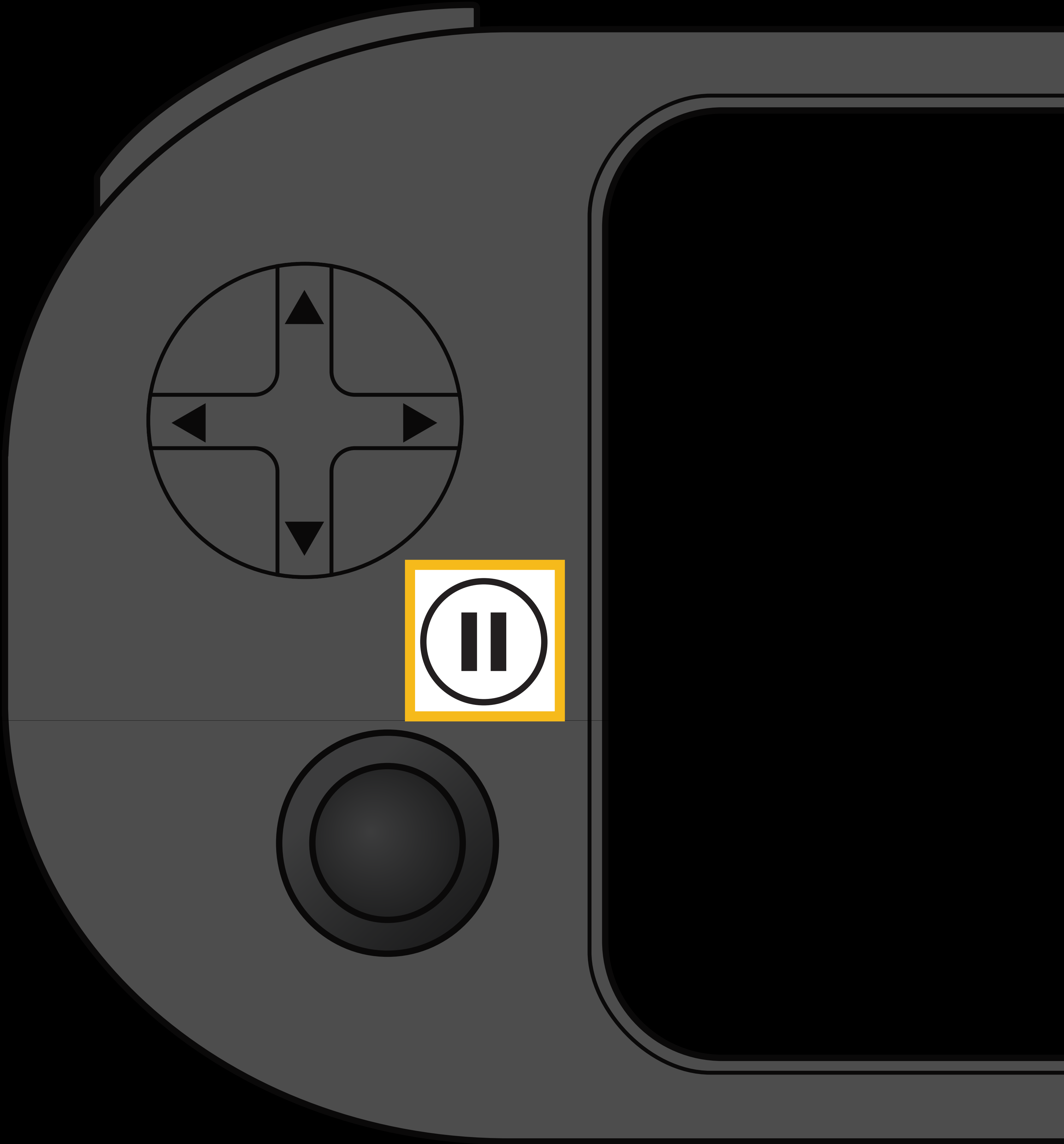
- Every controller includes Pause button
- Handling required
  - If game supports controllers
- Treat as a toggle
  - Active ➔ Pause
  - Paused ➔ Active
- Consider UI state



Additional Controls

Pause button

- Every controller includes Pause button
- Handling required
  - If game supports controllers
- Treat as a toggle
  - Active ➡ Pause
  - Paused ➡ Active
- Consider UI state



Additional Controls

Pause button example

```
- (void)setupControllers
{

    // ...

    // Add Pause Handler
    self.myController.controllerPausedHandler = ^(GCController *controller)
    {
        // Pause button pressed
        [self togglePauseResumeState];
    }
}
```

Additional Controls

Pause button example

```
- (void)setupControllers
{

    // ...

    // Add Pause Handler
    self.myController.controllerPausedHandler = ^(GCController *controller)
    {
        // Pause button pressed
        [self togglePauseResumeState];
    }
}
```

Additional Controls

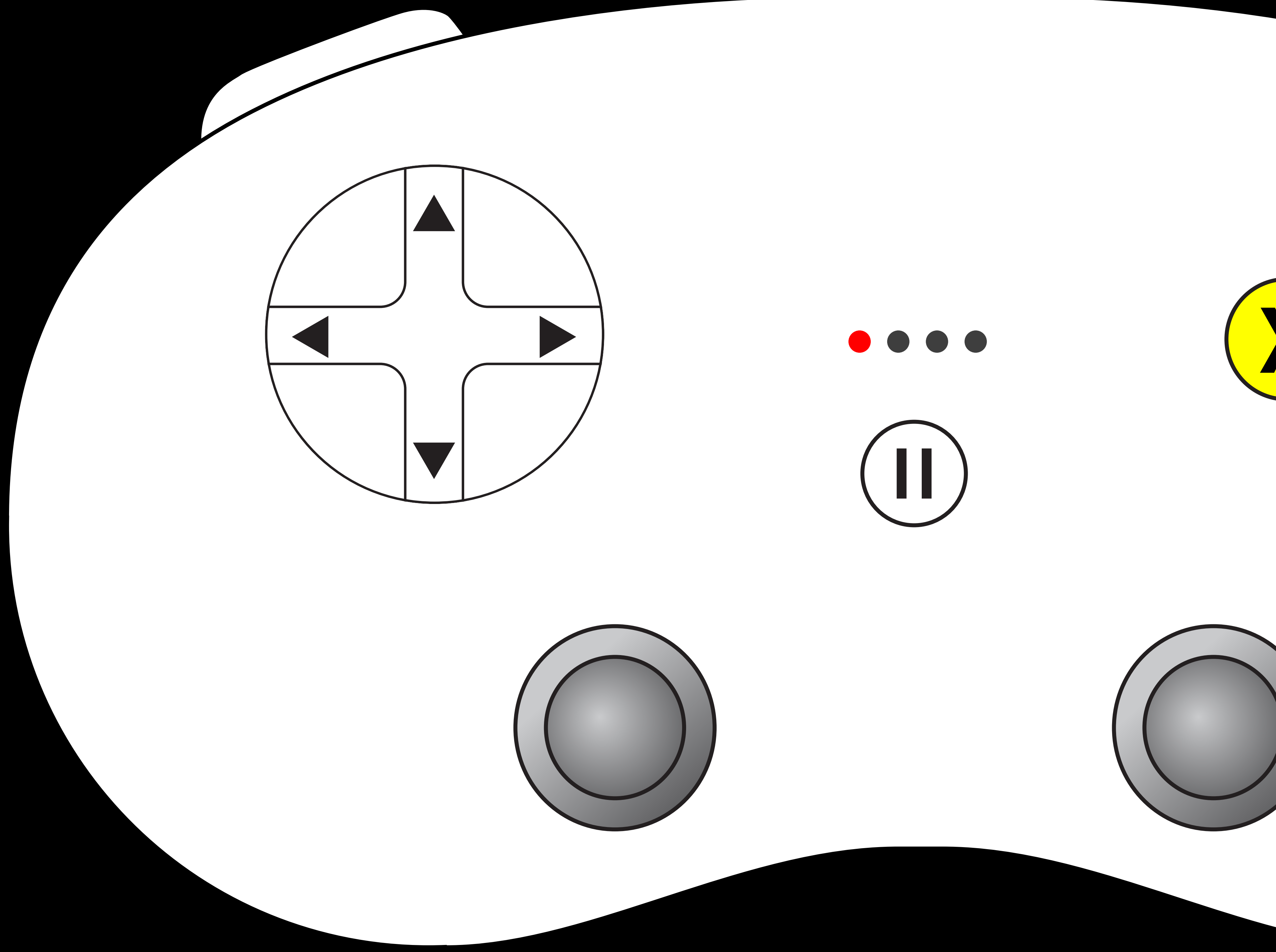
Pause button example

```
- (void)setupControllers
{

    // ...

    // Add Pause Handler
    self.myController.controllerPausedHandler = ^(GCController *controller)
    {
        // Pause button pressed
        [self togglePauseResumeState];
    }
}
```

Additional Controls

Player indicator LEDs

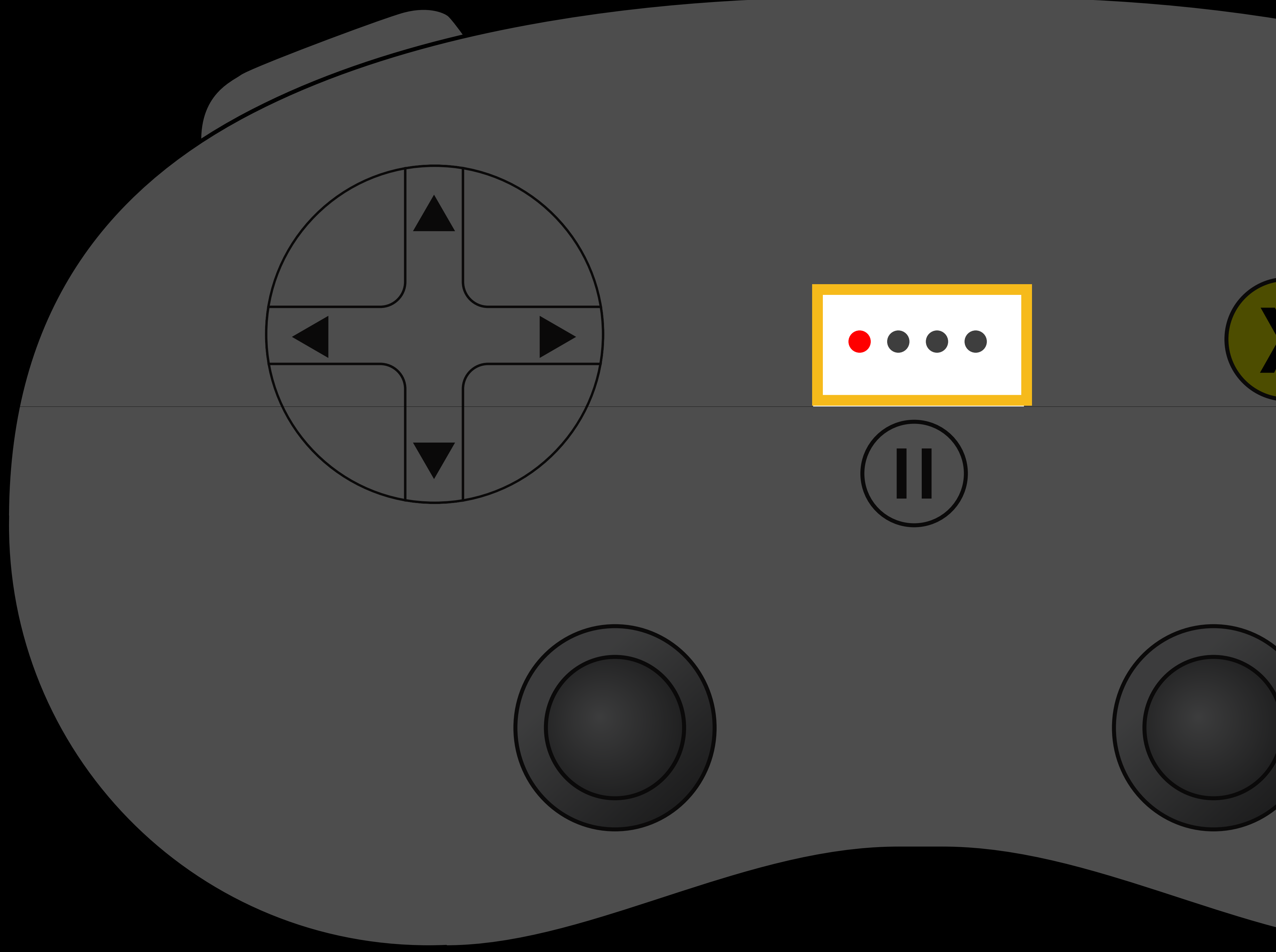
- Controllers may include player indicators
  - Four LEDs
  - API to set/get
  - Persistent
- Always set LEDs
  - For controllers being used
  - Player index



Additional Controls

Player indicator LEDs

- Controllers may include player indicators
  - Four LEDs
  - API to set/get
  - Persistent
- Always set LEDs
  - For controllers being used
  - Player index



Additional Controls

Player indicator example

- Single-controller game

```
// If unset, illuminate first LED
if (self.myController.playerIndex == GCControllerPlayerIndexUnset)
{
    self.myController.playerIndex = 0;    // Zero-based index
}
```

- Multiple-controller game

```
// If any are unset, present UI to assign players to controllers
if ((self.myControllers[0].playerIndex == GCControllerPlayerIndexUnset) ||
    (self.myControllers[1].playerIndex == GCControllerPlayerIndexUnset))
{
    [self launchControllerPicker];
}
```

Additional Controls

Player indicator example

- Single-controller game

```
// If unset, illuminate first LED
if (self.myController.playerIndex == GCControllerPlayerIndexUnset)
{
    self.myController.playerIndex = 0;    // Zero-based index
}
```

- Multiple-controller game

```
// If any are unset, present UI to assign players to controllers
if ((self.myControllers[0].playerIndex == GCControllerPlayerIndexUnset) ||
    (self.myControllers[1].playerIndex == GCControllerPlayerIndexUnset))
{
    [self launchControllerPicker];
}
```

Additional Controls

Player indicator example

- Single-controller game

```
// If unset, illuminate first LED
if (self.myController.playerIndex == GCControllerPlayerIndexUnset)
{
    self.myController.playerIndex = 0;    // Zero-based index
}
```

- Multiple-controller game

```
// If any are unset, present UI to assign players to controllers
if ((self.myControllers[0].playerIndex == GCControllerPlayerIndexUnset) ||
    (self.myControllers[1].playerIndex == GCControllerPlayerIndexUnset))
{
    [self launchControllerPicker];
}
```

Best Practices

Best Practices

Controllers enhance gameplay

- Games can not require a controller
  - Game controllers are optional
  - Games supporting controllers must also work without controllers
- Design first for native input
  - iOS: Touch, motion
  - OS X: Keyboard, mouse

Best Practices

Follow standard conventions

- Button A is action, button B is cancel
- When connected
    - Move to controller-based input
    - Illuminate player indicator
    - Remove on-screen control overlays
- When disconnecting
    - Pause gameplay
    - Return to regular controls

Best Practices

Think through your input

- Touch
    - Ideal for direct manipulation
    - i.e., taps, gestures, swipes
- Controller
    - Good for precise controls
    - i.e., moves, actions
- Touch and form-fitting controller together
    - Fine-grain controls plus direct manipulation

In the Lab

Developer Preview

Prototype Logitech Controller

More Information

Allan Schaffer
Graphics and Game Technologies Evangelist
aschaffer@apple.com

Apple Developer Forums
http://devforums.apple.com/

Developer Documentation
http://developer.apple.com/library/

Related Sessions

Introduction to Sprite Kit	Presidio Wednesday 11:30AM
Designing Games with Sprite Kit	Mission Wednesday 2:00PM
Advances in OpenGL ES	Mission Thursday 9:00AM

Labs

Game Controllers Lab	Graphics and Games Lab B **Tuesday 4:30PM**
Game Controllers Lab	Graphics and Games Lab B **Wednesday 9:00AM**
Sprite Kit Lab	Graphics and Games Lab B **Wednesday 3:15PM**

WWDC2013