Intro to Input Handling

Keyboard Input – Step 1

(get something working)

Handling Keyboard Input – Step 1

1. Define keyboard event handler
   - Based on `KeyEvent.key`, update object
2. Register handler for keyboard event
3. Profit!

Define Keyboard Event Handler

```
function onKeyDown(e) {
    //console.log(`${e.key} : ${e.code}`);
    if (e.key === 'a') {
        moveLeft(2);
    }
    else if (e.key === 'd') {
        moveRight(2);
    }
    else if (e.key === 'w') {
        moveUp(2);
    }
    else if (e.key === 's') {
        moveDown(2);
    }
}
```

Update Object

```
function moveLeft(distance) {
    center.x -= distance;
}

function moveRight(distance) {
    center.x += distance;
}

function moveUp(distance) {
    center.y -= distance;
}

function moveDown(distance) {
    center.y += distance;
}
```

Register Handler for Keyboard Event

```
window.addEventListener('keydown', onKeyDown);
```

Is this good enough?

Is this good enough?

(not yet, but getting there)

What are the Issues?

- Update is asynchronous to our game loop...we aren’t in control
- Update based on how often the event is fired by the browser
- Update is fixed per event firing
- Weak design

Keyboard Input – Step 2

(update based on elapsed time)

What are the Issues?

- Update is asynchronous to our game loop...we aren’t in control
- Update based on how often the event is fired by the browser
- Update is fixed per event firing
- Weak design

Maintain Elapsed Time

```
let elapsedTime = 0;
let lastTimeStamp = performance.now();
```

```
function gameLoop(time) {

    elapsedTime = time - lastTimeStamp;
    lastTimeStamp = time;
    update();
    render();

    requestAnimationFrame(gameLoop);
};
```

Update Based on Elapsed Time

```
function onKeyDown(e) {
    if (e.key === 'a') {
        moveLeft(elapsedTime);
    }
    else if (e.key === 'd') {
        moveRight(elapsedTime);
    }
    else if (e.key === 'w') {
        moveUp(elapsedTime);
    }
    else if (e.key === 's') {
        moveDown(elapsedTime);
    }
}
```

Update Object Based on Elapsed Time

```
function moveLeft(elapsedTime) {
    center.x -= moveRate * elapsedTime;
}

function moveRight(distance) {
    center.x += moveRate * elapsedTime;
}

function moveUp(distance) {
    center.y -= moveRate * elapsedTime;
}

function moveDown(distance) {
    center.y += moveRate * elapsedTime;
}
```

Is this good enough?

Is this good enough?

(not yet, but getting there)

Keyboard Input – Step 3

(let’s take control)

What are the Issues?

- Update is asynchronous to our game loop...we aren't in control
- Update based on how often the event is fired by the browser
- ~~Update is fixed per event firing~~
- Weak design

Capturing the Input – An Input Manager

```
let input = (function() {
    function Keyboard() {
        let that = {
            keys : {}
        };
        function keyPress(e) {
            that.keys[e.key] = e.timeStamp;
        }
        function keyRelease(e) {
            delete that.keys[e.key];
        }
        window.addEventListener('keydown', keyPress);
        window.addEventListener('keyup', keyRelease);

        return that;
    }

    return {
        Keyboard : Keyboard
    };
}());
```

Create an Input Manager

```
let myInput = input.Keyboard();
```

Process the Input

```
function processInput(elapsedTime) {
    myObject.handleInput(myInput, elapsedTime);
}
```

```
function handleInput(input, elapsedTime) {
    if (input.keys.hasOwnProperty('a')) {
        moveLeft(elapsedTime);
    }
    if (input.keys.hasOwnProperty('d')) {
        moveRight(elapsedTime);
    }
    if (input.keys.hasOwnProperty('w')) {
        moveUp(elapsedTime);
    }
    if (input.keys.hasOwnProperty('s')) {
        moveDown(elapsedTime);
    }
};
```

Is this good enough?

Is this good enough?

(not yet, but getting there)

Keyboard Input – Step 4

(semantic separation)

What are the Issues?

- ~~Update is asynchronous to our game loop…we aren't in control~~
- ~~Update based on how often the event is fired by the browser~~
- ~~Update is fixed per event firing~~
- Weak design

Semantic Separation

1. Register to receive input events
2. Process registered input handlers

Input Manager – Allow Registering for Events

```
let that = {
    keys : {},
    handlers : {}
};


that.registerCommand = function(key, handler) {
    that.handlers[key] = handler;
};
```

Client – Register for Events

```
myKeyboard.registerCommand('a', myTexture.moveLeft);
myKeyboard.registerCommand('d', myTexture.moveRight);
myKeyboard.registerCommand('w', myTexture.moveUp);
myKeyboard.registerCommand('s', myTexture.moveDown);
```

```
that.rotateRight = function(elapsedTime) {
    spec.rotation += spec.rotateRate * (elapsedTime / 1000);
};
that.rotateLeft = function(elapsedTime) {
    spec.rotation -= spec.rotateRate * (elapsedTime / 1000);
};
that.moveLeft = function(elapsedTime) {
    spec.center.x -= spec.moveRate * (elapsedTime / 1000);
};
that.moveRight = function(elapsedTime) {
    spec.center.x += spec.moveRate * (elapsedTime / 1000);
};
```

Taking Control

1. Create something to capture and buffer the input
2. Using this thing, process the input at the desired time

Input Manager – Process Registered Handlers

```
that.update = function(elapsedTime) {
    for (let key in that.keys) {
        if (that.keys.hasOwnProperty(key)) {
            if (that.handlers[key]) {
                that.handlers[key](elapsedTime);
            }
        }
    }
};
```

Is this good enough?

Is this good enough?

(reasonable, but still room for improvement)

What are the Issues?

- ~~Update is asynchronous to our game loop...we aren't in control~~
- ~~Update based on how often the event is fired by the browser~~
- ~~Update is fixed per event firing~~
- ~~Weak design~~
- Ability to unregister from an event
- Multiple subscribers per event
- Input patterns
  - One time only
  - Repeat based on elapsed time (a firing rate)
    - currently fires every frame...remember, that's bad!
- Mouse and other input devices