BoardKey

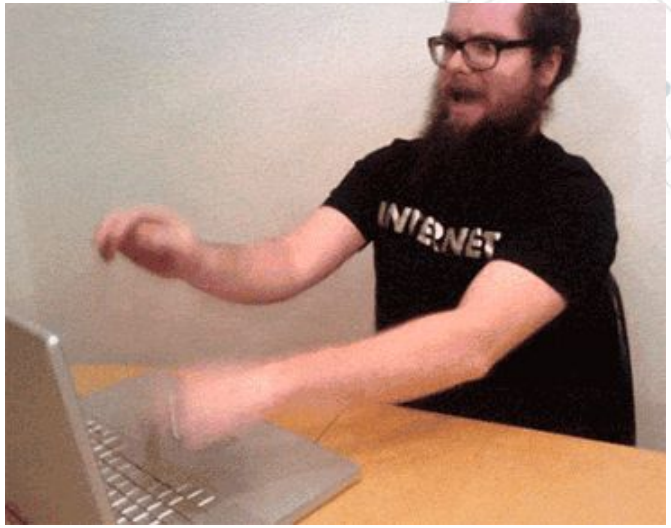
Team Bryson

Joel Mashita, Bryson Li, Andrew Gao,
Wilson Chang, Eric Oh

Motivation

- A few key presses away from
- 1) an important email
- 2) an essay due midnight

- Temporary fix
- Convenient
- Virtual Keyboard



A broken keyboard can be a real struggle

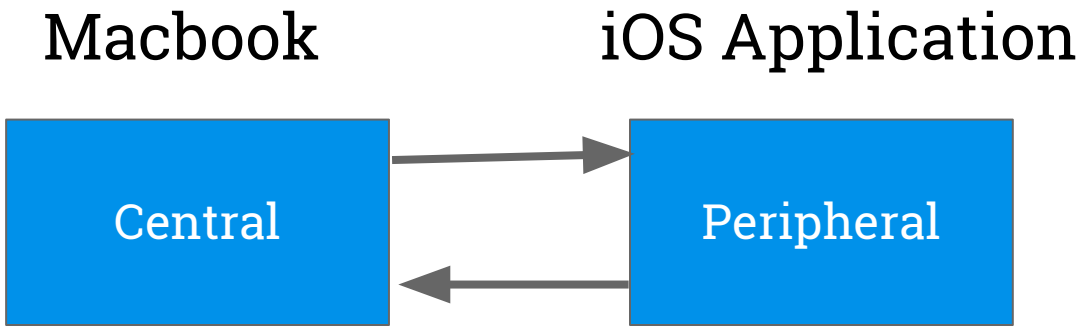
Project Goals

1. Establish Bluetooth connection (phone and laptop)
2. Create virtual keyboard UI for phone
3. Simulate keypress events on the laptop
4. Combine the two through Bluetooth

Why Bluetooth?

- Bluetooth is built-in in our phone and laptop.
- Good for short range wireless communication.
- Better for device to device connections.
- It has low power consumption (Bluetooth Smart)
- Bluetooth devices have the capability to automatically detect each other.

How we used Bluetooth



1. Peripheral broadcasts the service it provides
2. Central asks for the service
3. Peripheral sends the information

Experiments 1: Connection

Central

```
startBLECentral
centralManagerDidUpdateState
Scanning for peripherals
<CBPeripheral: 0x6000001081f0, identifier = 9A7056C2-2BB1-4D46-8B6E
state = disconnected>
Connected to servicenil
<CBService: 0x604000276f00, isPrimary = NO, UUID = AA40E3C9-D777-4E
<CBCharacteristic: 0x6000000aafe0, UUID = 5FF12413-BC42-4B51-ACBE-E
value = (null), notifying = NO>
5FF12413-BC42-4B51-ACBE-EEB5B305B468: properties contains .read
5FF12413-BC42-4B51-ACBE-EEB5B305B468: properties contains .notify
<CBCharacteristic: 0x6000000aab60, UUID = 6682C4F0-61EF-473F-9329-E
value = (null), notifying = NO>
6682C4F0-61EF-473F-9329-B2BE9875911D: properties contains .notify
```

Peripheral

```
Discoverable name : Awesome_M117
Discoverable service :
AA40E3C9-D777-4B2B-8A4F-9B78B1E8D605
peripheralManagerDidUpdateState
Create service...
Charac. read :
5FF12413-BC42-4B51-ACBE-EEB5B305B468
Charac. write :
6682C4F0-61EF-473F-9329-B2BE9875911D
peripheralManager didAdd service
service:
AA40E3C9-D777-4B2B-8A4F-9B78B1E8D605
Advertisement datas:
["kCBAdvDataServiceUUIDs": [AA40E3C9-D777
"Awesome_M117"]
Starting to advertise.
peripheralManagerDidStartAdvertising OK
```

Experiment 2: Key Presses

CGKeyCode / Virtual Key Code

CGEventSource, CGEvent, post/ "send" event

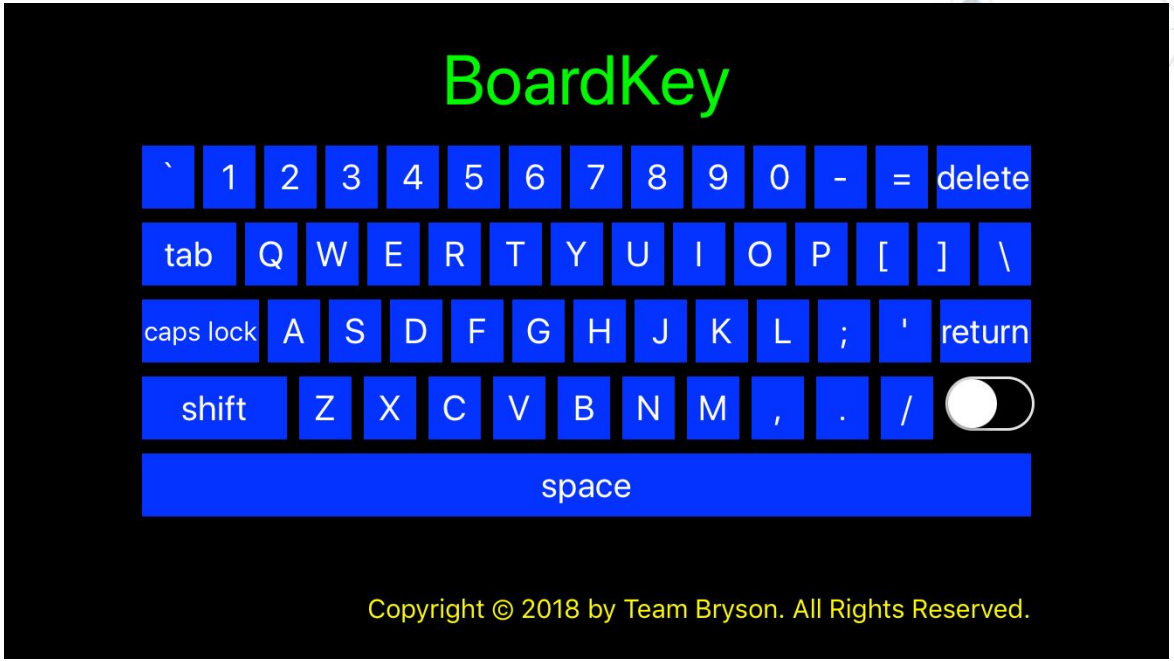
```
func executeKeypress (value: String)
{
    print("Executing Keypress")
    keyboardKeyDown(key: str2code(str: value))
}
```

```
func str2code(str: String) -> CGKeyCode {
    switch (str) {
    case ("A"): return 0x00
    case ("S"): return 0x01
    case ("D"): return 0x02
    case ("F"): return 0x03
    case ("H"): return 0x04
```

```
func keyboardKeyDown(key: CGKeyCode) {
    let source = CGEventSource(stateID: .hidSystemState)
    let event = CGEvent(keyboardEventSource: source, virt
    event?.post(tap: .cghidEventTap)
}
```

Experiment 3: Putting it all Together

```
B
Executing Keypress
Read Char
O
Executing Keypress
Read Char
A
Executing Keypress
Read Char
R
Executing Keypress
Read Char
D
Executing Keypress
Read Char
space
Executing Keypress
Read Char
K
Executing Keypress
Read Char
E
Executing Keypress
Read Char
Y
Executing Keypress
```



Demo:
Submit Essay

Results

The iOS app is able to connect with macOS via Bluetooth. Inputs on the iPhone convert to keystroke inputs on the Macbook. iOS display shows a virtual keyboard, in which each virtual key corresponds to its respective keystroke. Not all possible keystrokes are represented in the app. Certain inputs, like repeated inputs of the same key by holding it down, do not work as they might if used on the Macbook hardware.

Discussion

One deviation from the expected result is that holding down a key in the app does not correspond to repeated inputs on the computer. This is because with our current design, each key event is processed individually. The corresponding keystroke is not sent to the computer until the key-up event is recognized, following a key-down. This also doesn't allow for a Shift key to be implemented the same way it is on a regular keyboard. One way to achieve upper-case letters would be to have a toggle to an entire keyboard of upper case letters, as is present in current mobile keyboards.

Another desirable feature is to be able to scan bluetooth devices and connect to a selected one. Currently, we can only pair a single device to a single computer by hard-coding the connection into the app. A bluetooth scanning feature could be achieved if the computer could find and choose one device running the app to connect to.

Conclusion

Through this project, we discovered the challenges that come with designing a Bluetooth enabled app that connects to a computer. Though some features could be worked on independently, it was much more straightforward to work on the majority of the project linearly, as shown in the experiments. We were also able to find the limitations of the tools being used, including Bluetooth and Apple development capabilities.

Thank you.

Q & A