Nigel Gilbert

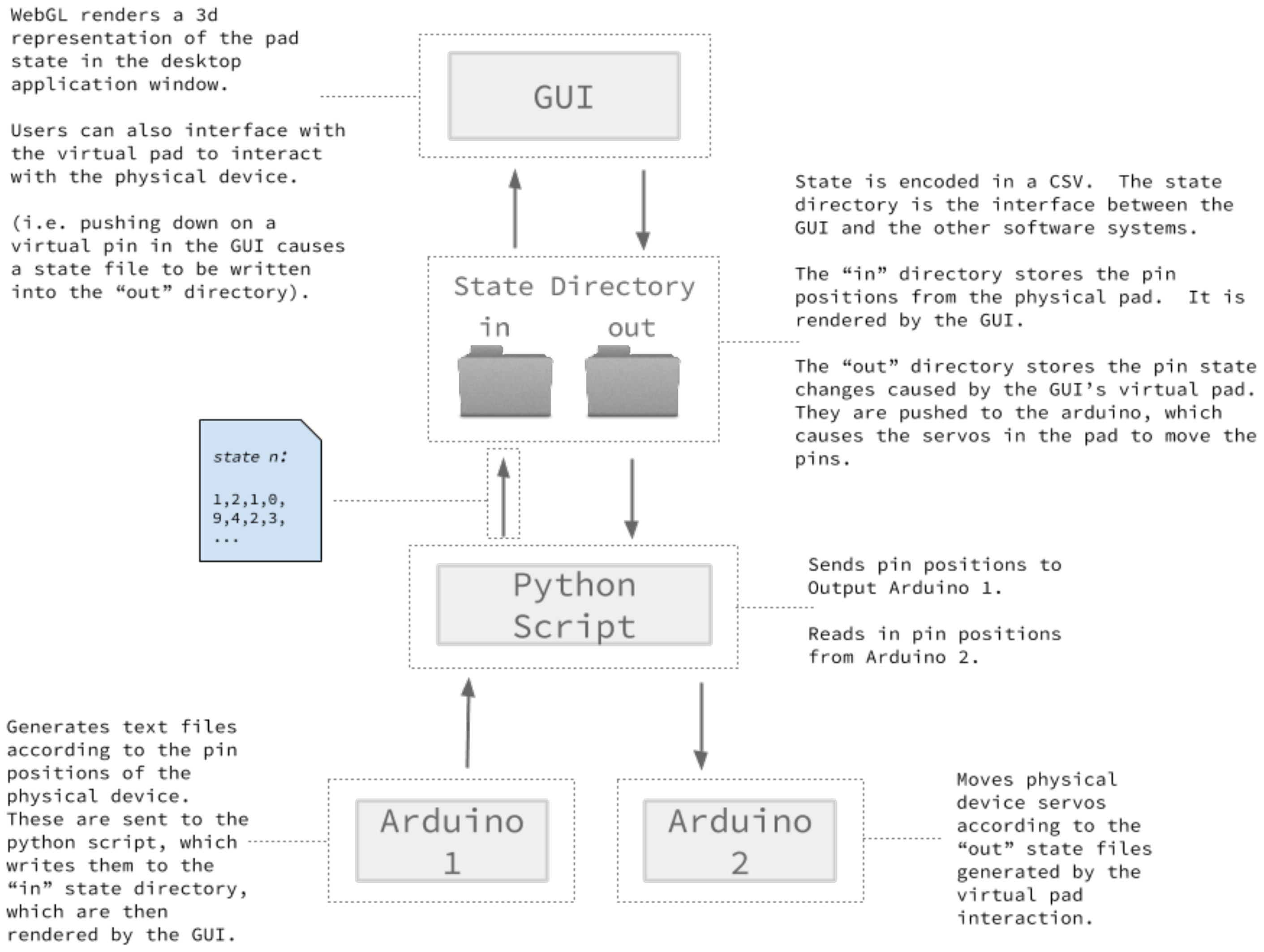
Senior Design Documentation

May 2nd, 2016

A. GUI Directory Overview

```
├── app/                - Directory of the application.
│   │
│   ├── app.html        - The html pg. that is rendered in the app’s chromium
│   │                     window.  It draws the Three.js scene and includes
│   │                     dependencies.
│   │
│   ├── board/          - Includes the js modules for the hexagonal grid.
│   │
│   ├── controllers/    - Click handlers for controls to the Three.js scene.
│   │
│   └── lib/            - Libraries for client app (Three.js and Tween.js).
│
├── constants.js        - Defines the events that are handled by the
│                         bidirectional communication channel (aka IPC in
│                         electron apps) between the client app and the main
│                         process.  The IPC uses Node’s EventEmitter API.
│
├── demos               - Directory of the scripts that modify or manage
│   │                     the application state.
│   │
│   └── demo.js         - The demo that we showed at the career fair.
│
├── main.js             - There are two threads in an Electron application:
│                         the GUI and the main process.  This is the entry
│                         into the main process.  It configures keybindings
│                         for the application and starts the demo and state
│                         watcher.
│
├── node_modules        - Node dependencies (i.e. Electron, React, etc.).
│
├── package.json        - Configuration for Node’s package manager.  Also
│                         defines various scripts for starting the app as well
│                         as development utilities.
│
├── sidebar/            - The src for the React sidebar.  This is bundled with
│                         webpack, which concatenates this directory into a
│                         single bundle that is written to the app directory.
│
├── state/              - The directory to which state is written to.
│                         Watch.js looks at this folder, and then sends new
│                         state files to be rendered by the GUI application.
│
└── watch.js            - Watches state and then sends JSON state files to the
                          application window using the MainWindow Electron
                          object.
```

B. Bi-directional State Pipeline



WebGL renders a 3d representation of the pad state in the desktop application window.

Users can also interface with the virtual pad to interact with the physical device.

(i.e. pushing down on a virtual pin in the GUI causes a state file to be written into the “out” directory).

State is encoded in a CSV. The state directory is the interface between the GUI and the other software systems.

The “in” directory stores the pin positions from the physical pad. It is rendered by the GUI.

The “out” directory stores the pin state changes caused by the GUI’s virtual pad. They are pushed to the arduino, which causes the servos in the pad to move the pins.

Sends pin positions to Output Arduino 1.

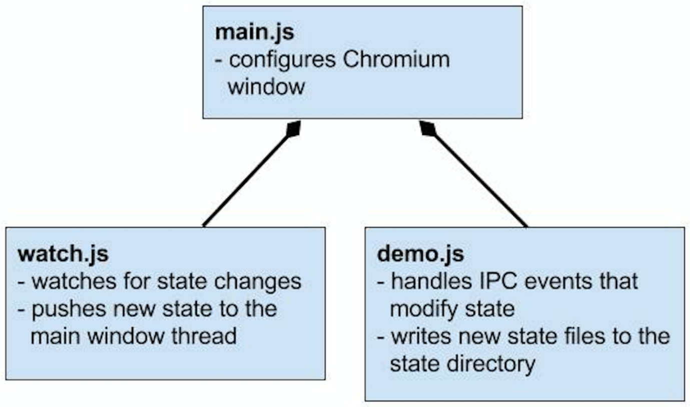
Reads in pin positions from Arduino 2.

Generates text files according to the pin positions of the physical device. These are sent to the python script, which writes them to the “in” state directory, which are then rendered by the GUI.

Moves physical device servos according to the “out” state files generated by the virtual pad interaction.

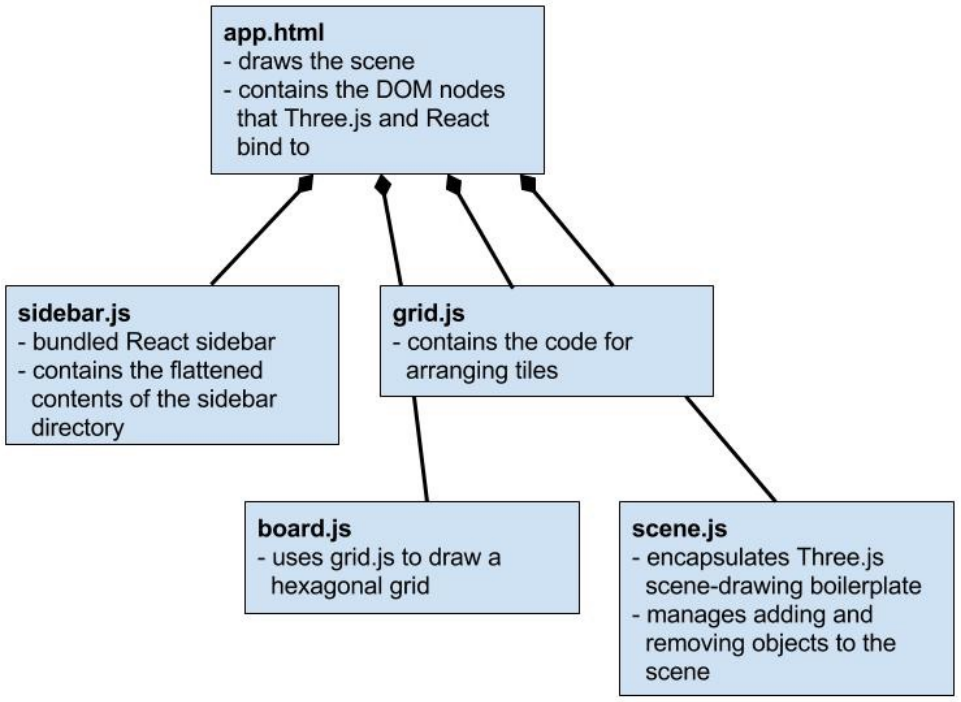
C. (abbreviated) UML diagrams

Main.js thread:



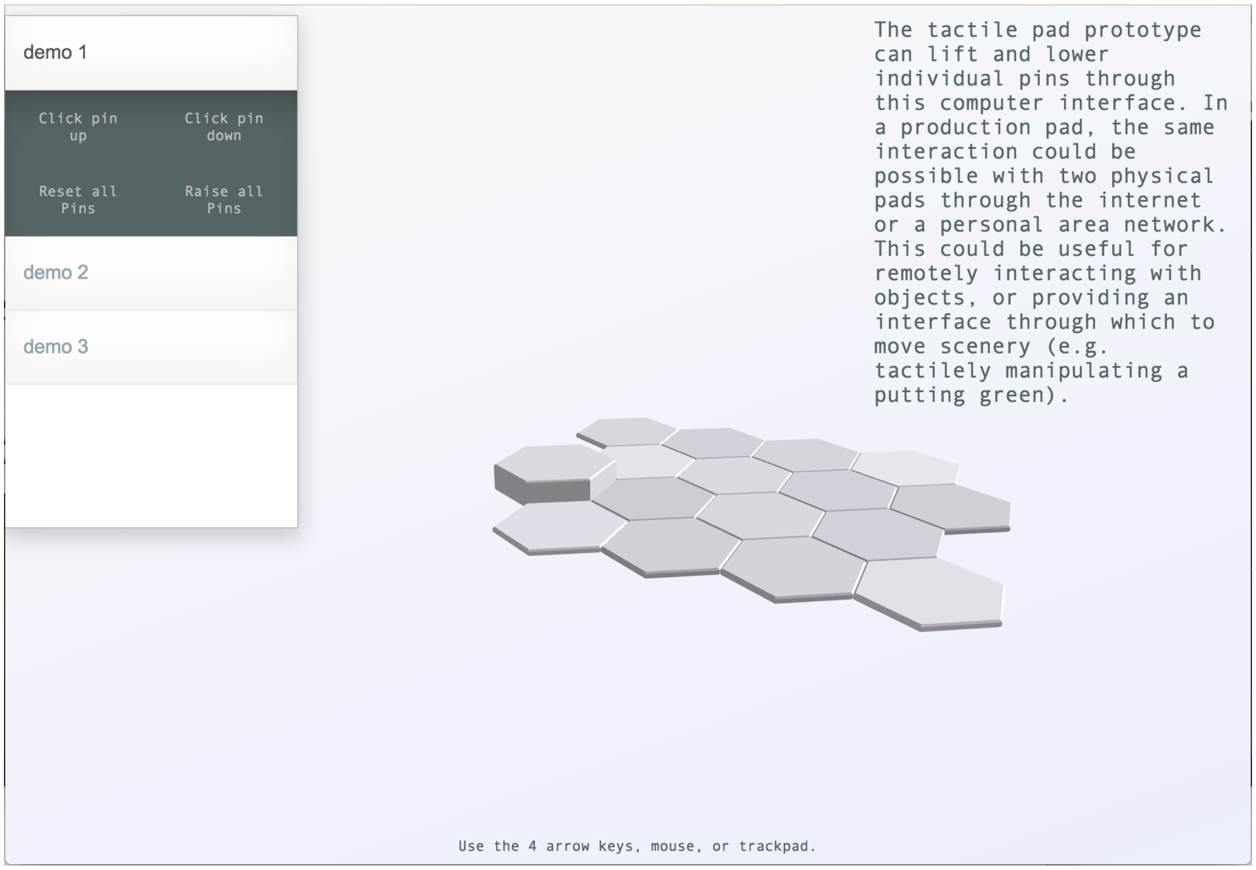
main.js
- configures Chromium window
watch.js
- watches for state changes
- pushes new state to the main window thread
demo.js
- handles IPC events that modify state
- writes new state files to the state directory

GUI thread:



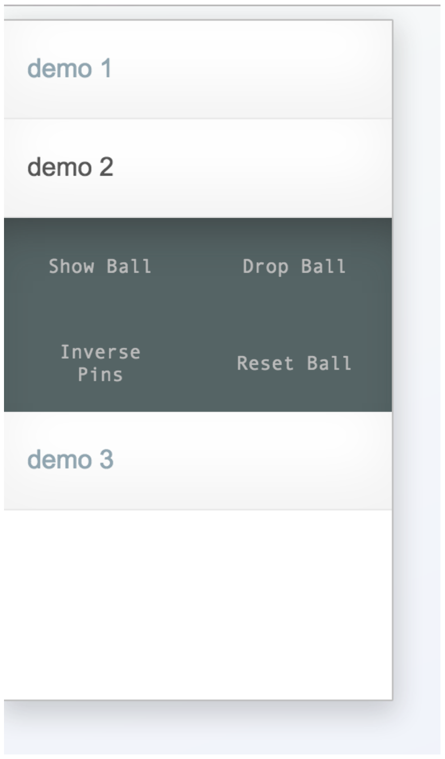
app.html
- draws the scene
- contains the DOM nodes that Three.js and React bind to
sidebar.js
- bundled React sidebar
- contains the flattened contents of the sidebar directory
grid.js
- contains the code for arranging tiles
board.js
- uses grid.js to draw a hexagonal grid
scene.js
- encapsulates Three.js scene-drawing boilerplate
- manages adding and removing objects to the scene

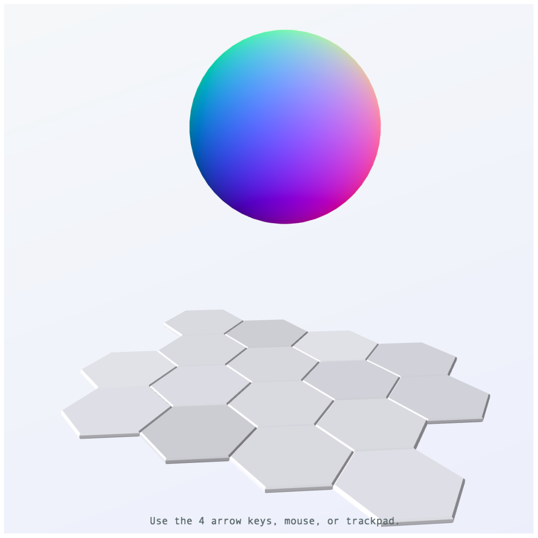
D. Screenshots



The tactile pad prototype can lift and lower individual pins through this computer interface. In a production pad, the same interaction could be possible with two physical pads through the internet or a personal area network. This could be useful for remotely interacting with objects, or providing an interface through which to move scenery (e.g. tactilely manipulating a putting green).

Use the 4 arrow keys, mouse, or trackpad.





E. Individual File Documentation

Note: I encourage anyone interested to open up the source code. I made a point of thoroughly commenting everything. The purpose and functionality of each files are commented clearly, and some are not specified here.

webpack.config.js:

Webpack is a module bundling and transpilation tool for Javascript. In this app, I used it to turn the ES2015 I used to program the React Sidebar into javascript that the Chromium window can run. This is a configuration file that specifies what operations should be done to transpile the source code (in this case, the “react-hot” loader is used). Check out https://webpack.github.io/docs/ for more information on how to use Webpack.

watch.js:

Watch handles sending state files to the app that is being rendered in the Chromium window. It uses a node package called Chokidar, which watches the state directory and executes a callback function whenever a new file is added. The callback reads the new file, and then sends it to the application window. There are 2 threads of execution in an electron app: the GUI window and the main thread. Communication between the main thread and the browser window is done through the “IPC”, or inter process communication event emitter. An event is a string descriptor and a payload, which can be anything. Events are handled with a callback, like so:

```
.on(“event”, function() {

     console.log(“hey an event just happened"

});
```

demo/demo.js:

This is the demo that was running during the senior design fair. It maintains the application state (a 2d array of numbers representing the pins elevation), and contains on-event handlers for the IPC that mutate the state. When clicks or user interactions occur in the GUI, the IPC sends an event (e.g. “reset”). This causes a state file to be written to the state directory, which then triggers the changes in the GUI and pad.