# Making games with Electron

Node.js, JEST, React and other magical words

# Who am I

Inga Pflaumer

Data analyst and Deployment specialist at SMG studio

Organiser of Sydney Unity Game Dev meetup

And Women In Game Development meetup

Corgi owner

# Do you want to make a game?

- Where to start
- What to look for
- Plan your architecture
- TDD your backed
- React your frontend
- Devops and Ship It!

# Where to start - board games

- Start with a simple board game you play. Why?
  - Logic is already there
  - Game balance is already there
  - Audience is already there

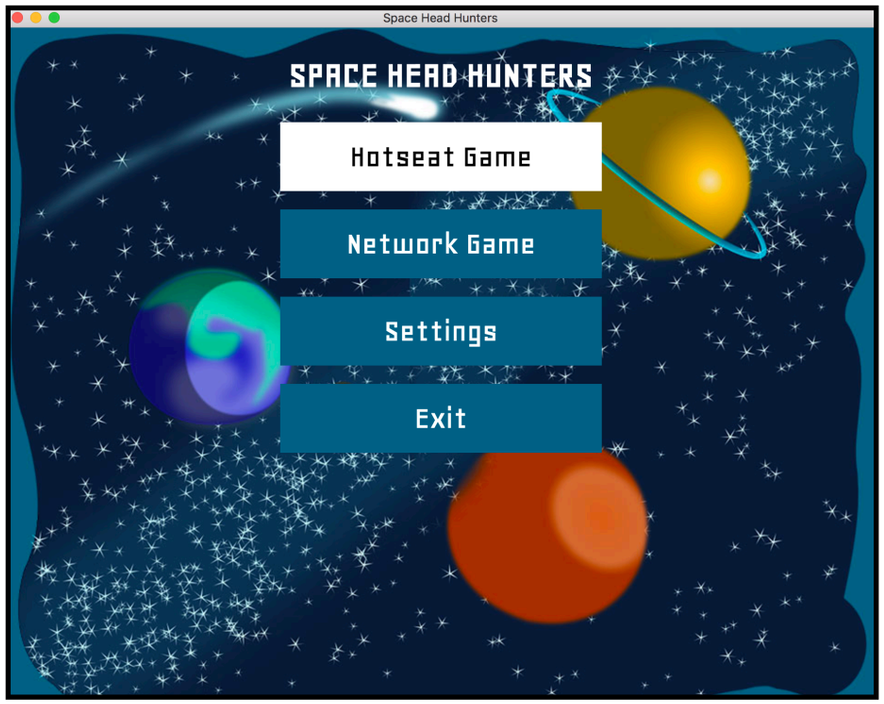
# Examples

- Space HeadHunters
  - Based on CROWS by Tyler Sigman

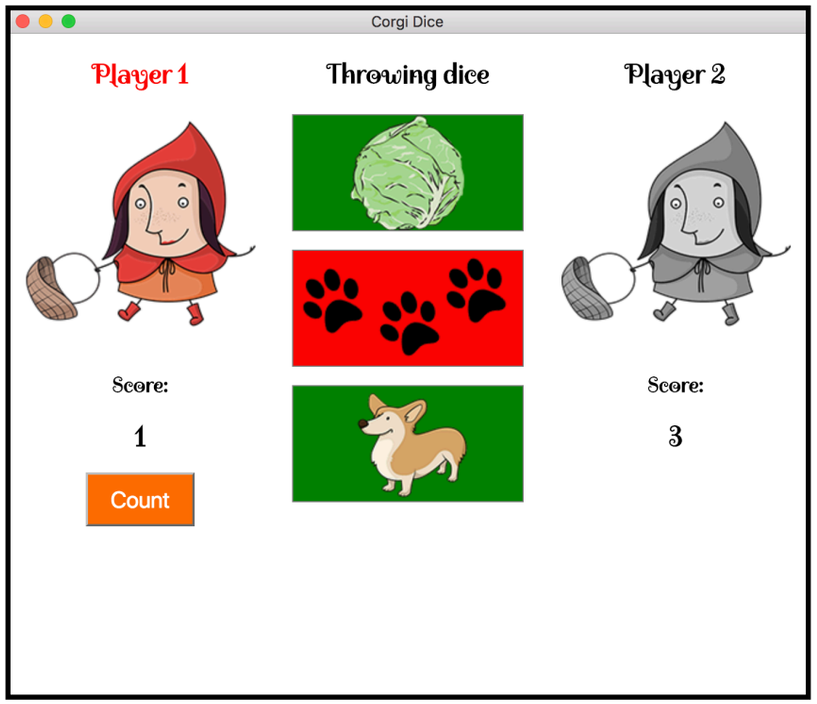

- Corgi Dice
  - Based on ZOMBIE DICE by Steve Jackson

# Examples

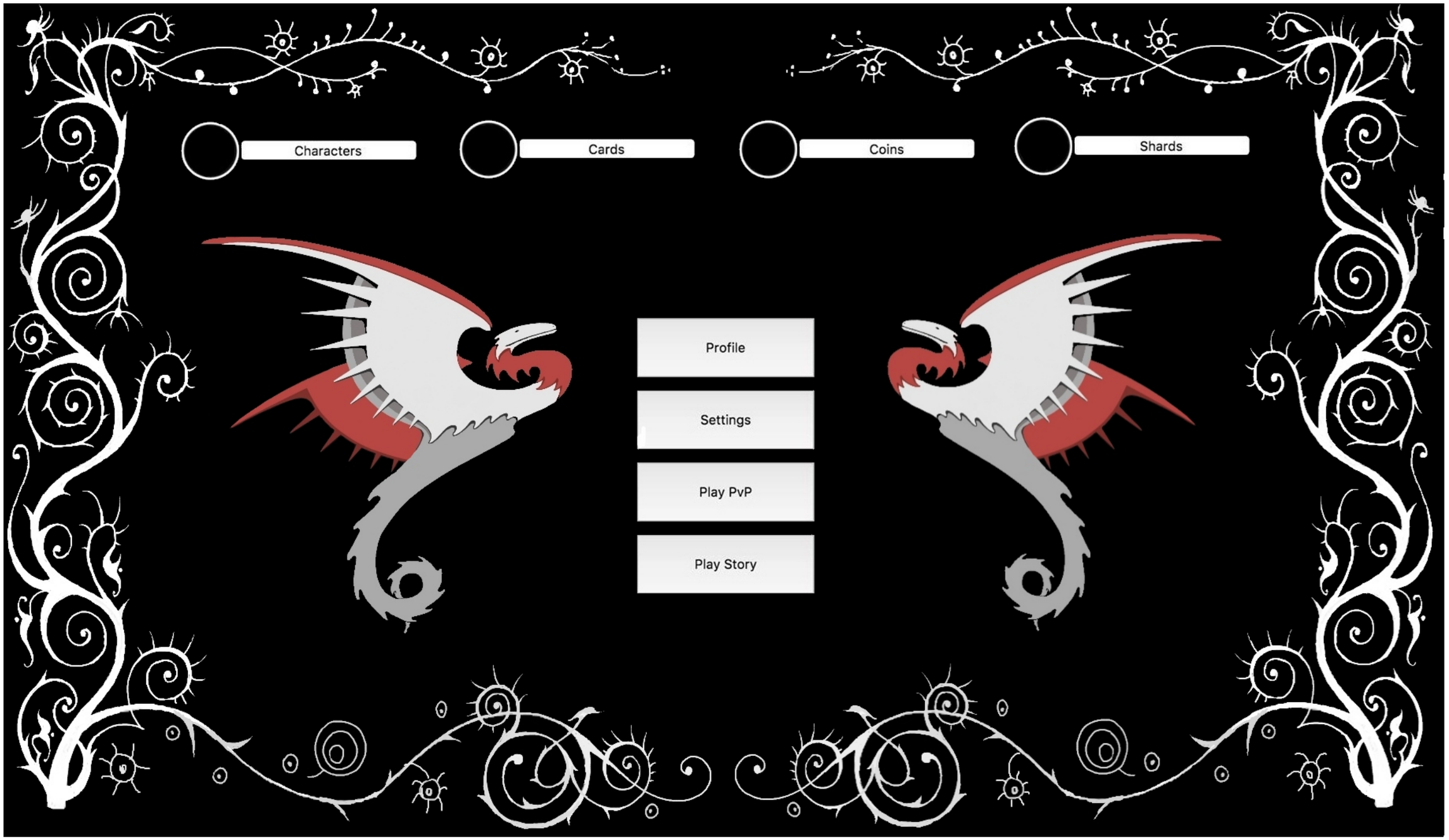
- Space HeadHunters
- https://github.com/Rukia3d/BITS_SpaceHeadHunters




- Corgi Dice
- https://github.com/Rukia3d/CorgiDice



# In Progress

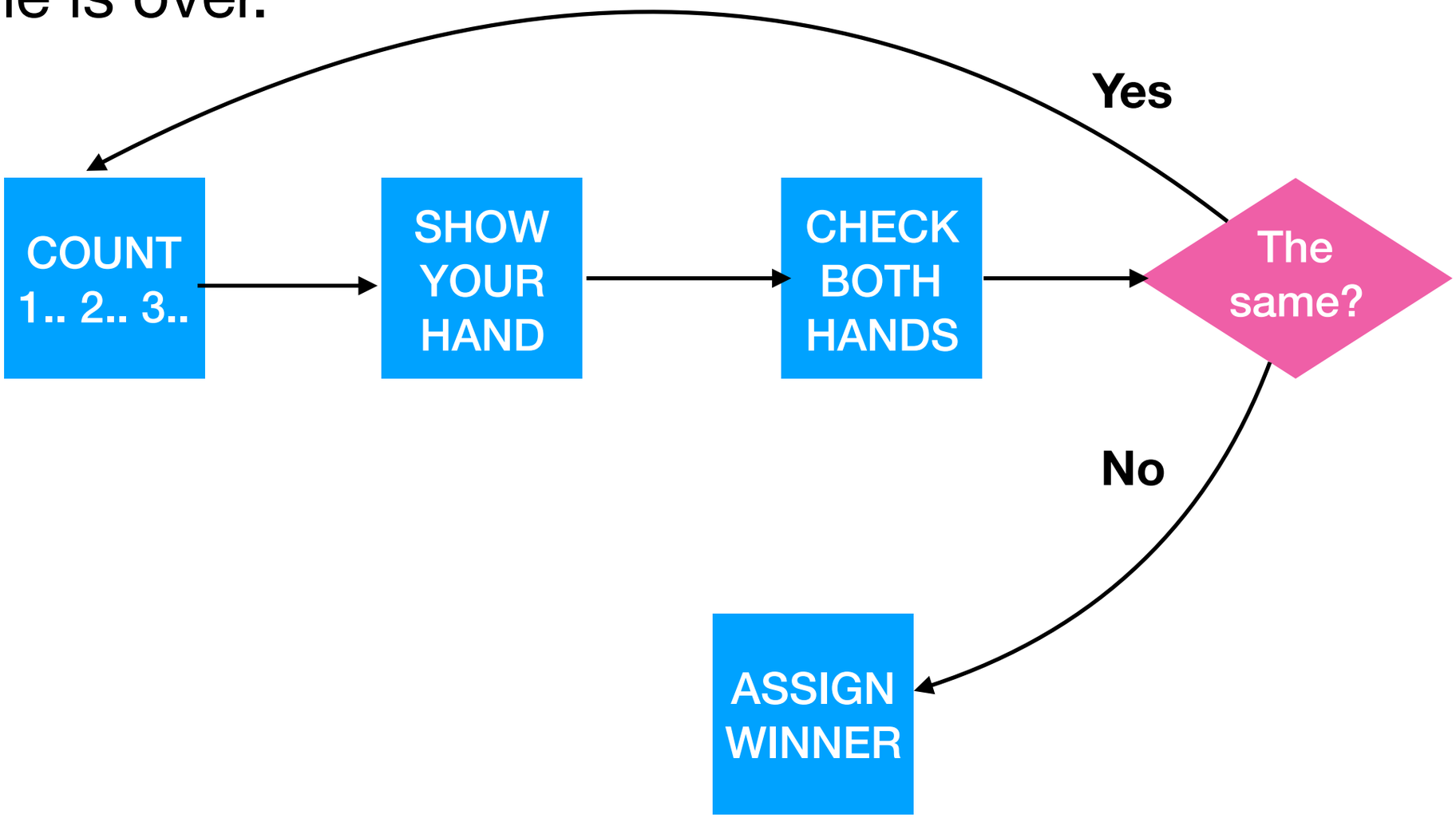

Characters
Cards
Coins
Shards
Profile
Settings
Play PvP
Play Story

# What to look for - rules

- Set of working logical rules
- All the conditions and loops are defined
- You game loop is somewhere there
    - FIND IT

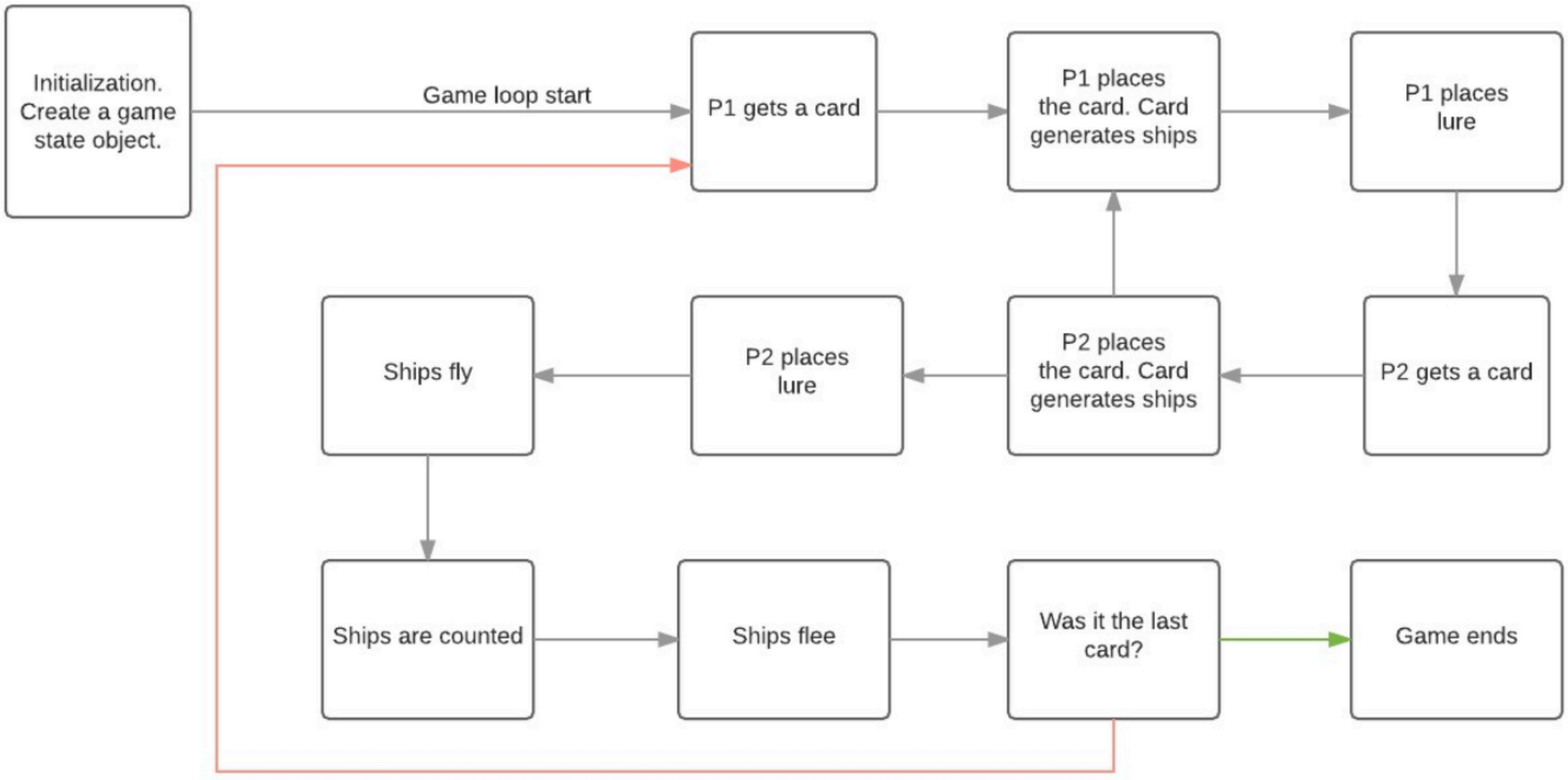
# Game Loop

- A sequence of events/actions that will repeat until the game is over.



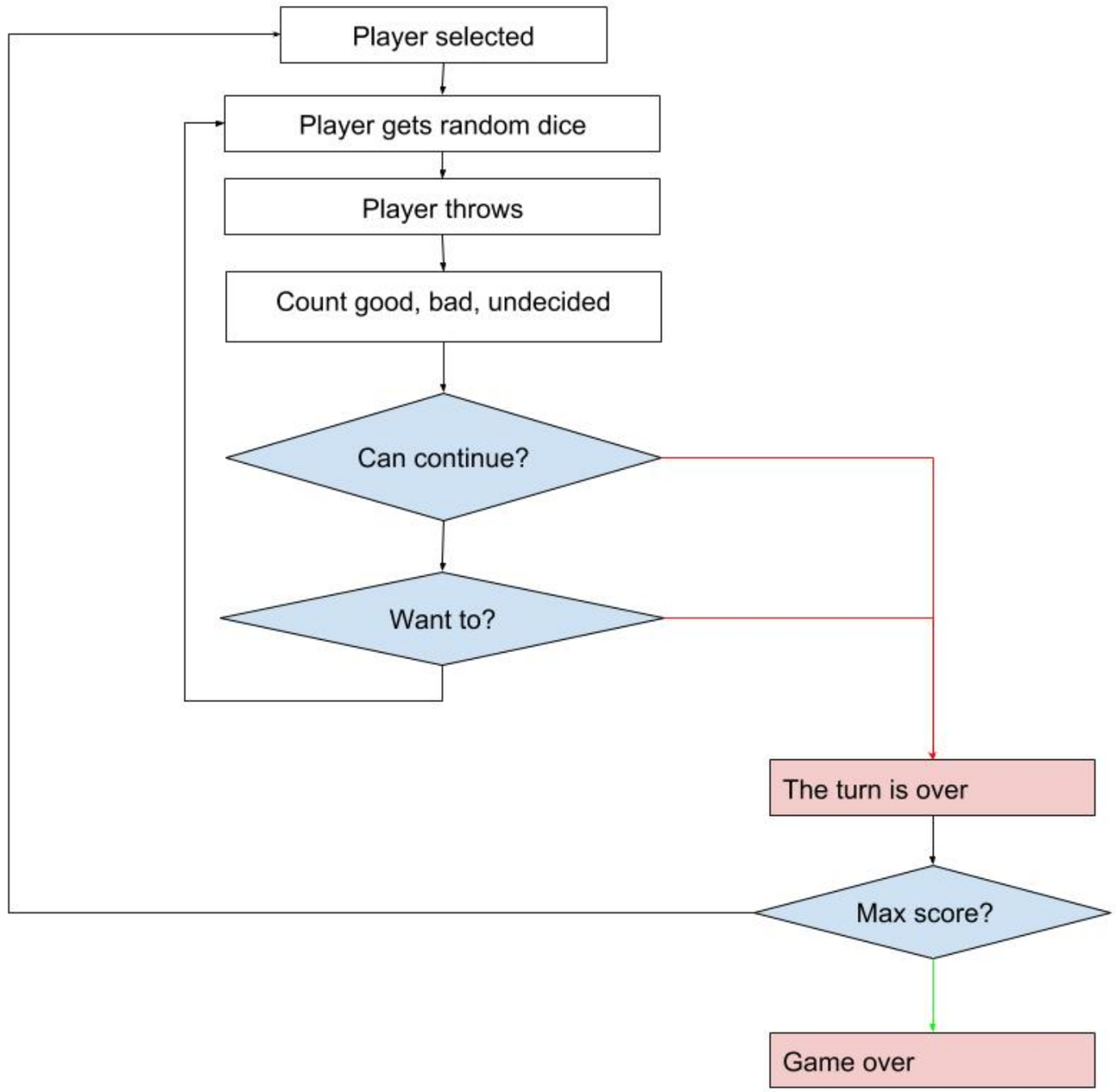
# Examples

- Space HeadHunters



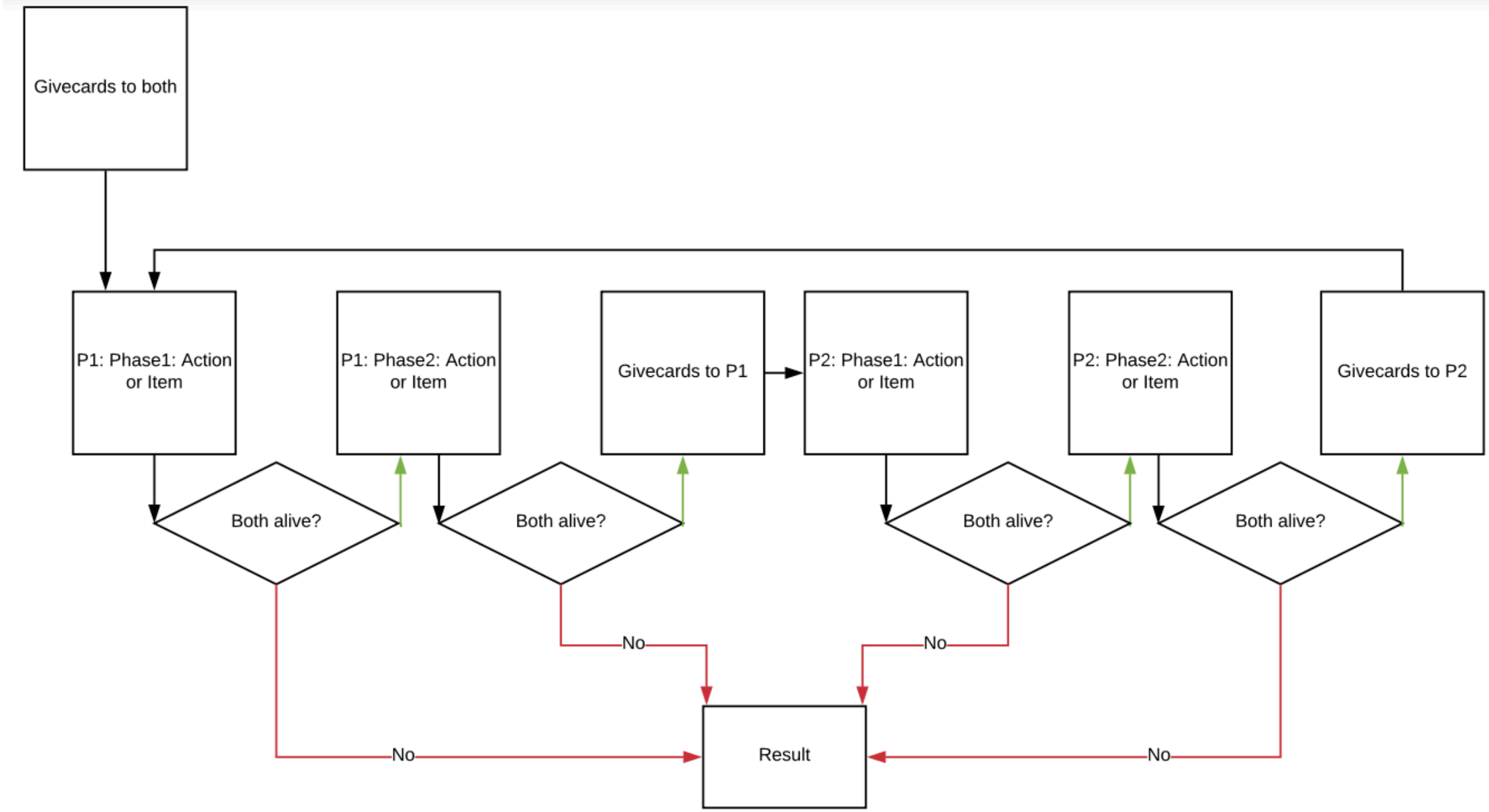
# Examples

- Corgi Dice



# Examples

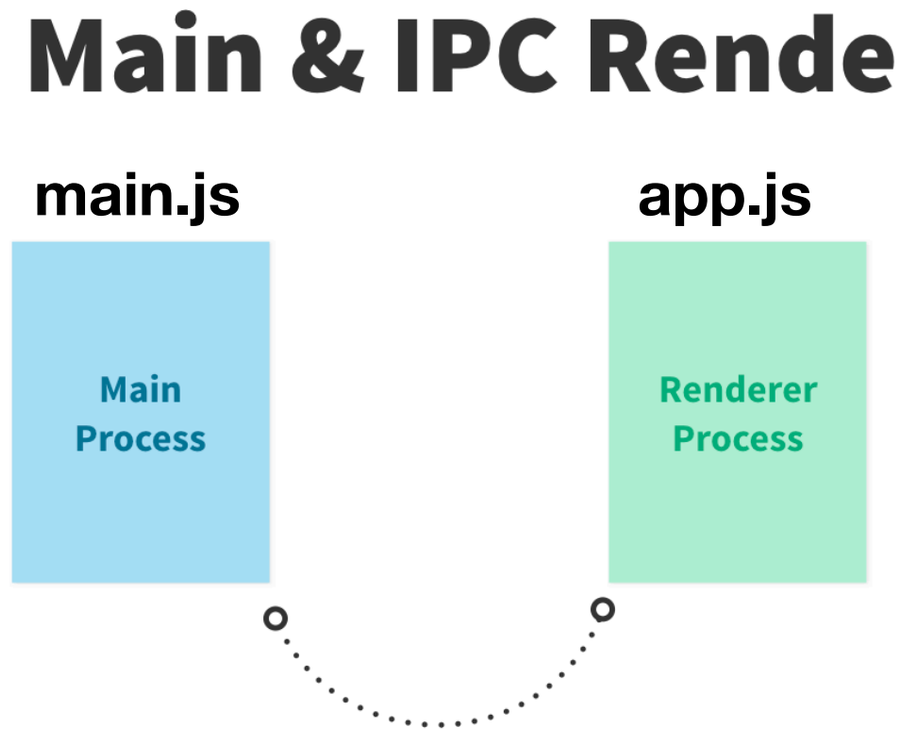
- Bitva



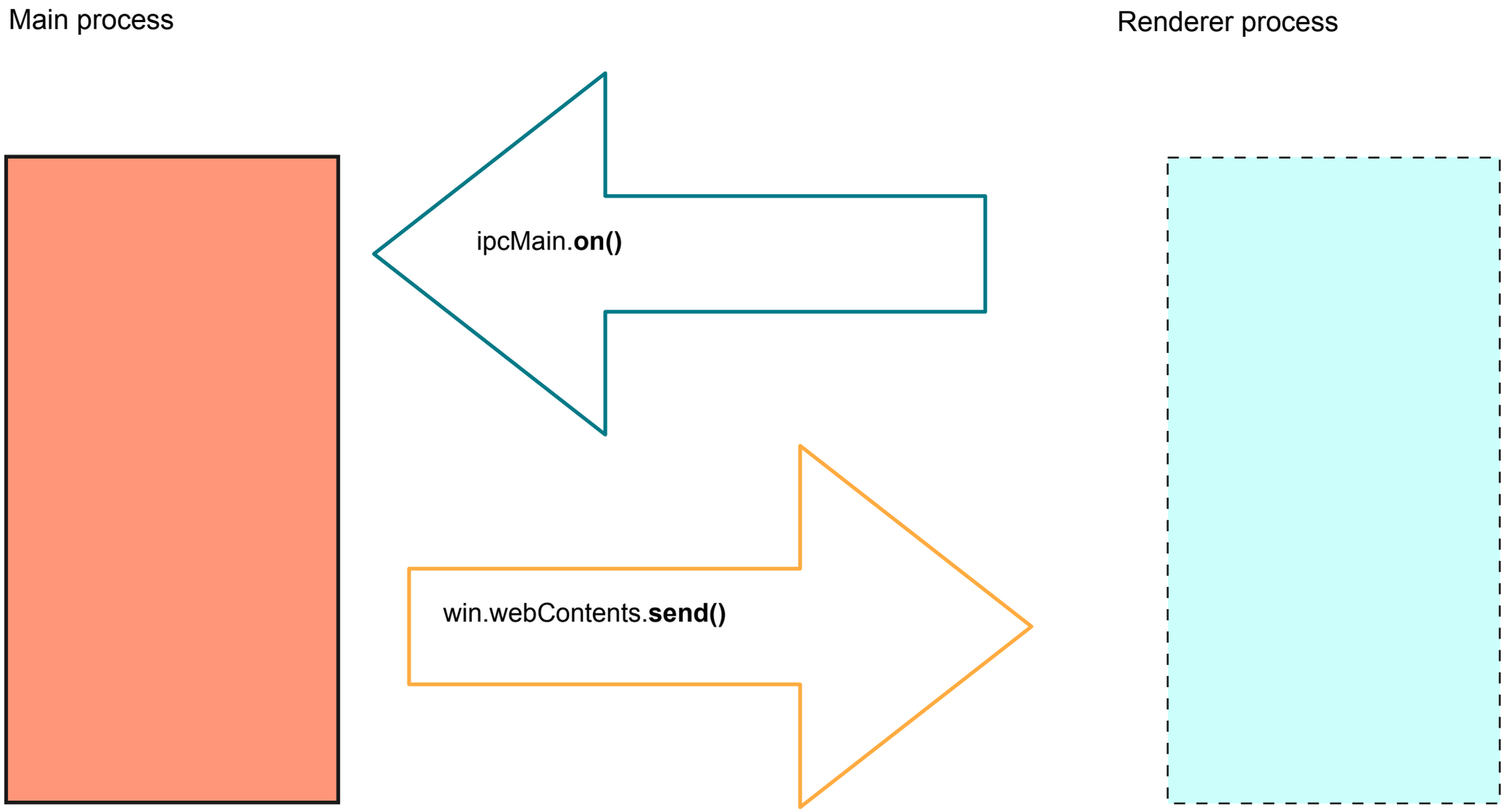
# Plan your architecture

- Your game loop contains STATES of your game
- The data about all the parts of the game can be called GSO
- Game State Object - all the data of your game in any given state

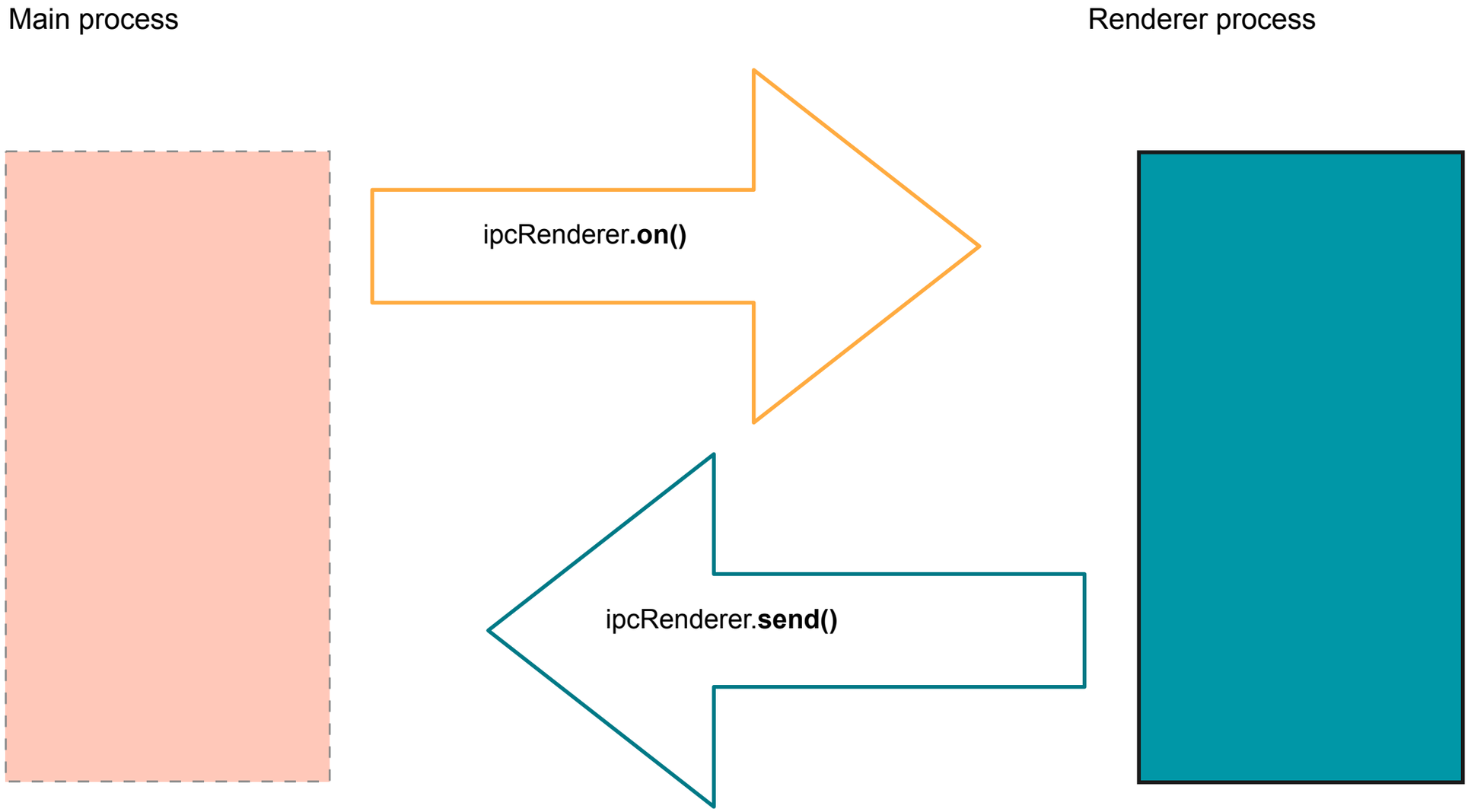
# Electron - frontend/backend communication



# frontend/backend communication



# frontend/backend communication

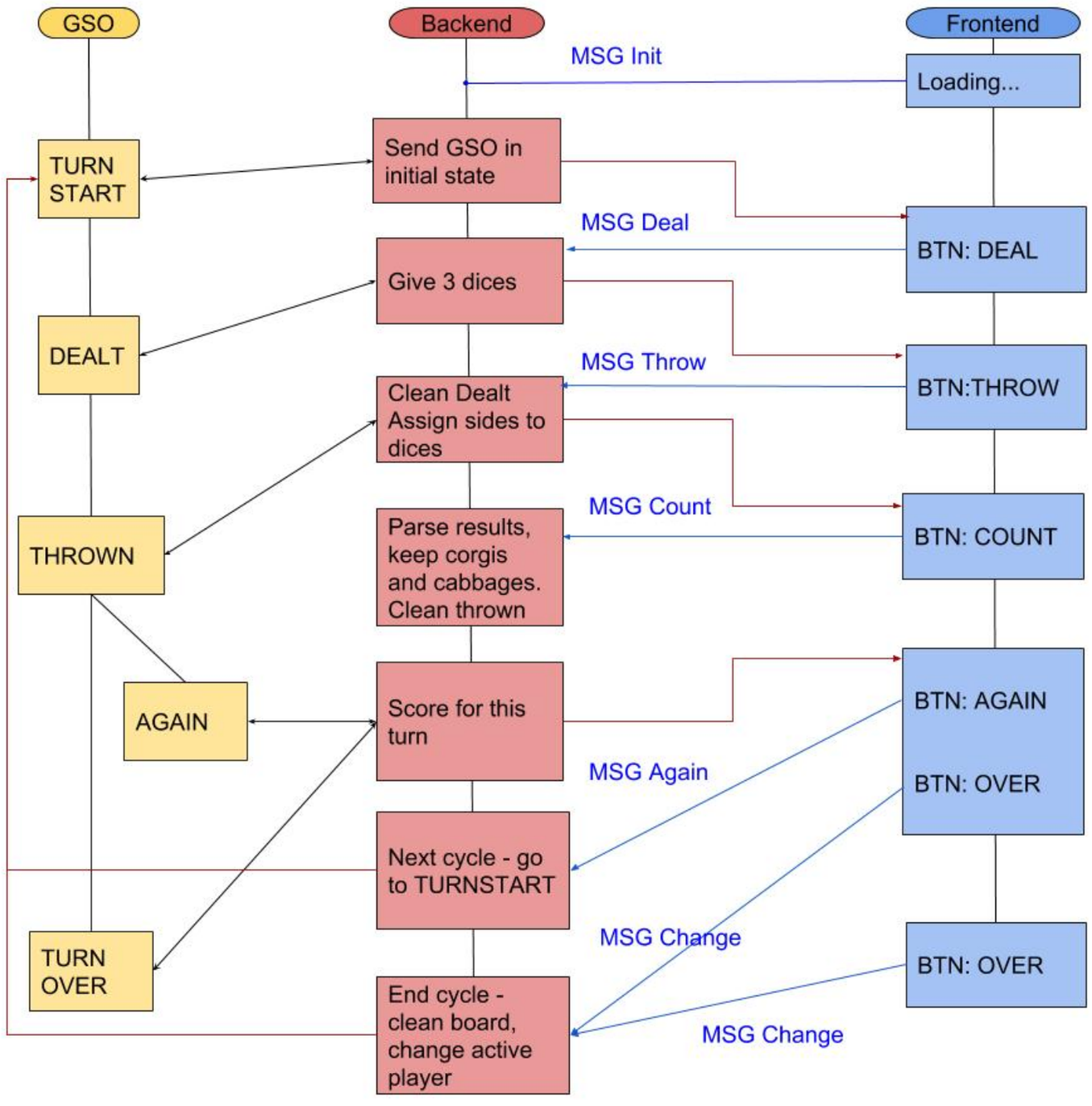


# Electron - frontend/backend communication

- Unidirectional flow
- Work separation (known inputs and outputs)
- Messaging system based on states

# Frontend: messages
# Backend: GSO

- Corgi Dice



# GSO

- Keeps all the data for the current state.
- Changed by  backend (Node.js)
- To what and how?
- TDD it!

# Testing with JEST

- You know your messages
- You know what receives them
- You know what you should get in return
- Write tests. Fail. Fix.

# Example

- Corgi Dice

engine.test.js — CorgiDice

OPEN FILES
- main.js — CorgiDice
- engine.test.js
- main.js — CorgiDice/Solutic

FOLDERS
- CorgiDice
  - __tests__
    - engine.test.js
  - node_modules
  - public
  - Solutions

main.js — CorgiDice | engine.test.js | main.js — CorgiDice/Solutions

```
1  const engine = require("../engine");
2
3  test.only("Init the game: active player is set, state is TURNSTART", ()=> {
4      var gso = {
5          "players": [
6              {"active": true},
7              {"active": false}
8          ],
9          "state": "TURNSTART"
10     }
11     var gso1 = engine.msgReceived("Init");
12     expect(gso1).toMatchObject(gso);
13 });
14
```

3. bash

```
MacBook-Pro:CorgiDice inga$
MacBook-Pro:CorgiDice inga$ jest engine.test.js
 FAIL  __tests__/engine.test.js
  ✕ Init the game: active player is set, state is TURNSTART (14ms)
  ○ skipped 11 tests

  ● Init the game: active player is set, state is TURNSTART

    ReferenceError: initGame is not defined

      76 |     switch(arg){
      77 |         case "Init":
    > 78 |             return initGame(); // GSO state: ANY
         |             ^
      79 |         case "Deal":
      80 |             return getDices(); // GSO state: DEALT
      81 |         case "Throw":

      at Object.msgReceived (engine.js:78:13)
      at Object.<anonymous>.test.only (__tests__/engine.test.js:11:20)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 11 skipped, 12 total
Snapshots:   0 total
Time:        1.444s
Ran all test suites matching /engine.test.js/i.
MacBook-Pro:CorgiDice inga$
```

# Example

- Corgi Dice

```
Ran all test suites matching /engine.test.js/i.
MacBook-Pro:CorgiDice inga$ jest engine.test.js
 PASS  __tests__/engine.test.js
  ✓ Init the game: active player is set, state is TURNSTART (6ms)
  ✓ Give dice - player didn't throw yet, state is DEALT (2ms)
  ✓ Give dice - player has 2 paws in hand (3ms)
  ✓ Give dice to second player - player didn't throw yet, state is DEALT (1ms)
  ✓ Throw the dices player got, set state to THROWN (2ms)
  ✓ Player threw 3 cabbages - his turn is over, set state to TURNEND (1ms)
  ✓ Player threw 3 paws, set state to AGAIN (2ms)
  ✓ Player has 1 corgi, 1 paw and 1 cabbage, it's second cabbage, state is AGAIN (1ms)
  ✓ Player has 2 corgis and 1 cabbage, it's third cabbage, state is TURNEND (1ms)
  ✓ Player wants to play again, state is TURNSTART (1ms)
  ✓ The turn is over, count points, player 1 was active (2ms)
  ✓ The turn is over, count points, player 2 was active (1ms)

Test Suites: 1 passed, 1 total
Tests:       12 passed, 12 total
Snapshots:   0 total
Time:        0.918s, estimated 1s
Ran all test suites matching /engine.test.js/i.
MacBook-Pro:CorgiDice inga$
```

# Frontend

- States mean… React
- Use create-react-app to setup frontend.
- Connect your app.js to main.js
- Send messages in ComponentDidMount()

# Example

- Corgi Dice

```
// Import electron and establis connection to use app.js as Renderer
const electron = window.require('electron');
const ipcRenderer  = electron.ipcRenderer;

class App extends Component {

    // Set the initial state, before contacting backend.
    // Bind the function to send messages
    constructor(props){
        super(props);
        this.state = {gso: {state: "loading"} }
        this.sendMessage = this.sendMessage.bind(this);
    }

    // Using electron Renderer send message to Main process.
    sendMessage(msg){
        ipcRenderer.send('MSG', msg);
    }

    // When the page loads, get GSO from backend, save it to state.
    componentDidMount(){
        ipcRenderer.on("GSO", (event, arg) => {
            this.setState({ gso: arg });
        });
        this.sendMessage("Init");
    }
```

# Connecting things

- Use Foreman to start React and then Electron
- Will freakout on Windows
- Fix it by using cross-env and wait-on
- Packaging is… a bit complicated

# ISSUE

- Electron creates a window and you tell it what to load.

- In development you want to load from react-create-app server.

- In production you want to use a static build that react-create-app made for you.

# DevOps

- npm run build will build you React app, but not Electron app
- electron-packager will build you Electron app, but with no React
- Solution:
  - build react app
    - teach your main.js to load from create-react-app server in DEV environment and from build in PRODUCTION environment
  - use electron-packager

# Main.js

```
function createWindow() {
    // Create the browser window.
    win = new BrowserWindow({
        width: 1366, height: 768, show: false, icon: path.join(__dirname, 'src/icons/png/64x64.png'),
    });

    // and load the index.html of the app.
    if (process.env.REACT_URL) {
        win.loadURL(process.env.REACT_URL);
    } else {
        win.loadURL(url.format({
            protocol: 'file:',
            slashes: true,
            pathname: path.join(__dirname, '/build/index.html')
        }));
    }
```

# Pack and ShipIt

Do some QA though, helpful event with TDD

# Thank you!

Find me here:

- Twitter: @IngaPflaumer
- Medium: @IngaPflaumer
- Sites:
  - https://inga.rocks
  - https://dropbearlabs.com