

- ✓ \* Build skeleton website

- ✓\* Build JSON parser
- ✓\* Build one-way branching structure
- ✓\* Build multi-directional branching structure
- ✓\* Update rooms based on story progress
- \* Build elaborate website
- ✓\* Write at least one story ending

The optional milestones I outlined were:

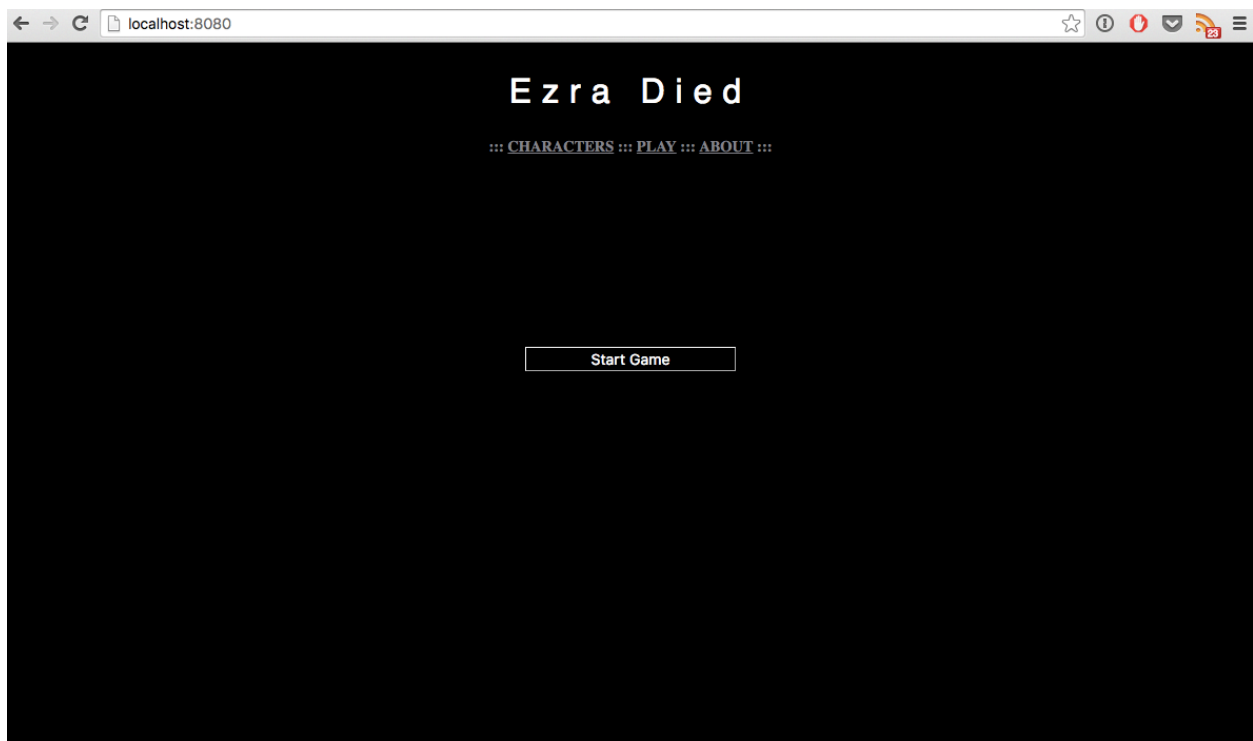
- ✓\* Make a text-input mini-game
- ½\* Make an inventory mini-game
- ✓\* Insert a music player
- \* Build a JSON-based character page
- \* Finish all story text
- \* Build prettier website
- ½\* Save game data in browser to allow for secret endings and loading old games

Overall I am very pleased with the progress that I made. The website and the game are fully functional and are ready to be filled with content. It could be prettier and it could be fancier, but it works well and it even has some extra features. I had time to explore several possibilities for implementing special features that ended up being dead-ends, too, so implementing correctly them will be faster when I work on this project next.

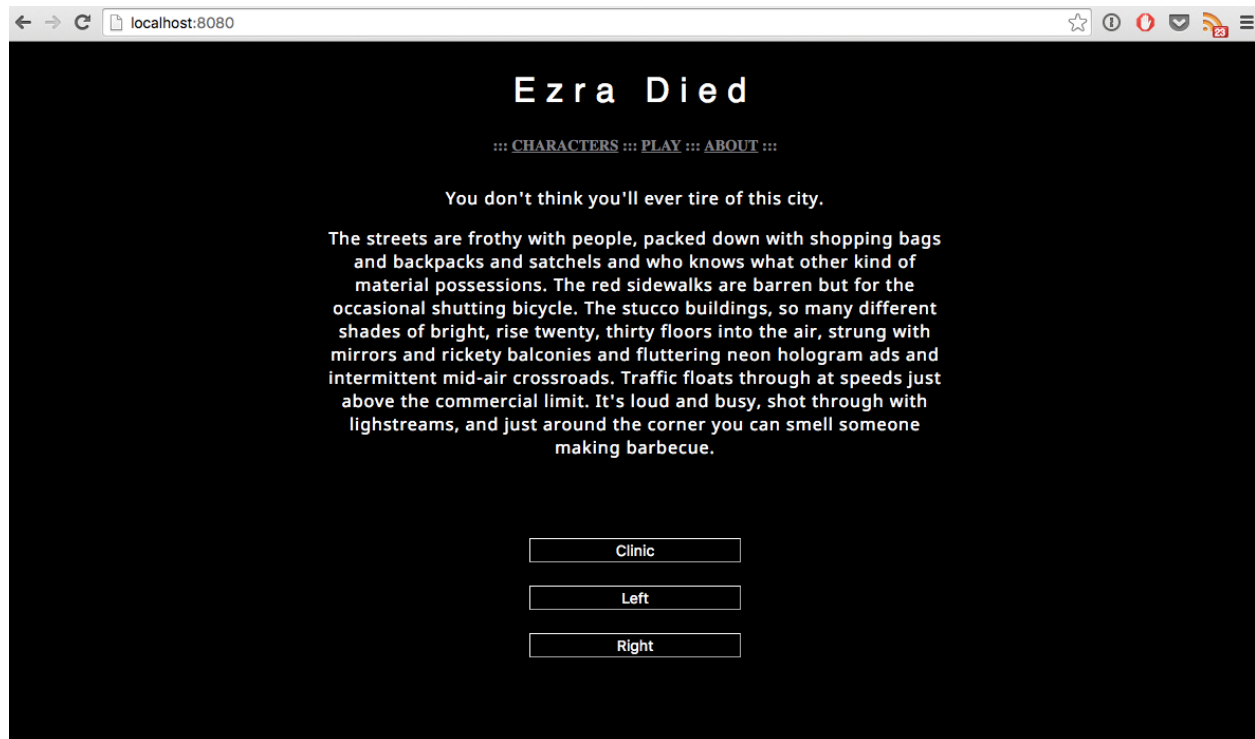
---

### 3. Final Product

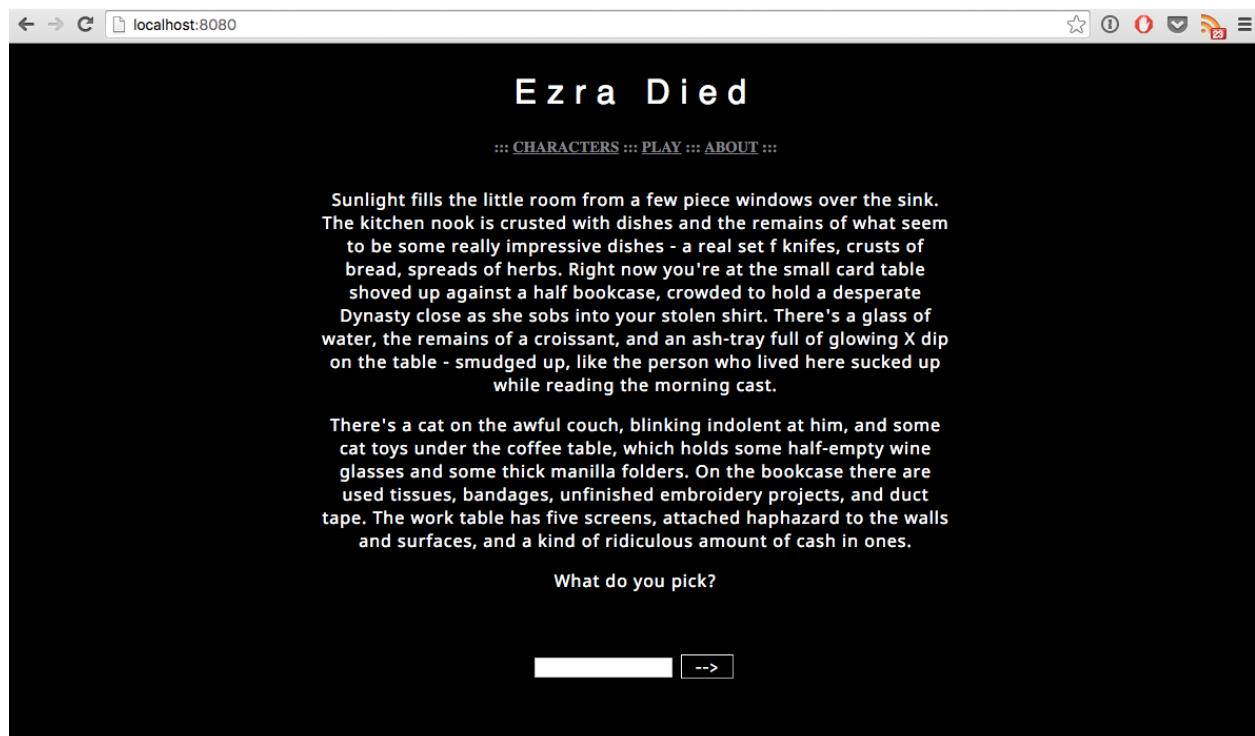
The final product is a plain but dramatic-looking website with a single page. Once you're running the Node.js server, navigate your browser to `localhost:8080` to load the page. For now, the game cannot save any data so it will always reload to the beginning of the game. There is also a rough outline of an "about" page, too. The "characters" link does nothing.



Players can click through and make decisions about where to take the main character. Some events are only available once, such as conversations. Some conversations will unlock areas in a different part of the world. Some rooms have music and others demand silence. Some have



password protected areas, which also function as “searching” mini-games. (In this example, you need to use the *water* in the room to destroy the chip.)



Formatting this was quite confusing for me, as I do not have a lot of sophisticated CSS experience, and after a few hours of meddling with it I decided to focus my attention on the JavaScript code instead. For now I have a rudimentary binary system – rooms tagged as “short” (stored in the “c” variable) are forcibly shoved down to 40% of the vertical space on the page, while rooms tagged as “long” are allowed to start at the top and fill the page as much as they can. Ideally, all rooms would follow the same rule, which would automatically vertically center any text, but that is easier said than done. For now I am manually tagging each entry and dealing with the sometimes awkward visuals this produces.

#### 4. Data Structure: Graph

(name)	
text:	<input type="text"/>
ver:	<input type="text"/>
c:	<input type="text"/>
paths:	
direction:	<input type="text"/>
text:	<input type="text"/>
direction:	<input type="text"/>
text:	<input type="text"/>

*Determines CSS style - see more in section 5*

Figure C. Static room

Here are memory models of the two kinds of “rooms” or “nodes” (vertices) that hold the entirety of the gameplay content. A generic, static room is exactly as pictured, while a room that has different versions of itself (in order to adapt to a player’s progress in the game) looks like a very different kind of object and instead holds an array of possible generic, static rooms (the edges of that vertex).

A vertex might also hold other information than shown here, such as a variable called “music” to hold a string of HTML in order to add a music player or an array of arrays to detail changes in game variables. JSON is extremely forgiving of these oddities, so vertices that have no need for those fields are not forced to store several useless null values. Instead, as shown in figure C, a room

(name)																	
ver:	<input type="text"/>																
index:	<input type="text"/>																
version:	<input type="text"/>																
<table border="1"> <tbody> <tr> <td>text:</td> <td><input type="text"/></td> </tr> <tr> <td>ver:</td> <td><input type="text"/></td> </tr> <tr> <td>c:</td> <td><input type="text"/></td> </tr> <tr> <td colspan="2">paths:</td> </tr> <tr> <td>direction:</td> <td><input type="text"/></td> </tr> <tr> <td>text:</td> <td><input type="text"/></td> </tr> <tr> <td>direction:</td> <td><input type="text"/></td> </tr> <tr> <td>text:</td> <td><input type="text"/></td> </tr> </tbody> </table>		text:	<input type="text"/>	ver:	<input type="text"/>	c:	<input type="text"/>	paths:		direction:	<input type="text"/>	text:	<input type="text"/>	direction:	<input type="text"/>	text:	<input type="text"/>
text:	<input type="text"/>																
ver:	<input type="text"/>																
c:	<input type="text"/>																
paths:																	
direction:	<input type="text"/>																
text:	<input type="text"/>																
direction:	<input type="text"/>																
text:	<input type="text"/>																
<table border="1"> <tbody> <tr> <td>text:</td> <td><input type="text"/></td> </tr> <tr> <td>ver:</td> <td><input type="text"/></td> </tr> <tr> <td>c:</td> <td><input type="text"/></td> </tr> <tr> <td colspan="2">paths:</td> </tr> <tr> <td>direction:</td> <td><input type="text"/></td> </tr> <tr> <td>text:</td> <td><input type="text"/></td> </tr> <tr> <td>direction:</td> <td><input type="text"/></td> </tr> <tr> <td>text:</td> <td><input type="text"/></td> </tr> </tbody> </table>		text:	<input type="text"/>	ver:	<input type="text"/>	c:	<input type="text"/>	paths:		direction:	<input type="text"/>	text:	<input type="text"/>	direction:	<input type="text"/>	text:	<input type="text"/>
text:	<input type="text"/>																
ver:	<input type="text"/>																
c:	<input type="text"/>																
paths:																	
direction:	<input type="text"/>																
text:	<input type="text"/>																
direction:	<input type="text"/>																
text:	<input type="text"/>																

*Should have been a boolean but instead is either -1 or 1*

Figure D. Multi-version room

that does not have those special values doesn't even have to declare them.

## 5. Components

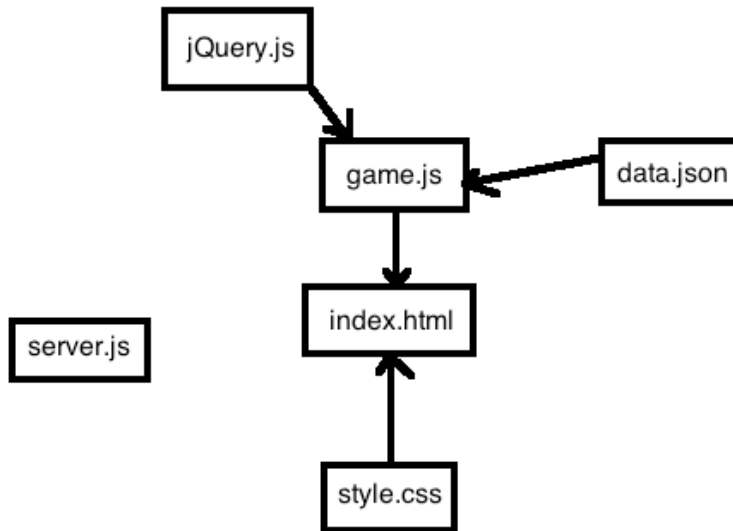


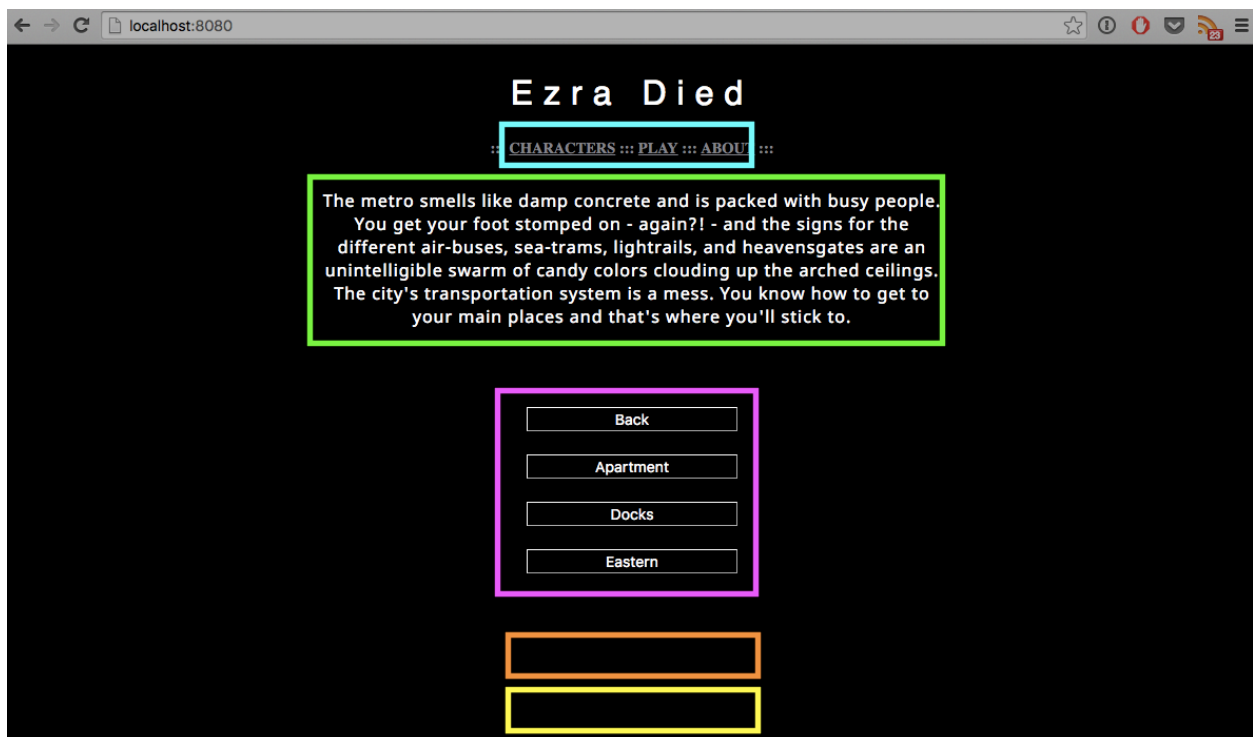
Figure E. Component interactions

The HTML on its own is almost entirely blank. It is styled by the CSS sheet – the fonts, the background color, the length of the buttons, et cetera – and has its content updated by the JavaScript code. The JavaScript parses through the JSON data provided. Meanwhile, the Node.js server keeps everything running.

### HTML AND CSS

There are several dynamic components in the HTML. The header and links always remain unchanged, but the boxes outlined above are rough approximations of the following divs:

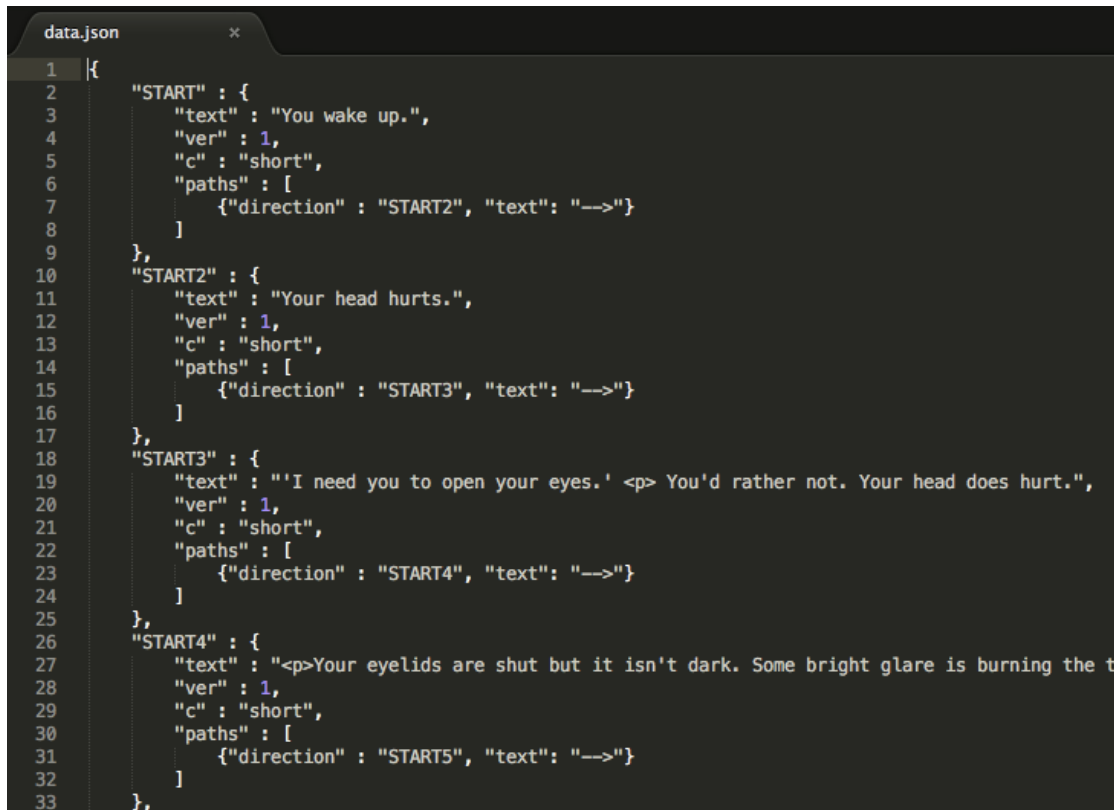
Blue - (unimplemented) inventory   Green - text   Pink - “paths”   Orange - input   Yellow - music



The code knows to feed the descriptive text into the text div, build buttons for each path in the buttons div, and add extras like music into their own respective divs.

## JSON

The data is essentially a very long list of objects. I have made them all follow the same pattern (the “room” memory models above) but that is not a requirement for JSON objects. It just makes sense in this particular application to have uniformity.



```
1  {
2    "START" : {
3      "text" : "You wake up.",
4      "ver" : 1,
5      "c" : "short",
6      "paths" : [
7        {"direction" : "START2", "text": "-->"}
8      ]
9    },
10   "START2" : {
11     "text" : "Your head hurts.",
12     "ver" : 1,
13     "c" : "short",
14     "paths" : [
15       {"direction" : "START3", "text": "-->"}
16     ]
17   },
18   "START3" : {
19     "text" : "'I need you to open your eyes.' <p> You'd rather not. Your head does hurt.",
20     "ver" : 1,
21     "c" : "short",
22     "paths" : [
23       {"direction" : "START4", "text": "-->"}
24     ]
25   },
26   "START4" : {
27     "text" : "<p>Your eyelids are shut but it isn't dark. Some bright glare is burning the t
28     "ver" : 1,
29     "c" : "short",
30     "paths" : [
31       {"direction" : "START5", "text": "-->"}
32     ]
33   },
34 }
```

## JAVASCRIPT

The main functions in the JavaScript code are load() and printCurrentNode(). Load() defines what behavior to do when the page is first loaded – how to retrieve data from the JSON file and what to display as the start screen – while printCurrentNode() first checks which version of the node the player has access to, then sends that version to printCurrentNodeVer() to fetch information from the variable currentNode and print it out in a way that makes sense. This means putting the story text in the text div, adding in any special components such as a input box or a music player, and writing the buttons that will lead the player to the next possible nodes.

---

## 6. Walkthrough

To complete this rudimentary version of Ezra Died, follow these steps:

- (1) Start the game and follow the directions as prompted
- (2) Explore the left side of the street outside the clinic as much as you like
- (3) Explore the right side of the street outside the clinic as much as you'd like
- (4) Eventually go into your apartment and talk to Dynasty

- (5) The answer to the puzzle is “water”
- (6) Go outside and reach the (abrupt) ending

---

## 7. Conclusion

This project surprised me in almost every way. What I expected to be hard – building objects to hold room data, traversing through the data – and what I expected to be easy – making the website, writing the JSON data, parsing the JSON data – were almost always flipped.

```
LinkedWebNode {
    String name;
    String text;
    LinkedWebNode[] links;
    String[] linksName;

    Public LinkedWebNode(String name,
        String text, LinkedWebNode[] links,
        String[] linksName) {

        Name =this.name;
        Text = this.text;

        Links = new LinkedWebNode[5];
        linksName = new String[5];

        For (int i = 0; i <
            this.links.length, i++) {
            If (search(this.links[i]
                == null) {
                Links[i] = new
                LinkedWebNode("error",
                    new LinkedWebNode[0]);
                linksName[i] = "error";
            } Else {
                Links[i] =
                search(this.links[i]);
                linksName[i] =
                this.linksName[i];
            }
        }
    }
}
```

*Figure F. Original pseudo-code*

I had been coding in Java for so long that I forgot JavaScript would not require me to make an interface for a room object. I spent a lot of time trying to figure out how to make an object *without* using an interface, when in reality it was as easy as copy and pasting JSON data over and over again. Figure F shows my original thoughts on how the data structure would work.

The fact that I was not running my code on a live internet server got in my way a lot. First it would not even let my code run, so I started a local server. Then it prohibited several embedded widgets and imported libraries from being served, because of the locality and settings on my server. This was disappointing and hopefully will be cleared up when I host it off my machine.

I was pleasantly surprised by how easy it was to retrieve and display the room data once it had been properly formatted. JavaScript is a beautifully concise language and it was good to work with it again.

The game.js file itself is extremely short and straightforward. The complications were all in *building* the code, but the final product itself is clear enough that that I think even a 101 student could be able to follow it.

In the future I want to complete the text for the game and upload the finished product to the internet. [ezradied.com](http://ezradied.com) is still unclaimed at the moment and I would love to host it publicly and have people enjoy the game. I want to make sure I have a way of saving game progress and of using quotation marks in the text before I put it on the internet, however. I would also work with

someone who knows CSS better so that the website is less glitchy and a lot more professional looking.

It would also be fantastic to make an entirely separate Java application that allows me to enter in data to the JSON data file using a GUI instead of painstakingly formatting it every time I need to make a new room. It would be useful for other people, as well, and I could package it and release it as open-source freeware for other people to put together their own simple browser games.

Overall I am so grateful to have been able to do this project for schoolwork and extremely excited to continue it on my own time. I have big plans, high hopes, and now a great data structures skill set that I'm confident will be enough to finish this game.

---

## 8. Works Cited

This project would never have been possible without the base code I found on JSFiddle (<http://jsfiddle.net/F2es9/>). Obviously the code is 95% different now, but the core idea – using a for loop that builds a series of buttons to direct the paths of the story – was incredibly useful and inspired the direction of the entire project.

I also owe a previous professor, Rula Khayrallah from Foothill College, due credit for the server.js file that allows the entire project to be run from a personal computer. Without it, the code could be perfect but it would still be impossible to play. I'm very grateful that I had a copy because I would have been unable to dedicate as much time to the extra milestones if I had to learn how to write server code from scratch.