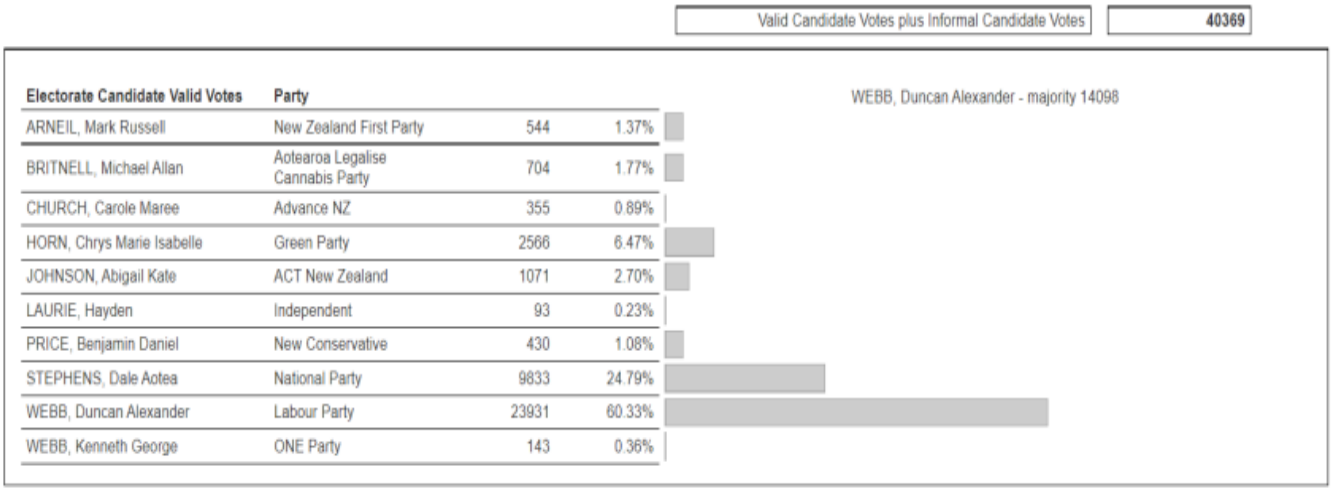
# BCDE102- Assessment 3 – Portfolio

Sample Example

# Introductory overview

My overarching goal for this assignment is to start working on a program modelled over the problem domain of the 2020 New Zealand election, using the event's results as my starting point for data collection. I need to design a program that both assigns Parties to their respective seats and displays the results of the election (for both 2020 Parties and 2020 Electorates) via an HTML webpage.

I have been told to use HTML, CSS and JavaScript to construct this webpage, and the webpage itself is meant to display four values: *Electorate Candidate Valid Votes, Party, seat count,* and *percentage* as a four-column table, coupled with a bar graph on the right-hand side to show visual results for comparison. In the top-right of the webpage, the value of *Valid Candidate Votes plus Informal Candidate Votes* must also be displayed. All this is to be done by my program "behind the scenes", so that the user only perceives a final product like the one provided below from my class:

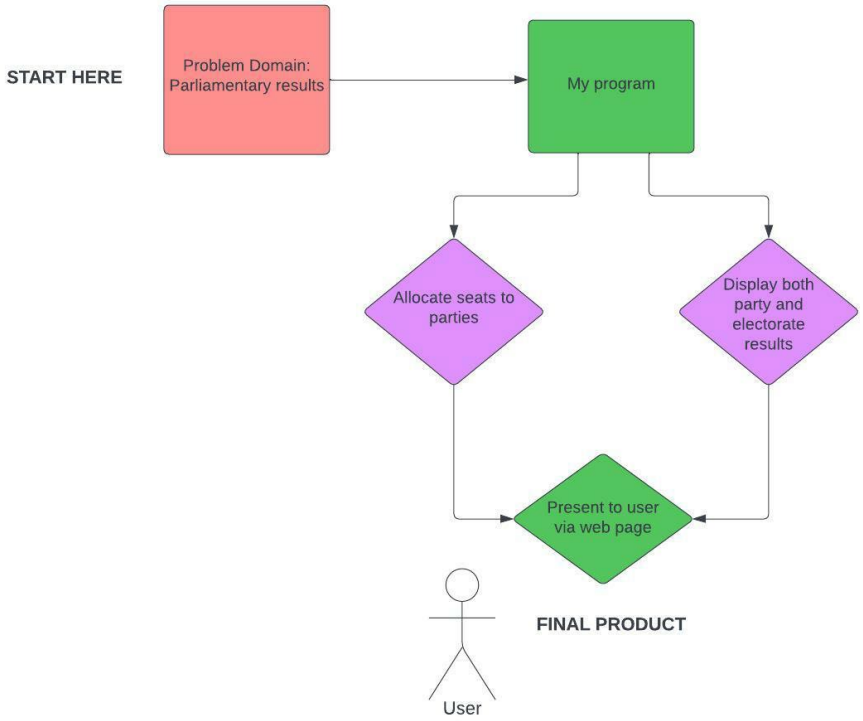


Valid Candidate Votes plus Informal Candidate Votes 40369

WEBB, Duncan Alexander - majority 14098

| Electorate Candidate Valid Votes | Party | | |
|---|---|---|---|
| ARNEIL, Mark Russell | New Zealand First Party | 544 | 1.37% |
| BRITNELL, Michael Allan | Aotearoa Legalise Cannabis Party | 704 | 1.77% |
| CHURCH, Carole Maree | Advance NZ | 355 | 0.89% |
| HORN, Chrys Marie Isabelle | Green Party | 2566 | 6.47% |
| JOHNSON, Abigail Kate | ACT New Zealand | 1071 | 2.70% |
| LAURIE, Hayden | Independent | 93 | 0.23% |
| PRICE, Benjamin Daniel | New Conservative | 430 | 1.08% |
| STEPHENS, Dale Aotea | National Party | 9833 | 24.79% |
| WEBB, Duncan Alexander | Labour Party | 23931 | 60.33% |
| WEBB, Kenneth George | ONE Party | 143 | 0.36% |

Using the already provided image and the brief assignment instructions, I have constructed a perceived initial 'flow' of the program before I start to work on it:




The final product should be a simple webpage with basic visual features, coded in JavaScript, and called within HTML/CSS.

# Iteration 1

## Goal

My current goal is to work on *Controller.js* and *Election.js* in order to Add *Election* as a Class and get it to display on an HTML webpage.

This will involve adding *Election* as an object instance within *Controller* and calling it in *Index.html* to show as output. I will need to construct a UML 2.0 diagram for a relationship between the classes, and I also need to create a set of *Jasmine* specifications for further checking to make sure it works. The final product will be a simple webpage displaying the *Election.js* class as a title on a blank, white webpage.

## Plan

No prior work has been done yet, but my working material will be the provided *Task1-Add Election file,* and a portion of the *Task2-Display Election* file as my foundational base.

The primary work will be to get the output from the Election class up and running, but to also write quite a few *Jasmine* tests to ensure the program is working correctly. Also, the UML 2.0 diagram needs to be done.

## Analysis

I am primarily analysing the SpecRunner output, since not much code has to be written during this iteration. It is a case of 'test and see'.

The purpose will be to ensure that the code is working according to my written *Jasmine* expectations.

### Design

Current design is a simple UML 2.0 Class Diagram relationship between *Controller*, *Election,* and *HTML*.

### Coding

Coding is going well. Only some basic JavaScript functions for both *Jasmine* and *Election* needed to be written.

### Testing

I am testing, in *Jasmine,* if both Election parameters are correct, and if the output of Election through *controller.setup()* is reaching the index.html page.

### Time Estimated for each Task:

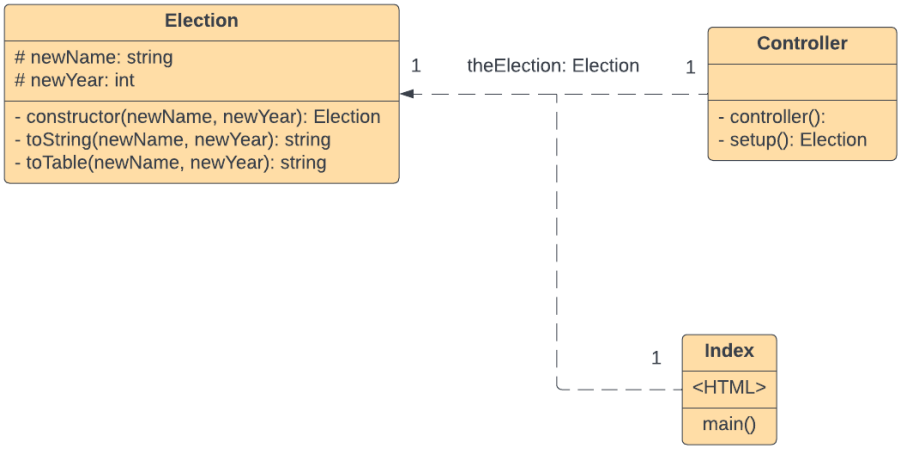
| Task | Task Details + Final product | Time Estimated | Actual Time |
|---|---|---|---|
| *Planning* | Planning class relationships between the three classes, some pseudocode | xxx | xxx |
| *Analysis* | Analysing class methods and attributes, a small sketch to summarise | xxx | xxx |
| *Design* | A digital UML 2.0 Class Diagram | xxx | xxx |
| *Coding* | Pairing Classes together, writing all *Jasmine* specifications; a program with conditions ready to test! | xxx | xxx |
| *Testing* | Testing all *Jasmine* specifications to ensure program is working. A completely green *Jasmine* screen. | xxx | xxx |
| | | **Total time taken:** | xxx |

### Provided Figures

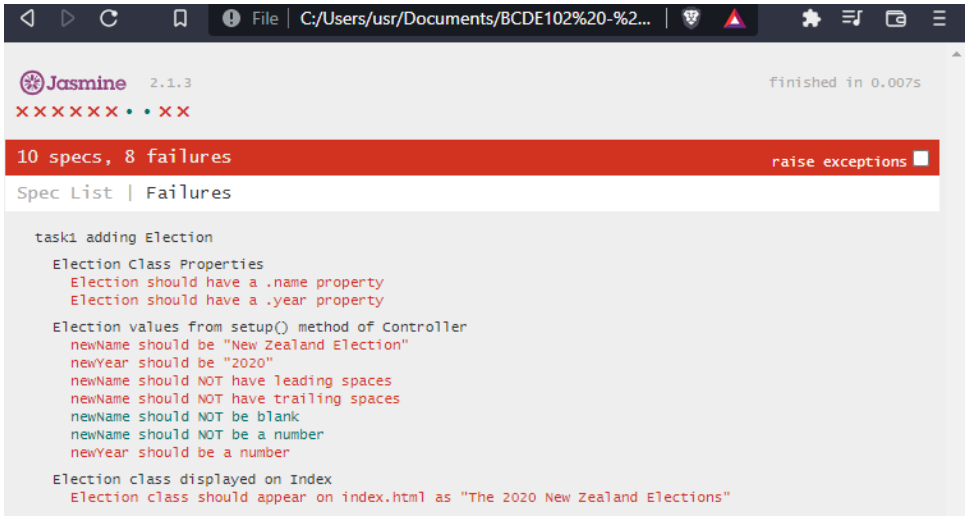
Pseudocode from the planning stage:

```
// pseudocode pre-project

// Class Controller
    // setup() for Election class

// A controller can control one election.

// Class Election
    // newName (string)
    // newYear (int)
    // toString() used for checking in jasmine
    // toTable() used by index.html (format as ${})

// An election can have one index.html page.

// "Class" index.html
    // <HTML>
    // use main() function to output Election.toTable() onto webpage

// Jasmine testing:
    // Test for property values of Class Election
        // Check type, and explicit value
        // Check for if blank or not
    // Modify given Jasmine code further to include index.html checking (regarding output)
```

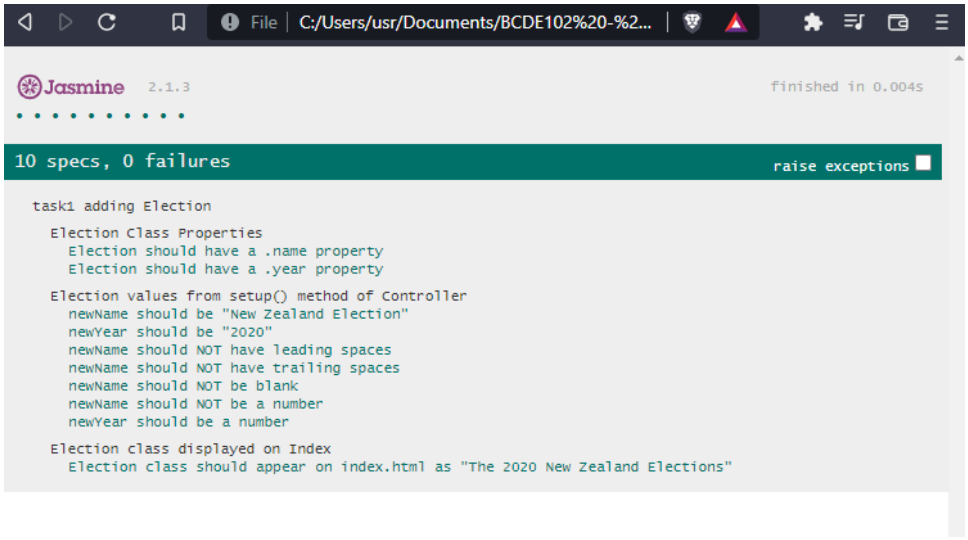
Current *UML 2.0* Class Diagram:



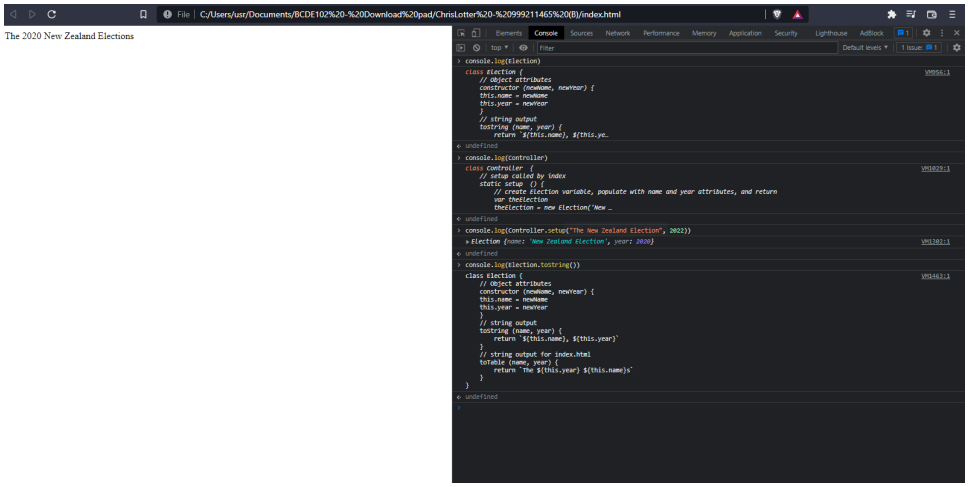

Before *Jasmine*:




After *Jasmine*:




*Index.html*:




***Style note: All JavaScript code is correct in standardjs format, EXCEPT for the usage of class names for Election and Controller; this is intentional, since the classes will be used in later iterations.***

## Unit testing

The unit testing is going very well. I was originally going to write a *.json* function within task1_addElectionSpec.js in order to get the SpecRunner to read output from the HTML page directly but realised that it was more convenient to check for *Election*'s *toTable()* output, since that is sent to Index.html anyways and saves a few lines of code. Aside from this, the rest is just simple formatting checks for attributes within *Election.js*.

## Project expectations vs reality

I expected the project to be quite easy to do before I started, as I knew I only needed to synthesize *Controller, Election,* and *Index* together for the final product of this iteration. The writing of the SpecRunner code took the most time, but it paid off to be careful; I was able to confirm that my class functions and parameters were being called and passed correctly. I expect to have to synthesize *Party* as a class into my program for the next iteration, with the appropriate *Jasmine* checking and display code for the HTML page.

### Performance Review

My actions were completely successful for this iteration, but there was a prior unsuccessful session; I recently tried to attempt this iteration beforehand with the massive task of doing both Parties and Electorates at the same time, as well as doing the basic implementation of the Election class and its display. That was a mistake, and I realize now that I should have planned out an 'iteration calendar' (which I did prior to this new, official attempt) to make sure that each week produced satisfactory 'chunks' of progress. After all, I can always update the UML 2.0 Class diagram as I go, and it was far easier to deal with just Election and display of Election for now. I learned this the hard way, but I am glad to have learned how to pace myself in iterative projects, which are a first for me. My individual performance was also good, and although I went over 30 minutes at one point, I know I can be more consistent for the next iteration. Finally, although our iteration is expected to be roughly five hours, I finished at just over two hours, but only because I had prior familiarity with this iteration already; I am sure, and expect, that I will work within the larger five-hour framework for iteration two.

# Iteration 2

## Goal

The goal for this iteration will be to add *Party.js* as a sub-class of Election, add all provided instances of *Party* to the *Controller,* and display all *Party* objects to the given index.html page to ensure that display is working correctly.

This will involve updating the UML 2.0 diagram, incorporating a UML 2.0 sequence diagram to display method interactions, and writing the necessary JavaScript code and Jasmine specifications before running through the testing phase. The final product will be the same white page from Iteration 1 with the election details, but also added will be an entire output of listed Party instances.

## Plan

We need to build upon Iteration 1 by updating the UML 2.0 diagram, and modify the previous code further, with more Jasmine tests added and an additional class of *Party* synthesized into the program for final output. A UML 2.0 sequence diagram will also need to be pulled up for planning.

## Analysis

My analysis will be primarily split between analysing the UML 2.0 diagram and the Jasmine testing that needs to be followed closely. I will also need to pay close reference to my previous UML 2.0 diagram, so that integration of the new Party class does not interfere with any prior class relationships or already tested Jasmine specifications.

## Design

Design will be a UML 2.0 sequence diagram and an update to the original UML 2.0 Class diagram from Iteration 1.

## Coding

Coding progress in reference to the state transition diagram, the UML 2.0 Class diagram (updated), and the UML 2.0 Sequence diagram, are going well; only Jasmine specifications need to be satisfied before the finalisation of the HTML webpage is done.

## Testing

We need to write a few *Jasmine* specifications to ensure we meet our final product. They are, in pseudo-code terms:

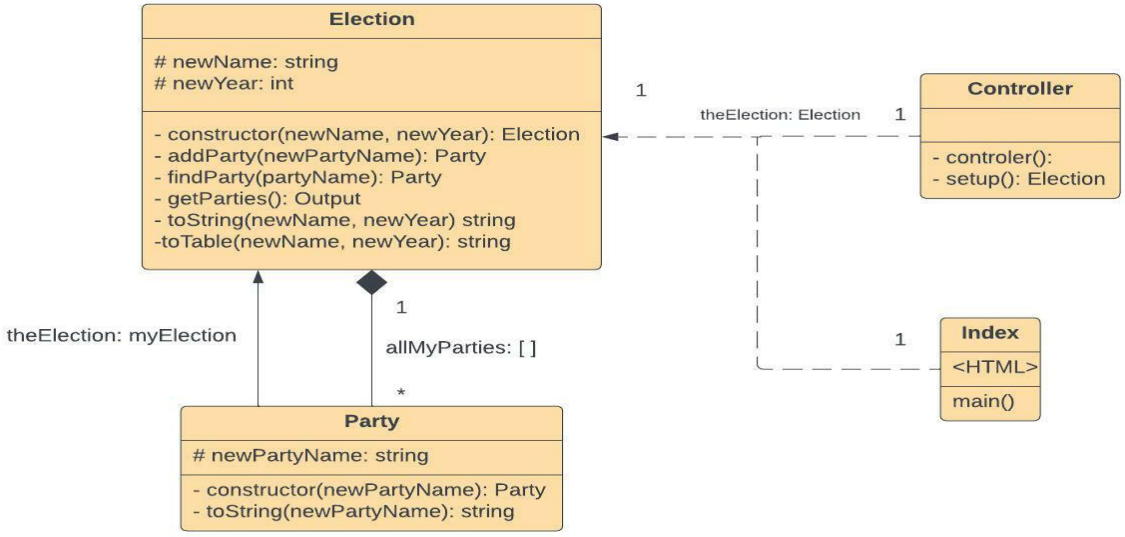
- Ensure Party exists as class
- Ensure Party class object has all necessary functions and properties
- Ensure party can be added by controller
- Ensure parties are added in correct order, and can be displayed in direct output

## Time Estimated for each Task

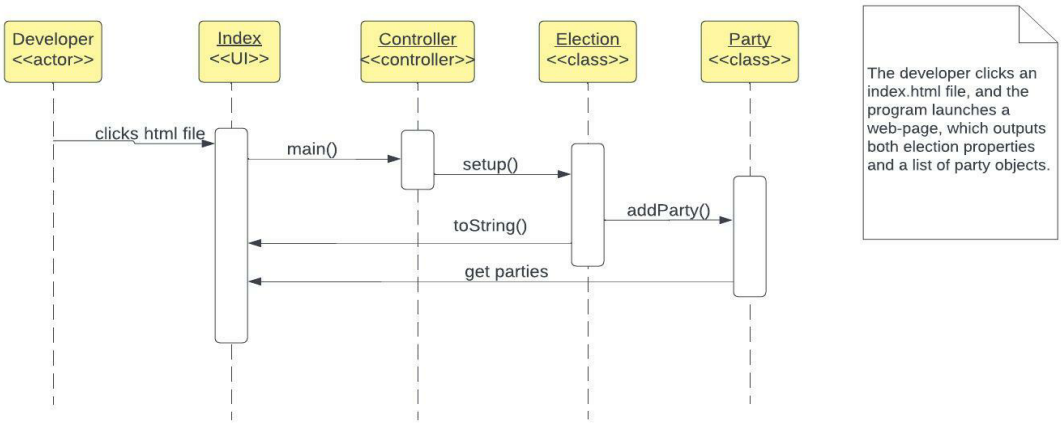
| Task | Task Details + Final product | Time Estimated | Actual Time |
|---|---|---|---|
| *Planning* | Planning design and relationships for state transition diagram | 30 Min | 10 Min |
| *Analysis* | Planning method relationships for UML 2.0 Sequence Diagram | 30 Min | 30 Min |
| *Design* | Updating UML 2.0 Class Diagram with Party class, making the UML 2.0 Sequence diagram, making the state transition diagram. | 30 Min | 30 Min |
| *Coding* | Writing all jasmine specifications. | 30 Min | 45 Min |
| *Testing* | Write *Party* class, Add *Party* class instances, update HTML output, synthesise other code so that it works together. | 30 Min | 60 + Min |
| | | **Total time taken:** Just under five hours | |

## Provided Figures

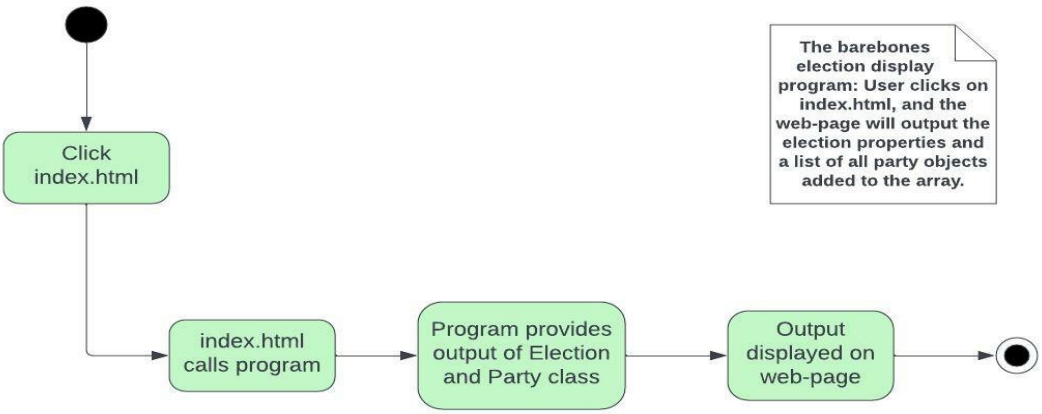
The updated UML 2.0 Class diagram:



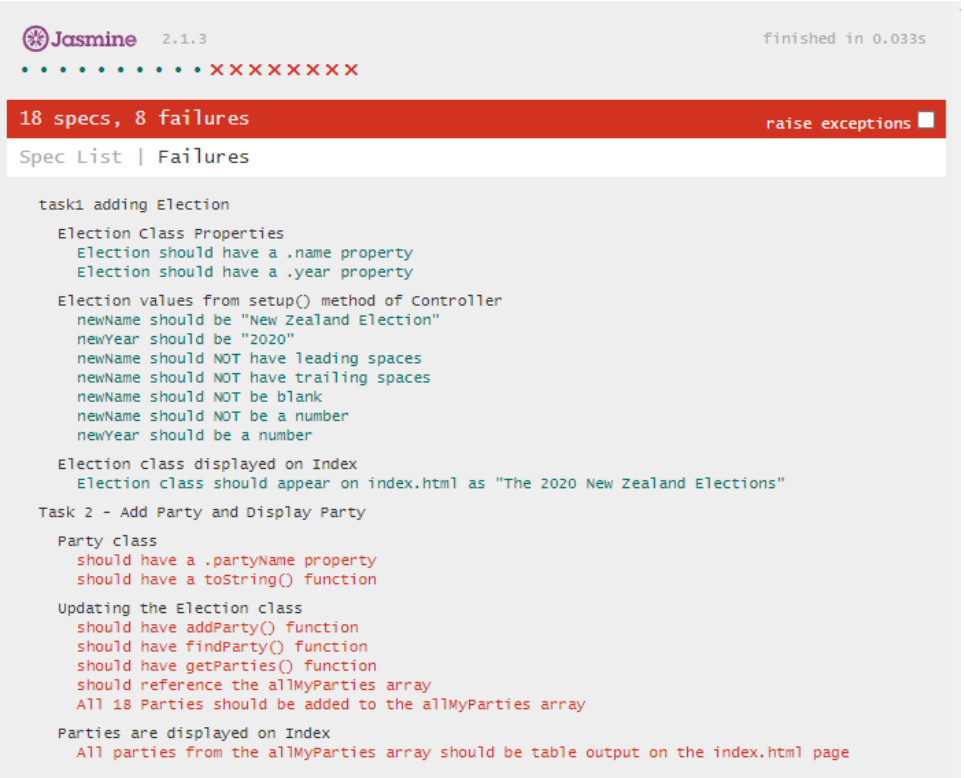

The newly added UML 2.0 sequence diagram:



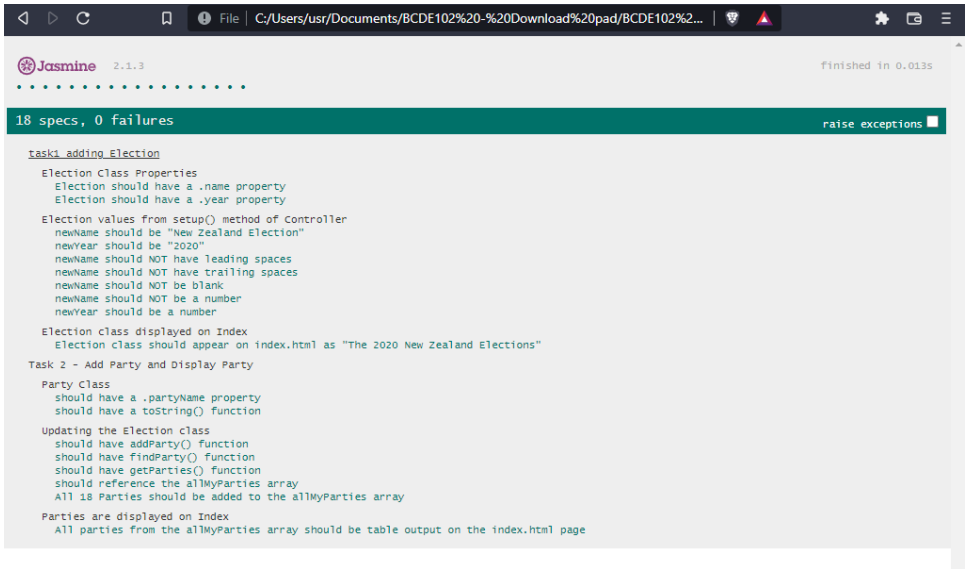

The state transition diagram:



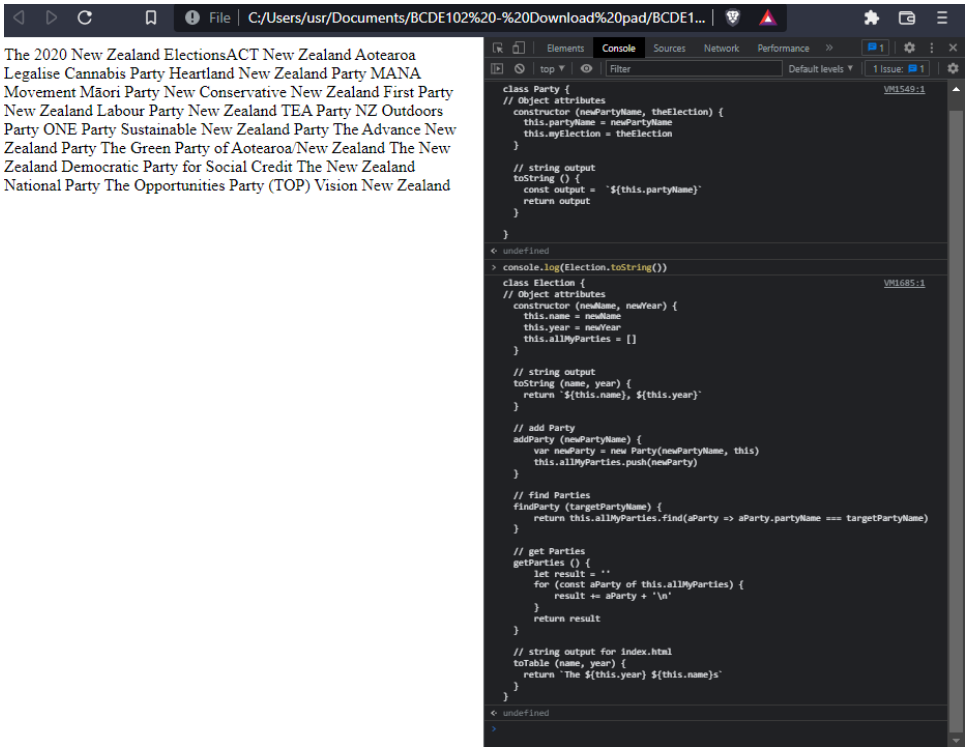
Before *Jasmine:*




After *Jasmine:*

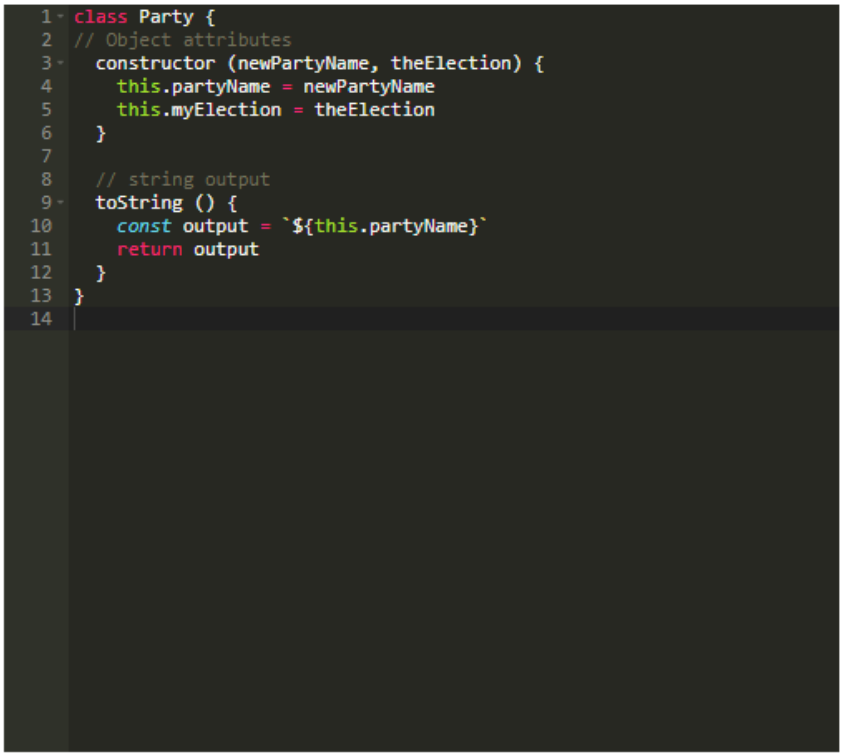



A screenshot of *Index.html*, the final product of Iteration 2: It is quite messy, but this is because it is still a dump from the *allMyParties* array. After CSS styling is done, the text will be much more formatted.




The following is the validation for both the JavaScript and HTML code in terms of style:

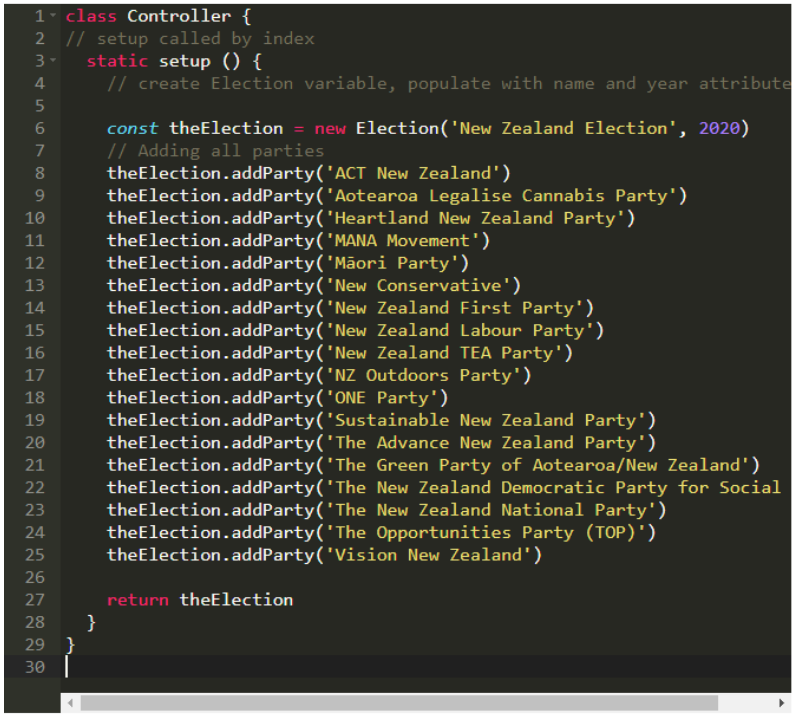
- Party and Controller JS files:



```
class Party {
// Object attributes
  constructor (newPartyName, theElection) {
    this.partyName = newPartyName
    this.myElection = theElection
  }

  // string output
  toString () {
    const output = `${this.partyName}`
    return output
  }
}
```

Correct errors using --fix

JavaScript Standard Style



```
class Controller {
// setup called by index
  static setup () {
    // create Election variable, populate with name and year attribute

    const theElection = new Election('New Zealand Election', 2020)
    // Adding all parties
    theElection.addParty('ACT New Zealand')
    theElection.addParty('Aotearoa Legalise Cannabis Party')
    theElection.addParty('Heartland New Zealand Party')
    theElection.addParty('MANA Movement')
    theElection.addParty('Māori Party')
    theElection.addParty('New Conservative')
    theElection.addParty('New Zealand First Party')
    theElection.addParty('New Zealand Labour Party')
    theElection.addParty('New Zealand TEA Party')
    theElection.addParty('NZ Outdoors Party')
    theElection.addParty('ONE Party')
    theElection.addParty('Sustainable New Zealand Party')
    theElection.addParty('The Advance New Zealand Party')
    theElection.addParty('The Green Party of Aotearoa/New Zealand')
    theElection.addParty('The New Zealand Democratic Party for Social
    theElection.addParty('The New Zealand National Party')
    theElection.addParty('The Opportunities Party (TOP)')
    theElection.addParty('Vision New Zealand')

    return theElection
  }
}
```

Correct errors using --fix

JavaScript Standard Style

The last three screenshots are for the Election.js file:

```
class Election {
// Object attributes
  constructor (newName, newYear) {
    this.name = newName
    this.year = newYear
    this.allMyParties = []
  }

  // string output
  toString (name, year) {
    return `${this.name}, ${this.year}`
  }

  // add Party
  addParty (newPartyName) {
    const newParty = new Party(newPartyName, this)
    this.allMyParties.push(newParty)
  }

  // find Parties
  findParty (targetPartyName) {
    return this.allMyParties.find(aParty => aParty.partyName === ta
  }

  // get Parties
  getParties () {
    let result = ''
    for (const aParty of this.allMyParties) {
      result += aParty + '\n'
    }
    return result
```

Correct errors using --fix

JavaScript Standard Style

```
utput
ame, year) {
{this.name}, ${this.year}`


y
ewPartyName) {
Party = new Party(newPartyName, this)
yParties.push(newParty)


ties
targetPartyName) {
is.allMyParties.find(aParty => aParty.partyName === targetPartyName)


ies
() {
t = ''
t aParty of this.allMyParties) {
+= aParty + '\n'

sult


utput for index.html
me, year) {
he ${this.year} ${this.name}s`
```

Correct errors using --fix

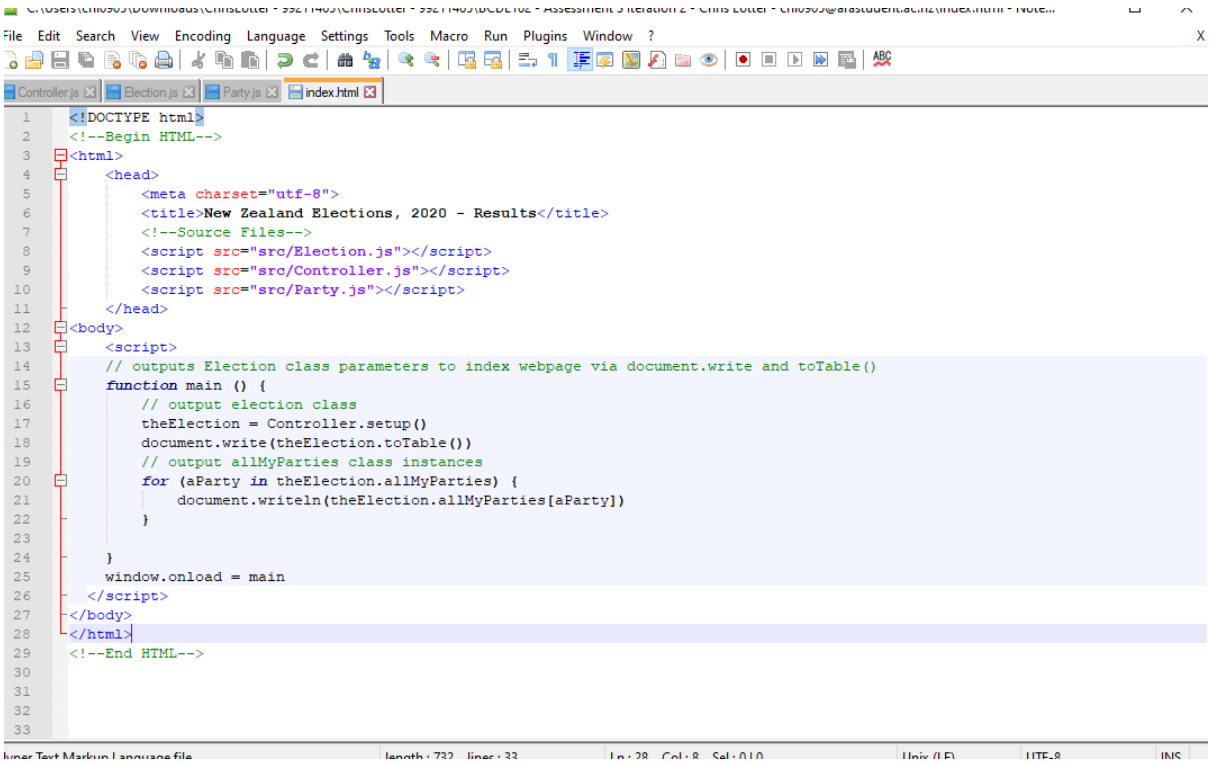
JavaScript Standard Style

```
  // string output
  toString (name, year) {
    return `${this.name}, ${this.year}`
  }

  // add Party
  addParty (newPartyName) {
    const newParty = new Party(newPartyName, this)
    this.allMyParties.push(newParty)
  }

  // find Parties
  findParty (targetPartyName) {
    return this.allMyParties.find(aParty => aParty.partyName === ta
  }

  // get Parties
  getParties () {
    let result = ''
    for (const aParty of this.allMyParties) {
      result += aParty + '\n'
    }
    return result
  }

  // string output for index.html
  toTable (name, year) {
    return `The ${this.year} ${this.name}s`
  }
}

```

Correct errors using --fix

**JavaScript Standard Style**

The last test is for the html file of *index.html*:




```
<!DOCTYPE html>
<!--Begin HTML-->
<html>
    <head>
        <meta charset="utf-8">
        <title>New Zealand Elections, 2020 - Results</title>
        <!--Source Files-->
        <script src="src/Election.js"></script>
        <script src="src/Controller.js"></script>
        <script src="src/Party.js"></script>
    </head>
<body>
    <script>
    // outputs Election class parameters to index webpage via document.write and toTable()
    function main () {
        // output election class
        theElection = Controller.setup()
        document.write(theElection.toTable())
        // output allMyParties class instances
        for (aParty in theElection.allMyParties) {
            document.writeln(theElection.allMyParties[aParty])
        }

    }
    window.onload = main
  </script>
</body>
</html>
<!--End HTML-->
```

## Unit Testing

Testing was unexpectedly very difficult. I took much longer than my expected 30 minutes chunk for this iteration because I had accidently written the Jasmine testing code incorrectly for the *Party* section of the specifications. I had to essentially go back and redo the Jasmine code, and make sure it read the Election class properties and Party class instances correctly, so that I could be sure that data was being placed within the *allMyParties* array. Testing comprised mostly 80% Jasmine testing, with the remaining 20% to html formatting, which ended up just being a block of text for now, as I ran out of time to really continue with it.

## Project expectations vs reality

I expected this project to go swimmingly but was held back by my time spent on writing Jasmine code, and then having to go back and fix some of the Jasmine code, which took up a significant amount of time. My goal of getting the Parties added to the class of Election has succeeded, however, and the final product, although I would prefer to reformat it to look nicer with CSS, looks decent enough and has accomplished the task of displaying all Parties within the array to index.html as output. I now have the steppingstone from which to add array relationships between *Party* and *Electorates* for the next iteration. In that iteration, I will also make sure to do some more CSS styling as well, to fix the displeasing appearance of text on my page.

## Performance Review

I could have done much better writing the Jasmine code or written it at a longer time-period, or just more efficiently. It would have saved me many 30-minute blocks if I had done so and caused a less tedious experience for the testing phase of the project, which was mainly just fixing jasmine code, which should have been unnecessary to do at that stage. I need to be faster at writing Jasmine code, but I also need to be more efficient in laying out *piece by piece* exactly what needs to be tested in SpecRunner, and knowing how to code it exactly with no hiccups.

Aside from the coding and testing stage, I am happy to say that I was very efficient in the planning and analysis stages; I knew exactly how to make my diagrams and was quick to sketch up some appropriate layouts for this current iterative plan. I also updated my previous UML 2.0 Class diagram, and will update these diagrams, if the conditions change, for the next iterations; they are designed to be integrated into future iterations, which is an aspect of the development project that I am interested in continuing.

CSS styling will be a priority for the next iteration, since iteration three will require more visual display with more data.