Peter Hanula
CogSci, Fall 2017

Sternberg Task

The Sternberg task is designed to assess how individuals store and retrieve random information from short-term memory (STM).

Many researchers of memory believe that there exists a STM system that holds information for a few seconds. If the information in STM is not transferred to long term memory (LTM) for more permanent storage, it vanishes. As evidence of the existence of STM grew, researchers started to explore its properties. In a series of articles starting in 1966, Saul Sternberg developed an experimental approach to explore how information was retrieved from STM. [In my report, STM and working memory (WM) are used interchangeably.]

The basic approach is simple. Participants were shown a short (one to six items) list of numbers and asked to memorize them. After putting the list to memory, a probe number was shown. The probe number was either one of the numbers in the list or a new number. The participant was to respond as quickly as possible, indicating whether the probe number was in the list or not. The response time of the participant should reflect the time spent searching STM to determine whether the probe number is part of the list. The key variable in the task is the LIST LENGTH manipulation of the memory set of items. By varying the number of items in the list, Sternberg hypothesized that he could test several theories of STM search. Further information about the original experiment can be found here.

Through his research, Sternberg was able to discover two main findings. The first being, **search is serial**. He found that response times grew linearly with increases in memory set size. For each additional item in the memory set, participants took (on average) an additional 38 ms to make their responses. Thus, it seems the probe item is compared one-by-one with each item in STM.

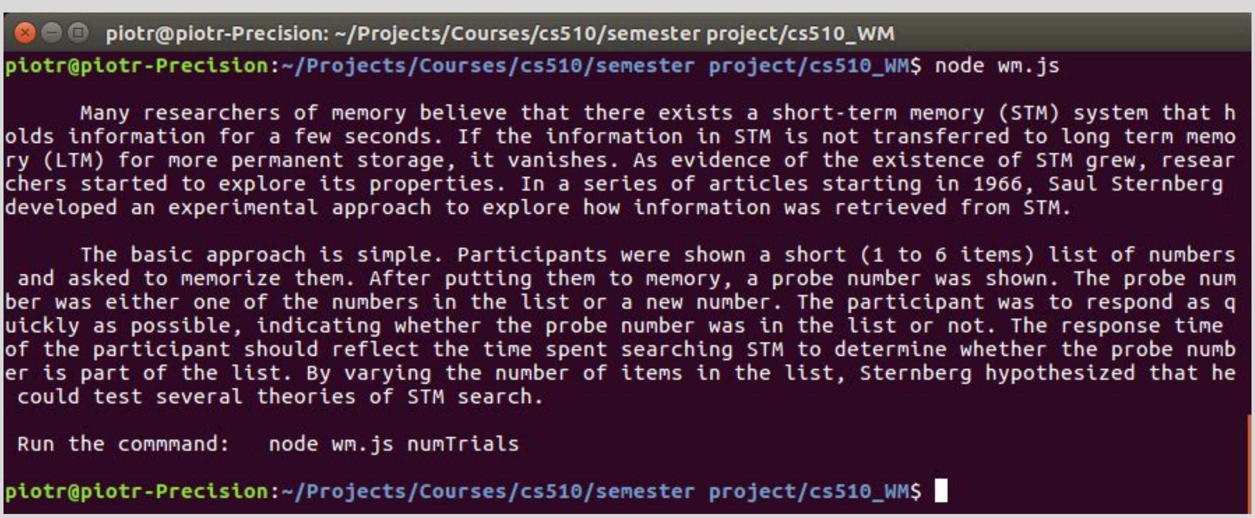
In general, serial memory processing can be either self-terminating or exhaustive. This leads us to Sternberg’s second main finding within his experiments, **search is always exhaustive**. When he compared response times for probe "Yes" and "No" trials (probe item was in the memory set or not, respectively), Sternberg found no differences in response times. At first glance, it might seem that a "Yes" trial could terminate as soon as the probe item is matched with the appropriate item in STM. The counterintuitive finding from Sternberg's study is that search of STM is always exhaustive. That is, the cognitive processes responsible for searching STM for a particular item search through all items in STM before reporting whether the probe item is in memory or not.

Computational Model

I decided to implement a computational model of the Sternberg Task. Instead of Python I used JavaScript, mainly because I wanted to incorporate a graphical user interface (GUI) after the computational model was developed. It is possible to run my model using a command-line interface or the GUI. I will go over both. The code can also be found here.

Command-Line Interface

Using Node.js, it is possible to run the computational model in a similar way that you would run the Python equivalent. Executing “node wm.js” without any arguments outputs a short introduction and further instructions.



```
piotr@piotr-Precision: ~/Projects/Courses/cs510/semester project/cs510_WM
piotr@piotr-Precision:~/Projects/Courses/cs510/semester project/cs510_WM$ node wm.js

      Many researchers of memory believe that there exists a short-term memory (STM) system that h
olds information for a few seconds. If the information in STM is not transferred to long term memo
ry (LTM) for more permanent storage, it vanishes. As evidence of the existence of STM grew, resear
chers started to explore its properties. In a series of articles starting in 1966, Saul Sternberg
developed an experimental approach to explore how information was retrieved from STM.

      The basic approach is simple. Participants were shown a short (1 to 6 items) list of numbers
 and asked to memorize them. After putting them to memory, a probe number was shown. The probe num
ber was either one of the numbers in the list or a new number. The participant was to respond as q
uickly as possible, indicating whether the probe number was in the list or not. The response time
of the participant should reflect the time spent searching STM to determine whether the probe numb
er is part of the list. By varying the number of items in the list, Sternberg hypothesized that he
 could test several theories of STM search.

 Run the command:   node wm.js numTrials

piotr@piotr-Precision:~/Projects/Courses/cs510/semester project/cs510_WM$
```

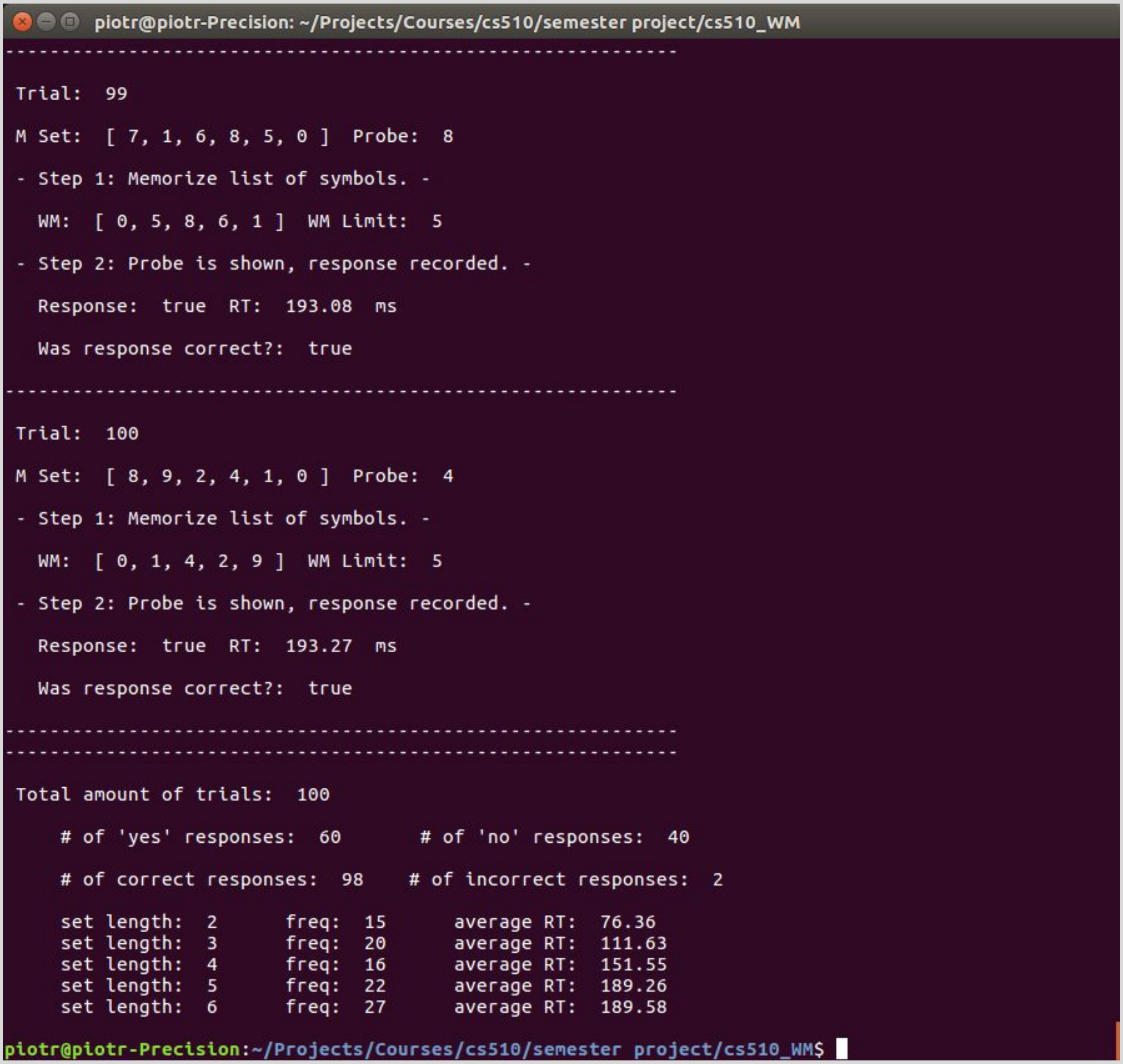
Executing “node wm.js numTrials” will run the Sternberg Task where “numTrials” represents the number of trials to run.

```
piotr@piotr-Precision: ~/Projects/Courses/cs510/semester project/cs510_WM

piotr@piotr-Precision:~/Projects/Courses/cs510/semester project/cs510_WM$ node wm.js 1
------------------------------------------------------------

 Trial:  1

 M Set:  [ 0, 7, 5, 1 ]  Probe:  1

 - Step 1: Memorize list of symbols. -

   WM:  [ 1, 5, 7, 0 ]  WM Limit:  9

 - Step 2: Probe is shown, response recorded. -

   Response:  true  RT:  148.87  ms

   Was response correct?:  true

------------------------------------------------------------
------------------------------------------------------------

 Total amount of trials:  1

     # of 'yes' responses:  1       # of 'no' responses:  0

     # of correct responses:  1    # of incorrect responses:  0

     set length:  4      freq:  1      average RT:  148.87

piotr@piotr-Precision:~/Projects/Courses/cs510/semester project/cs510_WM$
```

Example: running one trial

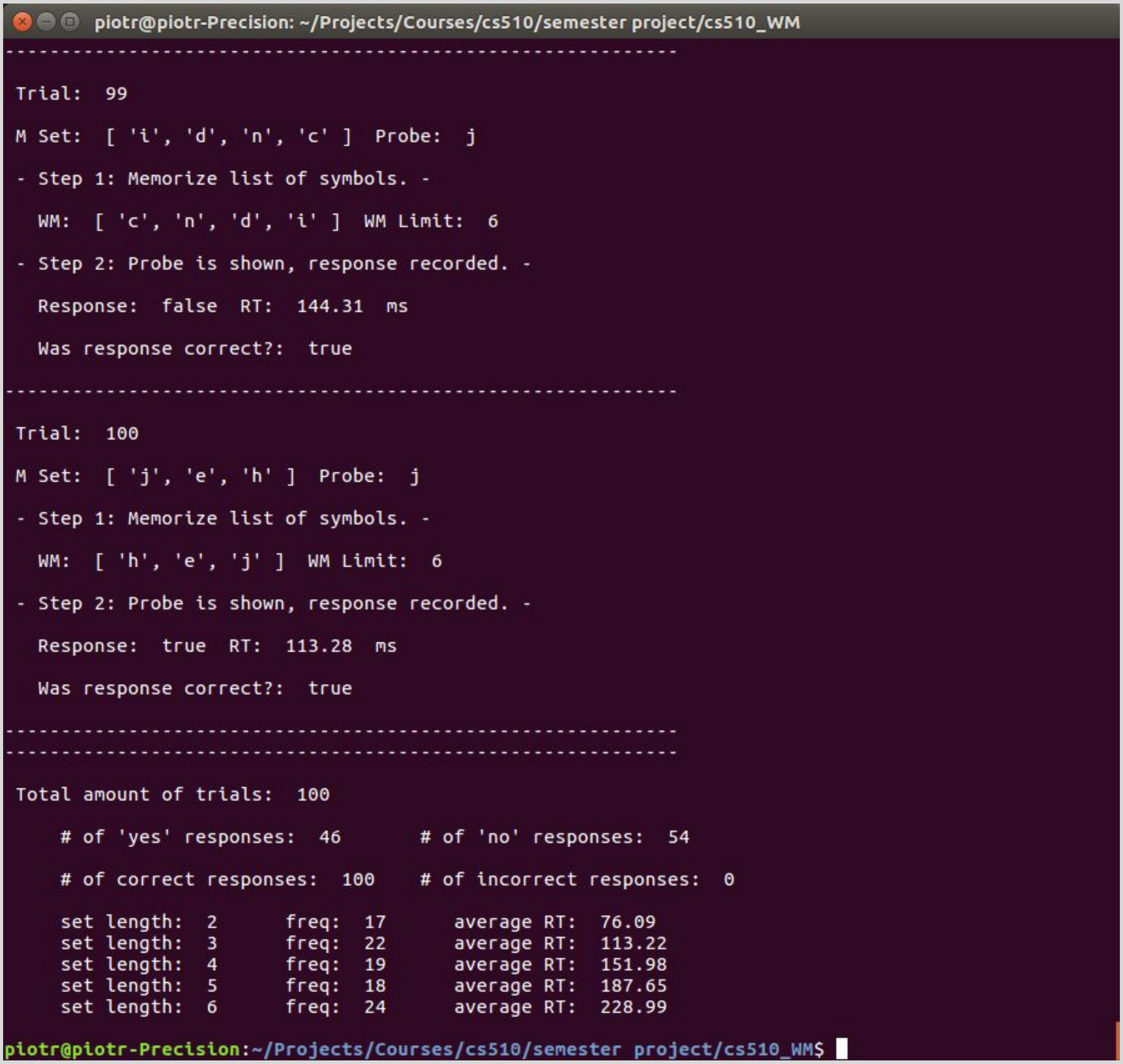
```
piotr@piotr-Precision: ~/Projects/Courses/cs510/semester project/cs510_WM
------------------------------------------------------------

 Trial:  99

 M Set:  [ 8, 2, 0, 9, 7, 1 ]  Probe:  8

 - Step 1: Memorize list of symbols. -

   WM:  [ 1, 7, 9, 0, 2, 8 ]  WM Limit:  9

 - Step 2: Probe is shown, response recorded. -

   Response:  true  RT:  230.34  ms

   Was response correct?:  true

------------------------------------------------------------

 Trial:  100

 M Set:  [ 2, 8, 0 ]  Probe:  2

 - Step 1: Memorize list of symbols. -

   WM:  [ 0, 8, 2 ]  WM Limit:  9

 - Step 2: Probe is shown, response recorded. -

   Response:  true  RT:  113.48  ms

   Was response correct?:  true

------------------------------------------------------------
------------------------------------------------------------

 Total amount of trials:  100

     # of 'yes' responses:  58       # of 'no' responses:  42

     # of correct responses:  100    # of incorrect responses:  0

     set length:  2      freq:  18      average RT:  75.21
     set length:  3      freq:  20      average RT:  114.55
     set length:  4      freq:  23      average RT:  152.68
     set length:  5      freq:  16      average RT:  189.91
     set length:  6      freq:  23      average RT:  227.09

piotr@piotr-Precision:~/Projects/Courses/cs510/semester project/cs510_WM$
```

Example: running 100 trials, note working memory limit is 9



```
piotr@piotr-Precision: ~/Projects/Courses/cs510/semester project/cs510_WM
------------------------------------------------------------

 Trial:  99

 M Set:  [ 7, 1, 6, 8, 5, 0 ]  Probe:  8

 - Step 1: Memorize list of symbols. -

   WM:  [ 0, 5, 8, 6, 1 ]  WM Limit:  5

 - Step 2: Probe is shown, response recorded. -

   Response:  true  RT:  193.08  ms

   Was response correct?:  true

------------------------------------------------------------

 Trial:  100

 M Set:  [ 8, 9, 2, 4, 1, 0 ]  Probe:  4

 - Step 1: Memorize list of symbols. -

   WM:  [ 0, 1, 4, 2, 9 ]  WM Limit:  5

 - Step 2: Probe is shown, response recorded. -

   Response:  true  RT:  193.27  ms

   Was response correct?:  true

------------------------------------------------------------
------------------------------------------------------------

 Total amount of trials:  100

     # of 'yes' responses:  60       # of 'no' responses:  40

     # of correct responses:  98    # of incorrect responses:  2

     set length:  2      freq:  15      average RT:  76.36
     set length:  3      freq:  20      average RT:  111.63
     set length:  4      freq:  16      average RT:  151.55
     set length:  5      freq:  22      average RT:  189.26
     set length:  6      freq:  27      average RT:  189.58

piotr@piotr-Precision:~/Projects/Courses/cs510/semester project/cs510_WM$
```

Example: running 100 trials, note working memory limit is 5



```
piotr@piotr-Precision: ~/Projects/Courses/cs510/semester project/cs510_WM
------------------------------------------------------------

 Trial:  99

 M Set:  [ 'i', 'd', 'n', 'c' ]  Probe:  j

 - Step 1: Memorize list of symbols. -

   WM:  [ 'c', 'n', 'd', 'i' ]  WM Limit:  6

 - Step 2: Probe is shown, response recorded. -

   Response:  false  RT:  144.31  ms

   Was response correct?:  true

------------------------------------------------------------

 Trial:  100

 M Set:  [ 'j', 'e', 'h' ]  Probe:  j

 - Step 1: Memorize list of symbols. -

   WM:  [ 'h', 'e', 'j' ]  WM Limit:  6

 - Step 2: Probe is shown, response recorded. -

   Response:  true  RT:  113.28  ms

   Was response correct?:  true

------------------------------------------------------------
------------------------------------------------------------

 Total amount of trials:  100

     # of 'yes' responses:  46       # of 'no' responses:  54

     # of correct responses:  100    # of incorrect responses:  0

     set length:  2      freq:  17      average RT:  76.09
     set length:  3      freq:  22      average RT:  113.22
     set length:  4      freq:  19      average RT:  151.98
     set length:  5      freq:  18      average RT:  187.65
     set length:  6      freq:  24      average RT:  228.99

piotr@piotr-Precision:~/Projects/Courses/cs510/semester project/cs510_WM$
```

Example: running 100 trials, using letters instead of digits

I will now go over the computational model in detail. The command-line version of the task is in the file “wm.js”. For simplicity, I will omit some unimportant code in this report. You can reference the JavaScript file for the full codebase.

```
/* ----------------------------------------

        Sternberg Task, each execution represents an average adult

        command line argument: # of trials

        node wm.js 54

---------------------------------------- */

var numTrials = 0; // number of trials to run

if (!isNaN(+process.argv[2])) // check if command-line argument is present
    numTrials = +process.argv[2]; // if yes, set numbr of trials to run

var trials = []; // stores all trials

// side note: every hundredish trials, limit increases by 1?
// magic 7 plus-or-minus 2, the limit of an average working memory
var wm_Limit = getRandomNumber(5, 9);

// the symbols to use for the trials, original experiment used digits
// however, this script supports other symbol sets
var symbols = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];
//var symbols = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p'];

// ----------------------------------------

if (numTrials == 0) { // if number of trials wasn't specified, print intro and directions
    console.log("\n      Many researchers of memory believe that there exists a short-term memory (STM
    console.log("\n      The basic approach is simple. Participants were shown a short (1 to 6 items)

    console.log("\n Run the commmand:   node wm.js numTrials\n");
} else { // else setup the experiment, run the trials, and print the data
    setup();
    printTrials();
    printStats();
}

// ----------------------------------------
```

```javascript
function setup() { // used to generate memory sets and run the task

    var currTrial = 1; // keep track of trials

    while (currTrial <= numTrials) { // run trials

        var memorySet = generateList(); // generate memory set

        var probe;

        // extra step to make sure probe has a 50/50 chance of being in the memory set

        var probeProb = getRandomNumber(0, 3);

        if (probeProb == 0)
            probe = memorySet[getRandomNumber(0, memorySet.length - 1)];
        else
            probe = symbols[getRandomNumber(0, symbols.length - 1)];

        // run Sternberg Task
        // pass the current trial number, the memory set, and the probe
        sternbergTask(currTrial, memorySet, probe);

        currTrial += 1; // increase trial number, needed for while loop
    }
}
```

```javascript

// THE STERBERG TASK
function sternbergTask(currTrial, memorySet, probe) {

    // Participant is shown a short (one to six items) list of numbers and asked to memorize them.
    // The list of numbers is stored in working memory

    var wm = memorizeSymbols(memorySet);

    // After putting the list to memory, a probe number is shown. The probe number is either
    // one of the numbers in the list or a new number. The participant has to respond as quickly
    // as possible, indicating whether the probe number was in the list or not. yes or no.

    var pResponse = seeProbeAndRespond(wm, probe);

    // validate the user's response to see if the probe was actually in the memory set or not

    var isCorrect = validateResponse(memorySet, probe, pResponse.response);

    // record and store trial

    var trial = {
        trialNumber: currTrial,
        memorySet: memorySet,
        probe: probe,
        wm: wm,
        wm_Limit: wm_Limit,
        pRes: pResponse.response,
        pResTime: pResponse.responseTime,
        isCorrect: isCorrect
    };

    trials.push(trial); // add trial to list of trials
}
```

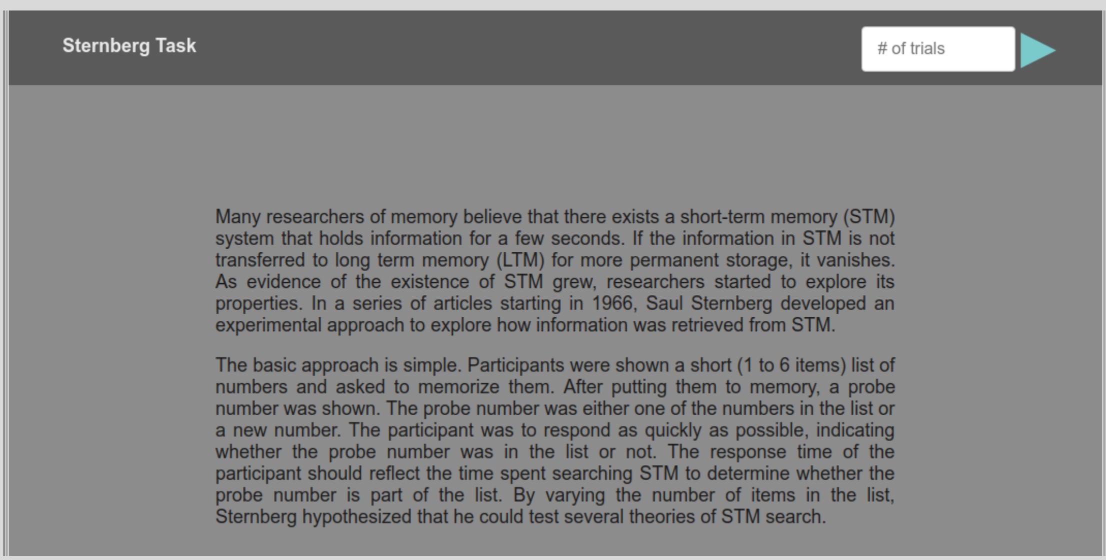
```javascript

function getRandomNumber(min, max) { // function used to generate random whole numbers
    min = Math.ceil(min);
    max = Math.floor(max);

    // the maximum is inclusive and the minimum is inclusive

    return Math.floor(Math.random() * (max - min + 1)) + min;
}

function generateList() { // function used to generate memory sets similar to original experiments

    var length = getRandomNumber(2, 6); // list length is between 2 and 6

    var symbolsCopy = symbols.slice(); // get a copy of the symbol set
    var indices = [];
    var tempList = []; // temporary list, that will be filled and returned

    if (symbolsCopy.length < length) { // safety check if symbol list is shorter than the list length
        console.log("Symbol set too small. Duplicating symbols to fill.")
        var tempLength = symbolsCopy.length;
        var multiplier = tempLength * Math.ceil((1.0 * length) / symbolsCopy.length);

        for (var x = 0; x < multiplier; x++) {

            symbolsCopy.push(symbolsCopy[x % tempLength]);
        }
    }

    for (var j = 0; j < length; j++) { // fill indices, used later for random order
        indices.push(j);
        tempList.push(symbolsCopy[0]);
    }

    for (var i = 0; i < length; i++) { // fill temporary list to return

        var randSymbSeed = getRandomNumber(0, symbolsCopy.length - 1); // get random symbol
        var randIndSeed = getRandomNumber(0, indices.length - 1); // get random index

        // fill memory set with random symbol at random index
        tempList[indices[randIndSeed]] = symbolsCopy[randSymbSeed];

        // delete the symbol and index already used to fill memory set
        symbolsCopy.splice(randSymbSeed, 1);
        indices.splice(randIndSeed, 1);
    }

    // return the temporary list, which is the final memory set for the current trial
    return tempList;
}
```

```
// function part of the model, user needs to memorize memory set
// coded to reflect how a human stores symbols in working memory
function memorizeSymbols(memorySet) {

    var tempWM = []; // working memory

    // user sees one symbol at a time and stores it in working memory
    for (var i = 0; i < memorySet.length; i++) {

        // new symbol is placed in beginning of working memory,
        // shifting older symbols to the right

        tempWM.unshift(memorySet[i]);

        // if working memory limit is reached, then oldest symbol
        // to the right gets deleted, just like a human

        if (i >= wm_Limit)
            tempWM.pop();
    }

    return tempWM; // return the working memory set
}

// function part of the model, user needs to see the probe and respond
function seeProbeAndRespond(wm, probe) {

    var r = false, rt = 0.0; // r = response, rt = response time

    // search is done in serial
    // search is exhaustive

    for (var i = 0; i < wm.length; i++) { // check WM for probe

        if (wm[i] == probe) // if found, response will be true
            r = true;

        // response time is increased for every symbol added to memory set.
        // these numbers are derived from the original experiment
        // average response time was 38 ms per symbol added

        rt += getRandomFloat(34.1, 41.7); // in milliseconds (ms)
    }

    // store the user's response and response time
    var pResponse = { response: r, responseTime: parseFloat(rt.toFixed(2)) };

    return pResponse; // return the response
}
```

```
// function used to validate user's response.
// was probe part of memory set?
function validateResponse(memorySet, probe, pResponse) {

    var correctAnswer = false;

    for (var i = 0; i < memorySet.length; i++) {

        if (memorySet[i] == probe)
            correctAnswer = true;
    }

    if (pResponse === correctAnswer) // check if user answered correctly
        return true;
    else
        return false;
}

// function used to generate random float numbers
function getRandomFloat(min, max) {
    return Math.random() * (max - min) + min;
}
```

Graphical User Interface

The GUI version uses the same core codebase as the command-line version, shown above. I then implemented the graphical components around the core codebase. Because the main focus of the class was to implement a computational model, I will not explain all the code required for the GUI, but you can still have a look at the code here, in particular the "wm_ui.js" file. Instead, I will share screenshots highlighting the GUI. You can also run your own trials here, since I have the GUI/web-app hosted on GitHub for all to see and use.



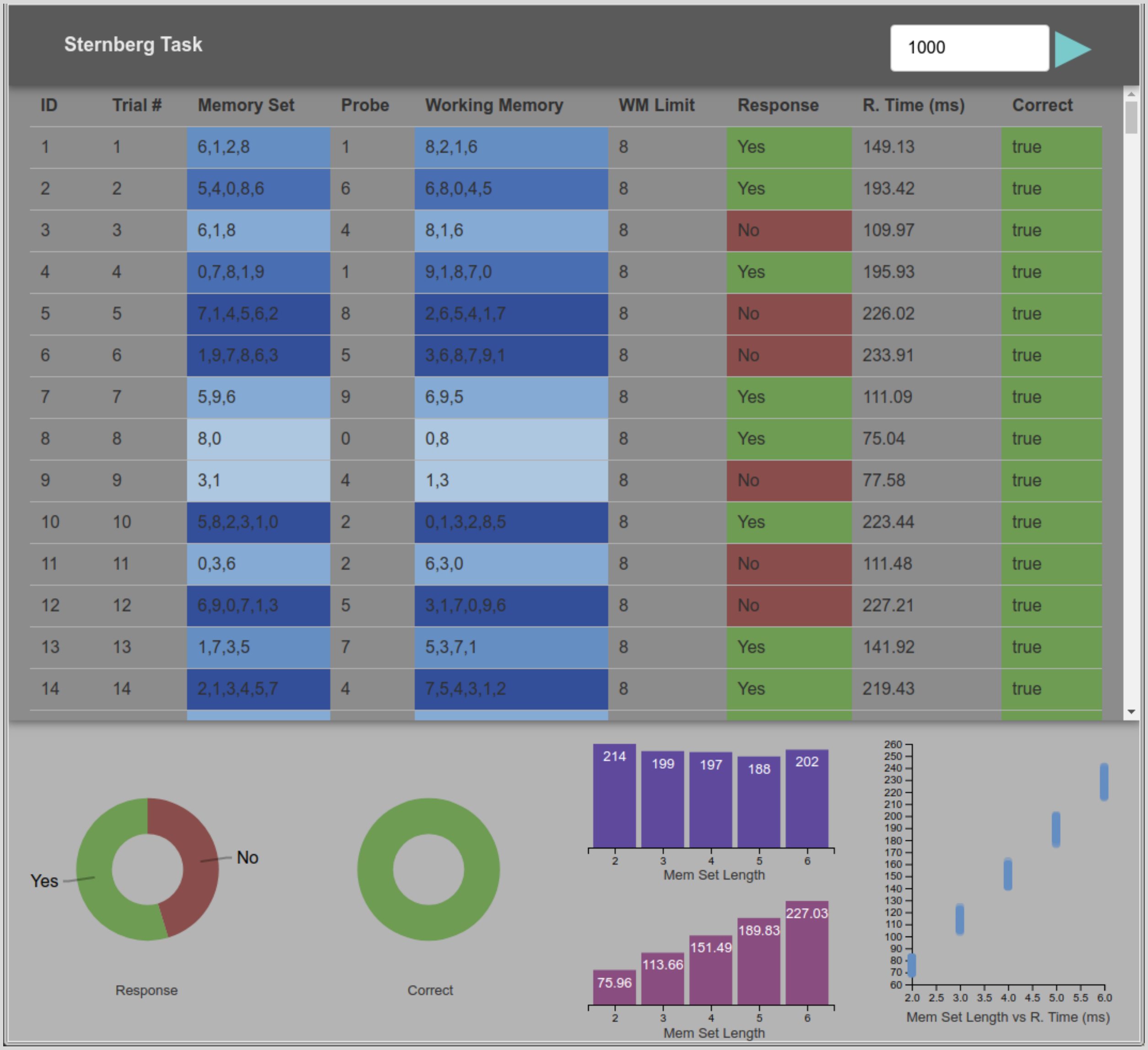
When the web application first starts, information about the Sternberg task is presented.

Sternberg Task

1000

ID	Trial #	Memory Set	Probe	Working Memory	WM Limit	Response	R. Time (ms)	Correct
1	1	0,4,3,1	0	1,3,4,0	5	Yes	156.87	true
2	2	3,0,4	2	4,0,3	5	No	110.79	true
3	3	3,8,4	1	4,8,3	5	No	108.12	true
4	4	3,2,1,8	8	8,1,2,3	5	Yes	151.79	true
5	5	9,4,8	4	8,4,9	5	Yes	112.52	true
6	6	2,3,4,7	7	7,4,3,2	5	Yes	149.11	true
7	7	4,9,7,5	6	5,7,9,4	5	No	155.27	true
8	8	2,6,0,8,5,3	0	3,5,8,0,6	5	Yes	186.84	true
9	9	5,4,1	6	1,4,5	5	No	117.41	true
10	10	6,7,5,2,9	7	9,2,5,7,6	5	Yes	194.84	true
11	11	2,5,7,0,1,6	8	6,1,0,7,5	5	No	186.43	true
12	12	2,5,6,0,3,9	3	9,3,0,6,5	5	Yes	193.71	true
13	13	5,7	0	7,5	5	No	77.22	true
14	14	3,0	4	0,3	5	No	77.41	true

Example showing 1000 trials, note working memory limit is 5



Sternberg Task

1000

ID	Trial #	Memory Set	Probe	Working Memory	WM Limit	Response	R. Time (ms)	Correct
1	1	6,1,2,8	1	8,2,1,6	8	Yes	149.13	true
2	2	5,4,0,8,6	6	6,8,0,4,5	8	Yes	193.42	true
3	3	6,1,8	4	8,1,6	8	No	109.97	true
4	4	0,7,8,1,9	1	9,1,8,7,0	8	Yes	195.93	true
5	5	7,1,4,5,6,2	8	2,6,5,4,1,7	8	No	226.02	true
6	6	1,9,7,8,6,3	5	3,6,8,7,9,1	8	No	233.91	true
7	7	5,9,6	9	6,9,5	8	Yes	111.09	true
8	8	8,0	0	0,8	8	Yes	75.04	true
9	9	3,1	4	1,3	8	No	77.58	true
10	10	5,8,2,3,1,0	2	0,1,3,2,8,5	8	Yes	223.44	true
11	11	0,3,6	2	6,3,0	8	No	111.48	true
12	12	6,9,0,7,1,3	5	3,1,7,0,9,6	8	No	227.21	true
13	13	1,7,3,5	7	5,3,7,1	8	Yes	141.92	true
14	14	2,1,3,4,5,7	4	7,5,4,3,1,2	8	Yes	219.43	true

Example showing 1000 trials, note working memory limit is 8

Final Remarks

Studying how individuals store and retrieve information from STM provides an important window into more general cognitive processing and human functioning. Sternberg's study and his analysis of the data had a major influence on models of memory and cognitive psychology in general. While more recent research has shown that the implications of the data on theories of memory search are not as straightforward as once believed, the experimental approach and theorizing is classic.

References

1. https://en.wikipedia.org/wiki/Serial_memory_processing
2. https://github.com/p10trH/cs510_WM
3. https://p10trh.github.io/cs510_WM/
4. https://rpadgett.butler.edu/nw221/sternberg_lab/index.html
5. http://pages.wustl.edu/dualmechanisms/sternberg-task#sbergdesc
6. https://en.wikipedia.org/wiki/Saul_Sternberg
7. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3945934/
8. https://coglab.cengage.com/labs/sternberg_search.shtml
9. http://www.psych.upenn.edu/~saul/hss.html
10. https://en.wikipedia.org/wiki/Mental_chronometry
11. https://en.wikipedia.org/wiki/Working_memory