RECOGNITION MEMORY EXPERIMENT FRAMEWORK

DESIGNERS:
M. RABE MMRABE@UVIC.CA
DR. S. LINDSAY SLINDSAY@UVIC.CA

DEVELOPER:
A. RICHARDSON
RICHARDSON.ASHLIN@GMAIL.COM

INSTITUTION:
UNIVERSITY OF VICTORIA

Contents

Date: May 12, 2017.

Overview

An online framework for parametric generation of Recognition Memory experiments to support researchers at the University of Victoria. The software is web based, self contained yet comprehensive, and reasonably flexible.

0.1. **Requirements.**

Server-side.

- Host:
  - An ordinary web server with Python/CGI enabled, is required.
  - Note: the system was tested with server: Apache/2.2.23 (Unix).

Client-side.

- For experiment participants:
  - A modern web browser (Firefox, Google Chrome, or Safari) on a desktop computer is required.
  - Note: the system was tested with Chrome v. 57.

- For administrators:
  - An FTP program is required for uploading experiment scripts (and downloading response data).
  - A text editor is required to edit experiment script files.
  - Limited technical knowledge about JavaScript is required to edit or modify experiments.

1. The System

The system, which may be downloaded from

https://github.com/ashlinrichardson/m3m0ry/archive/master.zip

has the following directory structure:

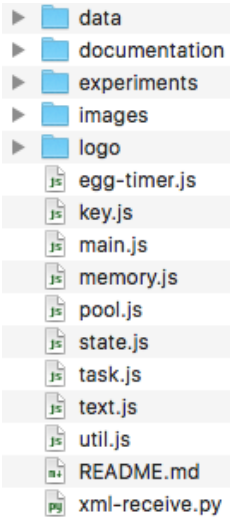

FIGURE 1.1.

where this document lives in the documentation/ folder. Additionally,

- **data**/ will contain CSV data files representing the user experience.
  - If all goes well, a data file should automagically appear in the **data**/ folder, each time a survey/experiment is completed.

  - Upon completion of a survey/experiment, the client-side JavaScript code submits (via util.js::xml_send()) a CSV data file to the web server, which receives the data using CGI/Python (via xml-receive.py).
  - The CSV file is saved with a name reflecting the date/time when the file was recorded, and a randomly-generated string to prevent "collisions".

- **images**/ contains image data used in experiments. To change the image data used in experiments, the administrator should:
  - upload new image data into the **images**/ folder, and
  - modify (an) experiment script(s) to reflect the filenames corresponding to the new image files added.
    - This is slightly technical, unless the image data obeys the usual numbered file-name convention.
- **experiments**/
  - contains a number of sub-folders, one for each of the included examples:
    - delay/
    - feedback/
    - instructions/
    - study-phase/
    - test-phase/
    - my-experiment/
  - Each subfolder contains a file **memory.html**, which always has the contents:

6 lines (5 sloc) | 70 Bytes

```
1 <html>
2 <body>
3 <script src="../../memory.js"></script>
4 </body>
5 </html>
```

FIGURE 1.2.

  - Then, supposing the project is uploaded to the main HTTP directory of a web server with URL **http://my-web-server.com**, the survey in the folder experiments/my-experiment/ represented by **experiments/my-experiment/my-experiment.js** will be accessed by navigating to the following address, in a web browser:
    - **http://my-web-server.com/experiments/my-experiment/memory.html**
  - **To create your own experiment,** we recommend editing the file **my-experiment.js** within the **my-experiment**/ folder
    - **To deploy your experiment on the web,** don't forget to upload your revised my-experiment.js to the server.

2. The Examples

2.1. experiments/instructions.

```
1 /* recognition memory experiment set-up */
2 var my_experiment = function(){
3
4   /* instruction slide */
5   instructions('welcome to the recognition memory experiment framework (press any key to
        continue)')
6
7   /* instruction slide */
```

```
8   instructions('here is what happens when you put in a lot of text— if you put in lots of
        text, it might go over the edge (press any key to continue)')
9
10  /* instruction slide */
11  instructions('this is an instructions slide (press any key to continue)')
12
13  /* instruction slide */
14  instructions('this is another instructions slide (press any key to continue)')
15
16  /* instruction slide -- fixed duration */
17  var x = instructions('this instructions slide will display for 5 seconds: if you press a
        key, it will do nothing')
18  x.set_expiry(5000)
19  x.key_expiry = false
20
21  /* instruction slide -- fixed duration or user intervention */
22  var y = instructions('this instructions slide will display for up to 5 seconds: if you
        press a key, the transition will happen before 5 seconds is up')
23  y.set_expiry(5000)
24  y.key_expiry = true
25
26  /* instruction slide */
27  instructions('this is a normal instructions slide (press any key to continue)')
28
29 }
```

2.2. **experiments/delay.**

```
1  /* recognition memory experiment set-up */
2  var my_experiment = function(){
3
4    /* instruction slide */
5    instructions('first delay phase (5 seconds): please press any key to start')
6
7    /* set up delay task: 5 seconds */
8    delay_task('please type names of as many countries as you can think of in 5 seconds,
         separated by spaces...press any key to begin',
9              5000 /* 5000 mS */)
10
11   /* instruction slide */
12   instructions('second delay phase (10 seconds): please press any key to start')
13
14   /* set up delay task: 10 seconds */
15   delay_task('please type names of as many countries as you can think of in 10 seconds,
         separated by spaces...press any key to begin',
16             10000 /* 10000 mS */)
17
18   /* instruction slide */
19   instructions('all done.. thank you.. please press any key to finish..')
20 }
```

2.3. **experiments/feedback.**

```
1  /* recognition memory experiment set-up */
2  var my_experiment = function(){
3
4    /* instructions */
5    instructions('feedback coming up... please press any key...')
6
7    /* feedback "task" */
8    feedback('please enter your affinity with the last stimulus on a scale of 1-5',
9            [49, 50, 51, 52, 53])
10
11   /* instructions */
```

```
12   instructions('thank you... more feedback coming up... please press any key...')
13
14   /* more feedback "task" */
15   feedback('please enter your affinity with the last stimulus on a scale of 0-9',
16            [49, 50, 51, 52, 53, 54, 55, 56, 57, 48])
17
18   /* instructions */
19   instructions('thank you... multiple choice style feedback coming up... please press any
         key...')
20
21   /* feedback "task" */
22   feedback('skill testing question: 10*10 is: a) 100 b) 200 c) 1000 d) 10000',
23            [65, 66, 67, 68])
24
25   /* instructions */
26   instructions('thank you.. please press any key to finish')
27 }
```

2.4. **experiments/study-phase.**

```
1  /* recognition memory experiment set-up */
2  var my_experiment = function(){
3
4    /* instructions */
5    instructions('study phase coming next:')
6    instructions('please remember each word/image and press any key')
7
8    /* set up a stimulus pool */
9    var p = stimulus_pool()
10
11   /* add images to stimulus pool */
12   for(var i = 0; i < 10; i++){
13     p.add(ctx.imgs[i])
14   }
15
16   /* add words to stimulus pool */
17   p.add('floccinaucinihilipilification')
18   p.add('supercalifragilisticexpialidocious')
19   p.add('umdiddlediddlediddleumdiddlei')
20
21   /* select portion of items from stimulus pool */
22   p.select(2, 2)
23
24   /* set up 'study phase': show selected portions of pool */
25   study_phase(p,  /* stimulus pool */
26               111 /* ISI (optional) */,
27               3000  /* SET (optional) */ )
28 }
```

2.5. **experiments/test-phase.**

```
1  /* recognition memory experiment set-up */
2  var my_experiment = function(){
3
4    /* set up some instruction slides */
5    instructions('study phase: please remember images and press any key')
6
7    /* set up a stimulus pool */
8    var p = stimulus_pool()
9
10   /* add images to stimulus pool */
11   for(var i = 0; i < 10; i++){
12     p.add(ctx.imgs[i])
13   }
```

```
14
15   /* add words to stimulus pool */
16   p.add('floccinaucinihilipilification')
17   p.add('supercalifragilisticexpialidocious')
18   p.add('umdiddlediddlediddleumdiddlei')
19
20   /* selection from stimulus pool (parameters are N, M) */
21   p.select(3, 3)
22
23   /* set up 'study phase': show selected portions of pool */
24   study_phase(p, 111)
25
26   /* some instructions before 'test phase' */
27   instructions('test phase coming up')
28   instructions('when you see an image/word, please press m or n')
29   instructions('please press m if you saw an image/word before')
30   instructions('please press n if you did not see the image/word before')
31
32   /* set up 'test phase' (user input recorded for whole randomized pool) */
33   test_phase(p, 333)
34 }
```

3. Sample Response Data

4. Source Code: Client Side

4.1. egg-timer.js.

```
1  /* via developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope/clearTimeout */
2  var egg_timer = {
3
4    /* callback */
5    setup: function(t_ms){
6
7      /* assert parameter is a number */
8      if(typeof this.timeoutID === "number"){
9        this.cancel()
10     }
11
12     /* what to do when the timer expires */
13     this.timeoutID = window.setTimeout(
14       function(){
15         var now = ctx.get_state()
16         var id = now.id
17         now.ding = true
18         if(now.key_expiry == false || now.expiry_ms > 0){
19           now.expire()
20         }
21       }.bind(this), t_ms
22     )
23   }, cancel: function(){
24     window.clearTimeout(this.timeoutID)
25     this.timeoutID = undefined
26   }
27 }
```

4.2. key.js.

```
1  /* convert from unicode to familiar symbol */
2  function unicode_from_key_event(e){
3    return e.charCode ? e.charCode : e.keyCode
4  }
5
```

```
/* keyboard status array (unicode format) */
var key_unicode = {}

/* keyboard event handler function */
function keyboard_module(){

  /* set up key-down event handler function */
  document.onkeydown = function(e){
    var unicode = unicode_from_key_event(e), key = String.fromCharCode(unicode)
    key_unicode[unicode] = true

    /* ignore caps-lock key */
    if(unicode == 20){

      /* enable this line to debug key codes: console.log("unicode", unicode) */
      return
    }

    /* when are we? */
    var now = ctx.get_state()

    /* record key press, if admissible */
    var admissible_keys = now.get_admissible_keys()
    if(admissible_keys.includes(unicode) || now.type == 'delay'){
      now.record_key_stroke(unicode)
    }

    /* by default, transition from a slide upon key-press */
    var go = true

    /* special treatment for delay task */
    if(now.type == 'delay'){
      if(now.txt == null){

        /* init */
        now.txt = ''
      }
      if(unicode == 8){

        /* backspace */
        var len = now.txt.length
        if(now.txt[len-1] != ' '){
          now.txt = now.txt.substring(0, len - 1)
        }
      }else if(unicode == 0){

        /* null */
      }else{

        /* add character */
        now.txt += key.toLowerCase()
      }

      /* redraw */
      update()
    }

    /* check if this state "requires" keyboard input */
    if(now.require_key() == true){
      if(admissible_keys.includes(unicode)){
        if(!(now.deja == undefined)){
          ctx.questions_total += 1
          if((now.deja == true && unicode == 77) || (now.deja == false && unicode == 78)){
            ctx.questions_correct += 1
          }
```

```
        }
      }else{
        /* block if a key was required but the one entered was not admissible */
        go = false
      }
    }
    if(now.ding == false && now.hold == true){
      go = false
    }

    /* t <-- t + 1 */
    if(now && now.key_expiry && go){
        ctx.clear_tmr()
        now.expire()
    }
  }
  return key_unicode
}
```

4.3. **main.js.**

```
var abs_path = '../../', ctx = canvas.getContext("2d")

/* background color, shape parameter and font size */
document.bgColor = "#FFFFFF", ctx.pad = 20, ctx.font_size = 30

/* canvas dimensions manipulation */
var less = function(x){
  return x - ctx.pad
}

ctx.w = function(){
  return less(window.innerWidth)
}

ctx.h = function(){
  return less(window.innerHeight)
}

/* canvas resize */
function resize(){
  canvas.width = ctx.w(), canvas.height = ctx.h()
}

/* load corporate logo */
ctx.symbol = load_img(abs_path + "logo/uvic_gray.png")

/* algo to draw scaled corporate logo */
ctx.draw_symbol = function(){
  var s_f = 5, pad = this.pad, s = this.symbol
  var ww = window.innerWidth, wh = window.innerHeight
  var w = ww - pad, h = wh - pad, w_s = s.width, h_s = s.height
  var wf = (ww - pad) / (s_f * w_s), lwf = w_s * wf, lhf = h_s * wf
  this.drawImage(s, w - lwf, h - lhf, lwf, lhf)
}

/* access current "state" (a state represents a particular "trial" in an experiment) */
ctx.set_state = function(s){
  last_state = null
  if(ctx.current_state != null){
    last_state = ctx.current_state
  }
  ctx.current_state = s

```

```
  /* sanity check */
  if(s != null){
    s.daddy = last_state
  }
  return(s)
}

/* access present "state" */
ctx.get_state = function(){
  return ctx.current_state
}

/* trigger update/plotting from window resize event */
window.onresize = function(event){
  update()
}

/* update the canvas (present the current "trial") */
function update(){
  resize()
  var now = ctx.get_state()
  if(now){
    now.show(ctx)
  }
}

/* "in" hook: plot the current trial */
window.onload = function(){
  update()
}

/* set up timer to coordinate transitions between trials */
ctx.egg_timer = egg_timer

ctx.clear_tmr = function(){
  ctx.egg_timer.cancel()
}

ctx.init_tmr = function(t_ms){
  ctx.egg_timer.setup(t_ms)
}

/* initialize reference to first and most-recently-initialized trials */
ctx.last_new_state = null, ctx.first_new_state = null

/* count number of questions answered correctly (this is redundant) */
ctx.questions_correct = 0, ctx.questions_total = 0

/* this function sets up the experiment (according to the user function my_experiment)
and we trigger this function after all the images have loaded. */
function run_after_loading_images(){

  /* set up an experiment according to user specs/code */
  my_experiment(ctx)

  /* in this part, we should record only the images that we actually need */

  instructions('thank you')

  ctx.last_state = ctx.last_new_state, ctx.first_state = ctx.first_new_state

  /* start at the very beginning, it's a very good place to start.. */
  ctx.set_state(ctx.first_state)

  /* respond to keyboard events */
```

```
  key_unicode = keyboard_module()

  /* start "stopwatch" */
  ctx.t0 = window.performance.now()

  /* go */
  ctx.get_state().start()
}

/* load some image files: need to change if the image database changes */
var n_imgs = 200, n_imgs_loaded = 0

/* load image data */
function load_img(fn){
  var img = new Image()
  img.onload = function(){

    /* have all images been loaded? */
    if(++n_imgs_loaded == n_imgs){

      /* proceed to init the experiment */
      run_after_loading_images()
    }
  }
  /* load the image */
  img.src = fn
  return img
}

/* load all of the image data */
ctx.load_imgs = function(n_imgs){

  /* ideally would only load the ones used */
  var imgs = new Array()
  for(var i = 1; i <= n_imgs; i++){
    var img_fn = abs_path + 'images/' + i + '.jpg'
    var my_img = load_img(img_fn)
    my_img.fn = 'images/' + i + '.jpg'
    imgs.push(my_img)
  }
  ctx.imgs = imgs
  return ctx.imgs
}

/* keep track of the "task-index" as the experiment is intialized */
var next_task_id = 0

/* this line "makes everything go" */
var my_images = ctx.load_imgs(n_imgs)
```

4.4. **memory.js.**

```
/* sleep function */
function sleep(ms){
  return new Promise(resolve => setTimeout(resolve, ms))
}

var js_added = -1, deps = []

/* j4v4scr1pt 4n4l0g 0f 1nclud3 st4t3m3nt */
function add_js(fn){
  var body = document.getElementsByTagName('body')[0], s = document.createElement('script')
  s.async = false, s.src = fn + '.js'

```

```
  /* wait until script is loaded before proceeding.. */
  s.onload = function(){
    if(++js_added < deps.length){
      add_js(deps[js_added])
    }
  }
  body.appendChild(s)
}

/* c411 411 th3 ch1ldr3n */
dependencies = ['text', 'key', 'util', 'task', 'pool', 'state', 'egg-timer']
for(var d in dependencies){
  deps.push('../../' + dependencies[d])
}
deps.push('my-experiment')
deps.push('../../main')
add_js(deps[0], '')
```

4.5. **pool.js.**

```
/* stimulus pool - object that has words or images added to it. Selections drawn randomly
    for "study phase"
by draw() method. That selection is shuffled back into the deck, for the "test phase" */
var next_pool_id = 0
function pool(){
  this.is_pool = true, this.pool_id = next_pool_id, this.ctx = ctx, this.stimuli = new Array
      ()
  ++ next_pool_id

  /* add a stimulus to the pool */
  this.add = function(stim){
    this.stimuli.push(stim)
    return stim
  }

  /* set number of samples for study phase */
  this.set_n = function(n){
    this.n = n
  }

  /* set number of additional samples to be included for test phase */
  this.set_m = function(m){

    /* subsequently to drawing "n" items from the pool (without replacement),
       a further "m" samples are drawn from the pool. For the test phase, the
      "n" and "m" selections are mixed together and shuffled. */
    this.m = m
  }

  /* get */
  this.get_n = function(){
    return this.n
  }

  /* get */
  this.get_m = function(){
    return this.m
  }

  /* remove any "blank" elements that appeared from drawing elements without
    replacement */
  this.remove_blanks = function(){
    this.stimuli = this.stimuli.filter(function(){return true})
  }
```

```
43
44  /* pseudorandom selection of size "n" */
45  this.draw_n = function(){
46
47    if(this.selection_n){
48      console.log('error: n-selection already made from this pool.')
49      return null
50    }
51    var n = parseInt(this.get_n())
52    if(n > this.stimuli.length){
53      console.log('error: n > this.stimuli.length')
54      return null
55    }
56    this.selection_n = new Array()
57    var rem = this.stimuli.length
58    for(var i = 0; i < n; i++){
59      var qx = rand() * parseFloat(rem --), idx = parseInt(qx)
60      this.selection_n.push(this.stimuli[idx])
61      delete this.stimuli[idx]
62      this.remove_blanks()
63    }
64  }
65
66  /* pseudorandom selection of size "m" */
67  this.draw_m = function(){
68
69    if(this.selection_m){
70      console.log('error: m-selection already made from this pool.')
71      return null
72    }
73    var m = parseInt(this.get_m())
74    if(m > this.stimuli.length){
75      console.log('error: m > this.stimuli.length')
76      return null
77    }
78    this.selection_m = new Array()
79    var rem = this.stimuli.length
80    for(var i = 0; i < m; i++){
81      var qx = rand() * parseFloat(rem --), idx = parseInt(qx)
82      this.selection_m.push(this.stimuli[idx])
83      delete this.stimuli[idx]
84      this.remove_blanks()
85    }
86  }
87
88  /* for initializing a test phase: mix "N"-selection and "M"-selection together */
89  this.reshuffle = function(){
90    var to_shuffle = [], i = 0
91
92    /* add the "N"-selection */
93    for(i = 0; i < this.selection_n.length; i++){
94      var dat_i = new Array()
95      dat_i.push(this.selection_n[i])
96      dat_i.push(true)
97      to_shuffle.push(dat_i)
98    }
99
100   /* add the "M"-selection */
101   for(i = 0; i < this.selection_m.length; i++){
102     var dat_i = new Array()
103     dat_i.push(this.selection_m[i])
104     dat_i.push(false)
105     to_shuffle.push(dat_i)
106   }
107
```

```
    /* "shuffle"-- randomize the ordering of the combined array */
    var shuffled = new Array(), deja_vu = new Array(), rem = to_shuffle.length
    while((rem --) > 0){
      var idx = parseInt(rand() * parseFloat(rem)), dat_i = to_shuffle[idx]
      shuffled.push(dat_i[0])
      deja_vu.push(dat_i[1])
      delete to_shuffle[idx]
      to_shuffle = to_shuffle.filter(function(){return true})
    }
    return [shuffled, deja_vu]
  }

  this.draw = function(){
    this.draw_n()
    this.draw_m()
    this.reshuffle()
  }

  /* set N, M parameters and make a selection */
  this.select = function(n,m){
    this.set_n(n)
    this.set_m(m)
    this.draw()
  }

  /* end of "pool::pool()" */
  return this
}

function stimulus_pool(){
  return new pool()
}
```

4.6. **state.js.**

```
/* global counter for states/ AKA frames/ AKA slides */
var state_id = -1

function get_id(){
  return ++ state_id
}

/* reference to 2d canvas graphics context */
function get_ctx(){
  return canvas.getContext("2d") //document.getElementsByTagName("canvas")[0].getContext("2d
      ");
}

/* state: generic object representing trial (like a card in "hypercard") */
function state(expiry_ms  =     0,  /* max. presentation time (mS) */
               key_expiry =  true,  /* force expiry by key-press (true <--> on) */
               intvl_ms   =     0,  /* interval btwn stimuli.. (ISI) 'blank slide' */
               img_idx    =    -1,  /* image data (if any) */
               txt        =  null,  /* text data (if any) */
               successor  =  null){
  var ctx = get_ctx()
  this.action = null, this.ding = false, this.hold = false, this.id = get_id()

  this.hold_on = function(){
    this.hold = true
  }

  /* is a key-press required to transition? */
  this.key_required = false
```

```
29
30  /* array to store admissible key-codes for data entry or transition to next "slide" */
31  this.admissible_keys = [77, 78]
32
33  this.get_admissible_keys = function(){
34    return this.admissible_keys
35  }
36
37  this.clear_admissible_keys = function(){
38    this.admissible_keys = new Array()
39  }
40
41  this.add_admissible_key = function(k){
42    this.admissible_keys.push(k)
43  }
44
45  /* this array will record the keystroke data received while residing in this state */
46  this.key_strokes = new Array()
47
48  this.record_key_stroke = function(k){
49    this.key_strokes.push(k)
50  }
51
52  this.set_pool_id = function(pid){
53    this.pool_id = pid
54  }
55
56  this.get_pool_id = function(){
57    return this.pool_id ? this.pool_id : ""
58  }
59
60  /* keep a reference to this state, if it's the first one ever.. */
61  if(ctx.first_new_state == null){
62    ctx.first_new_state = this
63  }
64
65  /* only applies if there's a "next" trial, if this is a trial */
66  this.intvl_ms = intvl_ms
67
68  /* numeric */
69  this.expiry_ms = expiry_ms
70
71  /* boolean */
72  this.key_expiry = key_expiry
73
74  /* global image index (images added as member of ctx) */
75  this.img_idx = img_idx, this.successor = null, this.predecessor = ctx.last_new_state
76
77  this.require_key = function(){
78    return this.key_required
79  }
80
81  var id = (this.predecessor == null) ? -1 : this.predecessor.id
82  ctx.last_new_state = this
83
84  if(this.predecessor != null){
85    this.predecessor.set_successor(this)
86  }
87
88  /* where are we going? */
89  this.set_successor = function(s){
90    this.successor = s
91  }
92
93  /* plot text or images */
```

```
94  this.show = function(){
95    if(this.action){
96      this.action(this)
97    }
98    var ctx = get_ctx()
99    ctx.clearRect(0, 0, ctx.w(), ctx.h())
100
101   /* bottom text */
102   if(this.txt2 && (!this.wrd_stim)){
103     // wrap_text(this.txt2, ctx, ctx.h() - (2 * ctx.font_size+20));
104   }
105
106   if(this.txt2){
107     wrap_text(this.txt2, ctx, ctx.h() - (2 * ctx.font_size + 20))
108   }
109
110   /* upper text */
111   if(this.txt){
112     wrap_text(this.txt, ctx, 0)
113   }
114
115   /* img or middle text (if word stim) */
116   if(this.img_stim){
117     draw_img(this.img_stim, ctx)
118   }
119
120   /* might need the wrap_text back on for long strings.. */
121   if(this.wrd_stim != null){  // wrap_text(this.wrd_stim, ctx, ctx.h()/2);
122
123     /* no wrap */
124     centre_text(this.wrd_stim)
125   }
126
127   /* logo of no image/ lower text present */
128   if(!this.txt2){
129     ctx.draw_symbol()
130   }
131 }
132
133 /* state expires by timer or key press */
134 this.set_expiry = function(t_ms){
135
136   /* follow clock or key to keep the show going */
137   this.expiry_ms = t_ms
138
139   /* state expires by key press */
140   if(t_ms <= 0){
141     this.key_expiry = true
142   }
143 }
144
145 /* enter a state (begin) */
146 this.start = function(){
147   var ctx = get_ctx()
148
149   if(this == ctx.last_state){
150       /* window.location.href == http://domain/memory/examples/test_phase/memory.html */
151       var href = window.location.href
152
153       /* go through all the states and record (in string format) the contents, as we'd
              like it to appear on the server */
154       var state_i = ctx.first_state, state_index = 0, message = "url,event_id,task_id,
              task_type,trial_id,duration(mS),start(yyyy:mm:dd:hh:mn:ss:mls),end(yyyy:mm:dd:hh
              :mn:ss:mls),isi,set,stim_type,stim_id,stim_pool_id,response\n"
```

```
155         for(var state_i = ctx.first_state; state_i != ctx.last_state; state_i = state_i.
                successor){
156           var stim_type = null, my_stim  = null, pi = ""
157
158           /* the right way to check if a variable is undefined or not */
159           if(typeof state_i.pool_id !== 'undefined'){
160             pi = JSON.parse(JSON.stringify(state_i.pool_id))
161           }
162
163           if(state_i.wrd_stim){
164             stim_type = "word", my_stim = state_i.wrd_stim
165           }
166
167           if(state_i.img_stim){
168             stim_type = "image", my_stim = state_i.img_stim.fn
169           }
170
171           if(!stim_type){
172             stim_type = ""
173           }
174
175           if(!my_stim){
176             my_stim = ""
177           }
178
179           /* for a given "state", record a line of data */
180           message += href + ","
181           message += state_index.toString() + ","       /* event_id: global index / line
                  number */
182           message += state_i.task_id + ","              /* task_id */
183           message += state_i.type + ","                 /* task_type */
184           message += state_i.trial_id + ","             /* trial_id */
185           message += Math.round(10. * (state_i.t1 - state_i.t0)) / 10. + ","
186           message += parse_date_time(state_i.start_date_time).toString() + ","
187           message += parse_date_time(state_i.end_date_time).toString() + ","
188           if(state_i.type == 'isi'){
189             message += state_i.expiry_ms.toString()
190           }
191           message += ","                                /* ISI */
192           if(!state_i.expiry_ms){
193             state_i.expiry_ms = ""
194           }
195           message += state_i.expiry_ms.toString() + "," /* SET */
196           message += stim_type.toString() + ","         /* stim_type */
197           message += my_stim.toString() + ","           /* stim_id */
198           message += pi.toString() + ","                /* stimulus-pool id */
199
200           var response = ""
201           for(var k in state_i.key_strokes){
202             response += String.fromCharCode(state_i.key_strokes[k])
203           }
204           message += response + ""                      /* response */
205
206           /* add a newline character */
207           message += "\n"
208           ++ state_index
209         }
210
211         /* remove last three elements from the array: take the page and navigate to: ../../
                xml-receive.py == http://domain/memory/xml-receive.py */
212         var words = href.split('/')
213         var nwords = words.length
214         var target = words.splice(0, nwords-3).join('/') + '/xml-receive.py'
215
216         /* send the message to the server-side script at URL: target */
```

```
        xml_send(message, target)
    }

    var ctx = get_ctx()

    /* start the clock.. */
    this.t0 = window.performance.now(), this.start_date_time = date_time()

    /* clear the timer */
    ctx.clear_tmr()

    /* plot the current trial */
    this.show(ctx)

    /* start the timer? */
    if(this.expiry_ms > 0){
      ctx.init_tmr(this.expiry_ms, this.expire)
    }
    return null
  }

  /* pr0c33d t0 th3 n3xt 5+4t3 */
  this.expire = function(){
    var ctx = get_ctx()

    /* st0p 4ll th3 cl0ck5 */
    ctx.clear_tmr()

    /* r3c0rd st0p t1m3 */
    this.end_date_time = date_time(), this.t1 = window.performance.now()
    var txt = this.txt, suc_txt = null, suc = this.successor

    if(suc!=null && suc.txt !=null){
      suc_txt = suc.txt
    }

    /* enter next state */
    if(this.successor!=null){
      ctx.set_state(this.successor)
      ctx.get_state().start()
    }
  }
  return this
}
```

4.7. **task.js.**

```
/* Event hierarchy: 1) Experiment (includes multiple tasks) 2) Task (includes multiple
    trials) 3) Trial (each task includes multiple basic events) */

/* instructions task (show a slide with a message on it) */
function instructions(txt){
  var my_task_id = next_task_id++

  /* initialize generic "trial" object */
  var x = new state()

  /* set associated text field */
  x.txt = txt

  /* no timer for the trial */
  x.set_expiry(0)
  x.type = 'instructions', x.task_id = my_task_id, x.trial_id = 0
  return x
```

```
}

/* study phase, formerly known as orientation task: multiple 'trials' / events occur here.. random selection of inputs... (for the test phase, the random selection is shuffled back into the pool).. */
function study_phase(my_pool, isi=0, time_limit=0, extra_feedback=false, extra_feedback_message="", extra_feedback_keys=[]){

  /* the above constructor (same with test_phase) can accept either a single stimulus pool (pool()),
    or an array of stimulus pools (pool()) */
  var my_pools = []
  if(my_pool.is_pool){
    my_pools.push(my_pool)
  }else{
    my_pools = my_pool
  }

  var trial_index = -1, my_task_id = next_task_id++
  this.ctx = ctx, this.p = my_pools, this.pool_ids = new Array()

  var my_selection = new Array()
  for(var a_pool in my_pools){
    var my_pool = my_pools[a_pool]
    this.pool_ids.push(my_pool.pool_id)
    for(var i  in my_pool.selection_n){
      var extra_feedback_this_slide = false
      if(extra_feedback != false){
        if(0 == i % parseInt(extra_feedback)){
          extra_feedback_this_slide = true
        }
      }
      my_selection.push([my_pool.selection_n[i], my_pool.pool_id, extra_feedback_this_slide])
    }
  }

  /* randomize the order of the array */
  shuffle(my_selection)

  for(var selection_ind in my_selection){

    /* increment the trial-index counter */
    ++ trial_index

    var a_selection = my_selection[selection_ind]

    /* data (word or image) assigned to "trial" */
    var data = a_selection[0], p_id = a_selection[1], extra_feedback_this_slide = a_selection[2]

    /* if ISI was set, prefix with a "blank" slide */
    if(isi > 0){
      var x = new state()
      x.set_expiry(isi)
      x.type = 'isi', x.wrd_stim = "", x.trial_id = trial_index, x.task_id = my_task_id
      x.set_pool_id(my_pool.pool_id)
      x.clear_admissible_keys()
      x.key_expiry = false
    }

    /* initialize generic "trial" object for each case */
    var x = new state()
    if(time_limit <= 0){
      x.set_expiry(0)
```

```
76        x.key_required = false
77      }else{
78        x.set_expiry(time_limit)
79        x.key_required = false
80      }
81
82      /* discern by image or word, respectively */
83      if( typeof(data) === 'object'){
84        x.img_stim = data
85      }else if(typeof(data) === 'string'){
86        x.wrd_stim = data
87      }
88      x.type = 'study_phase', x.trial_id = trial_index, x.task_id = my_task_id
89      x.set_pool_id(p_id)
90      if(extra_feedback_this_slide){
91        var x_f = feedback(extra_feedback_message, extra_feedback_keys)
92      }
93    }
94    return this
95  }
96
97  /* test phase, formerly known as recognition task - for this phase,
98  the random selection is shuffled back into the pool -- all elements
99  from the pool are shown (feedback is recorded).. */
100 function test_phase(my_pool, isi=0, time_limit=0, extra_feedback=false,
        extra_feedback_message="", extra_feedback_keys=[]){
101   var my_pools = []
102   if(my_pool.is_pool){
103     my_pools.push(my_pool)
104   }else{
105     my_pools = my_pool
106   }
107
108   var trial_index = -1, my_task_id = next_task_id++
109   this.ctx = ctx, this.p = my_pools, this.pool_ids = new Array()
110
111   var my_selection = new Array()
112   for(var a_pool in my_pools){
113     var my_pool = my_pools[a_pool]
114     this.pool_ids.push(my_pool.pool_id)
115     var trial_index = -1, shuffled_data = my_pool.reshuffle(), shuffled = shuffled_data[0],
          deja_vu = shuffled_data[1]
116     for(var i  in shuffled){
117       var extra_feedback_this_slide = false
118       if(extra_feedback != false){
119         if(0 == i % parseInt(extra_feedback)){
120           extra_feedback_this_slide = true
121         }
122       }
123       my_selection.push([shuffled[i], my_pool.pool_id, deja_vu[i], extra_feedback_this_slide
            ])
124     }
125   }
126   shuffle(my_selection)
127
128   for(var selection_ind in my_selection){
129     ++ trial_index
130
131     var a_selection = my_selection[selection_ind]
132     var data = a_selection[0], p_id = a_selection[1], deja = a_selection[2],
          extra_feedback_this_slide = a_selection[3]
133
134     /* if ISI was set, prefix with a "blank" slide */
135     if(isi > 0){
136       var x = new state()
```

```
137       x.set_expiry(isi)
138       x.type = 'isi', x.wrd_stim = "", x.trial_id = trial_index, x.task_id = my_task_id
139       x.set_pool_id(p_id)
140       x.clear_admissible_keys()
141       x.key_expiry = false
142     }
143
144     var x = new state()
145     x.key_required = true
146     if(time_limit <= 0){
147       x.set_expiry(0)
148     }else{
149       x.set_expiry(time_limit)
150     }
151
152     /* record within the object: do we have deja-vu? */
153     x.deja = deja
154
155     /* word or image? */
156     if( typeof(data) === 'object'){
157       x.img_stim = data
158     }else if(typeof(data) ==='string'){
159       x.wrd_stim = data
160     }
161     x.type = 'test_phase', x.trial_id = trial_index, x.task_id = my_task_id
162     x.set_pool_id(p_id)
163
164     if(extra_feedback_this_slide){
165       var x_f = feedback(extra_feedback_message, extra_feedback_keys)
166     }
167   }
168   var m = 'Thank you for completing this section. ', end = instructions(m)
169
170   end.action = function(me){
171     var msg = m + 'Your score: ' + ctx.questions_correct.toString() + '/' + ctx.
          questions_total.toString() + ". Please press any key."
172     me.txt = msg
173   }
174   return this
175 }
176
177 /* previously known as feedback task */
178 function feedback(txt, keys){
179   var my_task_id = next_task_id ++
180
181   var x = new state()
182   x.set_expiry(0)
183   x.txt = txt, x.key_required = true
184   x.clear_admissible_keys()
185   for(var i in keys){
186     x.add_admissible_key(keys[i])
187   }
188   x.type = 'feedback', x.trial_id = 0, x.task_id = my_task_id
189 }
190
191 /* list as many countries as possible during e.g., a 3-minute period (default, 30s) */
192 function delay_task(txt, delay_time=30000, isi_=500){
193   var my_task_id = next_task_id ++, isi = parseInt(isi_)
194
195   /* if ISI was set, prefix with a "blank" slide */
196   if(isi > 0){
197     var x = new state()
198     x.set_expiry(isi)
199     x.type = 'isi', x.wrd_stim = "", x.trial_id = 0, x.task_id = my_task_id
200     x.clear_admissible_keys()
```

```
    x.key_expiry = false
  }

  var y = instructions(txt)
  if(true){
    /* time [mS] */
    var x = new state()
    x.set_expiry(delay_time)
    x.key_expiry = false, x.txt = '', x.type = 'delay', x.trial_id = 0, x.task_id =
        my_task_id
  }
  return this
}
```

4.8. **text.js.**

```
/* wrap text around a window region— via ashblue */
function wrap_text(s, ctx, start_y=0){
  var myX = 10, myY = 50, line = '', lines = [], w = ctx.w(), h = ctx.h(), line_test = '',
      words = s.split(' '), font_size = ctx.font_size
  ctx.font = font_size +'px Arial'

  /* place words one by one */
  for(var j = 0; j < words.length; j++){
    line_test = line + words[j] + ' '

    /* wrap if over the edge */
    if(ctx.measureText(line_test).width > w){
      myY = lines.length * font_size + font_size
      lines.push({text: line, height: myY})
      line = words[j] + ' '
    }else{
      line = line_test
    }
  }

  /* catch last line if something left over */
  if(line.length > 0){
    current_y = lines.length * font_size + font_size
    lines.push({text: line.trim(), height: current_y})
  }

  /* plot text */
  for(var j = 0, len = lines.length; j < len; j++){
    ctx.fillText(lines[j].text, 0, lines[j].height + start_y)
  }
}

/* write centred text */
function centre_text(s){
  var font_size = ctx.font_size, textString = s
  ctx.font = 30 + 'px Arial'
  textWidth = ctx.measureText(textString).width
  ctx.fillText(textString, (canvas.width / 2) - (textWidth / 2), canvas.height / 2)
}
```

4.9. **util.js.**

```
/* cr34t3 a c4nv4s wh3r3 th3 m4g1c h4pp3ns */
var canvas = document.createElement('canvas')
document.body.appendChild(canvas)

/* get date and time */
function date_time(){
```

```
  return new Date()
}

/* seed for rand() below */
var seed = 5

/*random-number generator http://indiegamr.com/generate-repeatable-random-numbers-in-js/ :
    initial seed.. in order to work 'Math.seed' must NOT be undefined, so in any case, you
    HAVE to provide a Math.seed */
function rand(max, min){
  max = max || 1, min = min || 0
  seed = (seed * 9301 + 49297) % 233280
  return min + (seed / 233280) * (max - min)
}

/* pad to length n (with 0's on the left) */
function pad_n(x, n){
  var s = parseInt(trim(x)).toString(), m = s.length, d = n - m
  if(d > 0){
    s += '0'.repeat(d)
  }
  return s
}

/* via stackoverflow.com/users/4321/jw */
function get_keys(dictionary){

  /* keys recursive */
  var keys = []

  /* filter for direct ancestors */
  for(var key in dictionary){
    if(dictionary.hasOwnProperty(key)){
      keys.push(key)
    }
  }
  return keys
}

/* draw an image */
function draw_img(x, ctx){
  var cf = 4 * ctx.font_size
  var h = ctx.h() - cf, w = ctx.w()
  var lw = x.width, lh = x.height
  var sf = Math.min(w, h) / Math.max(lw, lh)
  var a = (w - lw * sf) / 2, b = (h - lh * sf) / 2
  var c = lw * sf, d = lh * sf, df =  (-20 + cf / 2)
  ctx.drawImage(x, a, b + df, c, d)
}

/* write the above to a standardized format */
function parse_date_time(today){

  /* most significant units first */
  var bits = [today.getFullYear(),
              today.getMonth() + 1,
              today.getDate(),
              today.getHours(),
              today.getMinutes(),
              today.getSeconds(),
              today.getMilliseconds()]

  /* pad with zeros */
  for(var i = 0; i < bits.length; i++){
    var n_pad = 2
```

```
    if(i == 0){
      n_pad = 4
    }
    if(i == 6){
      n_pad = 3
    }
    var bts = bits[i].toString()
    bits[i] = pad_n(bts, n_pad)
  }
  return(bits.join(':'))
}

/* "faster trim" via blog.stevenlevithan.com */
function trim(s){
  return s.toString().replace(/^\s\s*/,'').replace(/\s\s*$/,'')
}

/* send text format data (string s) via XML to receive script at url (string): xml_receive_script_url  */
function xml_send(s, xml_receive_script_url){

  /* xml http request object */
  var xhr = (window.XMLHttpRequest) ? new XMLHttpRequest() : new activeXObject("Microsoft.XMLHTTP")
  var data = new FormData()
  data.append("data", s)
  xhr.open('post', xml_receive_script_url, true)
  xhr.send(data)
}

/* Shuffle array in place, via http://stackoverflow.com/questions/6274339/how-can-i-shuffle-an-array
 * @param {Array} a items The array containing the items. */
function shuffle(a) {
  var j, x, i
  for(i = a.length; i; i--){

    /* use our seeded random number generator, so we get the same results every time */
    j = Math.floor(rand() * (1. * i))  /* j = Math.floor(Math.random() * i) */
    x = a[i - 1]
    a[i - 1] = a[j]
    a[j] = x
  }
}
```

5. Source Code: Server Side

The folder data/ in the directory structure: if it doesn't yet exist, the server-side python code will create it.

5.1. xml-receive.py.

```
#!/usr/bin/python
''' server-side python-CGI script to receive text data sent over
the internet by the client-side function util.js::xml_send()'''
import os
import cgi
import uuid
import datetime

# create /data folder if it does not yet exist
dat_f = os.getcwd() + '/data/'
if not os.path.exists(dat_f):
    os.mkdir(dat_f)

```

```
# retrieve CGI form data
dat = None
try:
    dat = str(cgi.FieldStorage().getvalue('data'))
except:
    pass

# write the data to file in the data/ folder
if dat:
    fn = dat_f + str(datetime.datetime.now().isoformat())
    open(fn + '_' + str(uuid.uuid4().hex) + '.txt', 'wb').write(dat)
```

6. Recommendations For Further Improvements

Here's a short point-form list of possible improvements to the software:

- Finish drag-and drop implementation, that
  - does not allow invalid experiments to be constructed
  - removes any technicality from the process of coding an experiment
- Smarter image loading
  - Only load the images that are actually used in the experiment
  - Automagically detect available images from folder