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 15, 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:

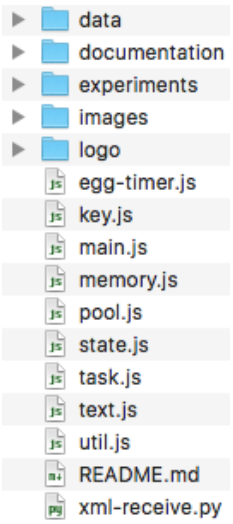

Figure 1.1.

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.**

```
/* recognition memory experiment set-up */
var my_experiment = function(){

  /* instruction slide */
  instructions('welcome to the recognition memory experiment framework (press any key to continue)')

  /* instruction slide */
  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)')

  /* instruction slide */
  instructions('this is an instructions slide (press any key to continue)')

  /* instruction slide */
  instructions('this is another instructions slide (press any key to continue)')

  /* instruction slide -- fixed duration */
  var x = instructions('this instructions slide will display for 5 seconds: if you press a key, it will do nothing')
  x.set_expiry(5000)
  x.key_expiry = false

  /* instruction slide -- fixed duration or user intervention */
  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')
  y.set_expiry(5000)
  y.key_expiry = true

  /* instruction slide */
  instructions('this is a normal instructions slide (press any key to continue)')

}
```

2.2. **experiments/delay.**

```
/* recognition memory experiment set-up */
var my_experiment = function(){

  /* instruction slide */
  instructions('first delay phase (5 seconds): please press any key to start')

  /* set up delay task: 5 seconds */
  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',
         5000 /* 5000 mS */)

  /* instruction slide */
  instructions('second delay phase (10 seconds): please press any key to start')

  /* set up delay task: 10 seconds */
  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',
         10000 /* 10000 mS */)

  /* instruction slide */
  instructions('all done.. thank you.. please press any key to finish..')
}
```

2.3. **experiments/feedback.**

```
/* recognition memory experiment set-up */
var my_experiment = function(){

  /* instructions */
  instructions('feedback coming up... please press any key...')

  /* feedback "task" */
  feedback('please enter your affinity with the last stimulus on a scale of 1-5',
          [49, 50, 51, 52, 53])

  /* instructions */
  instructions('thank you... more feedback coming up... please press any key...')

  /* more feedback "task" */
  feedback('please enter your affinity with the last stimulus on a scale of 0-9',
           [49, 50, 51, 52, 53, 54, 55, 56, 57, 48])

  /* instructions */
  instructions('thank you... multiple choice style feedback coming up... please press any
      key...')

  /* feedback "task" */
  feedback('skill testing question: 10*10 is: a) 100 b) 200 c) 1000 d) 10000',
           [65, 66, 67, 68])

  /* instructions */
  instructions('thank you.. please press any key to finish')
}
```

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

```
/* recognition memory experiment set-up */
var my_experiment = function(){

  /* instructions */
  instructions('study phase coming next:')
  instructions('please remember each word/image and press any key')

  /* set up a stimulus pool */
  var p = stimulus_pool()

  /* add images to stimulus pool */
  for(var i = 0; i < 10; i++){
    p.add(ctx.imgs[i])
  }

  /* add words to stimulus pool */
  p.add('floccinaucinihilipilification')
  p.add('supercalifragilisticexpialidocious')
  p.add('umdiddlediddlediddleumdiddlei')

  /* select portion of items from stimulus pool */
  p.select(2, 2)

  /* set up 'study phase': show selected portions of pool */
  study_phase(p,  /* stimulus pool */
              111 /* ISI (optional) */,
              3000  /* SET (optional) */ )
}
```

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

```
/* recognition memory experiment set-up */
var my_experiment = function(){

  /* set up some instruction slides */
  instructions('study phase: please remember images and press any key')

  /* set up a stimulus pool */
  var p = stimulus_pool()

  /* add images to stimulus pool */
  for(var i = 0; i < 10; i++){
    p.add(ctx.imgs[i])
  }

  /* add words to stimulus pool */
  p.add('floccinaucinihilipilification')
  p.add('supercalifragilisticexpialidocious')
  p.add('umdiddlediddlediddleumdiddlei')

  /* selection from stimulus pool (parameters are N, M) */
  p.select(3, 3)

  /* set up 'study phase': show selected portions of pool */
  study_phase(p, 111)

  /* some instructions before 'test phase' */
  instructions('test phase coming up')
  instructions('when you see an image/word, please press m or n')
  instructions('please press m if you saw an image/word before')
  instructions('please press n if you did not see the image/word before')

  /* set up 'test phase' (user input recorded for whole randomized pool) */
  test_phase(p, 333)
}
```

2.6. **experiments/my-experiment.**

```
/* recognition memory experiment set-up: customized/ complex experiment */
var my_experiment = function(){

  /* set up some instruction slides */
  instructions('study phase: please remember words/images and press any key')

  /* set up a stimulus pool */
  var p1 = stimulus_pool()

  /* add images to stimulus pool */
  for(var i = 65; i < 70; i++){
    p1.add(ctx.imgs[i])
  }

  /* set up a stimulus pool */
  var p2 = stimulus_pool()

  /* add images to stimulus pool */
  for(var i = 100; i < 105; i++){
    p2.add(ctx.imgs[i])
  }

  /* add words to stimulus pool */
  p1.add('floccinaucinihilipilification')
  p1.add('supercalifragilisticexpialidocious')
  p1.add('equanimity')

  /* add words to second stimulus pool */
  p2.add('compassion')
  p2.add('dogovarivatsya')
  p2.add('umdiddlediddlediddleumdiddlei')

  /* selection from stimulus pool (parameters are N, M) */
  p1.select(3, 3)
  p2.select(3, 3)

  /* need to bundle the two pools together, into an array */
  var two_pools = [p1, p2]

  /* set up 'study phase': show selected portions of pool */
  study_phase(two_pools,
              111, /* ISI */
              2000 /* SET */ )

  /* some instructions before 'test phase' */
  instructions('test phase coming up')
  instructions('when you see an image/word, please press m or n')
  instructions('please press m if you saw an image/word before')
  instructions('please press n if you did not see the image/word before')

  /* set up 'test phase' (user input recorded for whole randomized pool) */
  test_phase(two_pools, /* stimulus pools */
             1000, /* ISI */
             4000, /* SET */
             4, /* extra feedback (one for every 3 slides, approx.) */
             "Did you like the last picture? A=yes, B=no, C=maybe, D=not sure", /* message
                 for extra feedback */
             [65, 66, 67, 68] /* accepted keypresses for extra feedback */ )
}
```

3. Sample Response Data

4. Source Code: Client Side

4.1. egg-timer.js.

```
/* via developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope/clearTimeout */
var egg_timer = {

  /* callback */
  setup: function(t_ms){

    /* assert parameter is a number */
    if(typeof this.timeoutID === "number"){
      this.cancel()
    }

    /* what to do when the timer expires */
    this.timeoutID = window.setTimeout(
      function(){
        var now = ctx.get_state()
        var id = now.id
        now.ding = true
        if(now.key_expiry == false || now.expiry_ms > 0){
          now.expire()
        }
      }.bind(this), t_ms
    )
  }, cancel: function(){
    window.clearTimeout(this.timeoutID)
    this.timeoutID = undefined
  }
}
```

4.2. **key.js.**

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

/* 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){

    /* unicode vs. character representation */
    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 to buffer */
        now.txt += key.toLowerCase()
      }

      /* redraw */
      update()
    }
```

```
64
65      /* check if this state "requires" keyboard input */
66      if(now.require_key() == true){
67
68        /* is the key that was pressed, in the list of "admissible" keys? */
69        if(admissible_keys.includes(unicode)){
70
71          /* if we have a "deja-vu" variable, calculate a score */
72          if(!(now.deja == undefined)){
73            ctx.questions_total += 1
74
75            /* check for N or M keypress */
76            if((now.deja == true && unicode == 77) || (now.deja == false && unicode == 78)){
77              ctx.questions_correct += 1
78            }
79          }
80        }else{
81          /* block if a key was required but the one entered was not admissible */
82          go = false
83        }
84      }
85
86      /* t <-- t + 1 */
87      if(now && now.key_expiry && go){
88
89          /* clear the timer and "go next" */
90          ctx.clear_tmr()
91          now.expire()
92      }
93    }
94    return key_unicode
95  }
```

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()
```

```
64    var now = ctx.get_state()
65    if(now){
66      now.show(ctx)
67    }
68  }
69
70  /* "in" hook: plot the current trial */
71  window.onload = function(){
72    update()
73  }
74
75  /* set up timer to coordinate transitions between trials */
76  ctx.egg_timer = egg_timer
77
78  ctx.clear_tmr = function(){
79    ctx.egg_timer.cancel()
80  }
81
82  ctx.init_tmr = function(t_ms){
83    ctx.egg_timer.setup(t_ms)
84  }
85
86  /* initialize reference to first and most-recently-initialized trials */
87  ctx.last_new_state = null, ctx.first_new_state = null
88
89  /* count number of questions answered correctly (this is redundant) */
90  ctx.questions_correct = 0, ctx.questions_total = 0
91
92  /* this function sets up the experiment (according to the user function my_experiment)
93  and we trigger this function after all the images have loaded. */
94  function run_after_loading_images(){
95
96    /* set up an experiment according to user specs/code */
97    my_experiment(ctx)
98
99    instructions('thank you')
100
101   ctx.last_state = ctx.last_new_state, ctx.first_state = ctx.first_new_state
102
103   /* start at the very beginning, it's a very good place to start.. */
104   ctx.set_state(ctx.first_state)
105
106   /* respond to keyboard events */
107   key_unicode = keyboard_module()
108
109   /* start "stopwatch" */
110   ctx.t0 = window.performance.now()
111
112   /* go */
113   ctx.get_state().start()
114 }
115
116 /* load some image files: need to change if the image database changes */
117 var n_imgs = 200, n_imgs_loaded = 0
118
119 /* load image data */
120 function load_img(fn){
121   var img = new Image()
122   img.onload = function(){
123
124     /* have all images been loaded? */
125     if(++n_imgs_loaded == n_imgs){
126
127       /* proceed to init the experiment */
128       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.**

```
var next_pool_id = 0

/* 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" */
function pool(){

  /* keep count */
  ++ next_pool_id

  this.is_pool = true, this.pool_id = next_pool_id, this.ctx = ctx, this.stimuli = new Array()

  /* 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})
  }

  /* pseudorandom selection of size "n" */
  this.draw_n = function(){

    if(this.selection_n){
      console.log('error: n-selection already made from this pool.')
      return null
    }

    /* check the selection size */
    var n = parseInt(this.get_n())
    if(n > this.stimuli.length){
      console.log('error: n > this.stimuli.length')
      return null
    }
```

```
62
63      /* make a pseudorandom selection */
64      this.selection_n = new Array()
65      var rem = this.stimuli.length
66      for(var i = 0; i < n; i++){
67        var qx = rand() * parseFloat(rem --), idx = parseInt(qx)
68        this.selection_n.push(this.stimuli[idx])
69        delete this.stimuli[idx]
70        this.remove_blanks()
71      }
72    }
73
74    /* pseudorandom selection of size "m" */
75    this.draw_m = function(){
76
77      if(this.selection_m){
78        console.log('error: m-selection already made from this pool.')
79        return null
80      }
81
82      /* check the selection size */
83      var m = parseInt(this.get_m())
84      if(m > this.stimuli.length){
85        console.log('error: m > this.stimuli.length')
86        return null
87      }
88
89      /* make a pseudorandom selection */
90      this.selection_m = new Array()
91      var rem = this.stimuli.length
92      for(var i = 0; i < m; i++){
93        var qx = rand() * parseFloat(rem --), idx = parseInt(qx)
94        this.selection_m.push(this.stimuli[idx])
95        delete this.stimuli[idx]
96        this.remove_blanks()
97      }
98    }
99
100   /* for initializing a test phase: mix "N"-selection and "M"-selection together */
101   this.reshuffle = function(){
102
103     /* put the "N"-selection and "M" selection, together in array to_shuffle,
104       which will be shuffled */
105     var to_shuffle = [], i = 0
106
107     /* add the "N"-selection */
108     for(i = 0; i < this.selection_n.length; i++){
109       var dat_i = new Array()
110       dat_i.push(this.selection_n[i])
111       dat_i.push(true)
112       to_shuffle.push(dat_i)
113     }
114
115     /* add the "M"-selection */
116     for(i = 0; i < this.selection_m.length; i++){
117       var dat_i = new Array()
118       dat_i.push(this.selection_m[i])
119       dat_i.push(false)
120       to_shuffle.push(dat_i)
121     }
122
123     /* "shuffle"-- randomize the ordering of the combined array */
124     var shuffled = new Array(), deja_vu = new Array(), rem = to_shuffle.length
125     while((rem --) > 0){
126       var idx = parseInt(rand() * parseFloat(rem)), dat_i = to_shuffle[idx]
```

```
127         shuffled.push(dat_i[0])
128         deja_vu.push(dat_i[1])
129         delete to_shuffle[idx]
130         to_shuffle = to_shuffle.filter(function(){return true})
131       }
132       return [shuffled, deja_vu]
133     }
134
135
136     /* perform all of the above */
137     this.draw = function(){
138       this.draw_n()
139       this.draw_m()
140       this.reshuffle()
141     }
142
143     /* set N, M parameters and make a selection cf the above */
144     this.select = function(n,m){
145       this.set_n(n)
146       this.set_m(m)
147       this.draw()
148     }
149
150     /* end of "pool::pool()" */
151     return this
152 }
153
154 /* following the convention to wrap away the new() operator */
155 function stimulus_pool(){
156     return new pool()
157 }
```

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.id = get_id()

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

  /* array to store admissible key-codes for data entry or transition to next "slide":
    default: M, N */
  this.admissible_keys = [77, 78]

  this.get_admissible_keys = function(){
    return this.admissible_keys
  }

  this.clear_admissible_keys = function(){
    this.admissible_keys = new Array()
  }

  this.add_admissible_key = function(k){
    this.admissible_keys.push(k)
  }

  /* this array will record the keystroke data received while residing in this state */
  this.key_strokes = new Array()

  this.record_key_stroke = function(k){
    this.key_strokes.push(k)
  }

  this.set_pool_id = function(pid){
    this.pool_id = pid
  }

  this.get_pool_id = function(){
    return this.pool_id ? this.pool_id : ""
  }

  /* keep a reference to this state, if it's the first one ever.. */
  if(ctx.first_new_state == null){
    ctx.first_new_state = this
  }

  /* only applies if there's a "next" trial, if this is a trial */
```

```
63  this.intvl_ms = intvl_ms
64
65  /* numeric */
66  this.expiry_ms = expiry_ms
67
68  /* boolean */
69  this.key_expiry = key_expiry
70
71  /* global image index (images added as member of ctx) */
72  this.img_idx = img_idx, this.successor = null, this.predecessor = ctx.last_new_state
73
74  this.require_key = function(){
75    return this.key_required
76  }
77
78  var id = (this.predecessor == null) ? -1 : this.predecessor.id
79  ctx.last_new_state = this
80
81  /* sanity check: make sure the predecessor points here */
82  if(this.predecessor){
83    this.predecessor.set_successor(this)
84  }
85
86  /* where are we going? */
87  this.set_successor = function(s){
88    this.successor = s
89  }
90
91  /* plot text or images */
92  this.show = function(){
93
94    /* execute associated action, if we have one */
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   /* upper text */
102   if(this.txt){
103     wrap_text(this.txt, ctx, 0)
104   }
105
106   /* middle text */
107   if(this.txt2){
108     wrap_text(this.txt2, ctx, ctx.h() - (2 * ctx.font_size + 20))
109   }
110
111   /* img or middle text (if word stim) */
112   if(this.img_stim){
113     draw_img(this.img_stim, ctx)
114   }
115
116   /* might need the wrap_text back on for long strings.. */
117   if(this.wrd_stim){
118
119     /* no wrap */
120     centre_text(this.wrd_stim)
121   }
122
123   /* logo of no image/ lower text present */
124   if(!this.txt2){
125     ctx.draw_symbol()
126   }
127 }
```

```
128
129   /* state expires by timer or key press */
130   this.set_expiry = function(t_ms){
131
132     /* follow clock or key to keep the show going */
133     this.expiry_ms = t_ms
134
135     /* state expires by key press */
136     if(t_ms <= 0){
137       this.key_expiry = true
138     }
139   }
140
141   /* enter a state (begin) */
142   this.start = function(){
143     var ctx = get_ctx()
144
145     /* do data dump, if we're at the end */
146     if(this == ctx.last_state){
147
148         /* window.location.href == http://domain/memory/examples/test_phase/memory.html */
149         var href = window.location.href
150
151         /* go through all the states and record (in string format) the info we'd like to
                appear on the server */
152         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"
153         for(var state_i = ctx.first_state; state_i != ctx.last_state; state_i = state_i.
                successor){
154           var stim_type = null, my_stim  = null, pi = ""
155
156           /* "the right way to check if a variable is undefined or not" */
157           if(typeof state_i.pool_id !== 'undefined'){
158             pi = JSON.parse(JSON.stringify(state_i.pool_id))
159           }
160
161           /* assign "stimulus type" keyword */
162           if(state_i.wrd_stim){
163             stim_type = "word", my_stim = state_i.wrd_stim
164           }
165           if(state_i.img_stim){
166             stim_type = "image", my_stim = state_i.img_stim.fn
167           }
168           if(!stim_type){
169             stim_type = ""
170           }
171           if(!my_stim){
172             my_stim = ""
173           }
174
175           /* for a given "state", record a line of data */
176           message += href + ","
177
178           /* event_id: global index / line number */
179           message += state_index.toString() + ","
180
181           /* task_id */
182           message += state_i.task_id + ","
183
184           /* task_type */
185           message += state_i.type + ","
186
187           /* trial_id */
188           message += state_i.trial_id + ","
```

```
189           message += Math.round(10. * (state_i.t1 - state_i.t0)) / 10. + ","
190           message += parse_date_time(state_i.start_date_time).toString() + ","
191           message += parse_date_time(state_i.end_date_time).toString() + ","
192
193           /* ISI */
194           if(state_i.type == 'isi'){
195             message += state_i.expiry_ms.toString()
196           }
197           message += ","
198
199           if(!state_i.expiry_ms){
200             state_i.expiry_ms = ""
201           }
202
203           /* SET */
204           message += state_i.expiry_ms.toString() + ","
205
206           /* stimulus type */
207           message += stim_type.toString() + ","
208
209           /* stimulus id */
210           message += my_stim.toString() + ","
211
212           /* stimulus-pool id */
213           message += pi.toString() + ","
214
215           /* user response */
216           var response = ""
217           for(var k in state_i.key_strokes){
218             response += String.fromCharCode(state_i.key_strokes[k])
219           }
220           message += response + ""
221
222           /* add a newline character */
223           message += "\n"
224
225           /* go next */
226           ++ state_index
227         }
228
229         /* remove last three elements from array: take current page and navigate to:
230           ../../xml-receive.py == http://domain/memory/xml-receive.py */
231         var words = href.split('/')
232         var nwords = words.length
233         var target = words.splice(0, nwords-3).join('/') + '/xml-receive.py'
234
235         /* send the message to the server-side script at URL: target */
236         xml_send(message, target)
237       }
238
239       var ctx = get_ctx()
240
241       /* start the clock.. */
242       this.t0 = window.performance.now(), this.start_date_time = date_time()
243
244       /* clear the timer */
245       ctx.clear_tmr()
246
247       /* plot the current trial */
248       this.show(ctx)
249
250       /* start the timer? */
251       if(this.expiry_ms > 0){
252         ctx.init_tmr(this.expiry_ms, this.expire)
253       }
```

```
    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 && suc.txt){
      suc_txt = suc.txt
    }

    /* enter next state */
    if(this.successor){
      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]
```

```
58
59      /* data (word or image) assigned to "trial" */
60      var data = a_selection[0], p_id = a_selection[1], extra_feedback_this_slide =
            a_selection[2]
61
62      /* if ISI was set, prefix with a "blank" slide */
63      if(isi > 0){
64        var x = new state()
65        x.set_expiry(isi)
66        x.type = 'isi', x.wrd_stim = "", x.trial_id = trial_index, x.task_id = my_task_id
67        x.set_pool_id(my_pool.pool_id)
68        x.clear_admissible_keys()
69        x.key_expiry = false
70      }
71
72      /* initialize generic "trial" object for each case */
73      var x = new state()
74      if(time_limit <= 0){
75        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()
201     x.key_expiry = false
202   }
203
204   var y = instructions(txt)
205   if(true){
206     /* time [mS] */
207     var x = new state()
208     x.set_expiry(delay_time)
209     x.key_expiry = false, x.txt = '', x.type = 'delay', x.trial_id = 0, x.task_id =
          my_task_id
210   }
211   return this
212 }
```

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