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 11, 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:

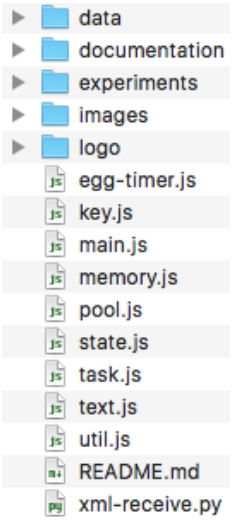

FIGURE 1.1.

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.

- **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(){
  /* set up some instruction slides */
  instructions('welcome to the recognition memory experiment framework. here is what happens when you put i
  instructions('this is an instructions slide')
  instructions('this is another instructions slide')
}
```

2.2. **experiments/delay.**

```
/* recognition memory experiment set-up */

var my_experiment = function(){

  /* set up some instruction slides */
  instructions('delay phase: please press any key to start')

  /* set up delay tasks */
  delay_task('please type in the names of as many countries as you can think of in 5 seconds, separated by
  delay_task('please type in the names of as many countries as you can think of in 30 seconds, separated by
  instructions('all done.. thank you')
}
```

2.3. **experiments/feedback.**

```
/* recognition memory experiment set-up */

var my_experiment = function(){

  /* instructions */
  instructions('feedback coming up...')

  /* 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...')

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

  /* instructions */
  instructions('thank you')
}
```

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

```
/* recognition memory experiment set-up */

var my_experiment = function(){

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

  /* set up a stimulus pool */
  var p = 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(3, 3)

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

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 = 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)
}
```

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
        /* console.log('ding from now(): id', id) */
        now.ding = true
        if(now.key_expiry ==false){
          now.expire()
        }
    }.bind(this), t_ms)
  }, cancel: function() {
    window.clearTimeout(this.timeoutID)
    this.timeoutID = undefined
  }
}
```

4.2. **key.js.**

```
/* convert form 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){
    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.record_key_stroke(unicode)
    }

    /* by default, transition from a slide upon key-press */
    var go = true
    if(now.type=='delay'){
      if(now.txt ==null){
        now.txt =''
      }
      if(unicode ==8 ){
        var len = now.txt.length
        if(now.txt[len-1] != ' '){
          now.txt = now.txt.substring(0, len - 1)
        }
      }else if(unicode == 0){
      }else{
        now.txt += key.toLowerCase()
      }
      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
```

```
64        }
65      }
66      if(now.ding==false && now.hold==true){
67        go = false
68      }
69
70      /* t <-- t + 1 */
71      if(now && now.key_expiry && go){
72          ctx.clear_tmr()
73          now.expire()
74      }
75    }
76    return key_unicode
77  }
```

4.3. **main.js.**

```
abs_path = '../../'
var history = [], canvas = document.getElementsByTagName("canvas")[0], ctx = canvas.getContext("2d")

/* background color */
document.bgColor = "#FFFFFF"

/* shape parameter */
ctx.pad = 20

/* font size */
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, ww = window.innerWidth, wh = window.innerHeight, w = ww -
  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

  /* should not happen.. */
  if(s != null){
    s.daddy = last_state
  }
  return(s)
}

/* access present "state" */
ctx.get_state = function(){
  var s = ctx.current_state
  var st = ''
  try{
    st = s.txt
  }catch(e){
    st = ''
```

```
64      }
65      return s
66    }
67
68    /* trigger update/plotting from window resize event */
69    window.onresize = function(event){
70      update()
71    }
72
73    /* update the canvas (present the current "trial") */
74    function update(){
75      resize()
76      var now = ctx.get_state()
77      if(now != null)
78        now.show(ctx)
79    }
80
81    /* "in" hook: plot the current trial */
82    window.onload = function(){
83      update()
84    }
85
86    /* set up timer to coordinate transitions between trials */
87    ctx.egg_timer = egg_timer
88    ctx.clear_tmr = function(){
89      ctx.egg_timer.cancel()
90    }
91    ctx.init_tmr = function(t_ms){
92      ctx.egg_timer.setup(t_ms)
93    }
94
95    /* initialize reference to first and most-recently-initialized trials */
96    ctx.last_new_state = null
97    ctx.first_new_state = null
98
99    /* count number of questions answered correctly (this is redundant) */
100   ctx.questions_correct = 0
101   ctx.questions_total = 0
102
103   /* load some image files: need to change if the image database changes */
104   var n_imgs = 200;
105   ctx.load_imgs = function (n_imgs){
106
107     /* ideally would only load the ones used */
108     var imgs = new Array()
109     for(var i=1; i <= n_imgs; i++){
110       var img_fn = abs_path + 'images/' + i + '.jpg'
111       var my_img = load_img(img_fn)
112       my_img.fn = 'images/' + i + '.jpg'
113       imgs.push(my_img)
114     }
115     ctx.imgs = imgs
116     return ctx.imgs
117   }
118   var my_images = ctx.load_imgs(n_imgs)
119
120   var next_task_id = 0;
121
122   /* set up an experiment according to user specs/code */
123   my_experiment(ctx)
124
125   instructions('thank you')
126
127   ctx.last_state = ctx.last_new_state
128   ctx.first_state = ctx.first_new_state
```

```
129
130   /* start at the very beginning, it's a very good place to start.. */
131   ctx.set_state(ctx.first_state)
132
133   /* respond to keyboard events */
134   key_unicode = keyboard_module()
135
136   /* start "stopwatch" */
137   ctx.t0 = window.performance.now()
138
139   /* go */
140   ctx.get_state().start()
```

4.4. **memory.js.**

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

/* cr34t3 a c4nv4s wh3r3 th3 m4g1c h4pp3ns */
var canvas = document.createElement('canvas')
document.body.appendChild(canvas)
var js_added = 0
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'
  var callback = function(){
    js_added += 1
    if(js_added < deps.length){
      add_js(deps[js_added])
    }
  }

  /* wait until script is loaded before proceeding.. */
  s.onload = callback
  var len = body.childNodes.length
  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"
var next_pool_id = 0
function pool(){
  this.is_pool = true
  this.pool_id = next_pool_id
  next_pool_id += 1
  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), an additional "m" samples are
    this.m = m
  }

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

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


  /* remove any "blank" elements (an operation needed due to an apparent curiosity of the language) that ap
  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
    }
    var n = parseInt(get_n())
    if(n > this.stimuli.length){
      console.log('error: n > this.stimuli.length')
      return null
    }
    this.selection_n = new Array()
    var rem = this.stimuli.length
    for(var i = 0; i < n; i++){
      var qx = rand() * parseFloat(rem), idx = parseInt(qx)
      rem -= 1
      this.selection_n.push(this.stimuli[idx])
      delete this.stimuli[idx]
      this.remove_blanks()
    }
```

```
64  }
65
66  /* pseudorandom selection of size "m" */
67  this.draw_m = function(){
68    if(this.selection_m){
69      console.log('error: m-selection already made from this pool.')
70      return null
71    }
72    var m = parseInt(get_m())
73    if(m > this.stimuli.length){
74      console.log('error: m > this.stimuli.length')
75      return null
76    }
77    this.selection_m = new Array()
78    var rem = this.stimuli.length
79    for(var i = 0; i < m; i++){
80      var qx = rand() * parseFloat(rem), idx = parseInt(qx)
81      rem -= 1
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
108   /* "shuffle"-- randomize the ordering of the combined array */
109   var shuffled = new Array(), deja_vu = new Array(), rem = to_shuffle.length
110   while(rem >0){
111     rem -= 1
112     var idx = parseInt(rand() * parseFloat(rem)), dat_i = to_shuffle[idx]
113     shuffled.push(dat_i[0])
114     deja_vu.push(dat_i[1])
115     delete to_shuffle[idx]
116     to_shuffle = to_shuffle.filter(function(){return true})
117   }
118   var ret = [shuffled, deja_vu]
119   return ret
120 }
121
122 this.draw = function(){
123   this.draw_n()
124   this.draw_m()
125   this.reshuffle()
126 }
127
128 /* set N, M parameters and make a selection */
```

```
129   this.select = function(n,m){
130     this.set_n(n)
131     this.set_m(m)
132     this.draw()
133   }
134
135   /* end of "pool::pool()" */
136   return this
137 }
```

4.6. **state.js.**

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

function get_id(){
  state_id += 1;
  return state_id;
}

/* reference to 2d canvas graphics context */
function get_ctx(){
  return 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,  /* 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){
  this.action = null
  this.ding = false
  var ctx = get_ctx()
  this.hold = false

  this.hold_on = function(){
    this.hold = true
  }
  this.id = get_id()
  this.key_required = false

  /* array to store admissible key-codes for data entry or transition to next "slide" */
  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(){
    if(this.pool_id)
      return this.pool_id
    else
      return ""
  }

```

```
64  /* keep a reference to this state, if it's the first one ever.. */
65  if(ctx.first_new_state == null){
66    ctx.first_new_state = this
67  }
68
69  /* only applies if there's a "next" trial, if this is a trial */
70  this.intvl_ms = intvl_ms
71
72  /* numeric */
73  this.expiry_ms = expiry_ms
74
75  /* boolean */
76  this.key_expiry = key_expiry
77
78  /* global image index (images added as member of ctx) */
79  this.img_idx = img_idx
80  this.successor = null
81
82  this.require_key = function(){
83    return this.key_required
84  }
85  this.predecessor = ctx.last_new_state;
86  var id = this.predecessor == null ? -1 : this.predecessor.id
87  ctx.last_new_state = this
88  if(this.predecessor != null){
89    this.predecessor.set_successor(this)
90  }
91
92  /* where are we going? */
93  this.set_successor = function(s){
94    this.successor = s
95  }
96
97  /* plot text or images */
98  this.show = function(){
99    if(this.action){
100     this.action(this)
101   }
102   var ctx = get_ctx()
103   ctx.clearRect(0, 0, ctx.w(), ctx.h())
104
105   /* bottom text */
106   if(this.txt2 && (!this.wrd_stim)){
107     //wrap_text(this.txt2, ctx, ctx.h() - (2 * ctx.font_size+20));
108   }
109   if(this.txt2){
110     wrap_text(this.txt2, ctx, ctx.h() - (2 * ctx.font_size + 20))
111   }
112
113   /* upper text */
114   if(this.txt){
115     wrap_text(this.txt, ctx, 0)
116   }
117
118   /* img or middle text (if word stim) */
119   if(this.img_stim){
120     x = this.img_stim
121     draw_img(x, ctx)
122   }
123
124   /* might need the wrap_text back on for long strings.. */
125   if(this.wrd_stim!=null){
126     // wrap_text(this.wrd_stim, ctx, ctx.h()/2);
127
128     /* no wrap */
```

```
129       centre_text(this.wrd_stim)
130     }
131
132     /* logo of no image/ lower text present */
133     if(!this.txt2){
134       ctx.draw_symbol()
135     }
136   }
137
138   /* state expires by timer or key press */
139   this.set_expiry = function(t_ms){
140
141     /* follow clock or key to keep the show going */
142     this.expiry_ms = t_ms
143
144     /* state expires by key press */
145     if(t_ms <= 0){
146       this.key_expiry = true
147     }
148   }
149
150   /* enter a state (begin) */
151   this.start = function(){
152     var ctx = get_ctx()
153
154     if(this == ctx.last_state){
155
156         /* go through all the states and record (in string format) the contents, as we'd like it to appear
157         var state_i = ctx.first_state, state_index = 0
158         var message = "event_id,task_id,task_type,trial_id,duration(mS),start(yyyy:mm:dd:hh:mn:ss:mls),end(
159         var pi;
160         for(var state_i = ctx.first_state; state_i != ctx.last_state; state_i = state_i.successor){
161           var stim_type = null;
162           var my_stim  = null;
163
164           /* the right way to check if a variable is undefined or not */
165           if(typeof state_i.pool_id !== 'undefined'){
166             pi = JSON.parse(JSON.stringify(state_i.pool_id))
167           }else{
168             pi = ""
169           }
170
171           if(state_i.wrd_stim){
172             stim_type = "word"
173             my_stim = state_i.wrd_stim
174           }
175
176           if(state_i.img_stim){
177             stim_type = "image"
178             my_stim = state_i.img_stim.fn
179           }
180
181           if(stim_type){
182           }else{
183             stim_type = ""
184           }
185
186           if(my_stim){
187           }else{
188             my_stim = ""
189           }
190
191           /* for a given "state", record a line of data */
192           message += state_index.toString() + ","       /* event_id: global index / line number */
193           message += state_i.task_id + ","              /* task_id */
```

```
194           message += state_i.type + ","                    /* task_type */
195           message += state_i.trial_id + ","                /* trial_id */
196           message +=  Math.round(10. * (state_i.t1 -  state_i.t0)) / 10. + ","
197           message += parse_date_time(state_i.start_date_time).toString() + ","
198           message += parse_date_time(state_i.end_date_time).toString() + ","
199           if(state_i.type == 'isi'){
200             message += state_i.expiry_ms.toString()
201           }
202           message += ","                                   /* ISI */
203           message += ","                                   /* SET */
204           message += stim_type.toString() + ","            /* stim_type */
205           message += my_stim.toString() + ","              /* stim_id */
206           message += pi.toString() + ","                   /* stimulus-pool id */
207           var response = ""
208           for(var k in state_i.key_strokes){
209             response += String.fromCharCode(state_i.key_strokes[k])
210           }
211           message += response + ""                         /* response */
212
213           /* add a newline character */
214           message += "\n"
215           state_index += 1
216         }
217
218         /* window.location.href == http://domain/memory/examples/test_phase/memory.html */
219         var href = window.location.href
220
221         /* remove last three elements from the array: take the page and navigate to: ../../xml-receive.py =
222         var words = href.split('/')
223         var nwords = words.length
224         var target = words.splice(0, nwords-3).join('/') + '/xml-receive.py'
225
226         /* send the message to the server-side script at URL: target */
227         xml_send(message, target)
228     }
229
230     var ctx = get_ctx()
231
232     /* start the clock.. */
233     this.t0 = window.performance.now()
234     this.start_date_time = date_time()
235
236     /* clear the timer */
237     ctx.clear_tmr()
238
239     /* plot the current trial */
240     this.show(ctx)
241
242     /* start the timer? */
243     if(this.expiry_ms > 0){
244       ctx.init_tmr(this.expiry_ms, this.expire)
245     }
246     return null
247   }
248
249   /* pr0c33d t0 th3 n3xt 5+4t3 */
250   this.expire = function(){
251     var ctx = get_ctx()
252
253     /* st0p 4ll th3 cl0ck5 */
254     ctx.clear_tmr()
255
256     /* r3c0rd st0p t1m3 */
257     this.end_date_time = date_time()
258     this.t1 = window.performance.now()
```

```
259     var txt = this.txt, suc_txt = null, suc = this.successor
260
261     if(suc!=null && suc.txt !=null){
262       suc_txt = suc.txt
263     }
264
265     /* enter next state */
266     if(this.successor!=null){
267       ctx.set_state(this.successor)
268       ctx.get_state().start()
269
270       /* this condition might only be good if we have the "score card"? not sure. Replace score card with t
271       if(this.successor.successor == null){
272       }
273     }
274     /* record data to csv-line record (global) here..? */
275
276   }
277   return this
278 }
```

4.7. **task.js.**

```
/* Event hierarchy: 1) Experiment (includes multiple tasks) 2) Task (includes multiple trials) 3) Trial (ea

/* 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 selectio
function study_phase(my_pool, isi=0){
  var my_pools = []
  if(my_pool.is_pool){
    my_pools.push(my_pool)
  }else{
    my_pools = my_pool
  }

  var trial_index = -1
  var my_task_id = next_task_id++

  /* record references to graphics context, and stimulus pool */
  this.ctx = ctx
  this.p = my_pools
  this.pool_ids = new Array()

  for(var a_pool in my_pools){
    var my_pool = my_pools[a_pool]
    this.pool_ids.push(my_pool.pool_id)

    /* iterate over selected elements of pool */
    for(var i  in my_pool.selection_n){
      trial_index ++

      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()

      /* need to add timed parameter to front-end API */
      x.set_expiry(0)

```

```
64        /* data (word or image) assigned to "trial" */
65        var data = my_pool.selection_n[i]
66
67        /* discern by image or word, respectively */
68        if( typeof(data) === 'object'){
69          x.img_stim = data
70        }else if(typeof(data) === 'string'){
71          x.wrd_stim = data
72        }
73        x.type = 'study_phase'
74        x.trial_id = trial_index
75        x.task_id = my_task_id
76        x.set_pool_id(my_pool.pool_id)
77      } /* for var i  in my_pool.selection_n */
78    } /* for var a_pool in my_pools */
79
80    return this
81  }
82
83  /* test phase, formerly known as recognition task - for this phase, the random selection is shuffled back i
84  function test_phase(my_pool, isi=false){
85    var my_pools = []
86    if(my_pool.is_pool){
87      my_pools.push(my_pool)
88    }else{
89      my_pools = my_pool
90    }
91
92    var trial_index = -1
93    var my_task_id = next_task_id++
94
95    this.ctx = ctx
96    this.p = my_pools
97    this.pool_ids = new Array()
98
99    for(var a_pool in my_pools){
100     var my_pool = my_pools[a_pool]
101     this.pool_ids.push(my_pool.pool_id)
102
103     var trial_index = -1, shuffled_data = my_pool.reshuffle(), shuffled = shuffled_data[0], deja_vu = shuff
104     for(var i in shuffled){
105       trial_index ++
106
107       if(isi > 0){
108         var x = new state()
109         x.set_expiry(isi)
110         x.type = 'isi'
111         x.wrd_stim = ""
112         x.trial_id = trial_index
113         x.task_id = my_task_id
114         x.set_pool_id(my_pool.pool_id)
115         x.clear_admissible_keys()
116         x.key_expiry = false
117       }
118
119       var x = new state()
120       x.set_expiry(0)
121       x.key_required = true
122       var data = shuffled[i], deja = deja_vu[i]
123
124       /* record within the object: do we have deja-vu? */
125       x.deja = deja
126
127       /* word or image? */
128       if( typeof(data) === 'object'){
```

```
129         x.img_stim = data
130       }else if(typeof(data) ==='string'){
131         x.wrd_stim = data
132       }
133       x.type = 'test_phase'
134       x.trial_id = trial_index
135       x.task_id = my_task_id
136       x.set_pool_id(my_pool.pool_id)
137     }
138   }
139   var m = 'Thank you for completing this section. '
140   var end = instructions(m)
141
142   end.action = function(me){
143     var msg = m + 'Your score: ' + ctx.questions_correct.toString() + '/' + ctx.questions_total.toString()
144     me.txt = msg
145   }
146   return this
147 }
148
149 /* previously known as feedback task */
150 function feedback(txt, keys){
151   var my_task_id = next_task_id++
152
153   var x = new state()
154   x.set_expiry(0)
155   x.txt = txt
156   x.key_required = true
157   x.clear_admissible_keys()
158   for(var i in keys){
159     x.add_admissible_key(keys[i])
160   }
161   x.type = 'feedback'
162   x.trial_id = 0
163   x.task_id = my_task_id
164 }
165
166 /* list as many countries as possible during e.g., a 3-minute period (default, 30s) */
167 function delay_task(txt, delay_time=30000){
168   var my_task_id = next_task_id++
169
170   var y = instructions(txt)
171   y.key_expiry = true
172   y.set_expiry(500)
173
174   /* keypress activated with minimum time */
175   y.hold_on()
176
177   /* time [mS] */
178   var thirty_seconds = 30000, x = new state()
179   x.set_expiry(delay_time)
180   x.key_expiry = false
181   x.txt = ''
182   x.type = 'delay'
183   x.trial_id = 0
184   x.task_id = my_task_id
185   return this;
186 }
```

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(
  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)
  }
}

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.**

```
/* get date and time */
function date_time(){
  return new Date()
}

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

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

/*random-number generator http://indiegamr.com/generate-repeatable-random-numbers-in-js/ : initial seed.. i
function rand(max, min) {
  max = max || 1
  min = min || 0
  seed = (seed * 9301 + 49297) % 233280
  var rnd = seed / 233280
  return min + rnd * (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, h = ctx.h() - cf, w = ctx.w(), lw = x.width, lh = x.height, sf = Math.min(w
(-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.getMinute

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

```
63      var n_pad = 2
64      if(i==0) n_pad = 4
65      if(i== 6) n_pad = 3
66      var bts = bits[i].toString()
67      bits[i] = pad_n(bts, n_pad)
68    }
69    return(bits.join(':'))
70  }
71
72  /* "faster trim" via blog.stevenlevithan.com */
73  function trim(s){
74    return s.toString().replace(/^\s\s*/,'').replace(/\s\s*$/,'')
75  }
76
77  /* send text format data (string s) via XML to receive script at url (string): xml-receive_script_url
    */
78  function xml_send(s, xml_receive_script_url){
79
80    /* xml http request object */
81    var xhr = (window.XMLHttpRequest) ? new XMLHttpRequest() : new activeXObject("Microsoft.XMLHTTP")
82    var data = new FormData()
83    data.append("data", s)
84    xhr.open( 'post', xml_receive_script_url, true)
85    xhr.send(data)
86  }
```

5. Source Code: Server Side

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)
```