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 14, 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.

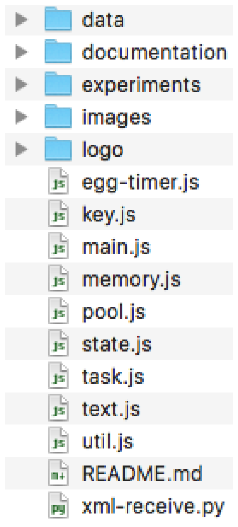
- For administrators:
  - An FTP program is required for uploading experiment scripts (and downloading response data).
  - A text editor is required to edit experiment script files.
  - Limited technical knowledge about JavaScript is required to edit or modify experiments.

1. The System

The system, which may be downloaded from

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

has the following directory structure:



```
data
documentation
experiments
images
logo
egg-timer.js
key.js
main.js
memory.js
pool.js
state.js
task.js
text.js
util.js
README.md
xml-receive.py
```

Figure 1.1.

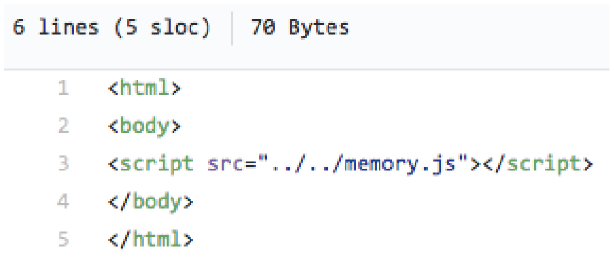
where this document lives in the documentation/ folder. Additionally,

- **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
<html>
<body>
<script src="../../memory.js"></script>
</body>
</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)
}
```

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()
    }

    /* check if this state "requires" keyboard input */
    if(now.require_key() == true){

      /* is the key that was pressed, in the list of "admissible" keys? */
      if(admissible_keys.includes(unicode)){

        /* if we have a "deja-vu" variable, calculate a score */
        if(!(now.deja == undefined)){
          ctx.questions_total += 1

          /* check for N or M keypress */
          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
      }
    }

    /* t <-- t + 1 */
    if(now && now.key_expiry && go){

        /* clear the timer and "go next" */
        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)

  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.**

```
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
  }

  /* make a pseudorandom selection */
  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)
    this.selection_n.push(this.stimuli[idx])
    delete this.stimuli[idx]
    this.remove_blanks()
  }
}

/* pseudorandom selection of size "m" */
this.draw_m = function(){

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

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

  /* make a pseudorandom selection */
  this.selection_m = new Array()
  var rem = this.stimuli.length
  for(var i = 0; i < m; i++){
    var qx = rand() * parseFloat(rem --), idx = parseInt(qx)
    this.selection_m.push(this.stimuli[idx])
    delete this.stimuli[idx]
    this.remove_blanks()
  }
}

/* for initializing a test phase: mix "N"-selection and "M"-selection together */
this.reshuffle = function(){

    /* put the "N"-selection and "M" selection, together in array to_shuffle,
      which will be shuffled */
    var to_shuffle = [], i = 0

    /* add the "N"-selection */
    for(i = 0; i < this.selection_n.length; i++){
      var dat_i = new Array()
      dat_i.push(this.selection_n[i])
      dat_i.push(true)
      to_shuffle.push(dat_i)
    }

    /* add the "M"-selection */
    for(i = 0; i < this.selection_m.length; i++){
      var dat_i = new Array()
      dat_i.push(this.selection_m[i])
      dat_i.push(false)
      to_shuffle.push(dat_i)
    }

    /* "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]
  }


  /* perform all of the above */
  this.draw = function(){
    this.draw_n()
    this.draw_m()
    this.reshuffle()
  }

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

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

/* following the convention to wrap away the new() operator */
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.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 */
  this.intvl_ms = intvl_ms

  /* numeric */
  this.expiry_ms = expiry_ms

  /* boolean */
  this.key_expiry = key_expiry

```

```
/* global image index (images added as member of ctx) */
this.img_idx = img_idx, this.successor = null, this.predecessor = ctx.last_new_state

this.require_key = function(){
  return this.key_required
}

var id = (this.predecessor == null) ? -1 : this.predecessor.id
ctx.last_new_state = this

/* sanity check: make sure the predecessor points here */
if(this.predecessor){
  this.predecessor.set_successor(this)
}

/* where are we going? */
this.set_successor = function(s){
  this.successor = s
}

/* plot text or images */
this.show = function(){

  /* execute associated action, if we have one */
  if(this.action){
    this.action(this)
  }
  var ctx = get_ctx()
  ctx.clearRect(0, 0, ctx.w(), ctx.h())

  /* upper text */
  if(this.txt){
    wrap_text(this.txt, ctx, 0)
  }

  /* middle text */
  if(this.txt2){
    wrap_text(this.txt2, ctx, ctx.h() - (2 * ctx.font_size + 20))
  }

  /* img or middle text (if word stim) */
  if(this.img_stim){
    draw_img(this.img_stim, ctx)
  }

  /* might need the wrap_text back on for long strings.. */
  if(this.wrd_stim){

    /* no wrap */
    centre_text(this.wrd_stim)
  }

  /* logo of no image/ lower text present */
  if(!this.txt2){
    ctx.draw_symbol()
  }
}

/* state expires by timer or key press */
this.set_expiry = function(t_ms){

  /* follow clock or key to keep the show going */
  this.expiry_ms = t_ms

  /* state expires by key press */
```

```
    if(t_ms <= 0){
      this.key_expiry = true
    }
  }

  /* enter a state (begin) */
  this.start = function(){
    var ctx = get_ctx()

    /* do data dump, if we're at the end */
    if(this == ctx.last_state){

        /* window.location.href == http://domain/memory/examples/test_phase/memory.html */
        var href = window.location.href

        /* go through all the states and record (in string format) the info we'd like to
            appear on the server */
        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"
        for(var state_i = ctx.first_state; state_i != ctx.last_state; state_i = state_i.
            successor){
          var stim_type = null, my_stim  = null, pi = ""

          /* "the right way to check if a variable is undefined or not" */
          if(typeof state_i.pool_id !== 'undefined'){
            pi = JSON.parse(JSON.stringify(state_i.pool_id))
          }

          /* assign "stimulus type" keyword */
          if(state_i.wrd_stim){
            stim_type = "word", my_stim = state_i.wrd_stim
          }
          if(state_i.img_stim){
            stim_type = "image", my_stim = state_i.img_stim.fn
          }
          if(!stim_type){
            stim_type = ""
          }
          if(!my_stim){
            my_stim = ""
          }

          /* for a given "state", record a line of data */
          message += href + ","

          /* event_id: global index / line number */
          message += state_index.toString() + ","

          /* task_id */
          message += state_i.task_id + ","

          /* task_type */
          message += state_i.type + ","

          /* trial_id */
          message += state_i.trial_id + ","
          message += Math.round(10. * (state_i.t1 - state_i.t0)) / 10. + ","
          message += parse_date_time(state_i.start_date_time).toString() + ","
          message += parse_date_time(state_i.end_date_time).toString() + ","

          /* ISI */
          if(state_i.type == 'isi'){
            message += state_i.expiry_ms.toString()
          }
          message += ","

          if(!state_i.expiry_ms){
            state_i.expiry_ms = ""
          }

          /* SET */
          message += state_i.expiry_ms.toString() + ","

          /* stimulus type */
          message += stim_type.toString() + ","

          /* stimulus id */
          message += my_stim.toString() + ","

          /* stimulus-pool id */
          message += pi.toString() + ","

          /* user response */
          var response = ""
          for(var k in state_i.key_strokes){
            response += String.fromCharCode(state_i.key_strokes[k])
          }
          message += response + ""

          /* add a newline character */
          message += "\n"

          /* go next */
          ++ state_index
        }

        /* remove last three elements from array: take current page and navigate to:
          ../../xml-receive.py == http://domain/memory/xml-receive.py */
        var words = href.split('/')
        var nwords = words.length
        var target = words.splice(0, nwords-3).join('/') + '/xml-receive.py'

        /* 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 && 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]

    /* 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)
      x.key_required = false
    }else{
      x.set_expiry(time_limit)
      x.key_required = false
    }

    /* discern by image or word, respectively */
    if( typeof(data) === 'object'){
      x.img_stim = data
    }else if(typeof(data) === 'string'){
      x.wrd_stim = data
    }
    x.type = 'study_phase', x.trial_id = trial_index, x.task_id = my_task_id
    x.set_pool_id(p_id)
    if(extra_feedback_this_slide){
      var x_f = feedback(extra_feedback_message, extra_feedback_keys)
    }
  }
  return this
}

/* test phase, formerly known as recognition task - for this phase,
the random selection is shuffled back into the pool -- all elements
from the pool are shown (feedback is recorded).. */
function test_phase(my_pool, isi=0, time_limit=0, extra_feedback=false,
    extra_feedback_message="", extra_feedback_keys=[]){
```

```
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)
  var trial_index = -1, shuffled_data = my_pool.reshuffle(), shuffled = shuffled_data[0],
      deja_vu = shuffled_data[1]
  for(var i  in shuffled){
    var extra_feedback_this_slide = false
    if(extra_feedback != false){
      if(0 == i % parseInt(extra_feedback)){
        extra_feedback_this_slide = true
      }
    }
    my_selection.push([shuffled[i], my_pool.pool_id, deja_vu[i], extra_feedback_this_slide
      ])
  }
}
shuffle(my_selection)

for(var selection_ind in my_selection){
  ++ trial_index

  var a_selection = my_selection[selection_ind]
  var data = a_selection[0], p_id = a_selection[1], deja = a_selection[2],
      extra_feedback_this_slide = a_selection[3]

  /* 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(p_id)
    x.clear_admissible_keys()
    x.key_expiry = false
  }

  var x = new state()
  x.key_required = true
  if(time_limit <= 0){
    x.set_expiry(0)
  }else{
    x.set_expiry(time_limit)
  }

  /* record within the object: do we have deja-vu? */
  x.deja = deja

  /* word or image? */
  if( typeof(data) === 'object'){
    x.img_stim = data
  }else if(typeof(data) ==='string'){
    x.wrd_stim = data
  }
  x.type = 'test_phase', x.trial_id = trial_index, x.task_id = my_task_id
  x.set_pool_id(p_id)
```

```

    if(extra_feedback_this_slide){
      var x_f = feedback(extra_feedback_message, extra_feedback_keys)
    }
  }
  var m = 'Thank you for completing this section. ', end = instructions(m)

  end.action = function(me){
    var msg = m + 'Your score: ' + ctx.questions_correct.toString() + '/' + ctx.
        questions_total.toString() + ". Please press any key."
    me.txt = msg
  }
  return this
}

/* previously known as feedback task */
function feedback(txt, keys){
  var my_task_id = next_task_id ++

  var x = new state()
  x.set_expiry(0)
  x.txt = txt, x.key_required = true
  x.clear_admissible_keys()
  for(var i in keys){
    x.add_admissible_key(keys[i])
  }
  x.type = 'feedback', x.trial_id = 0, x.task_id = my_task_id
}

/* list as many countries as possible during e.g., a 3-minute period (default, 30s) */
function delay_task(txt, delay_time=30000, isi_=500){
  var my_task_id = next_task_id ++, isi = parseInt(isi_)

  /* 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 = 0, x.task_id = my_task_id
    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 */
```

```javascript
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