Programming – course 8
09/01/2014
Martin SZINTE

Programming a behavioral experiment

Plan

1. Introduction
   - Elaborating a project
   - Choosing stimuli
   - Choosing the experimental method
   - Choosing the experimental variables
2. Determining the experimental program settings
   - Data file settings and functions paths
   - Screen settings
   - Visual angle conversion functions
   - Keyboard settings
   - Experimental constants settings
   - Experimental design settings
   - Instructions settings
3. Main loops
   - Main trial loop
   - Stimuli loop
   - Data collection
4. Experiment end
5. Conclusion

1. Introduction

In the preceding courses, we have learned how to use Matlab and the Psychtoolbox in order to program visual and auditory stimuli. From these tools we will here build a behavioral experiment.
For this purpose we will replicate an experiment made in NYU by Yaffa Yeshurun and Marisa Carrasco and publish in 1998 in Nature.

Yeshurun, Y., & Carrasco, M. (1998). Attention improves or impairs visual performance by enhancing spatial resolution. *Nature*, *396* (6706), 72-75.

You can find this paper in Materials/Reference folder.

ELABORATING A PROJECT

Under normal situation you should here determine the project you would like to program. However for the purpose of this course we will focus on Yeshurun and Carrasco project.

Their project concerned visual covert attention (attention that doesn't involve any movement) and on its role for the perception of visual stimuli.
Indeed Yeshurun and Carrasco reported (see reference above) three hypotheses potentially explaining the well known improvement of performance observed in different visual tasks when a position is cued before a target appears.

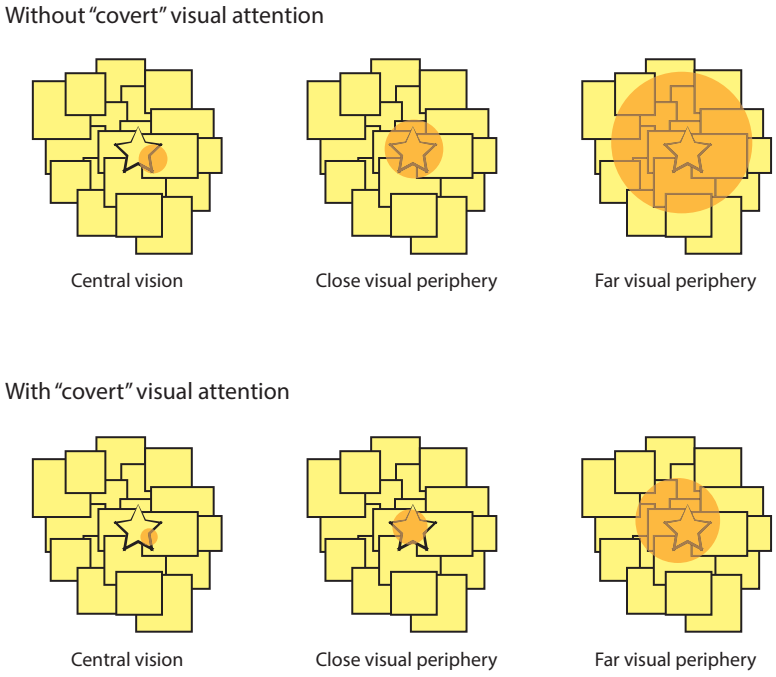
These hypotheses are:

- Improvements are the consequences of a reduction of visual noise (external or internal)

- Improvements are due to a change in decisional criteria
- Improvements are due to a perceptual increase of the signal at the cued location

Following previous research they decided to test the third hypothesis.
They therefore interpreted cued task improvement as an increase in the spatial resolution devoted for the attended position. To simplify the main model of their study, they describe it as follow.

Imagine the visual system as a series of filter of variable sizes: small for position close to our central vision (fovea) and big for very eccentric position. If we had to detect a stimulus of a certain size (a target) in a background composed of other stimuli, such filters in order to be used should match the target size.
Therefore around the fovea (central vision), it should be harder to detect a too big stimulus but the same stimulus will be easier to detect in close visual periphery until a certain point where performance will again drop as the filter will be now too big (see figure below).



The authors therefore suggested that attention reduce the filter size, increasing therefore the spatial resolution of both central and peripheral vision.
Such increase of spatial resolution should involve for a given stimuli size a first drop of performance in central vision together with an improvement in periphery.

We will then here reproduce their experiment starting by the stimuli, method and design determination.

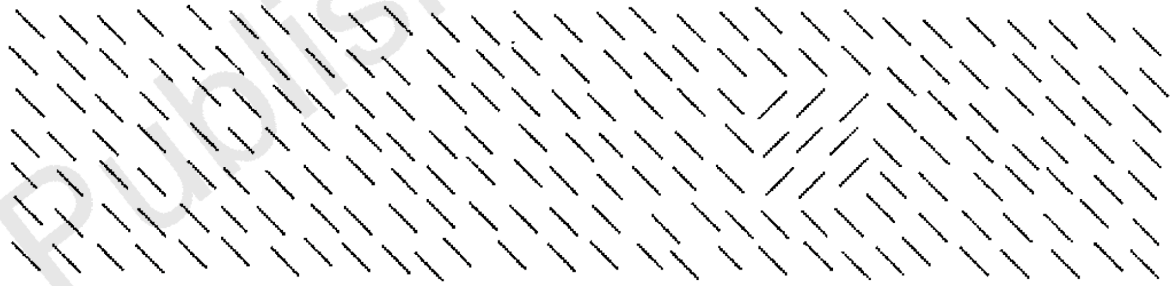
CHOOSING STIMULI

At the stage you would have to find a stimulus that match the hypothesis described above. We will however here simplify your task by simply using the stimuli proposed in the paper.

The authors indeed decided to use stimuli made of a background texture of oriented bars and a target texture composed of 3 by 3 bars oriented in a perpendicular direction relatively to the background (see figure below).

The same stimuli were used in other studies to show that detection of such texture is easier in close periphery that in central vision. The authors then simply added here an attentional task.

a

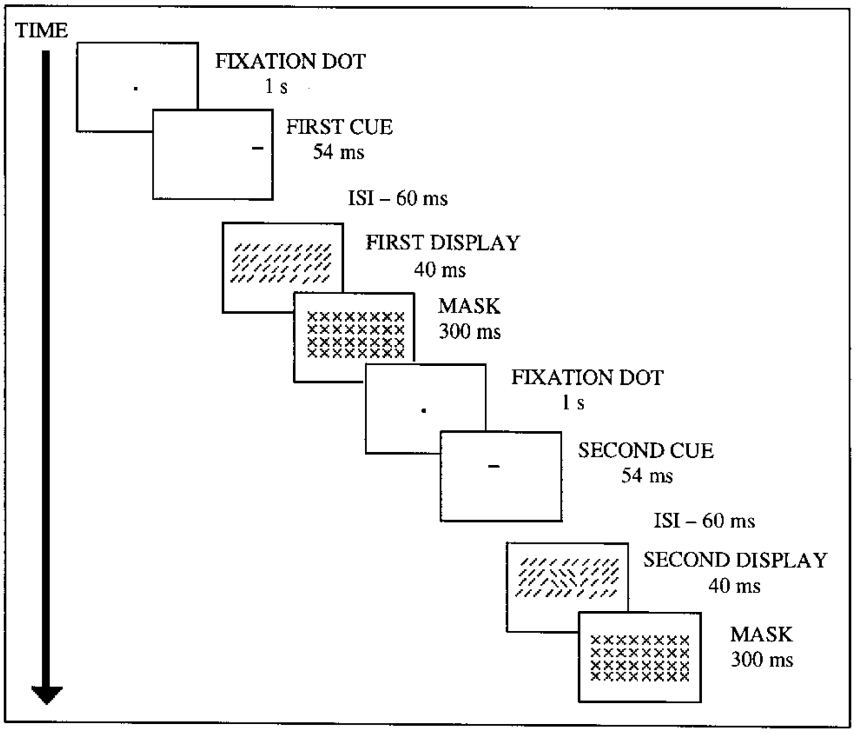


CHOOSING THE EXPERIMENTAL METHOD

It exists several ways to allocate visual attention to a certain position in a task involving texture detection. We will here describe and use the method chosen by Yeshurun and Carrasco (1998).

This method is called "2 interval forced choice" or "2IFC" task and is applied here to the detection of a texture after the presentation of a visual cue (see figure below).

b



For both interval, subjects fixate a central fixation cross; appears then a first cue followed by a texture containing or not the target texture; appears then a mask.

The cue doesn't indicate in which interval the target texture is presented but one of the two cues is always positioned at the target texture position.

At the end of each trial, subjects report as fast as possible via a keyboard if they think that the target texture was in the first or in the second interval.

The authors evaluated detection performances and reaction times for textures presented at different eccentricities to the left or to the right of the fixation cross.
They compared the results to a neutral condition in which instead of a single cue, two bars covered all possible position.

CHOOSING THE EXPERIMENTAL VARIABLES

The variables in this experiment were then:

1) Target interval (2 modalities):
    a. 1st interval
    b. 2nd interval
2) Attentional deployment condition (2 modalities):
    a. Local cue
    b. Neutral cue
3) Texture eccentricity (15 modalities)
    a. Central vision (0 degree)
    b. 1 degree in periphery
    …
    o. 14 degrees in periphery

Each trial will therefore be a combination of these 3 variables that will be presented randomly ten times each.

2. Determining the experimental program settings

As the stimuli, experimental design and variable are now chosen we can start to program the experiment using Matlab. For this purpose I will describe here my experimental template that you will be able to easily modify in order to program your next experiments.

This template is composed of several sub-functions organized in different folders.

Look for "Yeshurun98" folder in the material folder.

In the main experimental folder "Yeshurun98" you will find 7 different sub-folders:
"Config": this folder contains configuration functions (e.g. screen, keyboard)
"Conversion": this folder contains conversion functions (e.g converter of deg to pixel)
"Data": this folder contains the data that we will collect.
" Instructions": this folder contains subject instruction functions.
"Main": this folder contains the main functions driving all the other functions
"Stim": this folder contains stimuli functions
"Trials": this folder contains block and trial loop functions.

Look for "expLauncher.m" in Materials/Yeshurun98/Main/ folder, add it to your path and execute it through the command window to launch the experiment.

=> expLauncher;

```
%% General experimenter launcher %%
%  ============================  %
% Last edit : 09/01/2014
% By : Martin SZINTE
% Projet : Programming course - Experiment template
% Description : Adaptation of the Yeshurun and Carrasco (1998) Nature.

%% Initial settings
% Initial closing :
clear all;clear mex;clear functions;
close all;home;ListenChar(1);tic

% General settings
const.expName       = 'Yeshurun98';          % experiment name and folder
const.expStart         =  0;                                  %  Start  of  a  recording  exp
0 = NO   , 1 = YES

% Screen
const.desiredFD     = 60;                    % Desired refresh rate
const.desiredRes    = [1024,768];            % Desired resolution

% Path :
dir = (which('expLauncher'));cd(dir(1:end-18));

% Block definition
numBlockMain = 1;                            % number of block to play per run time
const.numBlockTot  = 10;                     % total number of block before analysis

% Subject configuration :
if const.expStart
    const.sjct = input(sprintf('\n\tInitials: '),'s');
    const.sjctCode = sprintf('%s_%s',const.sjct,const.expName);
    const.fromBlock  = input(sprintf('\n\tFrom Block nb: '));
    if const.fromBlock == 1;
        const.sjct_age = input(sprintf('\n\tAge: '));
        const.sjct_gender = input(sprintf('\n\tGender (M or F): '),'s');
    end
else
    const.sjct = 'Anon';const.fromBlock = 1;const.sjct_age = 'XX';const.sjct_gender = 'X';
end
const.sjctCode = sprintf('%s_%s',const.sjct,const.expName);


%% Main experimental code
for block = const.fromBlock:(const.fromBlock+numBlockMain-1)
    const.fromBlock = block;
    main(const);clear expDes
end
```

This function defines a few settings as simple variables (e.g. const.expStart => flags determining if the code should launch the experiment for real or just in debugging mode), checks subject details (e.g. initials, age, gender) and launches the core function *main.m* for a pre-defined amount of time.

Question 1: What is the role of the second part of the "if" loop?

The second part of this loop allows the experimenter to launch the code in a debugging mode.
In such mode the data will not be attributed to a certain subject and the execution of the code will be faster (no ask of subject details) in order to quickly debug the code.

Look for “main.m” in the Materials/Yeshurun98/Main/ folder.

```
function main(const)
% ----------------------------------------------------------------------
% main(const)
% ----------------------------------------------------------------------
% Goal of the function :
% Main code of experiment
% ----------------------------------------------------------------------
% Input(s) :
% const : struct containing subject information and saving files.
% ----------------------------------------------------------------------
% Output(s):
% none
% ----------------------------------------------------------------------
% Function created by Martin SZINTE (martin.szinte@gmail.com)
% Last update : 08 / 01 / 2014
% Project : Yeshurun98
% Version : -
% ----------------------------------------------------------------------

% File directory :
[const] = dirSaveFile(const);

% Screen configuration :
[scr] = scrConfig(const);

% Keyboard configuration :
[my_key] = keyConfig;

% Experimental design configuration :
[expDes] = designConfig(const);

% Experimental constant :
[const] = constConfig(scr,const);

% Instruction file :
[textExp,button] = instructionConfig;

% Open screen window :
[scr.main,scr.rect] = Screen('OpenWindow',scr.scr_num,[0 0 0],[], scr.clr_depth,2);
Screen('BlendFunction', scr.main, GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
priorityLevel = MaxPriority(scr.main);Priority(priorityLevel);

% Main part :
if const.expStart;ListenChar(2);end
GetSecs;runTrials(scr,const,expDes,my_key,textExp,button);

% End
overDone

end
```

This function is the core of this template. It launches sequentially:

dirSaveFile.m:	function creating the saving file directory and Matlab function path
scrConfig.m:	function defining screen configuration
keyConfig.m:	function defining keyboard configuration
designConfig.m:	function defining the experimental design
constConfig.m:	function defining experimental constants
instructionConfig.m:	function defining the subject instructions
runTrials.m:	function launching each trials
overDone.m:	quitting function

We will now see in details each of these functions.

DATA FILE SETTINGS AND FUNCTION PATHS

Look for “dirSaveFile.m” in the Materials/Yeshurun98/Main/ folder.

```
function [const] = dirSaveFile(const)
% ----------------------------------------------------------------------
% [const]=dirSaveFile(const)
% ----------------------------------------------------------------------
% Goal of the function :
% Make directory and saving files.
% ----------------------------------------------------------------------
% Input(s) :
% const : struct containing a lot of constant configuration
% ----------------------------------------------------------------------
% Output(s):
% const : struct containing a lot of constant configuration
% ----------------------------------------------------------------------
% Function created by Martin SZINTE (martin.szinte@gmail.com)
% Last update : 08 / 01 / 2014
% Project : Yeshurun98
% Version : -
% ----------------------------------------------------------------------

% Creates Directory
if ~isdir(sprintf('Data/%s',const.sjct));
    mkdir(sprintf('Data/%s',const.sjct));
    cd (sprintf('Data/%s',const.sjct));
else
    cd (sprintf('Data/%s',const.sjct));
end

if const.expStart
    expDir = sprintf('ExpData/Block%i',const.fromBlock);
    if ~isdir(expDir);
        mkdir(expDir);
        cd(expDir);
    else
        aswErase = input('\n This file allready exist, do you want to overwrite it ? (Y or N)
','s');
        if aswErase == 'N'
            error('Please restart the program with correct input.')
        elseif aswErase == 'Y'
            cd(expDir);
        else
            error('Incorrect input => Please restart the program with correct input.')
        end
    end

else
    const.c = clock;

    debugDir = sprintf('DebugData/%i-%i_trials/',const.c(2),const.c(3));
    if ~isdir(debugDir);
        mkdir(debugDir);
        cd (debugDir);
    else
        cd (debugDir);
    end

end

% Defines saving file names
const.scr_fileDat =         sprintf('scr_file%s.dat',const.sjctCode);
const.scr_fileMat =         sprintf('scr_file%s.mat',const.sjctCode);
const.aud_fileMat =         sprintf('aud_file%s.mat',const.sjctCode);
const.const_fileDat =       sprintf('const_file%s.dat',const.sjctCode);
const.const_fileMat =       sprintf('const_file%s.mat',const.sjctCode);
const.expRes_fileCsv =      sprintf('expRes%s.csv',const.sjctCode);
const.expRes_fileMat =      sprintf('expRes%s.mat',const.sjctCode);
const.design_fileMat =      sprintf('design%s.mat',const.sjctCode);

% Add path from the location of the data file folder
addpath('../../../../Config/');
addpath('../../../../Conversion/');
addpath('../../../../Data/');
addpath('../../../../Instructions/');
addpath('../../../../Main/');
```

```
addpath('../../../../Stim/');
addpath('../../../../Trials/');

end
```

In this function we will use subject details to generate a customized data folder where all settings and behavioral results will be stored. This function also changes the current directory of Matlab to this new folder and later adds all template sub-folders to the Matlab path.

Question 2: What is the role of the Matlab built-in function "mkdir"?

This function creates new directories in which we will store our different files. This way we avoid any later problem of data organization and data overwriting by automatizing the crating of a folder for each subject and experimental block.

SCREEN SETTINGS

Look for "scrConfig.m" in the Materials/Yeshurun98/Config/ folder.

```
function [scr]=scrConfig(const)
% ------------------------------------------------------------------------
% [scr]=scrConfig(const)
% ------------------------------------------------------------------------
% Goal of the function :
% Give all information about the screen and the monitor.
% ------------------------------------------------------------------------
% Input(s) :
% const : struct containing subject information and saving files.
% ------------------------------------------------------------------------
% Output(s):
% scr : struct containing all screen configuration.
% ------------------------------------------------------------------------
% Function created by Martin SZINTE (martin.szinte@gmail.com)
% Last update : 08 / 01 / 2014
% Project : Yeshurun98
% Version : -
% ------------------------------------------------------------------------

% Number of the exp screen:
scr.all = Screen('Screens');
scr.scr_num = max(scr.all);

% Size of the display :
[scr.disp_sizeX,scr.disp_sizeY] = Screen('DisplaySize',scr.scr_num);
scr.disp_sizeX = 400;scr.disp_sizeY=300;

% Screen resolution (pixel) :
[scr.scr_sizeX, scr.scr_sizeY]=Screen('WindowSize', scr.scr_num);
if (scr.scr_sizeX ~= const.desiredRes(1) || scr.scr_sizeY ~= const.desiredRes(2)) &&
const.expStart
    error('Incorrect screen resolution => Please restart the program after changing the
resolution to [%i,%i]',const.desiredRes(1),const.desiredRes(2));
end

% Frame rate : (fps)
scr.frame_duration =1/(Screen('FrameRate',scr.scr_num));
if scr.frame_duration == inf;
    scr.frame_duration = 1/60;
elseif scr.frame_duration ==0;
    scr.frame_duration = 1/60;
end
scr.fd = scr.frame_duration;

% Frame rate : (hertz)
scr.hz = 1/(scr.frame_duration);
if (scr.hz >= 1.1*const.desiredFD || scr.hz <= 0.9*const.desiredFD) && const.expStart
    error('Incorrect refresh rate => Please restart the program after changing the refresh
rate to %i Hz',const.desiredFD);
end

% Subject dist
scr.dist = 60;
```

```
% Center of the screen :
scr.x_mid = (scr.scr_sizeX/2.0);
scr.y_mid = (scr.scr_sizeY/2.0);
scr.mid = [scr.x_mid,scr.y_mid];

%% Saving procedure :
scr_file = fopen(const.scr_fileDat,'w');
fprintf(scr_file,'Resolution size X (pxl):\t%i\n',scr.scr_sizeX);
fprintf(scr_file,'Resolution size Y (pxl):\t%i\n',scr.scr_sizeY);
fprintf(scr_file,'Monitor size X (mm):\t%i\n',scr.disp_sizeX);
fprintf(scr_file,'Monitor size Y (mm):\t%i\n',scr.disp_sizeY);
fprintf(scr_file,'Subject distance (cm):\t%i\n',scr.dist);
fprintf(scr_file,'Frame duration (fps):\t%i\n',scr.frame_duration);
fprintf(scr_file,'Refresh Rate (hz):\t%i\n',scr.hz);
fclose('all');

% .mat file
save(const.scr_fileMat,'scr');

end
```

This function is essential for the good display of your stimuli.
It uses several Screen() sub-functions that automatically get and save your monitor settings.

This function determines:

`scr.scr_num`:	the monitor number on which the stimuli will be drawn
`scr.disp_sizeX/Y`:	the monitor size in mm
`scr.scr_sizeX/Y`:	the screen resolution in pixels
`scr.frame_duration`:	the screen frame duration in msec
`scr.hz`:	the screen frame rate in hertz
`scr.dist`:	the screen viewing distance in cm
`scr.mid`:	the screen center coordinates in pixels

All these values will be store in a Matlab structure “scr” and will be later used to define the size of our stimuli as well as their durations.

Question 3: Why should we define manually the monitor size?

In line 26 of *scrConfig.m* the monitor size is manually defined. Although Screen(‘DiplaySize’) of the preceding line is supposed to return such values, such function generally fails with old monitors and return fake values. Another reason to specify manually the screen size is when you use a resolution non-adapted to your screen (e.g. 4/3 resolution such as 1024x768 on a 16/9 screen) and creating margins on sides. In such situation the part of the monitor used for the stimuli display will not correspond to its physical size of your monitor and will lead to errors when visual degrees will be computed.

Question 4: What is the role of the saving procedure at the end of the function?

Although you might never use these values for your data analysis it is always good to record all settings of your experiment in case of. These values might become useful when you will have to write an experiment report or to check why for one particular subject data looks weird (e.g. it might happen that someone changes your screen settings in between subjects or blocks, you will be able to check such mistake only if you systematically save all settings).

VISUAL ANGLE CONVERTING FUNCTIONS

To describe your experiment you should specify the size of your stimuli.
You can then decide to speak of cm or mm, however these values are meaningless without the viewing distance. To simplify you can describe your visual stimuli in measure of visual angle degrees. Such measure includes both the size and the viewing distance.

The approximation calculation of a visual angle degree is the following:

$$\alpha = \frac{\boldsymbol{T} * \mathbf{57.3}}{\boldsymbol{d}}$$

α = measure in visual degrees
T = measure in cm
d = viewing distance

However we will prefer the exact formula to this approximated one and will then add to the template a series of converting functions able to convert:

- cm to pixel (see *Materials/Yeshurun98/Conversion/cm2pix.m*)
- pixel to cm (see *Materials/Yeshurun98/Conversion/pix2cm.m*)
- pixel to visual angle degrees (see *Materials/Yeshurun98/Conversion/pix2vaDeg.m*)
- visual angle degrees to cm (see *Materials/Yeshurun98/Conversion/vaDeg2cm.m*)
- visual angle degrees to pixel (see *Materials/Yeshurun98/Conversion/vaDeg2pix.m*)

Remark:
These conversion functions use screen information defined in *scrConfig.m* function and saved in the Matlab structure "scr".

Question 5: Why coding visual stimuli in visual angle degree?
It is not mandatory to defined stimuli in visual angle degree however when you will have to report your experiment such conversion will be essential. You can then of course imagine converting at posteriori but you will rarely fall on round values making the report a bit odd.

KEYBOARD SETTINGS

In this configuration functions we will determine the keyboard buttons (the PTB button value of each button) need for our subjects to report the interval where the target appeared. We will therefore only need to specify 2 buttons for example the right and left shift button of the keyboard together with the escape and space button for emergency quit and continue button.

Look for "keyConfig.m" in the Materials/Yeshurun98/Config/ folder.

```
function [my_key]=keyConfig
% ----------------------------------------------------------------------
% [my_key]=keyConfig
% ----------------------------------------------------------------------
% Goal of the function :
% Unify key names and return a structure containing each key names.
% ----------------------------------------------------------------------
% Input(s) :
% none
% ----------------------------------------------------------------------
% Output(s):
% my_key : structure containing all keyboard names.
% ----------------------------------------------------------------------
% Function created by Martin SZINTE (martin.szinte@gmail.com)
% Last update : 08 / 01 / 2014
% Project : Yeshurun98
% Version : -
% ----------------------------------------------------------------------

KbName('UnifyKeyNames');
my_key.escape       = KbName('Escape');
my_key.space        = KbName('Space');
my_key.rightShift   = KbName('RightShift');
my_key.leftShift    = KbName('LeftShift');

end
```

EXPERIMENTAL CONSTANTS SETTINGS

The experimental constants settings correspond to all settings that will not change across trials. In general they correspond to:

- Stimuli sizes color, background color
- Stimuli durations
- Values of the potential experimental variables
- Instruction text font and size
- …

Look for "constConfig.m" in the Materials/Yeshurun98/Config/ folder.

```
function [const]=constConfig(scr,const)
% ----------------------------------------------------------------------
% [const]=constConfig(const)
% ----------------------------------------------------------------------
% Goal of the function :
% Compute all constant data of this experiment.
% ----------------------------------------------------------------------
% Input(s) :
% scr : window pointer
% const : struct containg previous constant configurations.
% ----------------------------------------------------------------------
% Output(s):
% const : struct containing all constant data.
% ----------------------------------------------------------------------
% Function created by Martin SZINTE (martin.szinte@gmail.com)
% Last update : 08 / 01 / 2014
% Project : Yeshurun98
% Version : -
% ----------------------------------------------------------------------
```

```
% Instructions
const.text_size = 20       ;
const.text_font = 'Helvetica';

% Color Configuration :
const.red =     [200,   0,   0];
const.green =   [  0, 200,   0];
const.blue =    [  0,   0, 200];
const.orange =  [255, 150,   0];
const.gray =    [127, 127, 127];
const.colBG =   [127, 127, 127];
const.white =   [255, 255, 255];
const.black =   [  0,   0,   0];

% Time
const.my_clock_ini = clock;

% Fixation cross
const.sideFP_val     = 0.5;         [const.sideFP_X,const.sideFP_Y] =
vaDeg2pix(const.sideFP_val,scr);
const.thicknessFP_val  = 0.1;       [const.thicknessFP_X,const.thicknessFP_Y] =
vaDeg2pix(const.thicknessFP_val,scr);

% Background settings
const.jitterX_min_val = 0;                                          % min value of X
jitter (deg)
[const.jitterX_min,kiy] = vaDeg2pix(const.jitterX_min_val,scr);         % min value of X
jitter (pix)
const.jitterX_max_val = 0.3;                                        % max value of X
jitter (deg)
[const.jitterX_max,kiy] = vaDeg2pix(const.jitterX_max_val,scr);         % max value of X
jitter (deg)
const.jitterX_val = linspace(const.jitterX_min,const.jitterX_max,5);

const.jitterY_min_val = 0;                                          % min value of Y
jitter (deg)
[kix,const.jitterY_min] = vaDeg2pix(const.jitterY_min_val,scr);         % min value of Y
jitter (pix)
const.jitterY_max_val = 0.3;                                        % max value of Y
jitter (deg)
[kix,const.jitterY_max] = vaDeg2pix(const.jitterY_max_val,scr);         % max value of Y
jitter (pix)
const.jitterY_val = linspace(const.jitterY_min,const.jitterY_max,5);


const.stim_sizeX_val = 0.8;                                         % size X of one
element (deg)
[const.stim_sizeX,kiy] = vaDeg2pix(const.stim_sizeX_val,scr);           % size X of one
element (pix)
const.stim_sizeY_val = 0.8;                                         % size Y of one
element (deg)
[kix,const.stim_sizeY] = vaDeg2pix(const.stim_sizeY_val,scr);           % size Y of one
element (pix)

const.num_col = 29;                                                 % number of collumn
const.stimBG_sizeX_val = 31;                                        % size X of BG stimuli
(deg)
[const.stimBG_sizeX,kiy] = vaDeg2pix(const.stimBG_sizeX_val,scr);       % size X of BG stimuli
(pix)
const.spaceX = const.stimBG_sizeX/const.num_col;                    % size X of 1 stimiuli
(pix)

for t_col = 1:const.num_col
    const.mat_col(t_col) = scr.x_mid - (const.stimBG_sizeX/2) + ((t_col-1)*const.spaceX) +
(const.spaceX/2);
end

const.num_raw = 7;                                                  % number of raw
const.stimBG_sizeY_val = 7.5;                                       % size Y of BG stimuli
(deg)
[kix,const.stimBG_sizeY] = vaDeg2pix(const.stimBG_sizeY_val,scr);       % size Y of BG stimuli
(pix)
const.spaceY = const.stimBG_sizeY/const.num_raw;                    % size X of 1 stimiuli
(pix)

for t_raw = 1:const.num_raw
    const.mat_raw(t_raw) = scr.y_mid - (const.stimBG_sizeY/2) + ((t_raw-1)*const.spaceY) +
(const.spaceY/2);
end
```

```
% Cue settings
const.cueAtt_sizeX_val = 0.7;                                          % size X of the
attentionnal cue (deg)
[const.cueAtt_sizeX,kiy] = vaDeg2pix(const.cueAtt_sizeX_val,scr);      % size X of the
attentionnal cue (pix)
const.cueAtt_sizeY_val = 0.2;                                          % size Y of the
attentionnal cue (deg)
[kix,const.cueAtt_sizeY] = vaDeg2pix(const.cueAtt_sizeY_val,scr);      % size Y of the
attentionnal cue (pix)

const.cueNeu_sizeX_val = 31;                                           % size X of the
neutral cue (deg)
[const.cueNeu_sizeX,kiy] = vaDeg2pix(const.cueNeu_sizeX_val,scr);      % size X of the
neutral cue (pix)
const.cueNeu_sizeY_val = 0.2;                                          % size Y of the
neutral cue (deg)
[kix,const.cueNeu_sizeY] = vaDeg2pix(const.cueAtt_sizeY_val,scr);      % size Y of the
neutral cue (pix)

const.cue_addY_val = 0.3;                                              % ammount of Y shift
above or below FP (deg)
[kix,const.cue_addY] = vaDeg2pix(const.cue_addY_val,scr);              % ammount of Y shift
above or below FP (deg)

% Answer settings
const.radCircle_val = 1.5;                                             % radius of the answer
circle (deg)
[const.radCircle,kiy] = vaDeg2pix(const.radCircle_val,scr);            % radius of the answer
circle (pix)

% Experiental timing settings
const.T1  = 1.0;                % fixation time 1       = 1  sec
const.T2  = 0.050;              % fixation cue 1        = 50 msec (for 54 in article)
const.T3  = 0.0667;             % ISI                   = 67 msec (for 60 in article)
const.T4  = 0.0333;             % Stimuli 1             = 33 msec (for 35 in article)
const.T5  = 0.150;              % Mask 1                = 300 msec (for 300 in article)

% RQ: 300 msec demander mais double le temps bizarrement
% Solution : mettre 150 msec

const.T6  = const.T1;           % fixation time 1       = 1  sec
const.T7  = const.T2;           % fixation cue 2        = 50 msec (for 54 in article)
const.T8  = const.T3;           % ISI                   = 67 msec (for 60 in article)
const.T9  = const.T4;           % Stimuli 2             = 33 msec (for 35 in article)
const.T10 = const.T5;           % Mask 2                = 300 msec (for 300 in article)

const.numFrm_T1  =  round(const.T1/scr.frame_duration);
const.numFrm_T2  =  round(const.T2/scr.frame_duration);
const.numFrm_T3  =  round(const.T3/scr.frame_duration);
const.numFrm_T4  =  round(const.T4/scr.frame_duration);
const.numFrm_T5  =  round(const.T5/scr.frame_duration);
const.numFrm_T6  =  round(const.T6/scr.frame_duration);
const.numFrm_T7  =  round(const.T7/scr.frame_duration);
const.numFrm_T8  =  round(const.T8/scr.frame_duration);
const.numFrm_T9  =  round(const.T9/scr.frame_duration);
const.numFrm_T10 =  round(const.T10/scr.frame_duration);
const.numFrm_Tot =  const.numFrm_T1 + const.numFrm_T2 + const.numFrm_T3 + const.numFrm_T4 +
const.numFrm_T5 + ...
                    const.numFrm_T6 + const.numFrm_T7 + const.numFrm_T8 + const.numFrm_T9 +
const.numFrm_T10;

const.numFrm_T1_start  = 1;                              const.numFrm_T1_end  =
const.numFrm_T1_start  + const.numFrm_T1-1;
const.numFrm_T2_start  = const.numFrm_T1_end+1;          const.numFrm_T2_end  =
const.numFrm_T2_start  + const.numFrm_T2-1;
const.numFrm_T3_start  = const.numFrm_T2_end+1;          const.numFrm_T3_end  =
const.numFrm_T3_start  + const.numFrm_T3-1;
const.numFrm_T4_start  = const.numFrm_T3_end+1;          const.numFrm_T4_end  =
const.numFrm_T4_start  + const.numFrm_T4-1;
const.numFrm_T5_start  = const.numFrm_T4_end+1;          const.numFrm_T5_end  =
const.numFrm_T5_start  + const.numFrm_T5-1;
const.numFrm_T6_start  = const.numFrm_T5_end+1;          const.numFrm_T6_end  =
const.numFrm_T6_start  + const.numFrm_T6-1;
const.numFrm_T7_start  = const.numFrm_T6_end+1;          const.numFrm_T7_end  =
const.numFrm_T7_start  + const.numFrm_T7-1;
const.numFrm_T8_start  = const.numFrm_T7_end+1;          const.numFrm_T8_end  =
const.numFrm_T8_start  + const.numFrm_T8-1;
const.numFrm_T9_start  = const.numFrm_T8_end+1;          const.numFrm_T9_end  =
```

```
const.numFrm_T9_start  + const.numFrm_T9-1;
const.numFrm_T10_start = const.numFrm_T9_end+1;          const.numFrm_T10_end =
const.numFrm_T10_start + const.numFrm_T10-1;

%% Saving procedure :
const_file = fopen(const.const_fileDat,'w');
fprintf(const_file,'Subject initial :\t%s\n',const.sjct);
if const.fromBlock == 1
    fprintf(const_file,'Subject age :\t%s\n',const.sjct_age);
    fprintf(const_file,'Subject gender :\t%s\n',const.sjct_gender);
end
fprintf(const_file,'Date : \t%i-%i-
%i\n',const.my_clock_ini(3),const.my_clock_ini(2),const.my_clock_ini(1));
fprintf(const_file,'Starting time : \t%ih%i\n',const.my_clock_ini(4),const.my_clock_ini(5));
fclose('all');

% .mat file
save(const.const_fileMat,'const');

end
```

Question 6: What is the role of the Matlab built-in function "save"?

This function saves in .mat data format the content of a Matlab structure (e.g. here "const")

EXPERIEMENTAL DESIGN SETTINGS

This configuration function determines the experimental variables and their modalities.
It creates different vectors for the different variables and concatenates them inside a matrix of all possible combinations. Once this matrix is done, the function randomized combinations in order to determine expDes.expMat, a matrix defining each trial composition.

Look for "designConfig.m" in the Materials/Yeshurun98/Config/ folder.

```
function [expDes]=designConfig(const)
% ------------------------------------------------------------------------
% [expDes]=designConfig(const)
% ------------------------------------------------------------------------
% Goal of the function :
% Compute an experimental randomised matrix containing all variable data
% used in the experiment.
% ------------------------------------------------------------------------
% Input(s) :
% const : struct containing all constant configurations.
% ------------------------------------------------------------------------
% Output(s):
% expDes : struct containg all variable data randomised.
% ------------------------------------------------------------------------
% Function created by Martin SZINTE (martin.szinte@gmail.com)
% Last update : 08 / 01 / 2014
% Project : Yeshurun98
% Version : -
% ------------------------------------------------------------------------

%% Expemental variables

% Var 1 : Target eccentricity [27 modalitie(s)]
expDes.oneV =[1:27]';
    %  1 = 0 ecc
    %  2:14 = positive ecc.
    % 15:27 = negative ecc.

% Var 2 : Target interval [2 modalitie(s)]
expDes.twoV = [1;2];
    %  1 = 1st interval
    %  2 = 2nd interval

% Var 3 : Attention condition [2 modalitie(s)]
expDes.threeV = [1;2];
    %  1 = Cued trial
    %  2 = Neutral cue trial

% Rand 1 : Non-target eccentricity [27 modalities - 1]
```

```
expDes.oneR = [1:27]';

%% Experimental configuration :
expDes.var1_list = expDes.oneV;
expDes.nb_var1= numel(expDes.var1_list);
expDes.var2_list = expDes.twoV;
expDes.nb_var2= numel(expDes.var2_list);
expDes.var3_list = expDes.threeV;
expDes.nb_var3= numel(expDes.var3_list);

expDes.random1_list = expDes.oneR;
expDes.nb_random1= numel(expDes.random1_list);

expDes.nb_var  = 3;
expDes.nb_rand = 1;

expDes.nb_repeat = 1;
expDes.nb_trials = expDes.nb_var1 * expDes.nb_var2 * expDes.nb_var3 * expDes.nb_repeat;

expDes.timePauseMin = 15;
expDes.timePause = expDes.timePauseMin*60;

%% Experimental loop
trialMat = zeros(expDes.nb_trials,expDes.nb_var);
ii = 0;
for iv1=1:expDes.nb_var1
    for iv2=1:expDes.nb_var2
        for iv3=1:expDes.nb_var3
            for rr= 1:expDes.nb_repeat
                ii = ii + 1;
                trialMat(ii, 1) = iv1;
                trialMat(ii, 2) = iv2;
                trialMat(ii, 3) = iv3;
            end
        end
    end
end

rand('state',sum(100*clock));
trialMat = trialMat(randperm(expDes.nb_trials),:);
for t_trial = 1:expDes.nb_trials

    rand_var1 = expDes.var1_list(trialMat(t_trial,1),:);
    rand_var2 = expDes.var2_list(trialMat(t_trial,2),:);
    rand_var3 = expDes.var3_list(trialMat(t_trial,3),:);

    randVal1 = randperm(expDes.nb_random1); rand_random1 = expDes.oneR(randVal1(1));

    while rand_random1 == rand_var1
        randRandom1 = randperm(expDes.nb_random1);
        rand_random1 = randRandom1(1);
    end

    expDes.j = t_trial;
    expDes.expMat(expDes.j,:)=
[const.fromBlock,t_trial,rand_var1,rand_var2,rand_var3,rand_random1];
end


%% Saving procedure :
% .mat file
save(const.design_fileMat,'expDes');

end
```

INSTRUCTIONS SETTINGS

This last configuration function determines two Matlab structures: “textExp” and “button”.
These structures contained all instructions text lines as well as continuing and quitting lines instructions. Such function allows a rapid determination of instruction for your following projects, indeed you will simply have to change text lines in *instructionConfig.m* and the *instructions.m* function will automatically draw such text in a formatted way.

Look for “instructionConfig.m” in the Materials/Yeshurun98/Instructions/ folder.

```
function [textExp,button] = instructionConfig
% ------------------------------------------------------------------------
% [textExp,button] = instructionConfig
% ------------------------------------------------------------------------
% Goal of the function :
% Write text of calibration and general instruction for the experiment.
% ------------------------------------------------------------------------
% Input(s) :
% (none)
% ------------------------------------------------------------------------
% Output(s):
% textExp : struct containing all text of general instructions.
% button : struct containing all button instructions.
% ------------------------------------------------------------------------
% Function created by Martin SZINTE (martin.szinte@gmail.com)
% Last update : 08 / 01 / 2014
% Project : Yeshurun98
% Version : -
% ------------------------------------------------------------------------

%% Pause :
pause_l1 = 'Pause :';
pause_l2 = 'Please take a break.';
pause_b1 = '-----------------  PRESS [SPACE] TO CONTINUE  -----------------';

textExp.pause = {pause_l1;pause_l2};
button.pause = {pause_b1};

%% End :
end_l1 = 'Thank you ...';
end_b1 = '--------------------  PRESS [SPACE] TO QUIT  --------------------';

textExp.end = {end_l1};
button.end =  {end_b1};

%% Main instruction :

instruction_l1  =  'In two following sequences, a central white cross will appear on ';
instruction_l2  =  'the screen followed by a small or two long horizontal bars ';
instruction_l3  =  'indicating the possible location of a 3x3 texture array';
instruction_l4  =  'oriented differently compared to the background.';
instruction_l5  =  '';
instruction_l6  =  'During each trial keep fixating at the central white cross position, ';
instruction_l7  =  'draw your attention at the location of the horizontal bar and determine';
instruction_l8  =  'in which sequence does the 3x3 texture array appeared.';
instruction_l9  =  '';
instruction_l10 =  'If the 3x3 texture array was in the 1st sequence : press [RIGHT SHIFT]';
instruction_l11 =  'If the 3x3 texture array was in the 2nd sequence : press [LEFT SHIFT]';

instruction_b1 = '-----------------  PRESS [SPACE] TO CONTINUE  -----------------';

textExp.instruction1=
{instruction_l1;instruction_l2;instruction_l3;instruction_l4;instruction_l5;...

instruction_l6;instruction_l7;instruction_l8;instruction_l9;instruction_l10;instruction_l11};
button.instruction1 =  {instruction_b1};

end
```

Look for “instructions.m” in the Materials/Yeshurun98/Instructions/ folder.

```
function instructions(scr,const,my_key,text,button)
% ----------------------------------------------------------------------
% instructions(scr,const,my_key,text,button)
% ----------------------------------------------------------------------
% Goal of the function :
% Display instructions write in a specified matrix.
% ----------------------------------------------------------------------
% Input(s) :
% scr : main window pointer.
% const : struct containing all the constant configurations.
% text : library of the type {}.
% ----------------------------------------------------------------------
% Output(s):
% (none)
% ----------------------------------------------------------------------
% Function created by Martin SZINTE (martin.szinte@gmail.com)
% Last update : 08 / 01 / 2014
% Project : Yeshurun98
% Version : -
% ----------------------------------------------------------------------
while KbCheck; end
KbName('UnifyKeyNames');

push_button = 0;
while ~push_button;

    Screen('Preference', 'TextAntiAliasing',1);
    Screen('TextSize',scr.main, const.text_size);
    Screen ('TextFont', scr.main, const.text_font);
    Screen('FillRect', scr.main, const.gray);

    sizeT = size(text);
    sizeB = size(button);
    lines = sizeT(1)+sizeB(1)+2;
    bound = Screen('TextBounds',scr.main,button{1,:});
    espace = ((const.text_size)*1.50);
    first_line = scr.y_mid - ((round(lines/2))*espace);

    addi = 0;
    for t_lines = 1:sizeT(1)
        Screen('DrawText',scr.main,text{t_lines,:},scr.x_mid-
bound(3)/2,first_line+addi*espace, const.white);
        addi = addi+1;
    end
    addi = addi+2;
    for b_lines = 1:sizeB(1)
        Screen('DrawText',scr.main,button{b_lines,:},scr.x_mid-
bound(3)/2,first_line+addi*espace, const.orange);
    end
    Screen('Flip',scr.main);

    [ keyIsDown, seconds, keyCode ] = KbCheck;
    if keyIsDown
        if keyCode(my_key.space)
            push_button=1;
        elseif keyCode(my_key.escape) && ~const.expStart
            overDone;
        end
    end
end
```

With all this settings defined, our program is now ready to start the trials display and data saving.

3. Main loops

In the first part of this course we have seen all settings necessary to program a controlled behavioral experiment with Matlab. This second part introduces the main trials loop (*runTrials.m*) as well as the single trial loop (*runSingleTrial.m*).

MAIN TRIAL LOOP

The *runTrials.m* function contains the main trial loop launching each trial one after each other.
If you consider using additional apparatus (eye tracker, eeg, fMRI…) this function shoudl also be used to start and stop the recording apparatus before and after the main trial launch.

Look for "runTrials.m" in the Materials/Yeshurun98/Trials/ folder.

```
function runTrials(scr,const,expDes,my_key,textExp,button)
% ----------------------------------------------------------------------
% runTrials(scr,const,expDes,my_key,textExp,button)
% ----------------------------------------------------------------------
% Goal of the function :
% Main trial function, display the trial function and save the experi-
% -mental data in different files.
% ----------------------------------------------------------------------
% Input(s) :
% scr : window pointer
% const : struct containing all the constant configurations.
% expDes : struct containing all the variable design and configurations.
% my_key : keyborad keys names
% textExp : struct contanining all instruction text.
% button : struct containing all button text.
% ----------------------------------------------------------------------
% Output(s):
% none
% ----------------------------------------------------------------------
% Function created by Martin SZINTE (martin.szinte@gmail.com)
% Last update : 08 / 01 / 2014
% Project : Yeshurun98
% Version : -
% ----------------------------------------------------------------------

%% Gereral instructions:
instructions(scr,const,my_key,textExp.instruction1,button.instruction1);

%% Main Loop
expDone = 0;
newJ = 0;
startJ = 1;
while ~expDone
    for t = startJ:expDes.j
        trialDone = 0;
        while ~trialDone

            [resMat] = runSingleTrial(scr,const,expDes,my_key,t);
            if resMat == -1
                % Exit button  => send a new trial + save trial configuration for later
presentation
                trialDone = 1;
                newJ = newJ+1;
                expDes.expMatAdd(newJ,:) = expDes.expMat(t,:);
            else
                trialDone = 1;
                expResMat(t,:)= [expDes.expMat(t,:),resMat];
                csvwrite(const.expRes_fileCsv,expResMat);
            end
        end
    end
    %% If error of fixation of volontary missed trial
    if ~newJ
        expDone = 1;
    else
        startJ = expDes.j+1;
```

```
        expDes.j = expDes.j+newJ;
        expDes.expMat=[expDes.expMat;expDes.expMatAdd];
        expDes.expMatAdd = [];
        newJ = 0;
    end
end

const.my_clock_end = clock;
const_file = fopen(const.const_fileDat,'a+');
fprintf(const_file,'Ending time :\t%ih%i',const.my_clock_end(4),const.my_clock_end(5));
fclose('all');
instructions(scr,const,my_key,textExp.end,button.end);

end
```

Question 7: What is the role of "newJ"?

This variable counts the number of incorrect trials. In our case subjects are allowed to press space if they weren't ready for the trial. Such trial will later be replayed when all trials will be done.

Question8: How can we save our collected data?

You can decide to save your data in different formats, either in text using for example the Matlab built-in function fprintf(), in matrix Matlab format using save() or in comma separated value format using csvwrite().
As you can see, the main loop saves the data after each trial. Although you might think that it isn't necessary as you can save all at the end, I recommend saving and overwriting your file on each trial to avoid data loss after a possible program crash.

STIMULI LOOP

For each trial *runSingleTrial.m* will be played. This function draws the stimuli in function of the designed trial matrix "expDes.expMat" and of the trial meter "t".
It returns a variable "resMat" containing the trial response and reaction time.

Look for "runSingleTrial.m" in the Materials/Yeshurun98/Trials/ folder.

```
function [resMat]=runSingleTrial(scr,const,expDes,my_key,t)
% ------------------------------------------------------------------------
% [resMat] = runSingleTrial(scr,const,expDes,my_key,t)
% ------------------------------------------------------------------------
% Goal of the function :
% Main file of the experiment. Draw each sequence and return results.
% ------------------------------------------------------------------------
% Input(s) :
% scr : window pointer.
% const : struct containing all the constant configurations.
% expDes : struct containing all the variable design configurations.
% my_key : keyboard keys names.
% t : experiment meter.
% ------------------------------------------------------------------------
% Output(s):
% resMat(1) : experimental results
%          => = 1 : first interval
%          => = 2 : second interval
%          => = -1 : Voluntary brake
% resMat(2) : Reaction time
% ------------------------------------------------------------------------
% Function created by Martin SZINTE (martin.szinte@gmail.com)
% Last update : 08 / 01 / 2014
% Project : Yeshurun98
% Version : -
% ------------------------------------------------------------------------

while KbCheck; end
FlushEvents('KeyDown');

%% Compute and simplify var names :

ecc_target =    expDes.expMat(t,3);
```

```
ecc_random =    expDes.expMat(t,6);
tInterval =     expDes.expMat(t,4);
if tInterval == 1
    target_on1 = 1;
    target_on2 = 0;
elseif tInterval == 2
    target_on1 = 0;
    target_on2 = 1;
end

attCond = expDes.expMat(t,5);
if attCond == 1
    type = 1; % cued trials
    if tInterval == 1
        ecc_cue1 = ecc_target;
        ecc_cue2 = ecc_random;
    elseif tInterval == 2
        ecc_cue1 = ecc_random;
        ecc_cue2 = ecc_target;
    end
elseif attCond == 2
    ecc_cue1 = 1;
    ecc_cue2 = 2;
    type = 2; % neutral trials
end


%% Main loop

for tframes = 1:const.numFrm_Tot
    Screen('FillRect',scr.main,const.colBG);

    %% First interval
    % T1

    if tframes >= const.numFrm_T1_start && tframes <= const.numFrm_T1_end
        my_fixationCross(scr,const,const.black);
    end

    % T2
    if tframes >= const.numFrm_T2_start && tframes <= const.numFrm_T2_end
        my_cue(scr,const.black,const,type,ecc_cue1)
    end

    % T3
    if tframes >= const.numFrm_T3_start && tframes <= const.numFrm_T3_end
        % emtpty ISI
    end

    % T4
    if tframes >= const.numFrm_T4_start && tframes <= const.numFrm_T4_end
        if tframes == const.numFrm_T4_start
            const.randX_all = [];
            const.randY_all = [];
            for t_col = 1:const.num_col
                for t_raw = 1:const.num_raw
                    const.randX = randperm(5);
                    const.randY = randperm(5);
                    const.randX_all = [const.randX_all;const.randX];
                    const.randY_all = [const.randY_all;const.randY];
                end
            end
        end
        my_stim(scr,const,const.black,ecc_target,target_on1);
    end

    % T5
    if tframes >= const.numFrm_T5_start && tframes <= const.numFrm_T5_end
        my_mask(scr,const,const.black)
    end

    %% Second interval
    % T6
    if tframes >= const.numFrm_T6_start && tframes <= const.numFrm_T6_end
        my_fixationCross(scr,const,const.black);
    end

    % T7
    if tframes >= const.numFrm_T7_start && tframes <= const.numFrm_T7_end
```

```
        my_cue(scr,const.black,const,type,ecc_cue2)
    end

    % T8
    if tframes >= const.numFrm_T8_start && tframes <= const.numFrm_T8_end
        % ISI
    end

    % T9
    if tframes >= const.numFrm_T9_start && tframes <= const.numFrm_T9_end
        my_stim(scr,const,const.black,ecc_target,target_on2);
    end

    % T10

    if tframes >= const.numFrm_T10_start && tframes <= const.numFrm_T10_end
        my_mask(scr,const,const.black)
    end

    vbl = Screen('Flip',scr.main);

end
% Answer screen
[key_press,tRT]=getAnswer(scr,const,my_key);
tRT = tRT - vbl;

if key_press.rightShift == 1
    my_sound(1);
    resMat = [2,tRT];
elseif key_press.leftShift == 1
    my_sound(1);
    resMat = [1,tRT];
elseif key_press.space == 1
    my_sound(3);
    resMat = [-1,tRT];
elseif key_press.escape == 1
    overDone;
end

end
```

DATA COLLECTION

The *getAnswer.m* function puts loop the program in order to waits for the interaction with the subject. It codes the subject answer under a numeric value and collects the trial reaction time.

Look for "getAnswer.m" in the Materials/Yeshurun98/Trials/ folder.

```
function [key_press,tRt]=getAnswer(scr,const,my_key)
% ----------------------------------------------------------------------
% [key_press]=getAnswer(scr,const,expDes,t,my_key)
% ----------------------------------------------------------------------
% Goal of the function :
% Check keyboard press, and return flags.
% ----------------------------------------------------------------------
% Input(s) :
% scr : window pointer.
% const : struct containing all the constant configurations.
% my_key : keyboard keys names.
% tRt : machine time of button press
% ----------------------------------------------------------------------
% Output(s):
% key_press : struct containing key answer.
% ----------------------------------------------------------------------
% Function created by Martin SZINTE (martin.szinte@gmail.com)
% Last update : 08 / 01 / 2014
% Project : Yeshurun98
% Version : -
% ----------------------------------------------------------------------

key_press.leftShift = 0;
key_press.rightShift = 0;
key_press.push_button = 0;
key_press.escape = 0;
key_press.space = 0;
```

```
% Keyboard checking :
while ~key_press.push_button
    Screen('FillRect',scr.main,const.colBG);
    my_circle(scr,const.red,scr.x_mid,scr.y_mid,const.radCircle)
    my_circle(scr,const.colBG,scr.x_mid,scr.y_mid,9*const.radCircle/10)
    my_fixationCross(scr,const,const.black);

    Screen('Flip',scr.main);
    [keyIsDown, seconds, keyCode] = KbCheck;
    if keyIsDown
        if (keyCode(my_key.escape)) && ~const.expStart
            key_press.push_button = 1;
            key_press.escape = 1;
            tRt = seconds;
        elseif (keyCode(my_key.space))
            key_press.push_button = 1;
            key_press.space = 1;
            tRt = seconds;
        elseif (keyCode(my_key.rightShift))
            key_press.rightShift = 1;
            key_press.push_button = 1;
            tRt = seconds;
        elseif (keyCode(my_key.leftShift))
            key_press.leftShift = 1;
            key_press.push_button = 1;
            tRt = seconds;
        end
    end
end
end
```

4. Experimental end

When the main trial loop finishes the pre-defined number of trials, it launches *overDone.m*.
This very last function will close the PTB full-screen window as well as the settings text files. If you use some external apparatus, this function should be use to stop them and close/transfer any other external data files.

Look for “overDone.m” in the Materials/Yeshurun98/Main/ folder.

```
function overDone
% ----------------------------------------------------------------------
% overDone
% ----------------------------------------------------------------------
% Goal of the function :
% Close screen, listen keyboard and save duration of the experiment
% ----------------------------------------------------------------------
% Input(s) :
% none
% ----------------------------------------------------------------------
% Output(s):
% none
% ----------------------------------------------------------------------
% Function created by Martin SZINTE (martin.szinte@gmail.com)
% Last update : 08 / 01 / 2014
% Project : Yeshurun98
% Version : -
% ----------------------------------------------------------------------

ListenChar(1);
WaitSecs(2.0);

ShowCursor;
Screen('CloseAll');
timeDur=toc/60;
fprintf(1,'\nThis part of the experiment took : %2.0f min.\n\n',timeDur);

clear mex;
clear fun;

end
```

5. Conclusion

In this course we have learn how designing and programming an experiment using Matlab and the Psychtoolbox. I now hope that you will be able to use such template to create your own well-controlled experiment in very little time.