Programming – course 5
07/01/2014
Martin SZINTE

Using the Psychtoolbox (PART 2/2)

Plan

1. Installing de la Psychtoolbox (PTB)
2. What is the PTB?
3. The Screen function
   - Testing the PTB
   - Screen
   - Crash of the PTB
4. Help and demos of the PTB
   - Help section
   - PsychDemos
5. Essential functions of the PTB
   - FillRect / FrameRect
   - DrawLine / FillPoly
   - MakeTexture / DrawTexture
   - FillOval / FrameOval / DrawDots
   - DrawText
6. Take care of time and creating moving objects
   - Flip duration
   - Creating moving object
7. Use of the keyboard and mouse
   - Using the keyboard
   - Using the mouse

FILLOVAL/FRAMEOVAL/DRAWDOTS

These three sub-functions draw circle and oval on a full-screen window.
To use them follow such schemas:

```
Screen('FillOval', wPtr [,color] [,rect];
Screen('FrameOval', wPtr [,color] [,rect] [,penWidth] [,penHeight]);
Screen('DrawDots', wPtr, xy [,size] [,color] [,center] [,dot_type]);
```

FILLOVAL draws filled circles or ovals of a pre-defined size and color.
FRAMEOVAL draws famed circles or ovals with a pre-defined size, color and frame width.
DRAWDOTS draws very nice anti-aliased filled circles or ovals, however contrary to the two other drawing circle functions this function is very fast and allows the computation of thousand of them within the time of a screen refresh.

Question 7: How could we draw circles by specifying its center coordinates and radius instead of its RECT?

Most of the sub-functions of Screen use the LTBR coordinates system, however to draw a circle it is more intuitive to specify its center coordinates and its radius. To change the coordinate system of the PTB we will simply create our own circle drawing function "my_circle.m" that will automatically translate the center and radius into LTBR coordinates.

Look for "my_circle.m" in the material folder, add the function to your path and execute it through the command window.

```
function my_circle(scr,color,x,y,r,colorWidth,penWidth)
% --------------------------------------------------------------------
% my_circle(scr,color,x,y,r,[colorWidth],[penWidth])
% --------------------------------------------------------------------
% Goal of the function :
% Draw a colored (color) circle or oval in position (x,y) with radius
% (r) and with a colored contour (colorWidth) and width (penwidth).
% --------------------------------------------------------------------
% Input(s) :
% scr = window Pointer                                ex : w
% color = color of the circle in RBG or RGBA          ex :[0 0 0]
% x = position x of the center                        ex : x = 550
% y = position y of the center                        ex : y = 330
% r = radius for X (in pixel)                         ex : r = 25
% colorWidth = color of the contour of the circle     ex :[255 0 0]
% penWidth = size of the contour of the circle        ex : 10
% --------------------------------------------------------------------
% Output(s):
% none
% --------------------------------------------------------------------
% Function created by Martin SZINTE (martin.szinte@gmail.com)
% Last edit : 06 / 01 / 2014
% Project : Programming course
% Version : -
% --------------------------------------------------------------------

% if there isn't colorWidth argument the function will draw a filled circle
if nargin < 6
    % DrawDots makes nicer dots however it can only draw small dots (< 30pix)
    if r>30
        Screen('FillOval',scr,color,[(x-r) (y-r) (x+r) (y+r)]);
    else
        Screen('DrawDots', scr,[x,y],r*2,color,[],2)
    end

% if there is colorWidth argument the function will draw a framed circle
else
    Screen('FrameOval',scr,colorWidth,[(x-r) (y-r) (x+r) (y+r)],penWidth);
end

end
```

We can now test such function in a full-screen window of the PTB.

Look for “testCircle.m” in the material folder, add the function to your path and execute it through the command window.

=> testCircle(1)

```
function testCircle(type)
% --------------------------------------------------------------------
% testCircle(type)
% --------------------------------------------------------------------
% Goal of the function :
% Simple function that illustrates the FillOval and DrawDots functions.
% --------------------------------------------------------------------
% Input(s) :
% type : switch of the display
%                  if type == 1  :    filled circle
%                  if type == 2  :    frame circle
%                  else          :    both kind
% --------------------------------------------------------------------
% Output(s):
% (none)
% --------------------------------------------------------------------
% Function created by Martin SZINTE (martin.szinte@gmail.com)
% Last update : 06 / 12 / 2014
% Project : Programming courses
% Version : -
% --------------------------------------------------------------------

type;
scrAll = Screen('Screens');
scrNum = max(scrAll);
HideCursor;
[scr,scrRect] = Screen('OpenWindow',scrNum);
Screen('BlendFunction', scr, GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

priorityLevel = MaxPriority(scr);Priority(priorityLevel);

colBG = [255 255 255];              % background color
colCircle = [0 0 0];                % circles color
colWidth = [0 200 0];               % cicle frame color
penWidth = 10;                      % frame width

x1 = 200; y1 = 200;                 % first cicle center coordinates
x2 = 400; y2 = 300;                 % second cicle center coordinates
r1 = 15;                            % first cicle radius
r2 = 40;                            % second cicle radius

for timeFlip = 1:200

    Screen('FillRect',scr,colBG);

    if type == 1
        % draws filled circle
        my_circle(scr,colCircle,x1,y1,r1);
    elseif type == 2
        % draws framed circle
        my_circle(scr,colCircle,x2,y2,r2,colWidth,penWidth);
    else
        % draws both framed and filled circle
        my_circle(scr,colCircle,x1,y1,r1);
        my_circle(scr,colCircle,x2,y2,r2,colWidth,penWidth);
    end

    Screen(scr, 'Flip');

end

ShowCursor;
Screen('CloseAll');
```

DRAWTEXT

You will sooner or later need to display text for your experiment, either for your stimuli or simply to indicate to the subject the experiment instructions.
DRAWTEXT draws text on a full-screen window, however such Screen sub-function needs several others in order to draw formatted text with pre-defined size, font, color, position…

Look for "testText.m" in the material folder, add the function to your path and execute it through the command window.

=> testText;

```
function testText
% --------------------------------------------------------------------
% testCircle
% --------------------------------------------------------------------
% Goal of the function :
% Simple function that illustrates the DrawText sub-function.
% --------------------------------------------------------------------
% Input(s) :
% (none)
% --------------------------------------------------------------------
% Output(s):
% (none)
% --------------------------------------------------------------------
% Function created by Martin SZINTE (martin.szinte@gmail.com)
% Last update : 06 / 01 / 2014
% Project : Programming course
% Version : -
% --------------------------------------------------------------------

scrAll = Screen('Screens');
scrNum = max(scrAll);

HideCursor;
[scr,scrRect] = Screen('OpenWindow',scrNum);
Screen('BlendFunction', scr, GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
priorityLevel = MaxPriority(scr);Priority(priorityLevel);

colBG = [255 255 255];
textCol = [0 0 0];                                  % text color
my_font = 'Helvetica';                              % text font
text_size = 30;                                     % text font size
[scr_midX, scr_midY] = RectCenter(scrRect);         % screen center coordinates

% text to display
text_line1 = 'Two things are infinite: the universe and human stupidity.';
text_line2 = 'And I''m not sure about the universe.';
text_line3 = '';
text_line4 = '                                          Albert Einstein';

% create a cell with all text strings
text_all = {text_line1;text_line2;text_line3;text_line4};

Screen ('TextFont', scr, my_font);                  % defines text font
Screen('TextSize',scr, text_size);                  % defines text font size

lines = size(text_all,1);                           % number of lines of the text to
draw

bound = Screen('TextBounds',scr,text_all{1,:});     % determines left and right limit
of the first line
spaceL = ((text_size)*1.50);                        % defines space between lines
first_line = scr_midY - ((round(lines/2))*spaceL);  % defines position of the first
line to draw

for timeFlip = 1:1000

    Screen('FillRect', scr, colBG);
    addi = 0;

    % loop of text drawing
    for t_lines = 1:lines
```

```
        xText = scr_midX-bound(3)/2;
        yText = first_line+addi*spaceL;
        Screen('DrawText',scr,text_all{t_lines,:},xText,yText,textCol);
        addi = addi+1;
    end

    Screen('Flip',scr);

end

ShowCursor;
Screen('CloseAll');

end
```

The code above might look complex for such a simple outcome of formatted text. Drawing text in the PTB could indeed be summarized in 4 main points:

1. Define screen text font: Screen ('TextFont')
2. Define screen text size: Screen('TextSize')
3. Define text position: Screen('TextBounds')
4. Draw text on secondary screen: Screen('DrawText')

Question 8: What is the role of the variable "addi" contained in the text loop of testText.m?

This variable allows to count the number of lines already displayed and to then determine the y position of the following text line.

6. Control stimuli duration and creating moving objects

We just seen how displaying different kind of visual stimuli on a full-screen window using the PTB, however we generally selected randomly the duration during which each of these stimuli were displayed.
In order to now control stimuli duration we will first need to understand the way visual stimuli are physically displayed on a monitor.

FLIP DURATION

Since the beginning of these programming courses we used a main display loop made of a "for" looping for a random time a stimuli. Most of time however experiment involve relatively precise stimuli durations.
To specify the duration of a stimulus, we will have to determine the number of times that a code should be looped. In other term, we should determine how many time the secondary buffer screen should be flipped on the main monitor to get a specified duration, knowing the refresh rate of the monitor. All monitors got a specified refresh rate that corresponds to the number of time it could changes what it displayed during one second. For most of the flat modern screen, the refresh rate is of 60 Hz, meaning that they could change what is displayed 60 times per second. The consequence of such constraint is that the minimum duration of a stimulus on such monitors is of $1/60^{th}$ of a second, that is about 17 msec.
Therefore only durations that are multiple of 17 msec can be displayed (e.g. you can't have a 120 msec duration stimuli but at best a 117 msec or 133 msec, corresponding respectively to 7 or 9 flips).

Once you understand such physical constraint, you can use one of the two principal methods to control your stimuli durations:

1. A "for" loop counting screen flips (also called frame control)
2. A "*while*" loop using a clock (also called vbl control)

To illustrate these two procedures we will study the testTime.m function that in function of the input argument either use the first (1) or second (2) method.

Look for "testTime.m" in the material folder, add the function to your path and execute it through the command window.

=> testTime(1)

```
function testTime(type)
% --------------------------------------------------------------------
% testTime(type)
% --------------------------------------------------------------------
% Goal of the function :
% Simple function that illustrates the control of timing with the PTB
% using two types of programming versions.
% --------------------------------------------------------------------
% Input(s) :
% type : type of control of timing
%               if type = 1   :  frame control
%               if type = 2   :  vbl control
% --------------------------------------------------------------------
% Output(s):
% (none)
% --------------------------------------------------------------------
% Function created by Martin SZINTE (martin.szinte@gmail.com)
% Last update : 06 / 01 / 2014
% Project : Programming course
% Version : -
% --------------------------------------------------------------------

type;
scrAll = Screen('Screens');
scrNum = max(scrAll);
HideCursor;
[scr,scrRect] = Screen('OpenWindow',scrNum);
Screen('BlendFunction', scr, GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
priorityLevel = MaxPriority(scr);Priority(priorityLevel);

% sub-function determining the frame duration
frame_duration = Screen('GetFlipInterval', scr);

colBG = [255 255 255];
colRect = [0 0 0];
rectRect = [200, 200, 300, 300];
rectRect2 = [200, 400, 700, 500];
rectWidth = 10;

t1_sec = 3;                             % duration of the first stinulus in seconds
t2_sec = 6;                             % duration of the second stimulus in seconds

numFrameT1 = t1_sec/(frame_duration);   % number of frames/flips of the first stimulus
numFrameT2 = t2_sec/(frame_duration);   % number of frames/flips of the second
stimulus


% frame control of stimuli duration
if type == 1
    % T1: first time
    t0 = GetSecs;                       % Get current machine time to verify accuracy
of duration
    for timeFlipT1 = 1:numFrameT1
        Screen('FillRect',scr,colBG);
        Screen('FillRect',scr,colRect,rectRect);
        t1 = Screen(scr, 'Flip');
    end
    fprintf(1,'\n\tThe first stimulus duration: %1.3f sec',(t1-t0));
```

```
    fprintf(1,'\n\tError with desired duration: %1.3f msec',abs(t1_sec-(t1-t0))*1000);

    % T2
    for timeFlipT2 = 1:numFrameT2
        Screen('FillRect',scr,colBG);
        Screen('FrameRect',scr,colRect,rectRect2,rectWidth);
        t2 = Screen(scr, 'Flip');
    end
    fprintf(1,'\n\tThe second stimulus duration: %1.3f sec',(t2-t1));
    fprintf(1,'\n\tError with desired duration:  %1.3f msec\n',abs(t2_sec-(t2-
t1))*1000);


% vbl control of stimuli duration
elseif type == 2

    t0 = GetSecs;
    t1 = 0;
    while t1 < (t1_sec - frame_duration)
        Screen('FillRect',scr,colBG);
        Screen('FillRect',scr,colRect,rectRect);
        vbl1 = Screen(scr, 'Flip');
        t1 = vbl1-t0;
    end
    fprintf(1,'\n\tThe first stimulus duration: %1.3f sec',(vbl1-t0));
    fprintf(1,'\n\tError with desired duration: %1.3f msec',abs(t1_sec-(vbl1-
t0))*1000);

    t2 = 0;
    while t2 < (t2_sec- frame_duration)
        Screen('FillRect',scr,colBG);
        Screen('FrameRect',scr,colRect,rectRect2,rectWidth);
        vbl2 = Screen(scr, 'Flip');
        t2 = vbl2 - vbl1;
    end
    fprintf(1,'\n\tThe second stimulus duration: %1.3f sec',(vbl2-vbl1));
    fprintf(1,'\n\tError with desired duration: %1.3f msec\n',abs(t2_sec-(vbl2-
vbl1))*1000);

end

ShowCursor;
Screen('CloseAll');

end
```

Question 9: What is the best procedure to control stimuli duration?

For both procedures you should experience more or less big errors between the desired durations and the actual durations of the stimuli on the monitor.
This basically means that no procedure is perfect, however the accuracy is highly correlated with the monitor quality. When both procedures are equally inaccurate on flat monitors, frame control gives very accuracy timing on CRT monitors.

CREATING MOTION

To program a moving stimulus, we should first determine its speed, that is the distance it makes for a given duration.
As we now know how to control stimulus display duration, we will only need to process the distance to make for a specified duration. To illustrate such point, we will study the rotation of a circle and a bar in testClock.m.

Look for “testClock.m” and clock.jpeg in the material folder, add the function to your path and execute it through the command window.

=> testClock (60,2);

```
function testClock(t,numRep)
% ----------------------------------------------------------------------
% testClock(t,numRep)
% ----------------------------------------------------------------------
% Goal of the function :
% Display image a clock with a second hand turning in x seconde.
% ----------------------------------------------------------------------
% Input(s) :
% t : time to make a entire revolution in seconds
% numRep : number of revolutions
% ----------------------------------------------------------------------
% Output(s):
% (none)
% ----------------------------------------------------------------------
% Function created by Martin SZINTE (martin.szinte@gmail.com)
% Last update : 06 / 01 / 2014
% Project : Programming courses
% Version : -
% ----------------------------------------------------------------------

t;numRep;
scrAll = Screen('Screens');
scrNum = max(scrAll);
[scr,scrRect] = Screen('OpenWindow',scrNum);
Screen('BlendFunction', scr, GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
[scrCtrX, scrCtrY] = RectCenter(scrRect);
priorityLevel = MaxPriority(scr);Priority(priorityLevel);
frame_duration = Screen('GetFlipInterval', scr);  % get frame duration
HideCursor;

% Colors
white = [255 255 255];
colBG = white;
black = [0 0 0];

% Clock picture
pict = imread('clock.jpg');
t_handle = Screen('MakeTexture',scr,pict);

% Clock hand settings
width_line = 5;                         % width of the clock hand
length_line = 320;                      % lenght of the clock hand
col_line = [0,0,200];
distCircle = 130;
radCircle = 22;

matLine = [ones(length_line/2,width_line);...   % determines matrix of the clock hand
           zeros(length_line/2,width_line)];

matLine(length_line/2-distCircle-radCircle+1:length_line/2-distCircle+radCircle-1,:) =
0;

% determines each color and transparency matrix
matLineCol(:,:,1) = matLine.*col_line(1);
matLineCol(:,:,2) = matLine.*col_line(2);
matLineCol(:,:,3) = matLine.*col_line(3);
matLineCol(:,:,4) = matLine.*255;

hand_h = Screen('MakeTexture',scr,matLineCol);        % determines texture handle of
the clock hand


nFrameRot = round(t/frame_duration);    % number of frame per revolution
angle = 0:360/nFrameRot:360;            % rotation angle per frame

for trep = 1:numRep
    for timeFlip = 1:nFrameRot

        timeF = 0:nFrameRot-1;
        xCircle = scrCtrX + distCircle * cos((2*pi/nFrameRot)*timeF(timeFlip)-pi/2);
        yCircle = scrCtrY + distCircle * (-1) * -sin((2*pi/nFrameRot)*timeF(timeFlip)-
pi/2);

        Screen('FillRect',scr,colBG);
        Screen('DrawTexture',scr,t_handle);

        Screen('DrawTexture',scr,hand_h,[],[],angle(timeFlip));
        my_circle(scr,[],xCircle,yCircle,radCircle,col_line,width_line)
```

```
        my_circle(scr,col_line,scrCtrX,scrCtrY,radCircle/2)
        Screen('Flip',scr);

    end
end

ShowCursor;
Screen('CloseAll');

end
```

testTime() is a revision of the different notions we have seen throughout these last two courses. It involves the use Screen sub-function such as *MakeTexture* and *DrawTexture* to display the clock picture and a matrix of the clock hand (using transparency). It also calls previous functions such as *my_circle.m* to draw circle of different size, color and types.
Finally it involves the processing of rotational motion (i.e. defining duration and distance) as well as a very useful way of coding symmetrical stimuli (using trigonometry).

Question 10: What are the roles of the Matlab built-in functions "zeros" and "ones" used in testClock.m?

These built-in functions allow the creation of matrices composed either of zeros or ones. You can define the size of such matrices using these function input arguments (lines, column…).
In *testClock.m* these functions are used to determine the shape of the clock hand.

Remark:
testClock.m is a complex function illustrating different possibilities of the PTB, however the rationale behind moving objects is very simple. Moving objects are nothing else than objects for which coordinates change across the different flips of a monitor.

7. Use of the keyboard and mouse

In most of your experiments you will need a way to interact with your subject.
For such interaction you may use different complex devices such as a joystick, remotes, eye-trackers, or reaction time boxes, but most of the time you will simply use generic devices that are the keyboard and mouse.
To use such devices the PTB has several simple functions that we will discover in the following section.

USING THE KEYBOARD

It exists several built-in Matlab and PTB toolbox functions to collect data via a keyboard.
All these functions have some positive and negative aspects.
Choose carefully functions adapted to your needs using the following table:

Functions	Full-screen PTB?	Timing	Which button was pressed?	Wait for input?	Short description
Pause ()	No	No	No	Yes	Wait a pre-defined amount of time or a button press.
Input()	No	No	Yes	Yes	Wait a return and transmit it values
CharAvail()	Yes	No	No	No	Check that a letter keyboard button was pressed
GetChar()	Yes	Yes (un-precise)	Yes	Yes	Check for a text button press and return its value.
KbCheck()	Yes	Yes (precise)	Yes	No	Check that a button (any) is pressed and return its value as well as the moment it was pressed.
KbWait()	Yes	Yes (precise)	No	Yes	Wait for any button to be pressed.

Therefore, when we will need to collect information on the subject or to temporarily stop a program in Matlab (not during a full-screen PTB), we will prefer PAUSE and INPUT.

PAUSE

```
for counter = 1:30
   fprintf(1,'\tThe counter has a value of: %2.0f\n',counter);
   pause;
end
```

Question 11: What happens if we replace "pause" by "pause(1)"?
The input argument of the PAUSE function specifies the time during which we will break a code. This break will for example allow us to briefly see the results of a graph before printing the following one.

INPUT

This function waits for the input of an operator (the subject) and for the press of the return button. The entered input could either be a chain of character or a number.

Look for "magic.m" in the material folder, add this script to your path and execute it through the command window.

=> magic;

```
home;
val = input('Choose a number between 1 and 9:    ');
fprintf('\nMultiply it by 9');pause
fprintf('\nThis will give you: %i',val*9);pause

fprintf('\n\nNow subtract 5');pause
fprintf('\nThis will give: %i',val*9-5);pause

fprintf('\n\nIf you get a number with 2 digits, add them to each other until you
obtain a single digit');pause

fprintf('\n\nKnowing that 1 = A, 2 = B, 3 = C, 4 = D, ...')
fprintf('\nTake the letter corresponding to the result of your calculation')

fprintf('\nand think of a country name starting by such letter.');pause

fprintf('\n\nTake the last letter of the country name and think of a fruit name
starting by such letter. \n\n');pause

res = input('Were you thinking to a KIWI ? (Y/N)    ','s');
if strcmp(res,'Y')
    fprintf('The country you thought of was the Danemark, wasn''t it ?\n')
end
```

This example shows the different input arguments of the INPUT Matlab function.
The values entered by the operator are returned as output arguments of the INPUT function. You can then use such output to create interactivity between your codes and the operator.

GETCHAR / CHARAVAIL / KBCHECK

The functions PAUSE and INPUT can't be used during a full-screen window of the PTB.
We will then use the PTB functions GETCHAR, CHARAVAIL and KBCHECK.
To use them follow such schemas:

```
[ch, when] = GetChar([getExtendedData], [getRawCode])
[avail, numChars] = CharAvail
[keyIsDown, secs, keyCode, deltaSecs] = KbCheck([deviceNumber])
```

To illustrate these PTB functions we will study in *testButton.m* the use of keyboard button to modify text displayed on a full-screen window.
This function use both CHARAVAIL and GETCHAR in order to collect letter keyboard button and also KBCHECK to register keyboard button that doesn't return text when pressed.

Look for “testButton.m” in the material folder, add the function to your path and execute it through the command window.

=> testButton;

```
function testButton
% ---------------------------------------------------------------------
% testButton
% ---------------------------------------------------------------------
% Goal of the function :
% Function illustrating the use of KbCheck, GetChar and CharAvail
% ---------------------------------------------------------------------
% Input(s) :
% (none)
% ---------------------------------------------------------------------
% Output(s):
% (none)
% ---------------------------------------------------------------------
% Function created by Martin SZINTE (martin.szinte@gmail.com)
% Last update : 07 / 01 / 2014
% Project : Programming course
% Version : -
% ---------------------------------------------------------------------

clear all;
home;

% Keyboard settings
FlushEvents('KeyDown');             % clears any previous button press
KbName('UnifyKeyNames');            % gives similar values to each key for any OS
my_key.space = KbName('Space');     % defines the space button number
my_key.escape = KbName('escape');   % defines the escape button number
press_quit_button = 0;              % pre-defines quit button value
press_space = 0;                    % pre-defines space button value


% Screen settintgs
scrAll = Screen('Screens');
scrNum = max(scrAll);
HideCursor;
[scr.main,scrRect] = Screen('OpenWindow',scrNum);
Screen('BlendFunction', scr.main, GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
priorityLevel = MaxPriority(scr.main);Priority(priorityLevel);
[scr.x_mid, scr.y_mid] = RectCenter(scrRect);


% Text settings
colBG       = [255 255 255];
textCol     = [0 0 0];
my_font     = 'Calibri';
text_size   = 20;
text_line1  = 'First Name : ';
text_line2  = 'Last Name : ';
text_line3  = 'Gender (male/female) :';
text_line4  = 'Age : ';
text_line5  = '';
text_line6  = '>> Press space for next line and escape to quit<<';
text_all    = {text_line1;text_line2;text_line3;text_line4;text_line5;text_line6};
lines = size(text_all,1);
spaceL = ((text_size)*1.50);
first_line = scr.y_mid - ((round(lines/2))*spaceL);
Screen('TextSize', scr.main, text_size);
Screen ('TextFont', scr.main, my_font);

str_all = [];
t_ini   = GetSecs;
ListenChar(2);                  % stop printing text button

while ~press_quit_button

    Screen('FillRect', scr.main, colBG);

    addi = 0;
    for t_lines = 1:lines
        xTxt = 200;
        yTxt = first_line+addi*spaceL;
```

```
        Screen('DrawText',scr.main,text_all{t_lines,:},xTxt,yTxt,textCol);
        addi = addi+1;
    end

    % loop of text button press
    if CharAvail                % text checking loop
        str = GetChar;          % gets the latest character pressed
        str_all = [str_all,str];
    end

    % text modification
    if press_space == 0
        text_line1 = sprintf('First Name : %s',str_all);
        text_all =
{text_line1;text_line2;text_line3;text_line4;text_line5;text_line6};
    elseif press_space == 1
        text_line2 = sprintf('Last Name : %s',str_all);
        text_all =
{text_line1;text_line2;text_line3;text_line4;text_line5;text_line6};
    elseif press_space == 2
        text_line3 = sprintf('Gender (male/female) : %s',str_all);
        text_all =
{text_line1;text_line2;text_line3;text_line4;text_line5;text_line6};
    elseif press_space == 3
        text_line4 = sprintf('Age : %s',str_all);
        text_all =
{text_line1;text_line2;text_line3;text_line4;text_line5;text_line6};
    end

    [keyIsDown, secs, keyCode] = KbCheck;   % checks all button press
    % loop of action button press
    if keyIsDown
        if (keyCode(my_key.space))
            press_space = press_space + 1;
            str_all = [];
            while KbCheck;end
        elseif (keyCode(my_key.escape))
            press_quit_button = 1;
            t_end = secs;                   % gets time of the button press
            while KbCheck;end
        end
    end
    Screen('Flip',scr.main);
end

ShowCursor;
Screen('CloseAll');
ListenChar(1);                              % starts to print text keyboard button
% display full-screen window duration
fprintf(1,'\nYou took %3.0f sec to fill the form\n',t_end-t_ini);

end
```

Advice:
The sub-function ListenChar() allow you to avoid modifications in your Matlab code when an operator use button press during a full-screen window. Indeed when you launch a full-screen window via the PTB, the text cursor might stay inside your codes. In such case any button press of the opeartor will modify your code. You can avoid such situation by putting ListenChar(2) at the beginning of your codes and ListenChar(1) at the very end.

This very useful function however could lead to problems if your code crashes before executing the ListenChar(2) placed at its very end. In such case you will not be able to write any new code in your files or in the command window until you press CTRL+C.

USING THE MOUSE

SETMOUSE/GETMOUSE are two PTB functions allowing you to use the mouse in a full-screen window. To use them follow such schemas:

```
SetMouse(x,y,[windowPtrOrScreenNumber])
[x,y,buttons,focus] = GetMouse([windowPtrOrScreenNumber])
```

To illustrate such functions we will study *testMouse.m* where the mouse coordinates are used to modify the coordinates of a stimulus.

Look for "testMouse.m" in the material folder, add this function to your path and execute it through the command window.

=> testMouse;

```
function testMouse
% --------------------------------------------------------------------
% testMouse
% --------------------------------------------------------------------
% Goal of the function :
% Simple script illustrating the use of the mouse in the PTB.
% --------------------------------------------------------------------
% Input(s) :
% (none)
% --------------------------------------------------------------------
% Output(s):
% (none)
% --------------------------------------------------------------------
% Function created by Martin SZINTE (martin.szinte@gmail.com)
% Last update : 07 / 01 / 2014
% Project : Programming courses
% Version : -
% --------------------------------------------------------------------
ListenChar(2);
scrAll = Screen('Screens');
scrNum = max(scrAll);

% Screen settings
HideCursor;
[scr.main,scr.rectV] = Screen('OpenWindow',scrNum);
[scr.x_mid, scr.y_mid] = RectCenter(scr.rectV);
priorityLevel = MaxPriority(scr.main);Priority(priorityLevel);

% Stimulus settings
colBG = [255 255 255];
sqr_sizeX = 100;
sqr_sizeY = 100;
sqr_ctrX = scr.x_mid;
sqr_ctrY = scr.y_mid;
colSqr = [0 0 0];

% mouse settings
SetMouse(scr.x_mid,scr.y_mid)   % puts the mouse in screen center
press_left_mouse = 0;           % defines mouse left button value

while ~press_left_mouse
    Screen('FillRect',scr.main,colBG);

    [xMouseNew, yMouseNew, buttonMouse] = GetMouse();

    % get new mouse coordinates
    diffMouseX = (xMouseNew - sqr_ctrX);                        % determines x
difference between initial position and current one
    diffMouseY = (yMouseNew - sqr_ctrY);                        % determines y
difference between initial position and current one

    sqr_ctrX = sqr_ctrX + diffMouseX;                           % determines new x
```

```
mouse coordinate
    sqr_ctrY = sqr_ctrY + diffMouseY;                                % determines new y
mouse coordinate

    rectRect = [sqr_ctrX - sqr_sizeX/2,...                           % Rect coordinates of
the square
                sqr_ctrY - sqr_sizeY/2,...
                sqr_ctrX + sqr_sizeX/2,...
                sqr_ctrY + sqr_sizeY/2];

    Screen('FillRect',scr.main,colSqr,rectRect);

    Screen(scr.main, 'Flip');

    % mouse button loop check
    if sum(buttonMouse)~= 0
        if buttonMouse(1) == 1
            press_left_mouse = 1;
        end
    end
end

ListenChar(1);
ShowCursor;
Screen('CloseAll');

end
```