Experimental design and analysis

Experiment programming

https://www.lri.fr/~appert/eval/
Project: hypotheses to test

Pick two visual variables of your choice (e.g., color, size, shape, shadow, etc.). Let's call them $\text{VV}_1$ and $\text{VV}_2$.

Research hypotheses:

- $H_1$: $\text{VV}_1$ is preattentive
- $H_2$: $\text{VV}_2$ is preattentive
- $H_3$: $\text{VV}_1$ and $\text{VV}_2$ combined are less preattentive than $\text{VV}_1$ or $\text{VV}_2$ in isolation

In this correction, I use $\text{VV}_1 = \text{Size}$ and $\text{VV}_2 = \text{Color}$
Operationalization

We refine what *preattentive* means:

A visual variable is preattentive when the visual search time for the only object that differs from a collection because of this visual variable is not affected by the number of objects in the collection.
Operationalization

Visual search time

H₁: VV₁ is preattentive
H₂: VV₂ is preattentive

H₁ and H₂ would be supported by charts like that

(*) Number of objects in the collection
Operationalization

Visual search time

Measure

Low    Medium    Large    Object count (*)

(*) Number of objects in the collection
Operationalization

H₃: VV1 and VV2 combined are less preattentive than VV1 or VV2 in isolation

(It takes more time to spot a difference when the difference is along two visual variables than when it is along a single variable)
Operationalization

Visual search time

- $\text{VV1}$
- $\text{VV2}$
- $\text{VV1} \text{VV2}$

Difference in visual variables

Factor
Operationalization

Factors:
OC: ObjectCount
   \{Low, Medium, Large\}
VV: Difference in Visual Variables
   \{VV1, VV2, VV1VV2\}

Measure:
Visual Search Time

Task?
Operationalization - task

Stimulus: present a collection of objects where only one object is different from all the other objects.

Example with VV1=Size and VV2=Color
Operationalization - task

Response: the participant finds the different object

How to measure visual search time? Issues discussed last week:

Only visual search: stop timer at pointing (no!), stop timer at key press (yes!)

Make sure the participant spot the right object: 2-step task with first key press then click on placeholders

Avoid an "animation effect" by using placeholders that are clearly different from all objects in the collection
Experiment storyboard

Multiple shapes will get displayed. Only one shape is different from all other shapes.
1. Spot it as fast as possible and press Space bar;
2. Click on the placeholder over that shape.
Press enter key when ready to start.

Multiple shapes will get displayed. Only one shape is different from all other shapes.
1. Spot it as fast as possible and press Space bar;
2. Click on the placeholder over that shape.
Press enter key when ready to start.

...
Make the design formal with TouchStone (20')

Two constraints:

You need at least 30 measures per condition overall to run inferential statistics

You have access to 6 participants
Experiment programming
Experiment programming I/O

experiment design (TouchStone csv output)

Participant, Practice, Block, Trial, VV, OC

<table>
<thead>
<tr>
<th>Participant, Practice, Block, Trial, VV, OC</th>
<th>VisualSearchTime, ErrorCount</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, true, 0, 0, Size, Large</td>
<td>1632, 0</td>
</tr>
<tr>
<td>0, true, 0, 1, Size, Medium</td>
<td>1552, 1</td>
</tr>
<tr>
<td>0, true, 0, 2, Size, Small</td>
<td>2030, 0</td>
</tr>
<tr>
<td>0, false, 1, 0, Size, Large</td>
<td>1582, 0</td>
</tr>
<tr>
<td>0, false, 1, 1, Size, Large</td>
<td>1639, 0</td>
</tr>
</tbody>
</table>

experiment_touchstone1.csv

experiment program

log file (csv file for your statistical analyses)
Getting started

Download JavaScript code skeleton on class website

Class material

- Class 1: Introduction to Experimental Design (Slides)
- Class 2: Hands-on approach to Experimental Design (Slides)
  - Touchstone 1: Download
  - Touchstone 2: online tool, and introduction video (excerpt from the conference talk record)
- Class 3: Experiment Programming (Slides)
  - Project start code: Download
Getting started

experiment_touchstone1.csv and experiment_touchstone2.csv are just excerpts from TouchStone1 and TouchStone2 design files. You have to replace one or the other with your design.
At page loading time, function `createScene` is called. `createScene` then calls `loadData`. Two main files: `experiment.html` and `experiment.js`. We will modify `experiment.js` only.
**loadData** needs to access the CSV design file output by TouchStone, which is a local file on your system.

Use a (local) Web server to serve local files with HTTP

Launch an HTTP server in the TD's directory

```
> cd experiment_js/
> python -m http.server 8888  \(\text{python 3}\) or  \> python -m SimpleHTTPServer 8888  \(\text{python 2}\)
```

Access the page from your browser

Open Inspector and Console

Check cache is disabled
The program is designed to work with TouchStone 1 or 2

- Set the TouchStone version that you have used for the design in the first line of experiment.js

```
var touchstone = 1;
```

- `loadData` function will then either read `experiment_touchstone1.csv` or `experiment_touchstone2.csv`

```
var touchstone = 2;
var state = {
  NONE: 0,
  INSTRUCTIONS: 1,
}
```

- `d3.csv` creates an array for a csv table. The array contains one array per line. Values in these arrays can be accessed using the column header name.

For example, `ctx.trials[3]['VV']` returns the value in column "VV" of the fourth line in the csv file (⚠ as a string).
Clicking button GO calls function `startExperiment`

```javascript
var startExperiment = function(event) {
    ...
    // start first trial
    console.log("start experiment at "+ctx.cpt);
    nextTrial();
}
```

`ctx.cpt` is now the index just before the first trial to run in the trial table `ctx.trials`.

`nextTrial` function is called.
nextTrial calls displayInstructions

Multiple shapes will get displayed. Only one shape is different from all other shapes.

1. Spot it as fast as possible and press space bar;
2. Click on the placeholder over that shape.

Press Enter key when ready to start.
Keyboard events

Function `keyListener` gets called when a key is pressed

```javascript
var keyListener = function(event) {
  event.preventDefault();

  if(ctx.state == state.INSTRUCTIONS && event.code == "Enter") {
    d3.select("#instructions").remove();
    displayShapes();
  }

  // TODO makes the experiment progress when user press Space bar
}
```

When **Enter** key is pressed while instructions are displayed, we remove instructions and display the scene of shapes instead (function `displayShapes`)
Keyboard events

Function `keyListener` gets called when a key is pressed

```javascript
var keyListener = function(event) {
  event.preventDefault();

  if(ctx.state == state.INSTRUCTIONS && event.code == "Enter") {
    d3.select("#instructions").remove();
    displayShapes();
  }
}
```

When `Enter` key is pressed while instructions are displayed, we remove instructions and display the scene of shapes instead (function `displayShapes`)

We use library d3 to make DOM manipulations easy. For example,
```
d3.select("#instructions").remove();
```
removes element whose id is from the DOM
Display grid of shapes

Function `displayShapes` does the job

```javascript
var displayShapes = function() {
  ctx.state = state.SHAPES;
  var visualVariable = ctx.trials[ctx.cpt]["VV"]; 
  var oc = ctx.trials[ctx.cpt]["OC"]; 
  if(oc === "Small") {
    objectCount = 9;
  } else if(oc === "Medium") {
    objectCount = 25;
  } else {
    objectCount = 49;
  }
  console.log("display shapes for condition "+oc+","+visualVariable);
  var svgElement = d3.select("svg"); 
  var group = svgElement.append("g") .attr("id", "shapes") .attr("transform", "translate(100,100)"");
};
```

Access factor values for this trial

We will add all shapes to a group whose id is `shapes` so that we can remove all shapes later on with the single following line of code:

```javascript
d3.select("#shapes").remove();
```
Update function `keyListener` to remove shapes and display placeholders instead when participant presses `Space` bar in state `state.SHAPES`.

Function `displayPlaceholders` is provided.
Update function `displayPlaceholders` to remove placeholders and progress to next trial (call `nextTrial`) when participant clicks a placeholder. Ignore errors for now, progress to next trial in all cases.

```javascript
var displayPlaceholders = function() {
    ...
    placeholder.on("click",
        function() {
            // TODO
        });
};
```

Code called when a click occurs on a placeholder
Scene of objects

Let's take a closer look at function displayShapes

```javascript
var displayShapes = function() {
...

// 1. Decide on the visual appearance of the target
// In my example, it means deciding on its size (large or small) and its color (light or dark)
var randomNumber1 = Math.random();
var randomNumber2 = Math.random();
var targetSize, targetColor;
if(randomNumber1 > 0.5) {
    targetSize = 25; // target is large
} else {
    targetSize = 15; // target is small
}
if(randomNumber2 > 0.5) {
    targetColor = "DarkGray"; // target is dark gray
} else {
    targetColor = "LightGray"; // target is light gray
}
```

In order to avoid participants look for a specific object as opposed to look for the different object (i.e., threat to internal validity), we introduce some variation on the target appearance by setting it randomly to one of the four possible object appearances:

We decide on the target's appearance
Scene of objects

Let's take a closer look at function `displayShapes`

```javascript
var displayShapes = function() {
  ...

  // 2. Set the visual appearance of all other objects now that the target appearance is decided
  // Here, we implement the case VV = "Size" so all other objects are large (resp. small)
  // if target is small (resp. large) but have the same color as target.
  var objectsAppearance = [];
  for (var i = 0; i < objectCount-1; i++) {
    if(targetSize == 25) {
      objectsAppearance.push({
        size: 15,
        color: targetColor
      });
    } else {
      objectsAppearance.push({
        size: 25,
        color: targetColor
      });
    }
  }
}
```

We generate the list of other objects depending on the target's appearance.
Scene of objects

Let's take a closer look at function `displayShapes`

```javascript
var displayShapes = function() {
    ...

    // 3. Shuffle the list of objects (useful when there are variations regarding both visual variable) and add the target at a specific index
    shuffle(objectsAppearance);
    // draw a random index for the target
    ctx.targetIndex = Math.floor(Math.random()*objectCount);
    // and insert it at this specific index
    objectsAppearance.splice(ctx.targetIndex, 0, {size:targetSize, color:targetColor});

    We shuffle (*) the list of other objects and then insert the target at a specific index

    (*) explanation for shuffling later on
```
Scene of objects

Let's take a closer look at function `displayShapes`

```javascript
var displayShapes = function() {
    ...

    // 4. We create actual SVG shapes and lay them out as a grid
    // compute coordinates for laying out objects as a grid
    var gridCoords = gridCoordinates(objectCount, 60);
    // display all objects by adding actual SVG shapes
    for (var i = 0; i < objectCount; i++) {
        group.append("circle")
            .attr("cx", gridCoords[i].x)
            .attr("cy", gridCoords[i].y)
            .attr("r", objectsAppearance[i].size)
            .attr("fill", objectsAppearance[i].color);
    }
}
```

We actually display shapes as a SVG shapes laid out as a grid.
We use d3 library to manipulate the DOM (add elements and set their attributes' values)
TODO step 2-a

For now, function `displayShapes` ignores the actual value of $VV$ and simply implements the case $VV = "Size"

Adapt the code to your visual variable $VV_1$ (i.e., handle your case $VV = VV_1$)
SVG and visual variables

I used circles with Size and Color visual variables, but SVG provides you with different types of graphical shape and various graphical attributes.

- SVG code can be included directly in HTML documents
- Shapes: `rect`, `circle`, `ellipse`, `line`, `text`, `path`
- Styling: `fill`, `stroke`, `stroke-width`, `opacity`, `font-family`, `font-size`
- or use CSS rules
- Transparency can be controlled with `opacity` or `rgba(r,g,b,a)` color tuples

```
<svg width="100" height="100">
  <circle cx="50" cy="50" r="22" fill="blue" stroke="gray" stroke-width="4"/>
</svg>
```
SVG - Scalable Vector Graphics

(source: Emmanuel Pietriga)

<svg width="400" height="260">
  <!-- blue circle with a 5px-gray border-->
  <circle cx="50" cy="50" r="40" fill="blue" stroke="gray" stroke-width="5" />
  <!-- ellipse with a 4px redish border and no fill color-->
  <ellipse cx="100" cy="70" rx="30" ry="20" fill="none" stroke="#FF2244" stroke-width="4" />
  <!-- two rectangles partially overlapping, the one above (which us red) is semi-transparent-->
  <rect x="200" y="20" width="150" height="50" fill="#F00" />
  <rect x="220" y="30" width="150" height="50" fill="#F00" opacity=".5" />
  <!-- simple black line -->
  <line x1="50" y1="120" x2="150" y2="220" stroke="black" />
  <!-- simple text element -->
  <text x="200" y="180">Sample text</text>
  <!-- a quadratic bézier curve -->
  <path fill="none" stroke="#999" stroke-width="3" d="M10,250 Q380,250 380,120" />
</svg>
SVG - Scalable Vector Graphics

Affine Transforms

```xml
<rect x="10" y="10" width="100" height="50" style="fill:red"/>
<rect x="0" y="0" width="100" height="50" style="fill:green"
     transform="translate(10,80)" />
<rect x="0" y="0" width="100" height="50" style="fill:blue"
     transform="translate(10,150) scale(1.5) rotate(45 180 150)"/>
```
Many more possibilities, including, e.g., filters:

```xml
<svg width="200" height="150">
  <defs>
    <filter id="ds" x="0" y="0" width="200%" height="200%">
      <feOffset result="offOut" in="SourceAlpha" dx="20" dy="20"/>
      <feGaussianBlur result="blurOut" in="offOut" stdDeviation="10"/>
      <feBlend in="SourceGraphic" in2="blurOut" mode="normal"/>
    </filter>
  </defs>
  <rect x="10" y="10" width="90" height="90" fill="yellow" stroke="#333" filter="url(#ds)"/>
</svg>
```

Detailed SVG documentation:

Manipulating the DOM of a web page with d3

```javascript
var svgElement = d3.select("svg");
var group = svgElement.append("g")
  .attr("id", "shapes")
  .attr("transform", "translate(100,100)");

group.append("circle")
  .attr("cx", 50)
  .attr("cy", 50)
  .attr("r", 20)
  .attr("fill", "red");
```

Select first `svg` element and insert a `g` element as a child whose attr `id` is `shapes`
TODO step 2-b

Complement the code to make it work for your second visual variable (i.e., handle case $VV=V_{V2}$)
TODO step 2-c

Complement the code to make it work for the combination of your two visual variables (i.e., handle case $VV = VV_1 VV_2$)

This is where shuffling other objects in `displayShapes` makes sense

*example for $VV =$"ColorSize" with a target*

Other objects (generated by series of objects that have the three possible appearances)

Shuffle other objects

insert target at a specific index

Target
TODO step 3

Log measures by adding an array with values each time a trial ends at the end of the
\texttt{ctx.loggedTrials} array which contains one line for each trial that has been run until now.

\begin{verbatim}
ctx.loggedTrials = [
    ["DesignName","ParticipantID","TrialID","Block","Trial","VV","OC","visualSearchTime","ErrorCount"]];
...
ctx.loggedTrials.push(
    ["Preattention-experiment",1,1,1,1,"Size","Medium",1582,0]
)
\end{verbatim}
TODO step 3

Pay attention to column headers depending on the TouchStone version that you have used for your design

```javascript
ctx {
  ...
  loggedTrials:
    touchstone == 1 ?
    [["Participant","Practice","Block","Trial","VV","OC","visualSearchTime","ErrorCount"]]:
    [["DesignName","ParticipantID","TrialID","Block1","Trial","VV","OC","visualSearchTime","ErrorCount"]]
```

**TouchStone 1**

Participant,Practice,Block,Trial,VV,OC,visualSearchTime,ErrorCount

**TouchStone 2**

DesignName,ParticipantID,TrialID,Block,Trial,VV,OC,visualSearchTime,ErrorCount
TODO step 3

Button download log file calls function `downloadLogs` which turns `ctx.loggedTrials` into a csv that you can download.
TODO step 3-a

a) Log measure `visualSearchTime`, the function `Date.now()` can be useful for handling the timer. It returns the current time in ms.
b) Log measure ErrorCount: In case of error (wrong element clicked), just count an error but do not log anything. Restart a trial in the same condition (restart the timer). We want to have one correct completion time measure for each condition.

```
DesignName,ParticipantID,TrialID,Block,Trial,VV,OC,visualSearchTime,ErrorCount
...
Preattention-experiment,1,1,1,1,VV2,Medium,1582,2
...
```

The successful trial in this condition took 1582 ms. It was preceded by two erroneous trials in this condition.
TODO step 4

Make sure that your program stops when all trials for this participant are completed (i.e., when `ctx.trials[ctx.cpt]["Participant"]` is no longer the same value)