A decorative graphic on the left side of the slide featuring a blue parallelogram and a light green parallelogram, both tilted at an angle, set against a dark blue background with diagonal stripes.

The language of geometry: Fast comprehension of geometrical primitives and rules in human adults and preschoolers

By Marie Amalric, Liping Wang, Pierre Pica, Santiago Figueira, Mariano Sigman, Stanislas Dehaene

Presented by Julia, Anna, Eoghan and Leane



Introduction

The child's acquisition of language has been suggested to rely on the ability to build hierarchically structured representations from sequential inputs.

Research question: Does a similar mechanism underlie the acquisition of geometrical rules?

Why is this interesting? Might reveal a fundamental aspect of human cognition

Method

A formal language capable of describing, in a compact manner, all sequences of movements on a regular octagon.

0: stays in same location

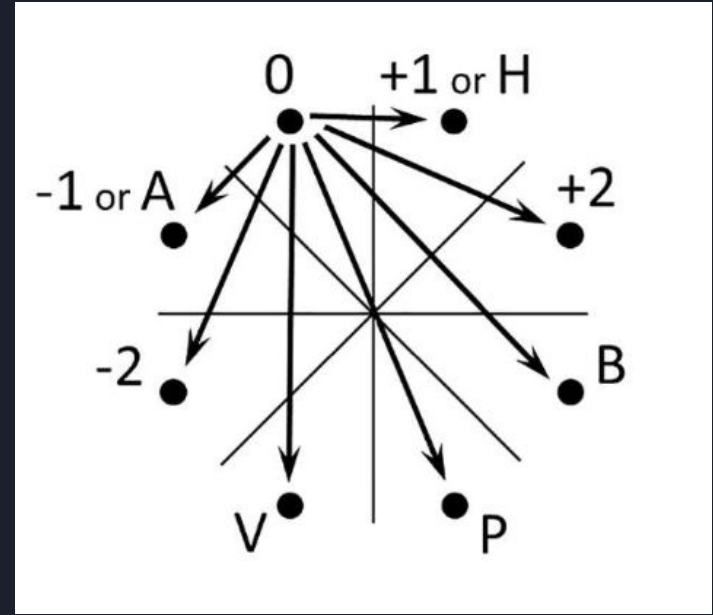
+1,+2: next element clockwise

-1,-2: next element counterclockwise

H,V: horizontal & vertical symmetry

P: rotational symmetry (+4)

A,B:symmetries around diagonal



Method

C ★ 2 random from 768 of max complexity ★ 1 exemplar randomly
 ★ both directions (CW&CCW) ★ All 4 axial symmetries (direction of rotation random)

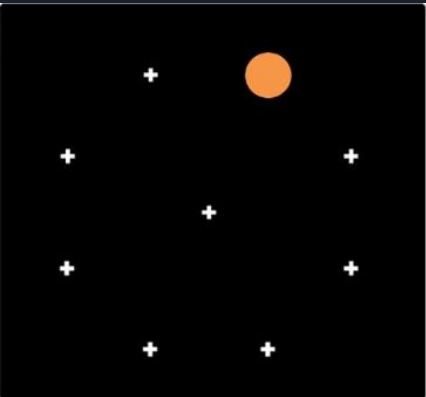
★ Repeat +1 (x2)	Repeat +2	★ Alternate (x2)	★ 2arcs (x2)
★ 2squares (x2)	★ 4segments (x4)	★ 4diagonals (x1)	2points
★ 2rectangles (x1)	★ 2crosses (x1)	★ irregular (x2)	4points

Total of 17
sequences

Experiment 1

Participants: 23 French adults (12 female; mean age 26.6; range 20–46)

1. The 8 dots of the octagon are always visible
2. First a simple CW or CCW “repeat +1”
3. For each participant the sequences are in a random order
4. First two elements of sequence are flashed (location 3)
5. Participant must guess the next location,
6. If incorrect, sequence is flashed again (1-3) with correct answer and participant must guess the following location (location 4)
7. Errors tracked from 3rd and 16th element of sequence
8. 17 sequences in total





Results

- Errors decrease over time (due to rote memory + anticipation)
- Error rate increases with sequence complexity
- Regular sequences learned better than irregular ones
- Quickly detected geometric regularities and generalized from few items

Results were different for preschoolers and uneducated adults

Experimental Environment and Static Stimuli

```
NUM_OF_DOT = 8
EXP_RADIUS = 150
DOT_RADIUS = 10
DOT_COLOR = "grey"
TRIGGERED_DOT_RADIUS = 15
TRIGGERED_DOT_COLOR = "yellow"

dot_positions = [
    (EXP_RADIUS * m.cos((t*2*m.pi/NUM_OF_DOT)+(m.pi/NUM_OF_DOT))),
    (EXP_RADIUS * m.sin((t*2*m.pi/NUM_OF_DOT)+(m.pi/NUM_OF_DOT)))
    for t in range(NUM_OF_DOT)]

dots = [stimuli.Circle(radius = DOT_RADIUS, colour = DOT_COLOR,
    position = p) for p in dot_positions]
```

- **Octagon Definition**
- **Visual Stimuli** : The `dots` list generates the 8 circles that are always visible
- **Feedback/Movement** : The `triggered_dots` list creates the dynamic elements used to flash successive locations during the sequence presentation, using `TRIGGERED_DOT_COLOR` ("yellow") and a larger radius (`TRIGGERED_DOT_RADIUS = 15`) to highlight the current position.

Sequences

Simple Sequences

```
C_sequences_ez = [(np.asarray([i for i in range(NUM_OF_DOT)]), "repeat cw"),  
                  (np.asarray([NUM_OF_DOT-i-1 for i in range(NUM_OF_DOT)]), "repeat ccw")]
```

Regular Sequences

```
✓ C_sequences = [(np.asarray([0,2,4,6,1,3,5,7]), "2squares cw"),  
                  (np.asarray([0,6,4,2,1,7,5,3]), "2squares ccw"),  
                  (np.asarray([0,2,1,3,2,4,3,5,4,6,5,7,6,0,7,1]), "alternate cw"),  
                  (np.asarray([0,6,7,5,6,4,5,3,4,2,3,1,2,0,1,6]), "alternate ccw"),  
                  (np.asarray([0,4,1,5,2,6,3,7]), "4diagonales cw"),  
                  (np.asarray([0,4,7,3,6,2,5,1]), "4diagonales ccw"),  
                  (np.asarray([1,2,3,4,0,7,6,5]), "2arcs cw"),  
                  (np.asarray([0,7,6,5,1,2,3,4]), "2arcs ccw"),  
                  (np.asarray([0,1,7,2,6,3,5,4]), "4segments cw"),  
                  (np.asarray([1,0,2,7,3,6,4,5]), "4segment ccw"),  
                  (np.asarray([0,5,4,1,6,3,2,7]), "2rectangles"),  
                  (np.asarray([0,4,5,1,2,6,7,3]), "2crosses")  
                  ]
```



Irregular sequence

All possible combinations are created for an initial length (position \times direction). Each sequence is filtered to meet the criteria for irregularity:

- No constant regular pattern
- No immediate repetitions or repetitive sub-sequences
- No symmetry
- No visually predictable patterns (simple alternations, identical jump patterns)
- Sequences that are too close to regular sequences

Trials

```
def run_trial(sequence_name, sequence):  
    present_instructions(FIRST_INSTRUCTION)  
  
    present_for(*dots)  
    present_for(*dots + [triggered_dots[sequence[0]]])  
    present_for(*dots + [triggered_dots[sequence[1]]])  
    timed_draw(*dots)  
    good_dot = wait_for_dot_click(sequence[2], dot_positions)  
    i = 2  
    while good_dot and i <= MAX_SEQ_SIZE :  
        timed_draw(*dots + [clicked_dots[sequence[i%NUM_OF_DOT]]])  
        i+=1  
        good_dot = wait_for_dot_click(sequence[i%NUM_OF_DOT], dot_positions)  
        i+=1  
  
    while not(good_dot) and i <= MAX_SEQ_SIZE :  
        present_for(*dots)  
        for j in range(i):  
            present_for(*dots + [triggered_dots[sequence[j%NUM_OF_DOT]]])  
            timed_draw(*dots)  
        good_dot = wait_for_dot_click(sequence[i%NUM_OF_DOT], dot_positions)  
        while good_dot and i <= MAX_SEQ_SIZE :  
            timed_draw(*dots + [clicked_dots[sequence[i%NUM_OF_DOT]]])  
            i+=1  
            good_dot = wait_for_dot_click(sequence[i%NUM_OF_DOT], dot_positions)  
        i += 1  
    # exp.data.add([sequence_name, sequence[0], i - 3])  
    present_instructions(END_INSTRUCTION_SEQUENCE)
```



Click function

```
def wait_for_dot_click(correct_dot_idx, dot_positions, radius=DOT_RADIUS):  
    mouse = io.Mouse(show_cursor=True)  
    while True:  
        _, (x, y), _ = mouse.wait_press()  
        clicked_dot = None  
        for idx, (dot_x, dot_y) in enumerate(dot_positions):  
            dist = m.sqrt((x - dot_x) ** 2 + (y - dot_y) ** 2)  
            if dist <= radius:  
                clicked_dot = idx  
                break  
        if clicked_dot is not None:  
            return clicked_dot == correct_dot_idx
```