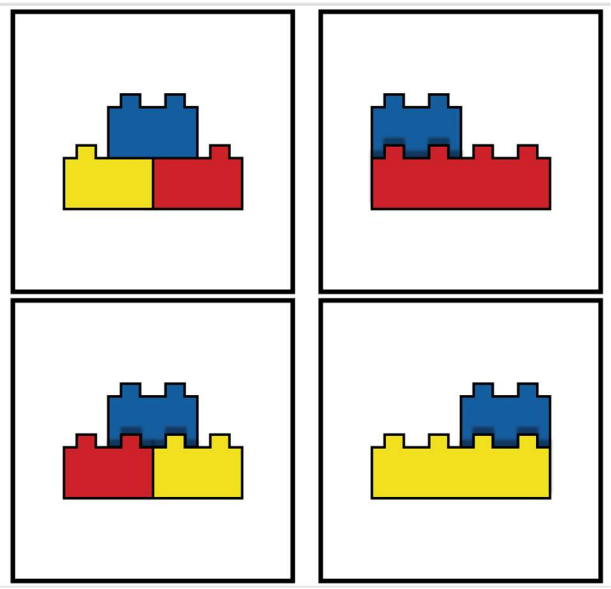
Contents

OVERVIEW

This study explores differences in integration of spatial information when building 3D Duplo constructions from 2D schematics of the construction from different perspectives.

Participants are shown four schemas of the same construction from its four sides (example below) and must build the construction. They must use shadows to understand depth and integrate information from all four pictures to build models. Participants complete 8 constructions at varying complexity and difficulty. Participants complete models at their own pace (up to 30 mins for adults, up to 45 for children).

The videos we'll be coding will come from the scene-view of the eye-trackers used in the study. Our aim is to code each move each participant makes whilst building, so that we can create construction paths (from first move to complete construction) for each construction each participant makes.



COLUMNS

ID

This column provides meta-data about the video that is being coded and which participant. It is really important because it's being printed along with the other data and it allows us to know which participant was coded. Otherwise – we would not know. Please make sure there is only one cell in the video and it covers ALL the video.

<id>,<tdate>, <bdate>,<sex>

ONSET
First frame of video

OFFSET
Last frame of video (don't change)

CONTENTS (ARGUMENTS) OF ID VARIABLE
Get information from coversheets or questionnaires

Any missing data should be recorded as "."

<id>
Participant ID is written in as S#XX where:

XX is the participant number For numbers less than 10, should appear with a starting 0. So participant 1 is "01".

<tdate>
Subject's test date

Format: mm/dd/yy

<bdate>
Subject's date of birth.

Format: mm/dd/yy

<sex>
male 'm'

female 'f'

TASK
The aim of this column is to capture what task the participant is currently doing. There are 8 tasks – each corresponds to a model that the participant is asked to build.

<model_type>, <correct_final_ci>

ONSET:

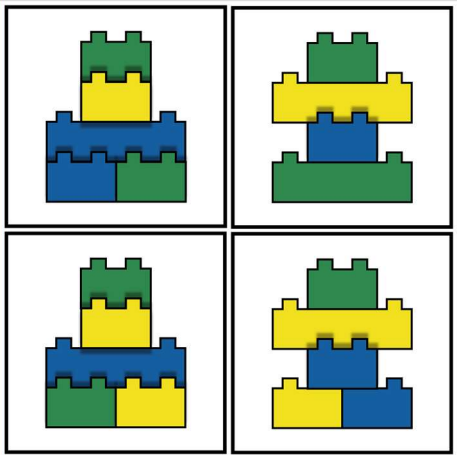
The first frame where the 4 point of views of the model appears

OFFSET:

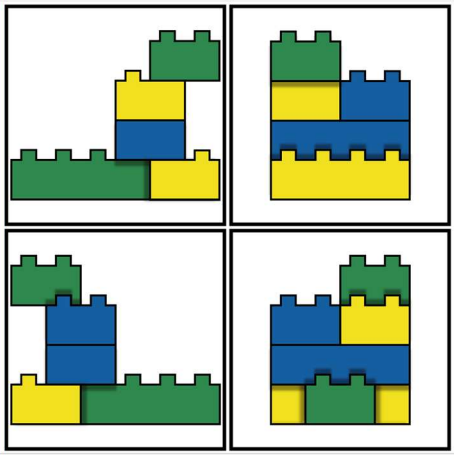
The first frame where the participant shows the experimenter the model (few moments after that, the experimenter or the participant deconstruct the model the participant built)

<model_type>

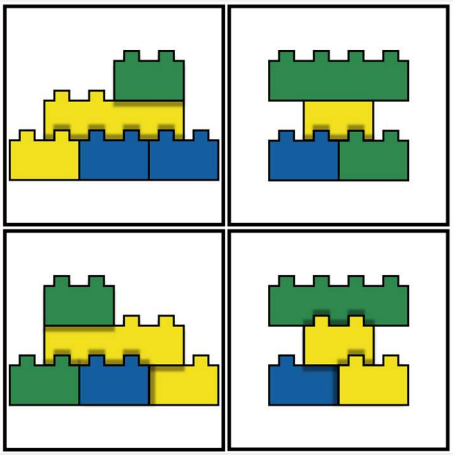
m1 = first model –



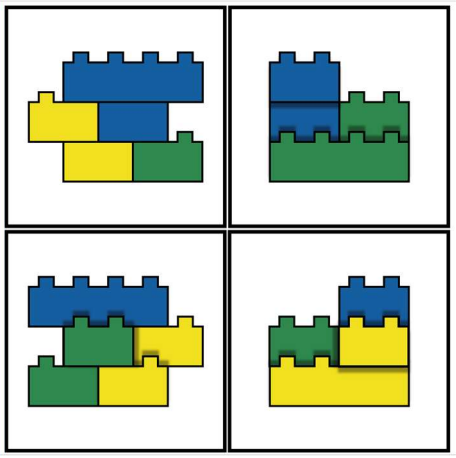
m2 = first model –



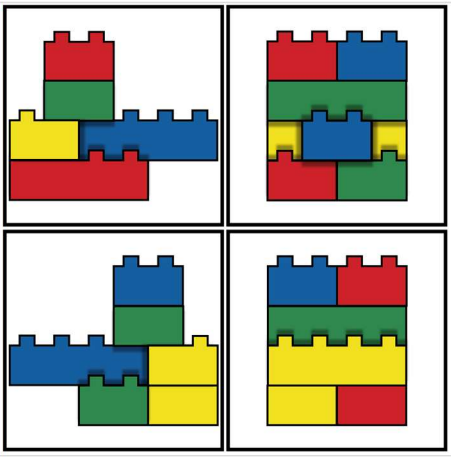
m3 = first model –



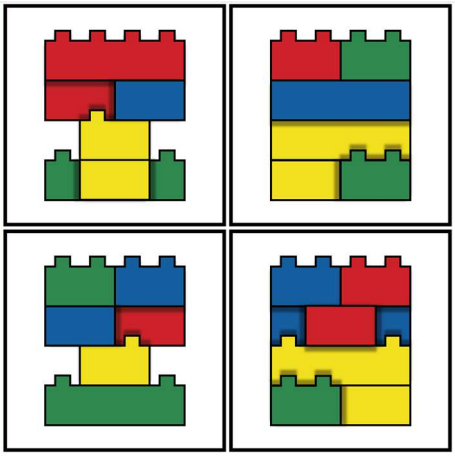
m4 = first model –



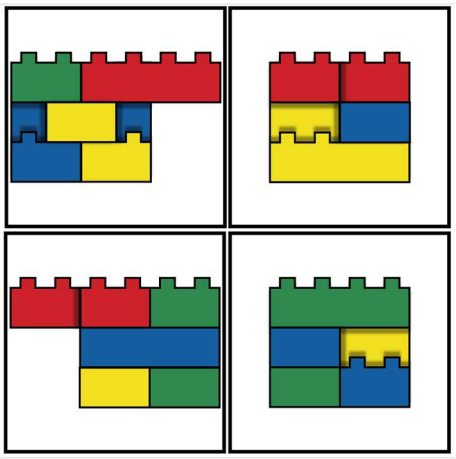
m5 = first model –



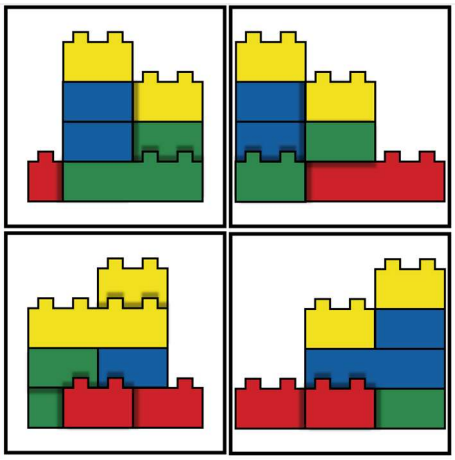
m6 = first model –



m7 = first model –



m8 = first model –



<correct_final_ci>

This code includes whether the final model the participant created was correct or incorrect. Use the pictures above, the Final constructions powerpoint and pictures, and your own DUPLO blocks, to check if the model is correct.

c = correct model

i = incorrect model

CONSTRUCT_ACTION

In this column, we code when the participant completes an action in relation to the construction within a trial. The columns include **point cells** that indicate the time when participant performed the action. The code includes the spatial relation between the new added block and the rest of the model. This is defined as ONLY when blocks are **interlocked** with each other. A new block can have relations with maximum 6 blocks (usually not more than 2), so there is option to add relations of up to 6 blocks.

To code the spatial relation between the new block and the construction we think about two things:

1. The overall relation between the new block and the model (was it added or removed, above or below).
2. The specific relation between the new block and the blocks in the model (what blocks was it interlocked with, in what orientation, in what exact configuration).

Configuration (nature) of relation is coded using offset of Duplo pips (i.e. how offset are the blocks from each other in two plains, left-right, front-back).

We ALWAYS code from the perspective of the participant.
We ALWAYS code the relation FROM the new block TO the related block. i.e. for left-right, we code how much we would

have to move the NEW block TO the related block in order for them to be flush. (If we need to move the NEW block left for the blocks to be flush, then it is a left move).

ONSET = OFFSET = The first frame where the interlock was completed.

NOTE: This should be as close as possible to the moment where the blocks fully interlock.

|<action_cd>,<target_rgbls>,<verticality_ab>, <orientation_longblock_hv> <relation1_color_rgbls>,<relation1_frontback_offset>,<relation1_leftright_offset> , <relation2_color_rgbls>,<relation2_frontback_offset>,<relation2_leftright_offset> , <relation3_color_rgbls>,<relation3_frontback_offset>,<relation3_leftright_offset> , <relation4_color_rgbls>,<relation4_frontback_offset>,<relation4_leftright_offset> , <relation5_color_rgbls>,<relation5_frontback_offset>,<relation5_leftright_offset> , <relation6_color_rgbls>,<relation6_frontback_offset>,<relation6_leftright_offset>

Codes:

<action_cd>

This code includes the action that was performed. If this happens out of sight (e.g. behind the model), try and code it as close to the moment you think it happened (this can often be the case for falls). **NOTE** for falls only code if the block has completely fallen off. Blocks are not so sturdy and a lot of the time they will slightly fall when new blocks are being added, we should not be coding these as falls.

c=construction (adding new block relations to the construct)

d=deconstruction (removing a block relation from the construct)

f = fall (a block has fallen or clicked off)

<target_rgbls>

This code includes which target block was added (in case of construction action) or removed (in case of deconstruction action). This is always the block ADDED to the model (model being defined as a group of blocks). If it is the first move, you can decide which block is the target.

rs = the small red block

bs = the small blue block

gs = the small green block

ys = the small yellow block

rl = the large red block

bl = the large blue block

gl = the large green block

yl = the large yellow block

<verticality_ab>
This code includes whether the target block was added to the top or bottom of the construct. This is same for deconstruction (i.e. if the block is deconstructed from on top of the model).

a = the target block was added/deconstructed above the construct

b = the target block was added/deconstructed bellow the construct

<orientation_longblock_hv>
This code includes whether the new added block was placed in a horizontal or vertical orientation **if it is a long block** (if it is a short block, its orientation will not matter and should be coded '.'). Note, this should be coded from the participant's perspective (as it will change generally).

h = the target block was added in a horizontal position (from participant's perspective)

v = the target block was added in a vertical position (from participant's perspective)

. = the target block does not have an orientation (small block).

<relation1_color_rgbls>
This code includes which relational block the target block was added or removed from IN THE CONSTRUCTION. This code CANNOT be '.' This is coded similarly to <target_rgbls>

(rs, bs, gs, ys, rl, bl, gl, yl).

<relation1_orientation_longblock_hv>
This code includes whether the relational block is in a horizontal or vertical orientation **if it is a long block** (if it is a short block, its orientation will not matter and should be coded '.'). Note, this should be coded from the participant's perspective (as it will change generally).

h = the target block was added in a horizontal position (from participant's perspective)

v = the target block was added in a vertical position (from participant's perspective)

. = the target block does not have an orientation (small block).

<relation1_frontback_offset>
This code includes the front-back offset between the target block and the FIRST relational block. Offset is calculated by looking at pip alignment. For front-back, we want to understand how many pips front-back the target block would have to move to be FLUSH with the relational block. 0 for no offset (blocks are flush IN THIS DIMENSION). Positive numbers if the target block is in **FRONT (BELOW)** of the relational block from the participant's perspective. Negative numbers if the target block is **BEHIND (ABOVE)** the relational block from the participant's perspective. The actual number is the number of pips it must be moved. Examples are below.

Tip: think of it as vector movements on a pos/neg x/y axis, FROM target block TO relational block.

(-3, -2, -1, 0, 1, 2, 3)

<relation1_leftright_offset>
Similar to the previous code – this code includes the left-right offset between the target block and the FIRST relational block. Offset is calculated by looking at pip alignment. For front-back, we want to understand how many pips front-back the target block would have to move to be FLUSH with the relational block. 0 for no offset (blocks are flush IN THIS DIMENSION). If the target block is to the **LEFT** of the relational block (i.e. must be moved right), the offset is positive. If the target block is to the **RIGHT** of the relational block (i.e. must be moved left), the offset is negative. The actual number is the number of pips it must be moved. Examples are below.

Tip: think of it as vector movements on a pos/neg x/y axis, FROM target block TO relational block.

(-3, -2, -1, 0, 1, 2, 3)

<relation2/3/4/5/6_color_rgbls>

<relation2/3/4/5/6_orientation_longblock_hv>
<relation2/3/4/5/6_frontback_offset>

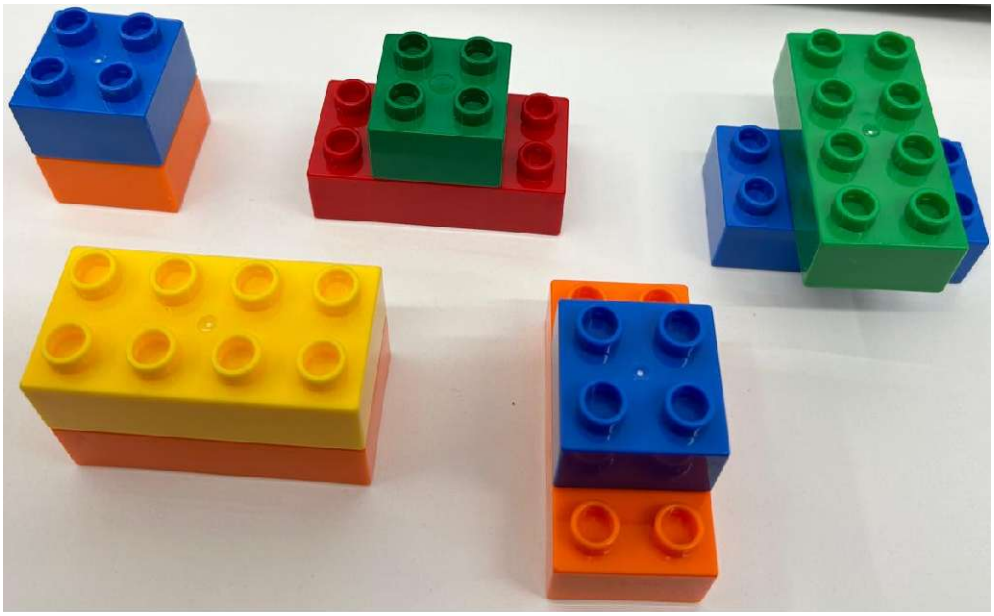
<relation2/3/4/5/6_leftright_offset>
The rest of these codes are used to code further relational blocks IF THERE ARE ANY. Otherwise they should be coded ".". There can be up to 5 other relational blocks.

Construct-Action TIPS:
REMEMBER: we are using this code to understand the construct path of each participant. It is important to be careful and specific about what each move is. It is REALLY important to be clear about what perspective you are coding from, as this will change the code.

- **The best way to understand the offset code is by thinking about vector movements made FROM target block TO relational block from ABOVE the participants perspective**
- **Always code from THE PARTICIPANT'S PERSPECTIVE. If they move the construction as they are constructing or deconstructing, try and pick the exact moment and stick to that perspective.**

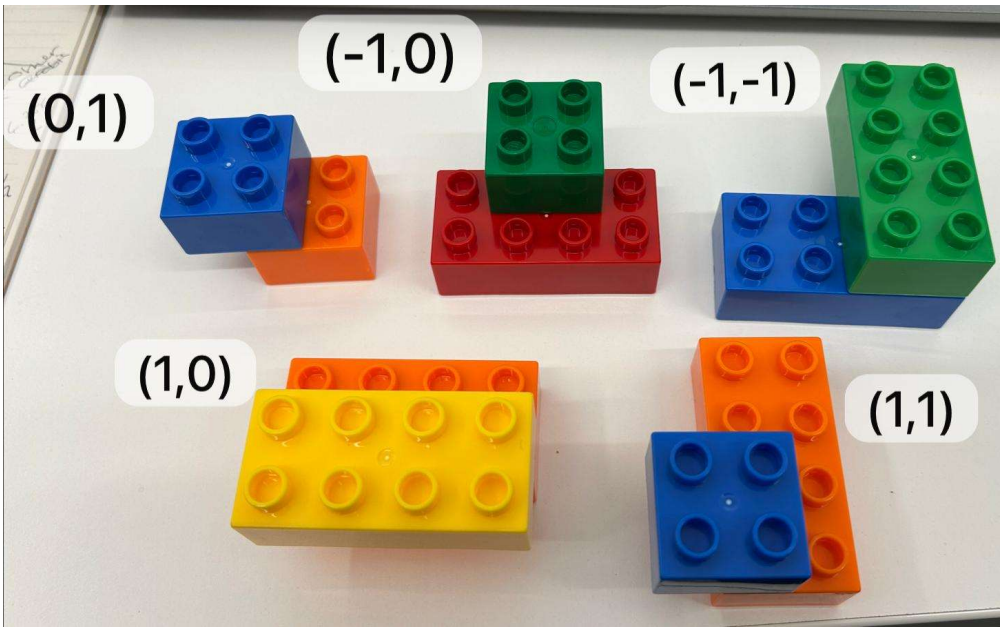
Basic Offset Rules

Flush Blocks

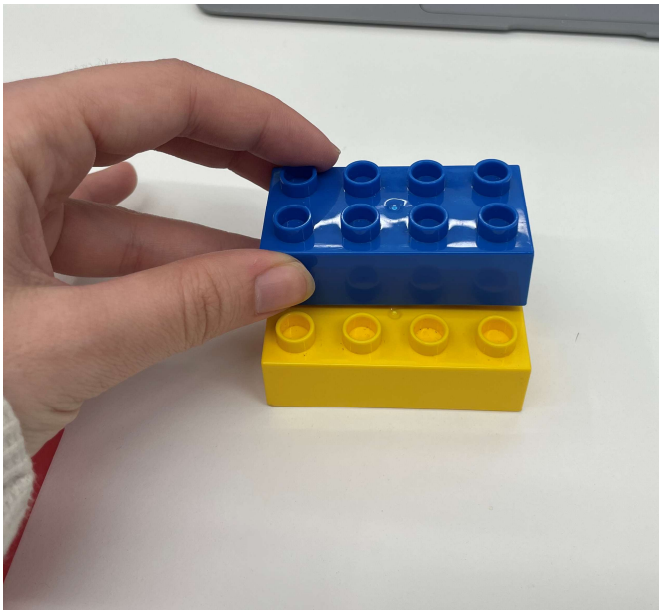


These are ALL examples of flush blocks.

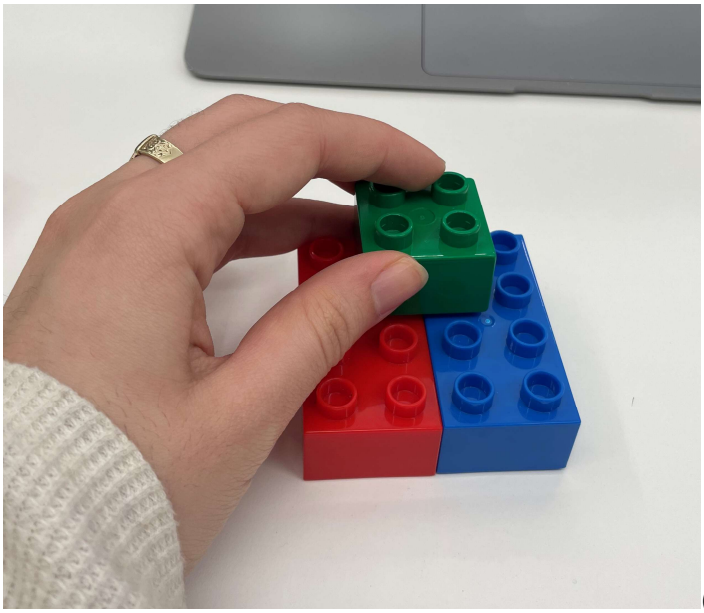
Offset by 1
These are examples of variations of an offset of 1/-1 in front-back and left-right. Important to remember that the same offset can look different with different blocks or blocks of different orientations.



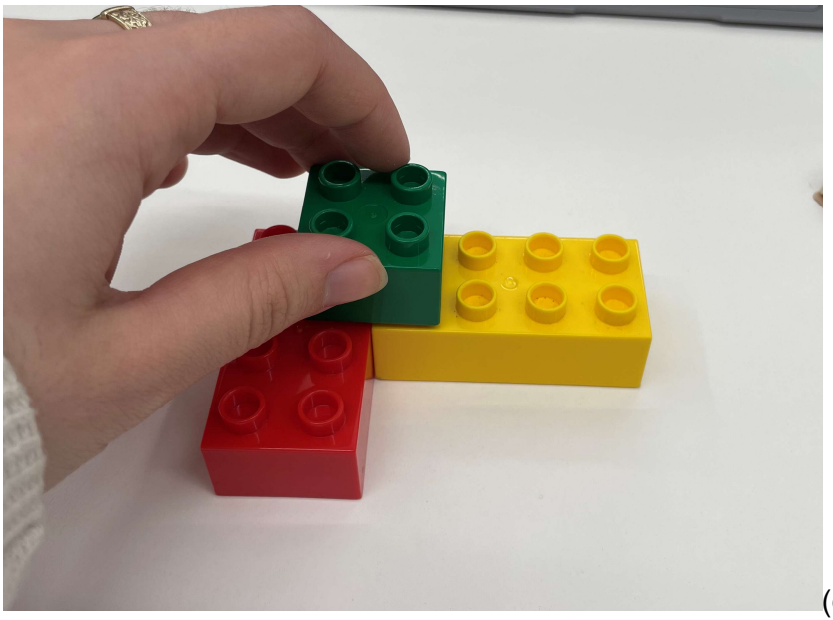
Here are few examples of codes and how they look like in the video, please MAKE SURE you UNDERSTAND why this is the code for this particular image:



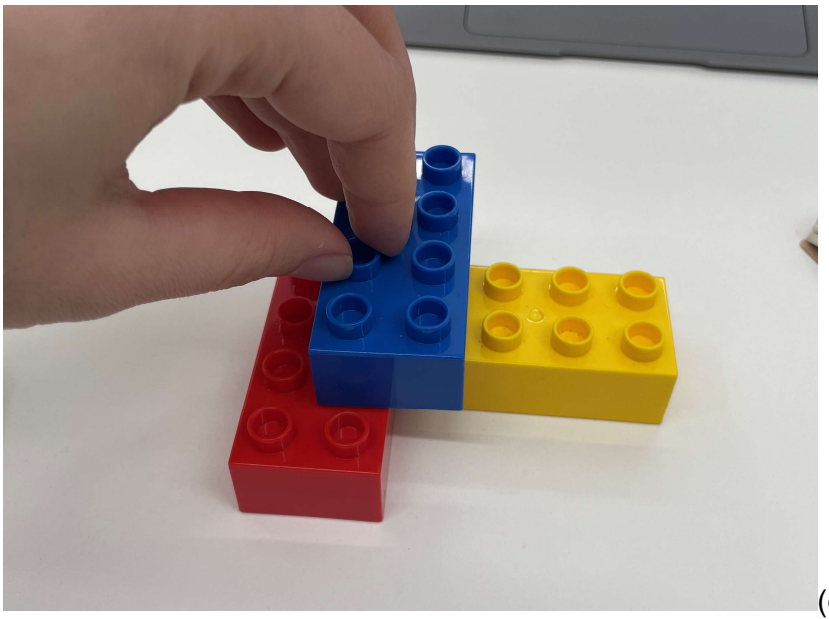
(c, bl, a, h, yl, h, -1, 0)



(c, gs, a, ., rl, v, -1, -1, bl, v, -1, 1)



(c, gs, a, ., rl, v, -1, -1, yl, h, 0, 2)



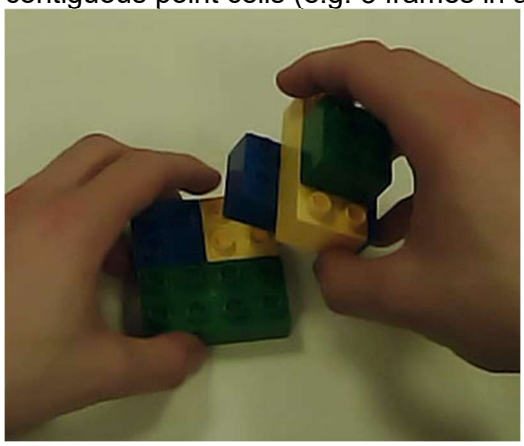
(c, bl, a, v, rl, v, -1, -1, yl, h, 0, 2)

(c, gs, a, ., rl, h, -1, -1, yl, h, -1, 1)

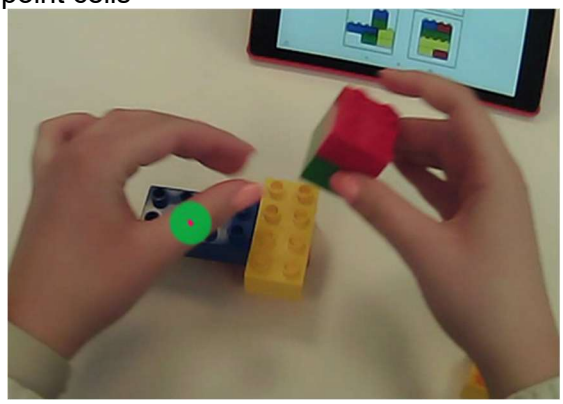
SPECIAL CASES

- **If participant connects several blocks at once:**
    - If it is one target block being connected to several blocks this requires normal code with more than one relational block

- If the target block(s) are laid out (see below), and all added at same time, we will use contiguous point cells (e.g. 3 frames in a row for the three moves)

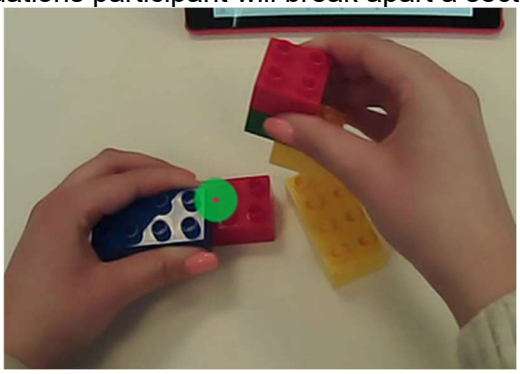


- If participant interlocks TWO ‘constructions’ (sets of blocks interlocked): pick one set to be the main model, making the other ‘construction’ a set of target blocks. Then code them using contiguous point cells

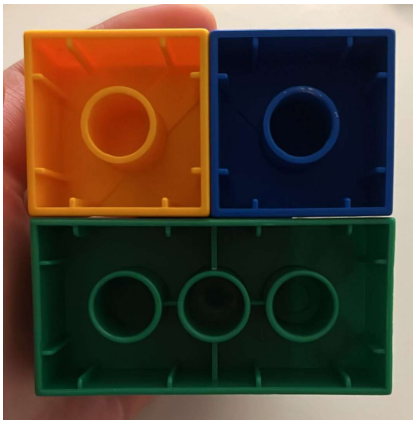


  - E.g. participant adds gs and rs (already connected) to an already connected main model (under her hands). Gs and rs are the target blocks here and should be coded as two separate moves in contiguous point cells

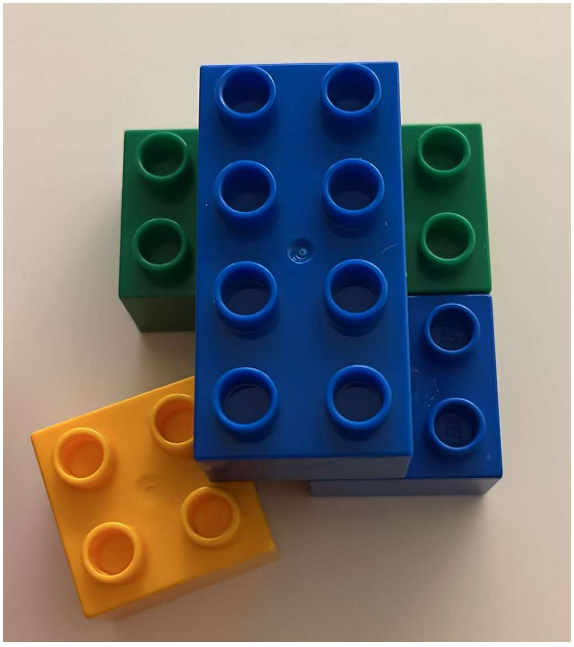
- **If participant disconnects several blocks at once:**
  - In some situations participant will break apart a section of the model to redo:



    - E.g. here they have pulled the rs,gs,ys off of the main model (currently bl, rl). For this situation, we would code 3 contiguous point cells reflecting three separate deconstructions (even though it’s one move). We can recognize it as one move still since the cells are contiguous.
- If participant connects blocks UPSIDE DOWN:
  - This creates quite a complex coding situation because we have to subvert the participants perspective in order to code it



- E.g. in this picture the ys has just been added to the construction from underneath
  - To code this, we want to code what this move would look like if the model was upright. That means TURNING IT TOWARD YOU (like image A below)



- IMAGE A

PERSPECTIVE_ORIENTING

In this column, we code when participants oriented the perspective of the construction by turning it with their hands to look at a different perspective.

ONSET = **OFFSET** = The first frame where the participant shifted perspective. PLEASE NOTE: You should code ***one turn at a time.*** This means that each perspective change is one cell only. If the participant turning perspective fast, you should code by looking at the video slowly and code each turn separately.

<direction_clockwise_anticlock>

This code indicates the direction of the rotation.

c = clockwise
a= anticlockwise

. = same frame as first move.

. = first construction move of sequence. i.e. either the first move of each trial, or if they take the whole model apart and start again, same frame as first reconstruction move.

REPEAT - if NO BLOCKS ARE CONNECTED (EVERYTHING HAS BEEN DISCONNECTED) – THERE SHOULD BE A . in PERSP ORIENTING AT THE SAME FRAME OF THE FIRST CONSTRUCTION MOVE AFTER.