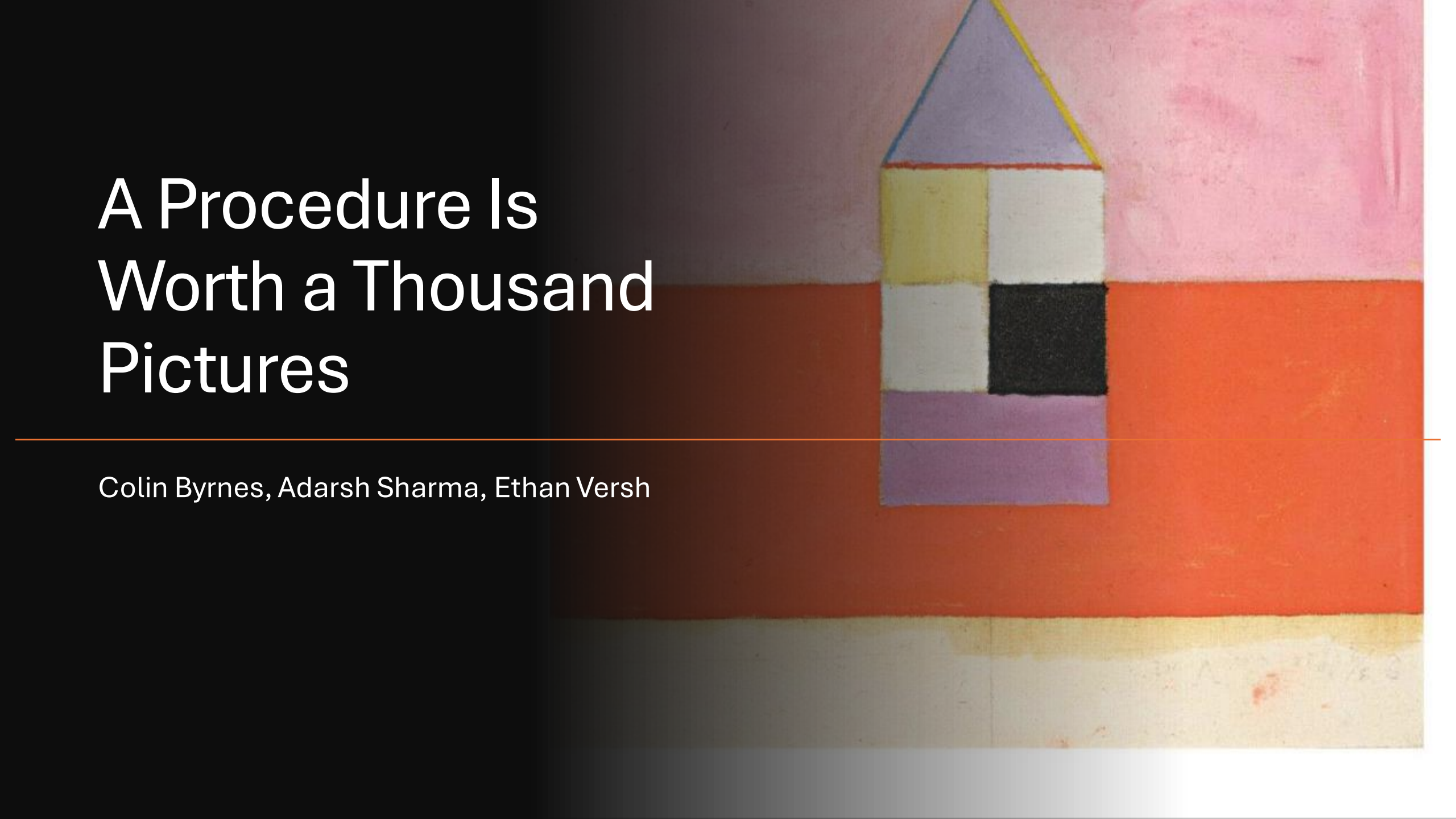
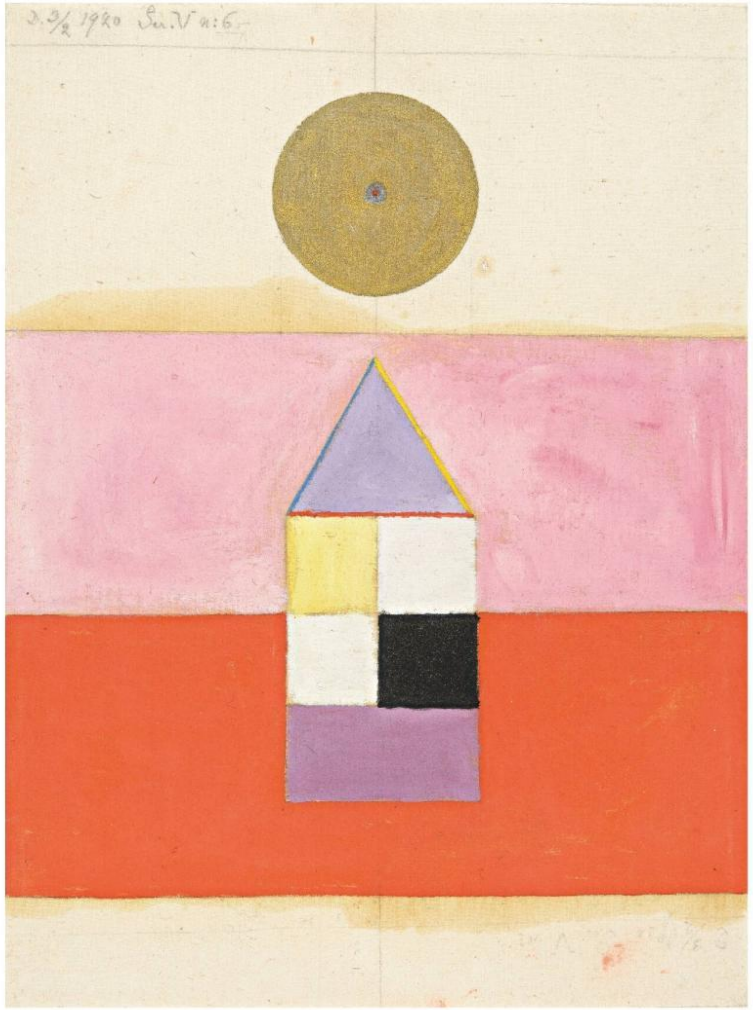
A Procedure Is Worth a Thousand Pictures

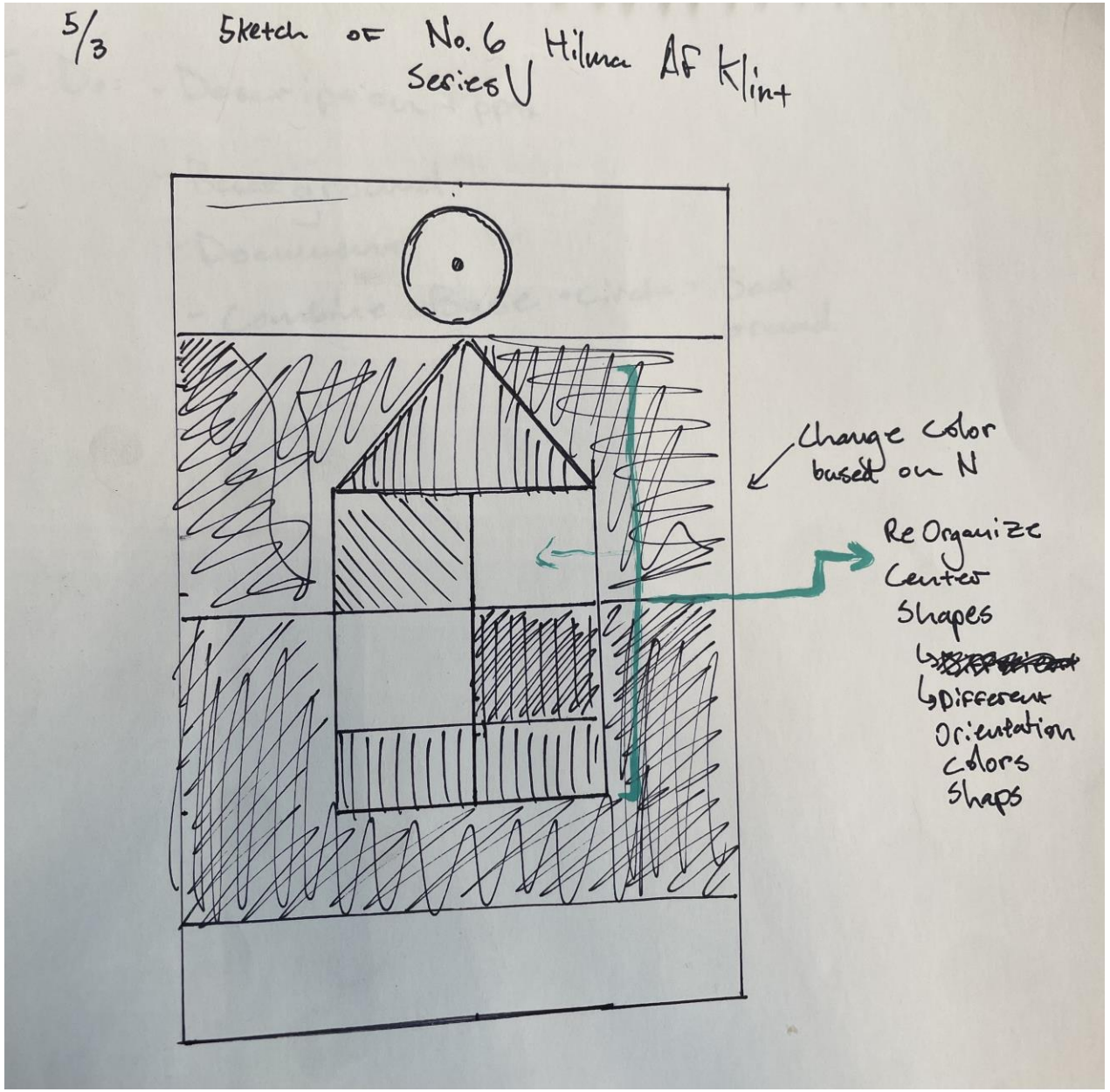
Colin Byrnes, Adarsh Sharma, Ethan Versh



Original Image



Sketch of Image & Goals



Design Plans

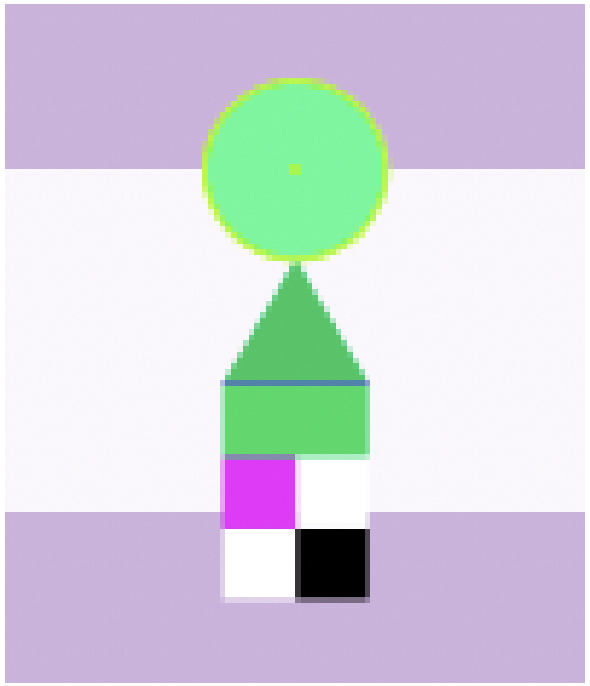
- **Base Image Creation**
  - Procedures **base1** through **base8** construct house-like structures from basic geometric shapes.
- **Image Selection**
  - **base-unit** selects one of the house-like structures based on the input **n**.
    - Uses **sum-of-digits** as helper procedure which calculates the sum of digits **n**.
- **Image Modification**
  - The **base** procedure modifies the selected image, potentially flipping it or adding house-like structures depending on **n**.
- **Dynamic Circle Images**
  - **circle-unit** creates a layered circle image with dynamic colors that change based on **n**, utilizing the **color-mod** procedure.

Design Plans con't

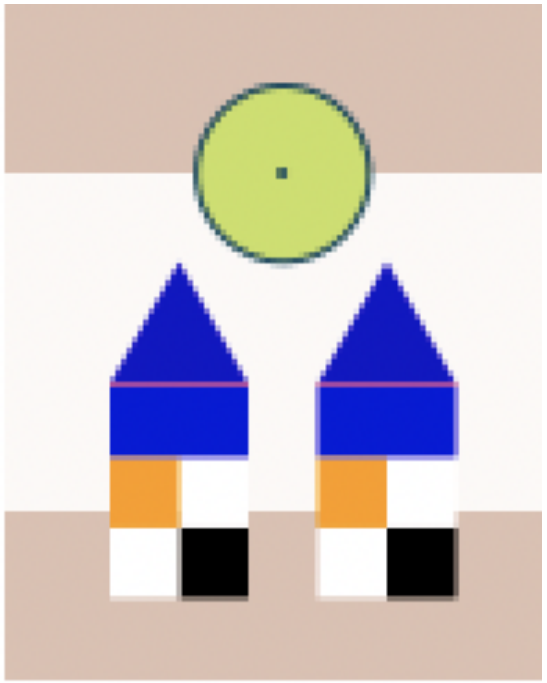
- **Circle Arrangement**
  - **circle** arranges these circle images in various configurations depending on **n**.
- **Background Creation**
  - **canvas** creates a vertically stacked background of colored rectangles using **color-mod2, create-rectangles, and stack-rectangles** as helper procedures, with colors varying based on **n** using **color-mod2.**
- **Main Image Composition**
  - **center-piece** constructs the main image by stacking a circle image above a base image.
- **Final Image Assembly**
  - **image-series** overlays the center piece onto the background canvas, completing the image.
    - **The rotate-hue** function adjusts the overall hue of the final image based on **n**.
- **Procedural Generation Techniques**
  - The procedures use recursion, conditional statements, and dynamic color manipulation to make sure each image is unique for given parameters (**n**, **width**, **height**).

Examples

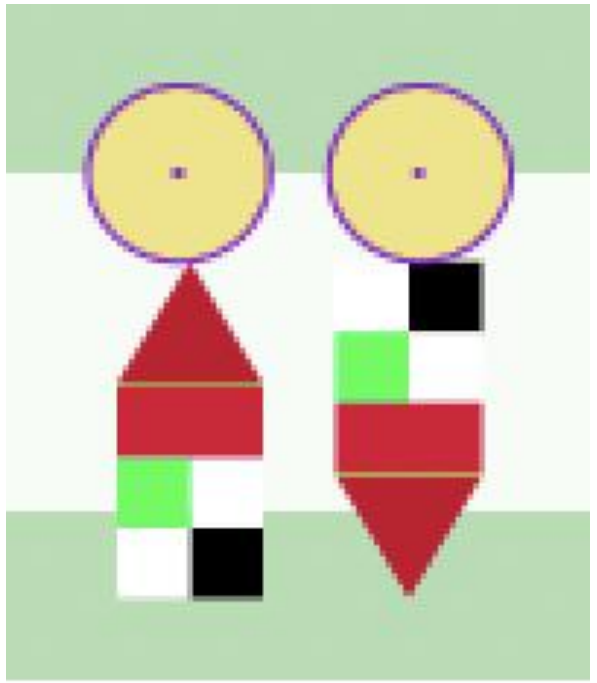
(image-series 250 100 100)



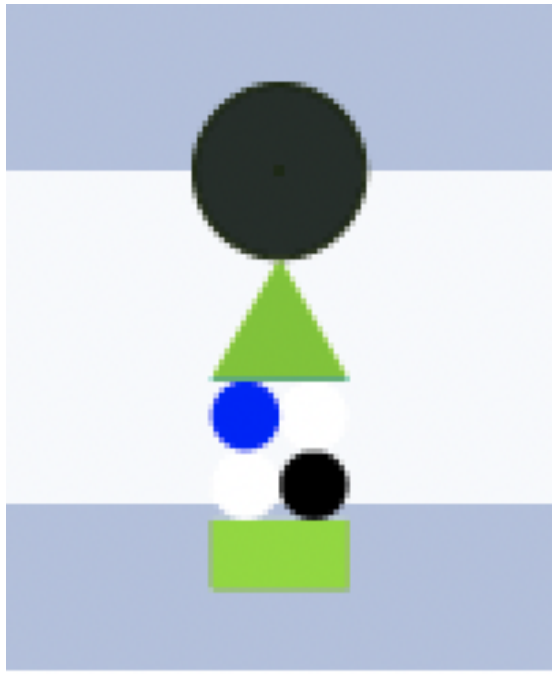
(image-series 350 100 100)



(image-series 450 100 100)



(image-series 550 100 100)



Notable Code

```
;;; (create-rectangles colors width height) -> image?
;;;   colors : (listof color?)
;;;   width : real?
;;;   height : real?
;;; Creates a list of rectangles with colors based on the input `colors`, with dimensions `width` and `height`.

(define create-rectangles
  (lambda (colors width height)
    (if (null? colors)
        '()
        (cons (solid-rectangle (* width 1) (* height 0.3) (car colors))
              (create-rectangles (cdr colors) width height)))))

;;; (stack-rectangles rects) -> image?
;;;   rects : (listof image?)
;;; Stacks a list of rectangles vertically to create the final canvas image.

(define stack-rectangles
  (lambda (rects)
    (if (null? rects)
        (solid-rectangle 0 0 (rgb 0 0 0 0)) ; Empty rectangle
        (above (car rects) (stack-rectangles (cdr rects))))))


;;; (color-mod2 digit) -> rgb?
;;;   digit : real?
;;; Generates an RGB color based on the input `digit` by applying modular arithmetic.

(define color-mod2
  (lambda (digit)
    (rgb (remainder (* digit 3) 255)
         (remainder (* digit 5) 255)
         (remainder (* digit 7) 255))))
```

```
;;; (canvas n width height) -> image?
;;;   n : real?
;;;   width : real?
;;;   height : real?
;;; Creates a canvas image with four rectangles of different colors based on the input `n`, with dimensions `width` and `height`.

(define canvas
  (lambda (n width height)
    (stack-rectangles (create-rectangles (map color-mod2 '(244 254 254 244)) width height))))
```

Q&A