Język dla grafiki 2D

Planning and drawing

In this programming language shapes/pictures are evaluated lazily. We declare our shapes, their relations and construct them in an abstract viewport, defined only in relations to each other.

When you want to force evaluation of this abstract image, call draw on a shape. This shape and all the shapes related to it will get evaluated against a defined viewport, giving them concrete pixel dimensions.

Declaring shapes

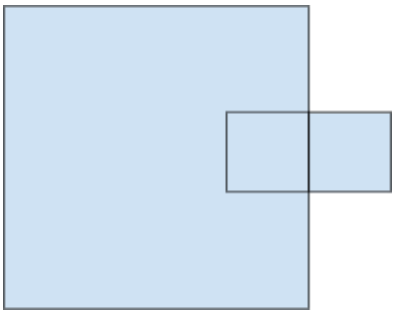
#viewport 1920 1080 // deklaracja rozmiaru viewportu w jakim rysujemy
square Big (50%); // this 50% is the first and only dimension of this square, so its by default relative to 1st dimension of this shape's parent viewport
rect R (20%, 30%); // width is 20% of viewport width, height is 30% of viewport height
rect Example (20% of 2, 30% of 1); // now width is 20% of second viewport dimension, so its 20% of height and viceversa for height
circle C (25%);
triangle [20%, 30%]; // these are bounding box dimensions, no complex shape math for now
square Unusual (50%) [30%]; // square, only single defining dimension given in round braces, but will get limited by bounding box dimensions given in square brackets; bounding box dimensions are always evaluated last, so they have "the final say"
// TODO: ROTATIONS

Shape relations

1) *right, left, on, under*

Unusual right Big outer;
Unusual right Big inner;
Big.left.outer == Unusual;

draw Big; // will also force Unusual to get drawn
// produces this image (actually CompileError, Unusual is in two places at once(Should it even be a compile error??)):



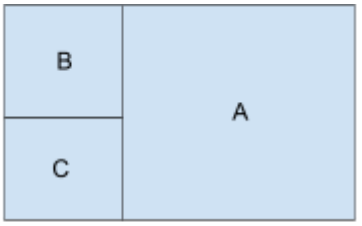
Small on Big; // Small centered in its viewport, Small is drawn on top of Big

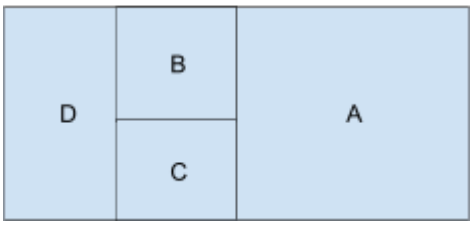
Small under Big; // Small centered inside Big, small is drawn under Big (is invisible in this case)

---

square A, B, C, D;

B left A;
C left A;
D left A;
D under B;
D under C; // D now doesn't have to share left side space; B and C will each be half the size of A's side, drawn to the left of it, but D will be on its own level, taking up the entire left side, being drawn below B and C





---

square A;
square B;
A left B;
draw A;
// Now, some users might expect two, symmetrically aligned squares, with the common edge centered in the middle of parent viewport; this is good default behaviour

square A;
draw A;

square B;
A left B; // A will not move as it is already drawn; B will try to fit in

draw A; // This will look for new shapes attached to A's graph and try to draw them

2) in

square A;
square B;

B in A; // now A is the viewport for B; all % dimensions given to B will now be a % of A dimensions

3) Code reusability

S() {
    // This code is planning shapes in an abstract viewport
    // Calling S will generate an abstract object
    square A;

```
        square B;
}

shape S1 = S();  // point in the middle of parent viewport; no area no color, so it doesn't
"really" get drawn; but it still might be used to orient other shapes around it;
draw S1;

square B;
B left S1;
draw S1;
```

3) Another feature would be image math; this would allow merging shapes together using operators

```
PictureFrame() {
        square Frame (100%);
        circle Hole (30%);
        Frame - Hole;
}


rect Sq (100%, 50%);
square Sq2 (50%);
circle C (30%);

Sq left Sq2;
draw Sq;
```