goto; conference aarhus

ESCAPING A RUT WITH ARRAY THINKING

David Leibs
Oracle Labs

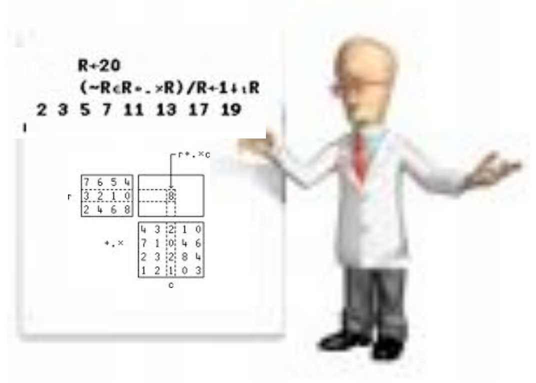
INTERNATIONAL SOFTWARE DEVELOPMENT CONFERENCE

gotocon.com

Escaping a rut with Array Thinking

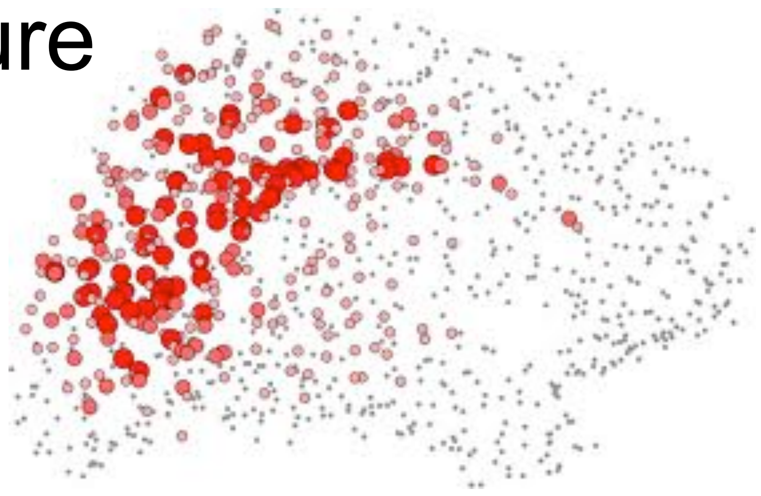
David Leibs
Oracle Labs

Agenda: We will discuss

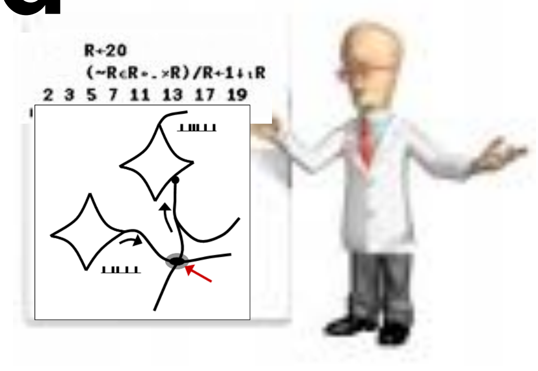


```
     R←20
     (~R∊R∘.×R)/R←1↓⍳R
  2 3 5 7 11 13 17 19
```

- Pseudo Neuroscience and metaphor
- Our propensity to stick with the groove
- Path Dependence
- Quick look at how I am wired
- A quick introduction to Array Programming
- Take a look at some ideas for the future

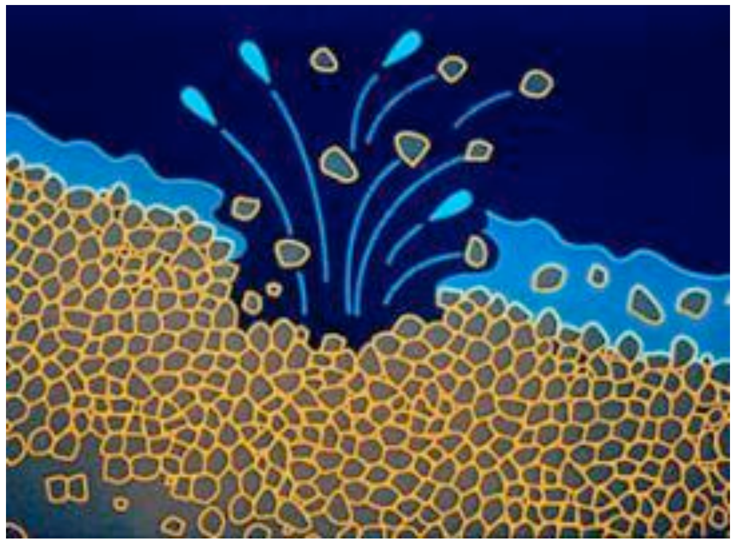


A metaphor for our perception and learning

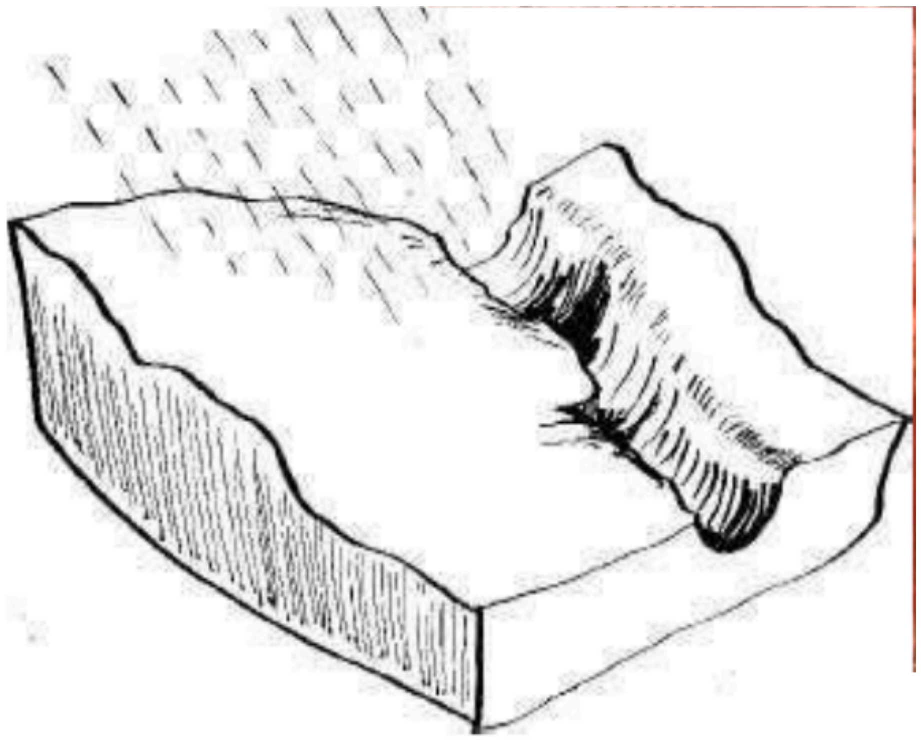


```
      R←20
      (~R∊R∘.×R)/R←1↓⍳R
2 3 5 7 11 13 17 19
```

Water shapes the land



- Water falls randomly
- Gravity starts a groove
- Once a groove starts it is reinforced
- It becomes a rut



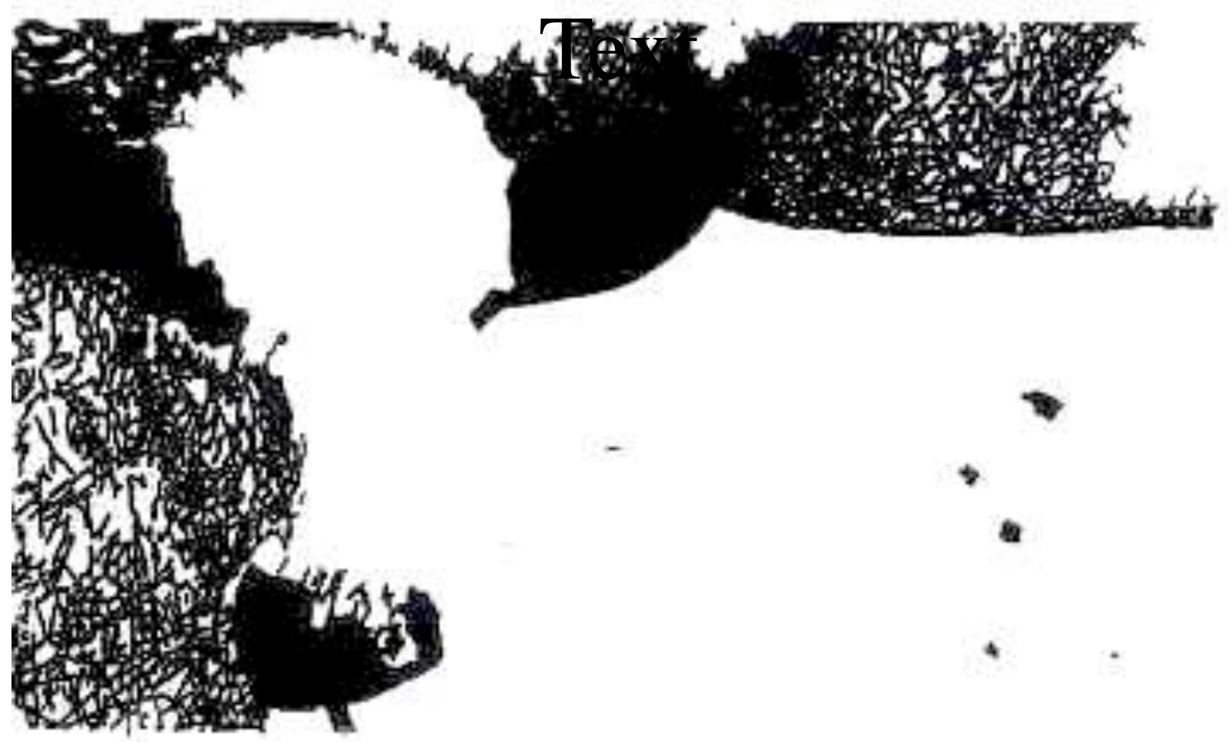
Over time water carves grand canyons

Learning, Practice, and Perception

- We are very influenced by what we first learn
- As we practice that to which we are drawn to we “fall into a groove”

Learning, Perception, and Practice

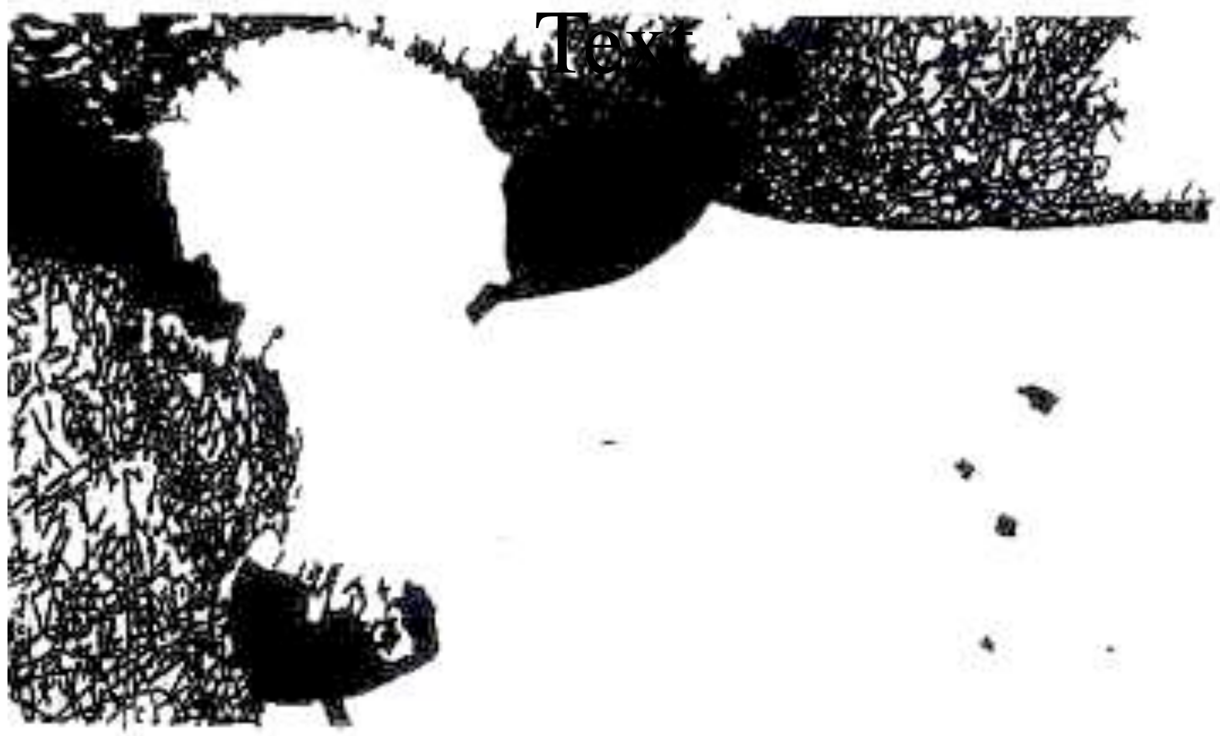
What Do You See?



ORACLE

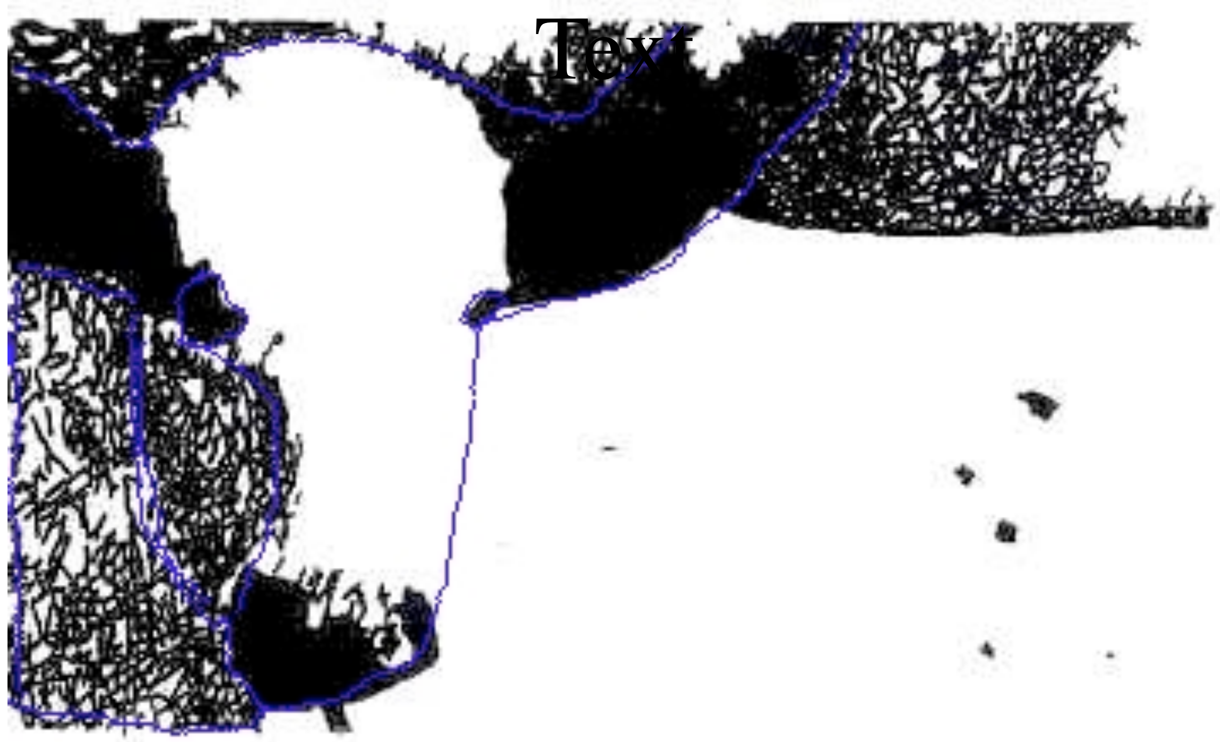
Learning, Practice, and Perception

Can you see the cow?



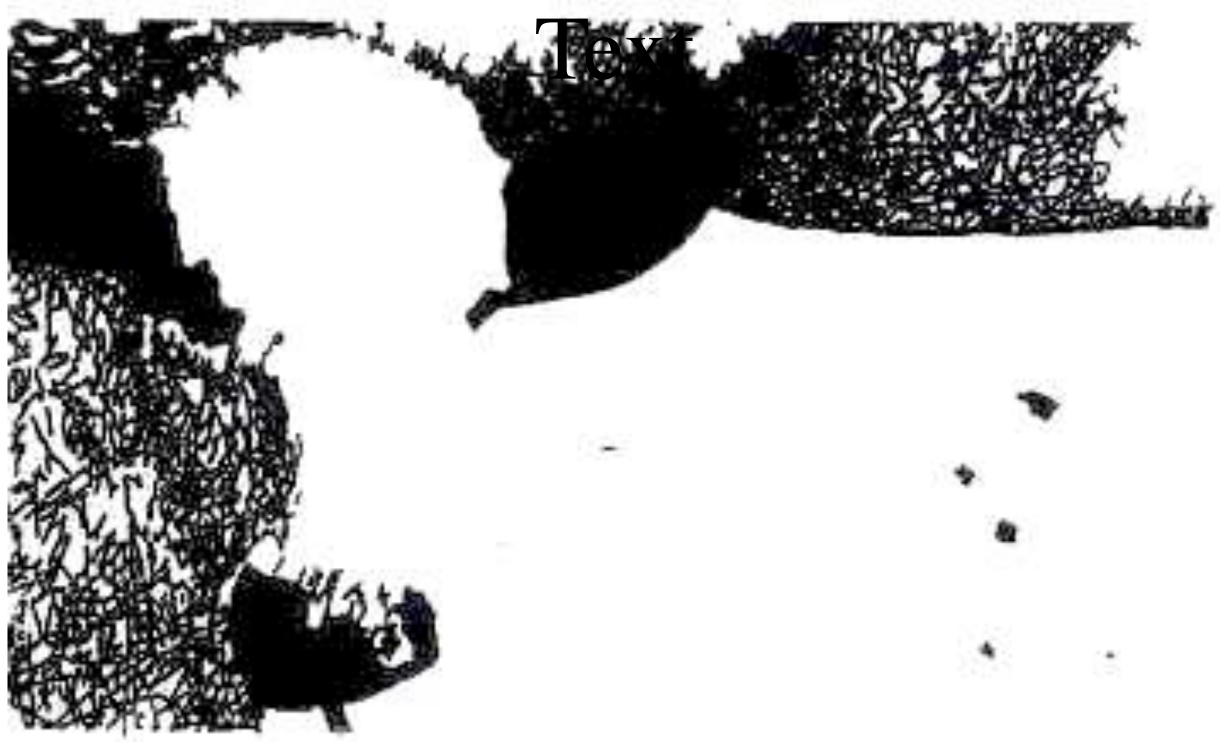
Learning, Practice, and Perception

Can you see the cow?



Learning, Practice, and Perception

Now, can you not see the cow?



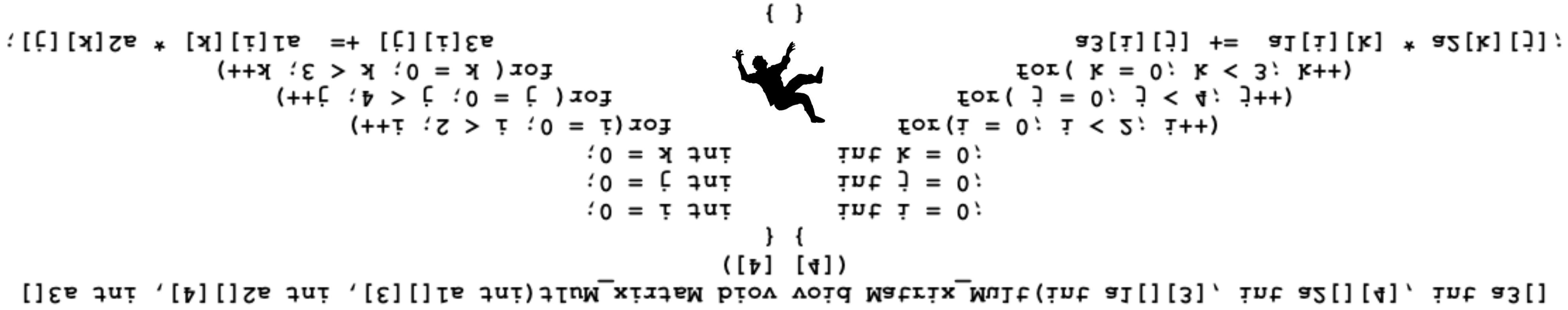
Ultimately we find ourselves at the bottom of a canyon

And we find ourselves at the bottom of a for loop

To most programmers this looks normal

```
void Matrix_Mult(int a1[][3], int a2[][4], int a3[]
[4])
{
   int i = 0;
   int j = 0;
   int k = 0;
   for(i = 0; i < 2; i++)
       for( j = 0; j < 4; j++)
           for( k = 0; k < 3; k++)
               a3[i][j] +=  a1[i][k] * a2[k][j];
}
```

And we find ourselves at the bottom of a ~~for loop~~ a canyon



```
void Matrix_Mult(int a1[][3], int a2[][4], int a3[][4])
{
    int i = 0;
    int j = 0;
    int k = 0;
    for(i = 0; i < 2; i++)
        for( j = 0; j < 4; j++)
            for( k = 0; k < 3; k++)
                a3[i][j] += a1[i][k] * a2[k][j];
}
```

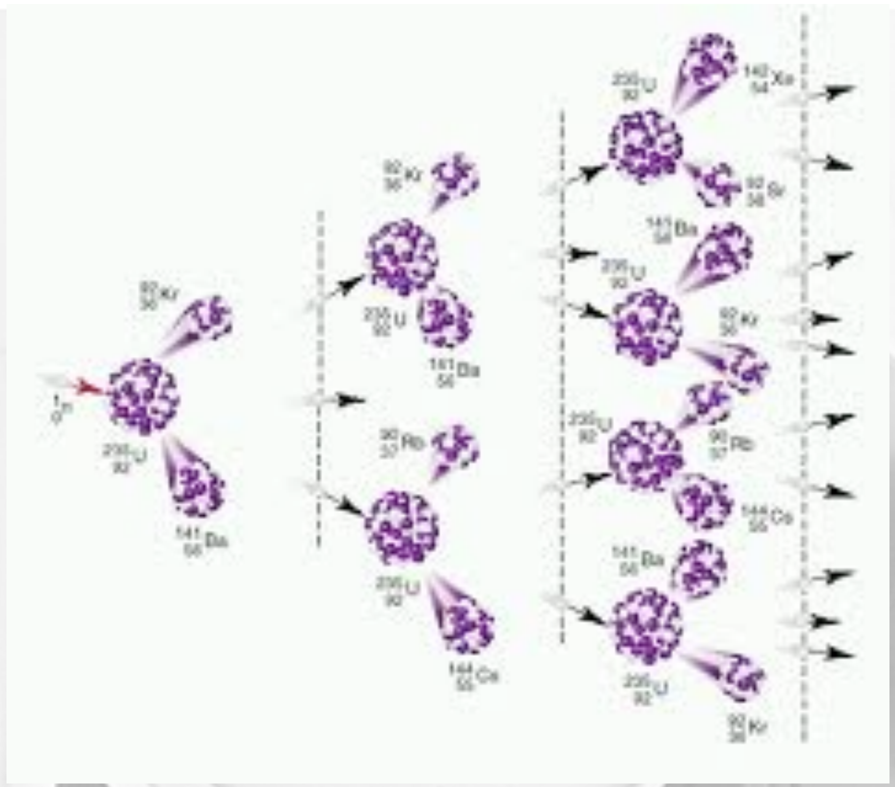
It’s important to remember to climb out and look at different canyons

Because there is great beauty out there!

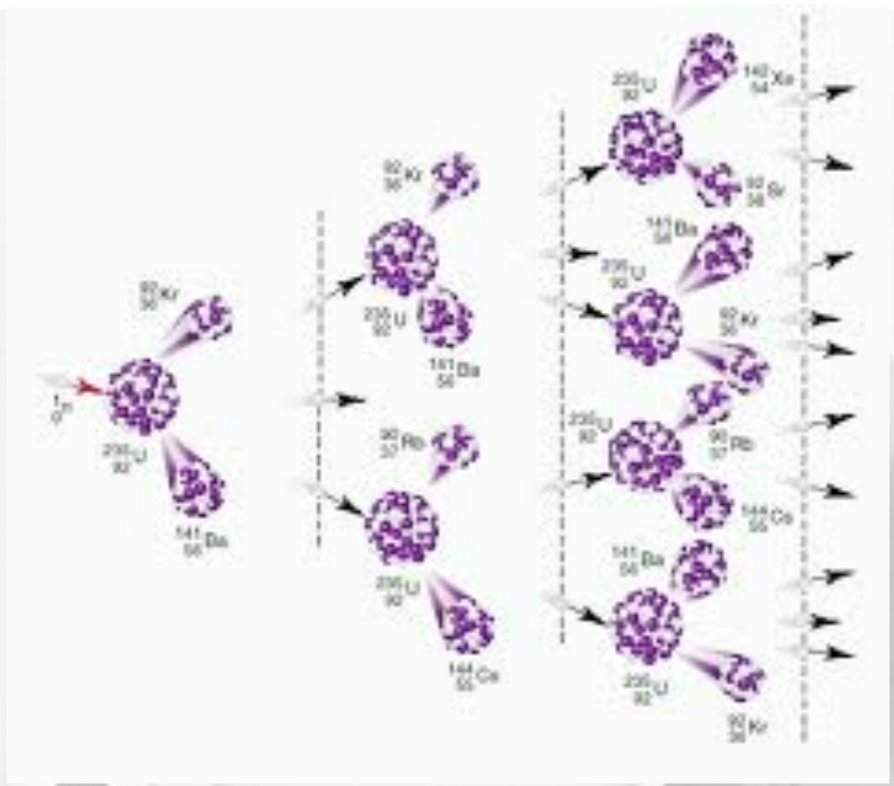
A quick look at how am I wired?

I Played with clothespins and watched TV

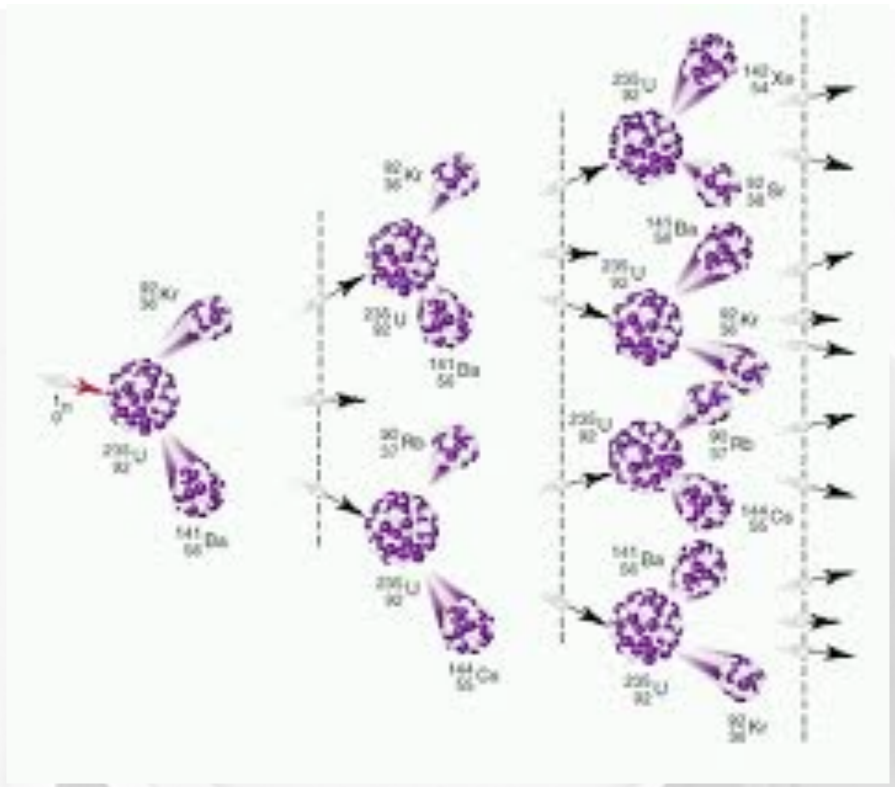
I set off chain reactions



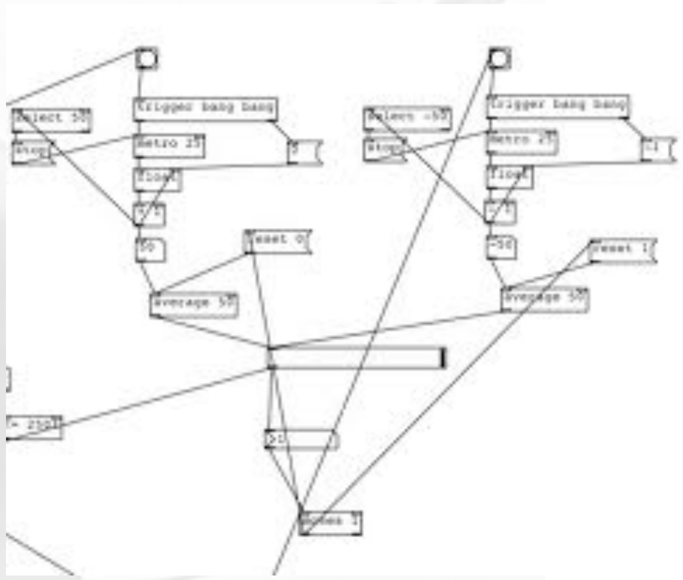
And prepared to be a scientist (fight monsters)



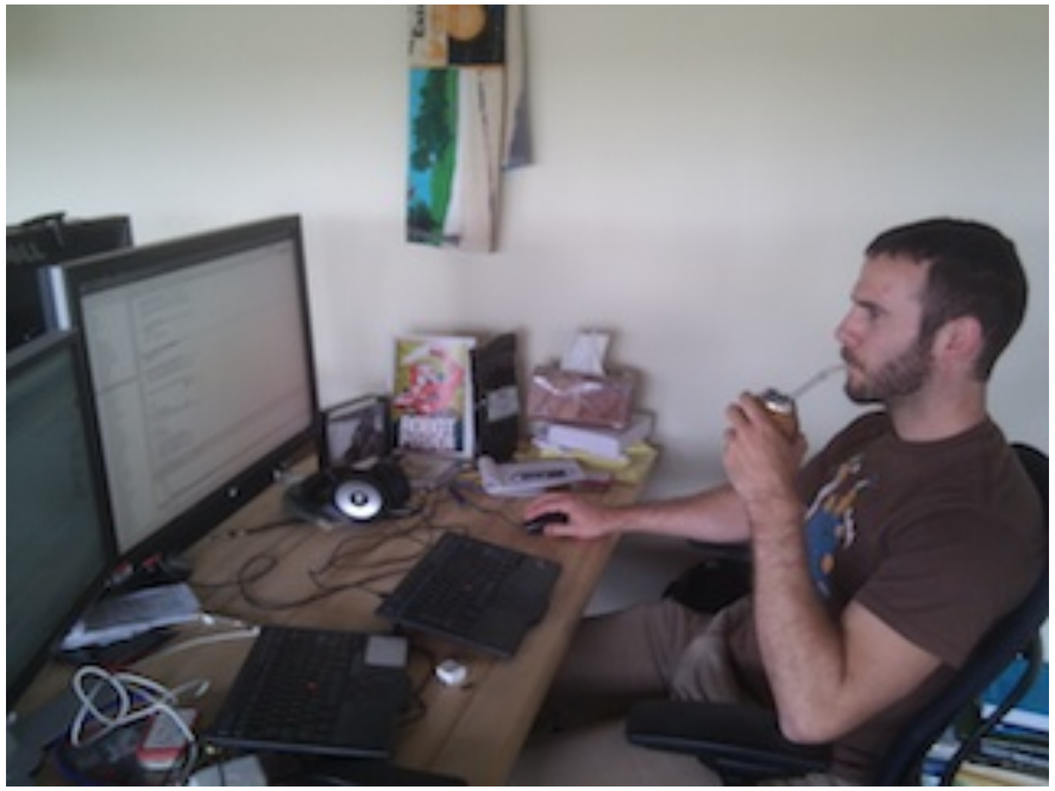
Finally I was drawn to mathematics



What became of such a neglected child?



And look at what I did to my own child!



Alright, about those grooves

Let us look a something beautiful from our past

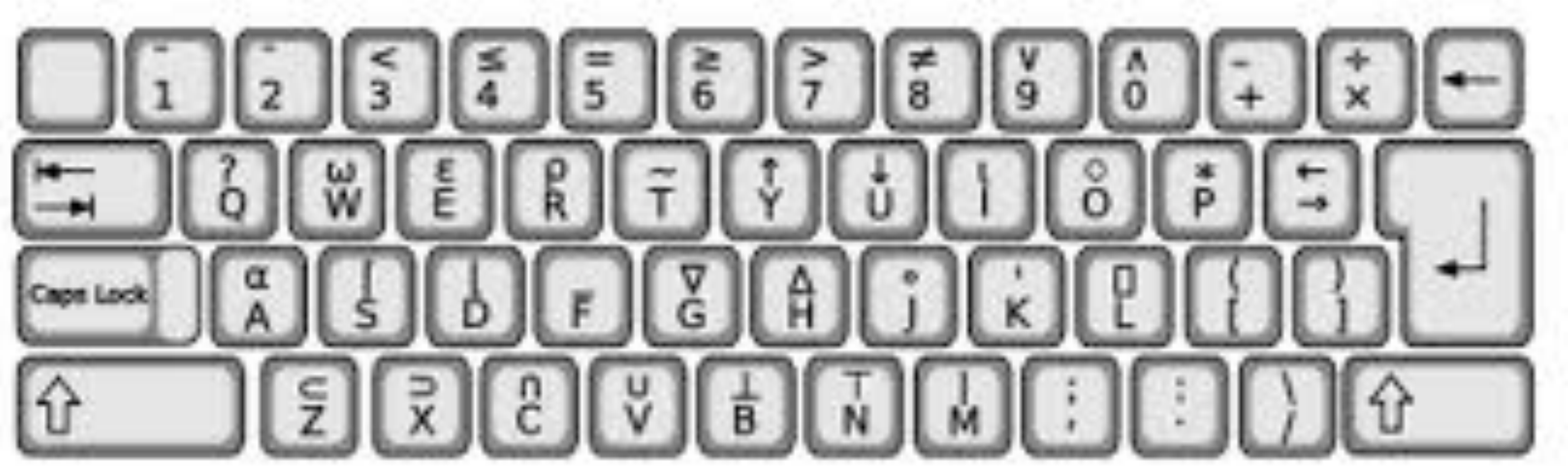
Iverson Notation and APL

- Looked for a better notation for math
- Spent years on a paper design
- Wrote a wonderful book
- Didn't get tenure
- At IBM with Adin Falkoff created an executable math notation called "APL"

ORACLE

Quick Overview

- Had hieroglyphic symbols
- Its own Selectric print head
- Its own keyboard
- No precedence rules for functions (just too many)
- Right to left evaluation
- Workspace
- Operate on multi-dimensional arrays



ORACLE

Functions and Operators

- Functions defined on scalars
- Operators defined on functions
- Extended to Arrays in Four Ways
  - element-by-element with possible extension of rank
  - reduction
  - inner product
  - outer product

$$\nabla \cdot \mathbf{E} = \frac{\rho}{\varepsilon_0}$$

$$\nabla \cdot \mathbf{B} = 0$$

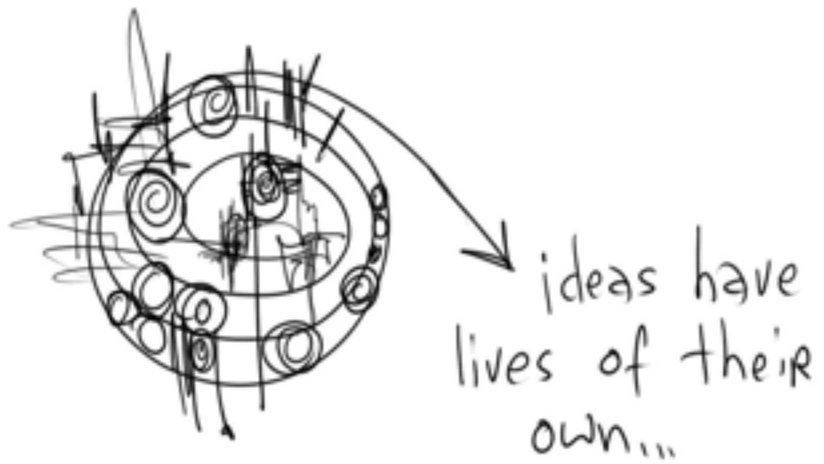
$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t}$$

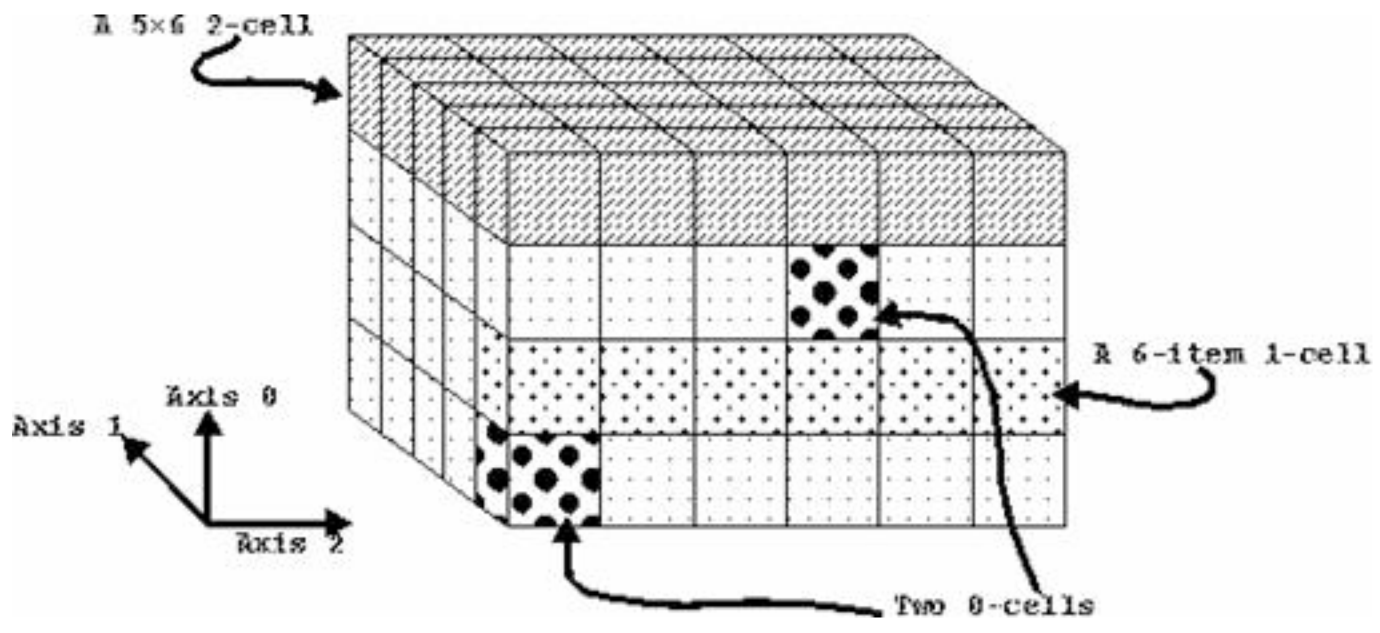
$$\nabla \times \mathbf{B} = \mu_0 \mathbf{J} + \mu_0 \varepsilon_0 \frac{\partial \mathbf{E}}{\partial t}$$

ORACLE

APL's Great Idea

- Noun rank combines with verb rank
- Frames, Items, and Cells





APL Performance

- Interpreted
- Lots of Optimized primitives
- The overhead of interpreted code was low relative to time spent in primitives (total = setup + execution)
  - setup ~ 2.5 milliseconds
  - execute time for scalar operation ~ 50 to 250 microseconds
  - ~ 1000 element arrays
  - ~ (2500 + 150000) * 10^-6
- Ran on Time Share system (50 users on IBM 360)
- Performance on iPhone is amazing

What Ever Happened to APL

- APL grew rapidly in the 1970s and declined in 1980s
- Lots of use in Statistics, Actuarial, and Financial
- Array Languages are still somewhat popular
- Has descendants: A+, J, K, Q
- Influenced:
  - Fortran 90
  - MATLAB
  - R
  - MSFT Accelerator
  - Intel’s Array Building Blocks

State of the Art in APL is J

- It can look a bit “alien”
- Encourages programming without loops
- Encourages programming without variables
- My 10 by 3 working subset of R

am =: amean =: +/ % #	gm =: gmean =: # %: */	hm =: hmean =: % @ am @: %
dev =: - amean	ss =: +/ @: *: @ dev	var =: ssp % <:@#ssp % <:@#
sd =: %: @ var	fr=: +/"1 @ (=/)	frtab=: [,.fr
io=: [:<:[:+/[</]	midpts=: [:-:2:+/\]	FR=: [: +/"1 {@[=/]
cfr=: i.@(<:@$@[) fr io	cfrtab=: midpts@[,.cfr	EACH=: &>
bars=: #&'*' EACH @ fr	barchart=: (": EACH @ [) ,. [: ' '&,. bars	vbarchart=: [: I. [: I: [: '^'&,.bars
barchartv =: (": EACH @ [) I.@I:@,. [: '-'&,. bars	stem=: 10&* @ <. @ %&10	leaf=: 10&I
SLtab=: ~.@stem ;"0 stem </. leaf	stemfrtab=: ~.@stem ,. stem #/. leaf	midindices=: (<.,>.)@-:@<:@#
Q2=: median=:[: am midindices { sort	Q1=: [: median] #~ median >]	Q3=: [: median] #~ median <]
five=: (<./,Q1,Q2,Q3,>./)	ArrayMaker =: ". ;. _2	mp =: dot =: +/ . *

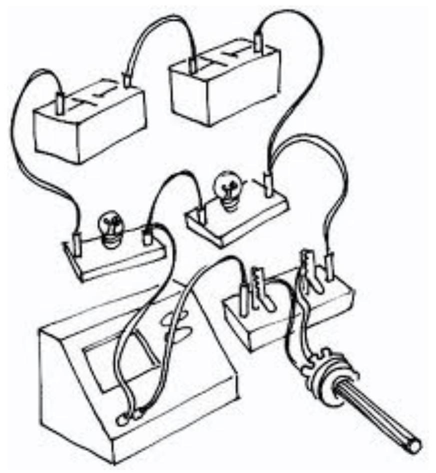
Optimizations: It can be fast

Phil Abrams APL Machine (1970)

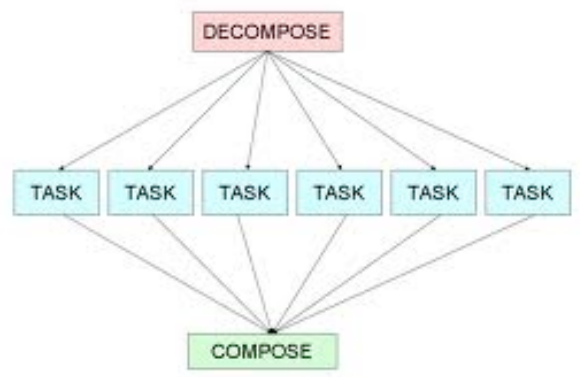
- High Level machine appropriate for APL1
- Drag-along
  - Defer the process of evaluation of operands and operators as long as possible (Lazy Evaluation)
  - take(3, 2 * -V)
  - A+B+C+D
- Beating
  - The transformation of code to reduce the amount of data manipulation during expression evaluation
- Envisioned “multiple copies of key evaluation algorithms working simultaneously on different parts of an expression

ORACLE

Most APL Primitives can be Parallel



- Willhoft-1991: Most APL2 Primitives Can Be Parallelized
  - “APL2 exhibits a high degree of parallelism”
  - “94 of the 101 primitives APL2 operations can be implemented in parallel”
  - “40-50 percent of the code in “real” applications is parallel code”
- Bernecky-1993
  - Good Properties for parallelism:
    - array orientation
    - adverbs and conjunctions
    - consistent syntax and semantics

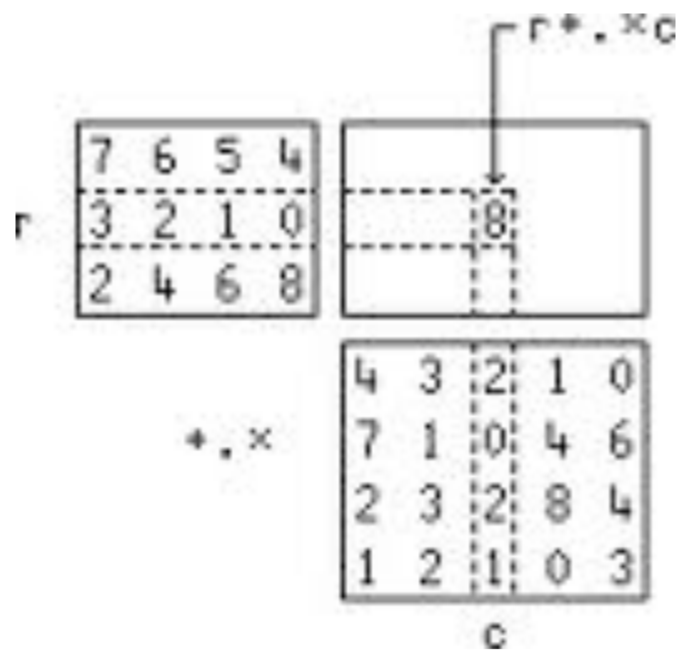


ORACLE

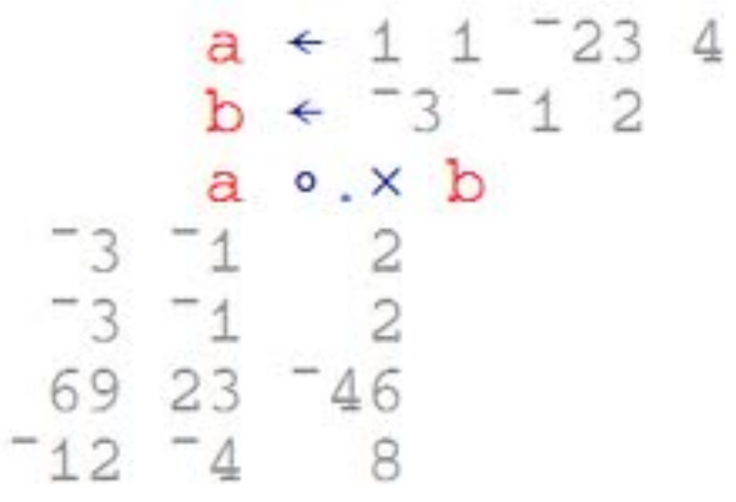
Training Our Thinking

Very useful for training data-para thinking

Inner Product

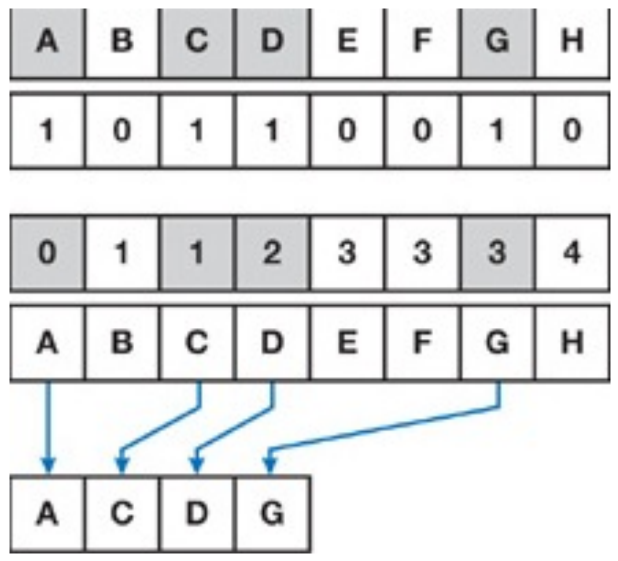


Outer Product



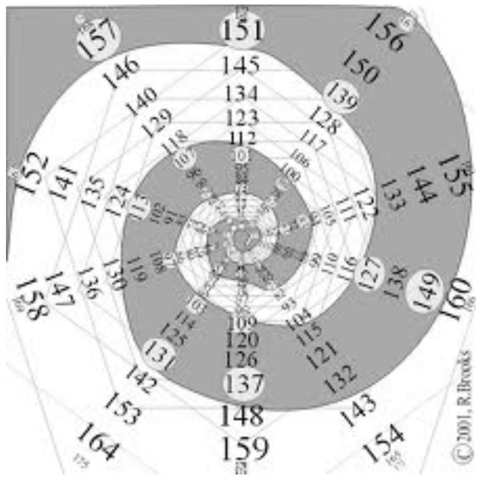
```
    a ← 1 1 ¯23 4
    b ← ¯3 ¯1 2
    a ∘.× b
¯3  ¯1    2
¯3  ¯1    2
69  23  ¯46
¯12 ¯4    8
```

Compress and Scan



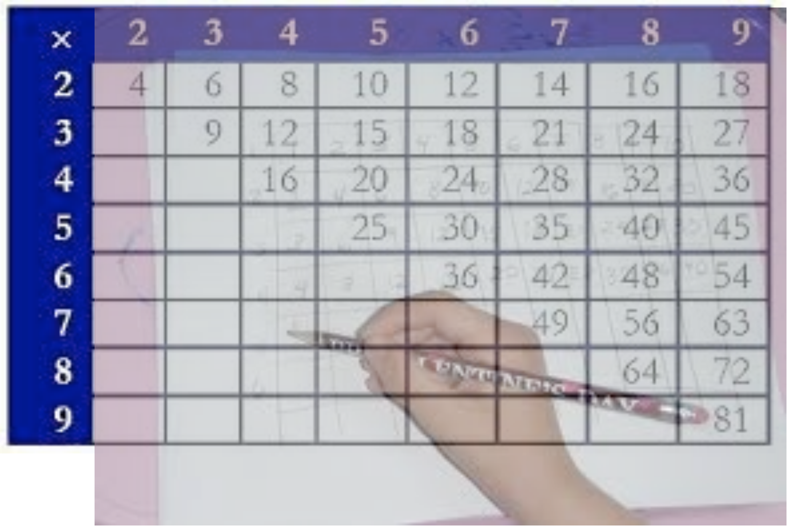
Monday, October 1, 12

Array Programming Encourages Beautiful Loopless Big Thinking

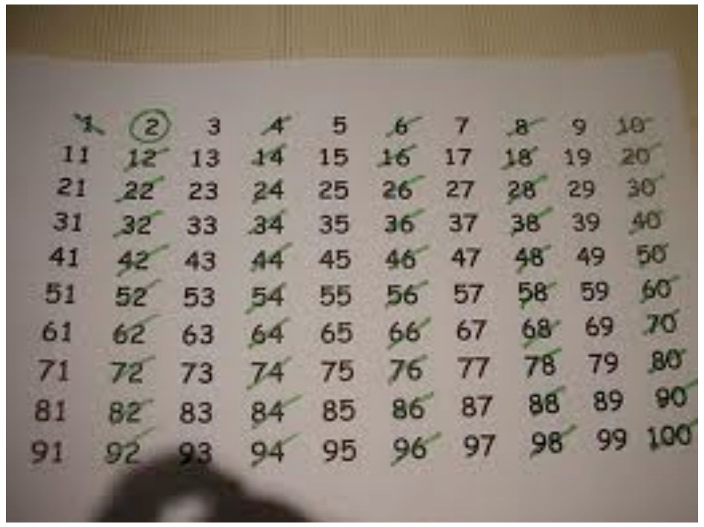


Create a List of Prime Numbers

```
(~ R ∈,R ∘ . × R)/R ← 1 ↓ ιR
```



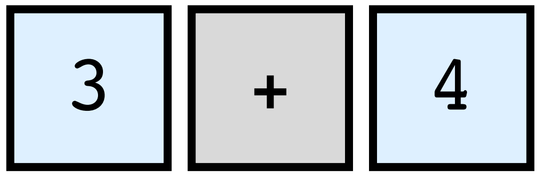
×	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18
3		9	12	15	18	21	24	27
4			16	20	24	28	32	36
5				25	30	35	40	45
6					36	42	48	54
7						49	56	63
8							64	72
9								81



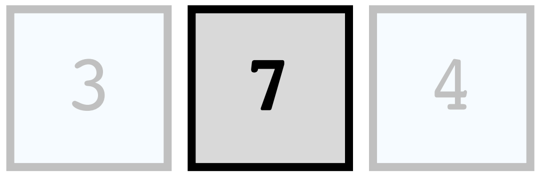
1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

A Taste of Array Programming

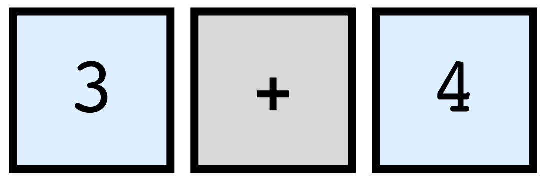
Simple Arithmetic



Simple Arithmetic



Simple Arithmetic

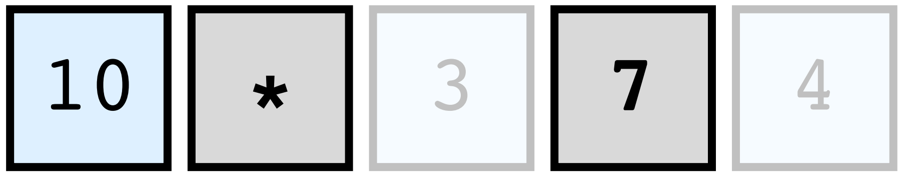


7

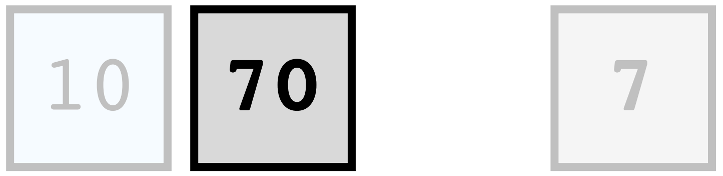
Evaluate right to left

`10` `*` `3` `+` `4`

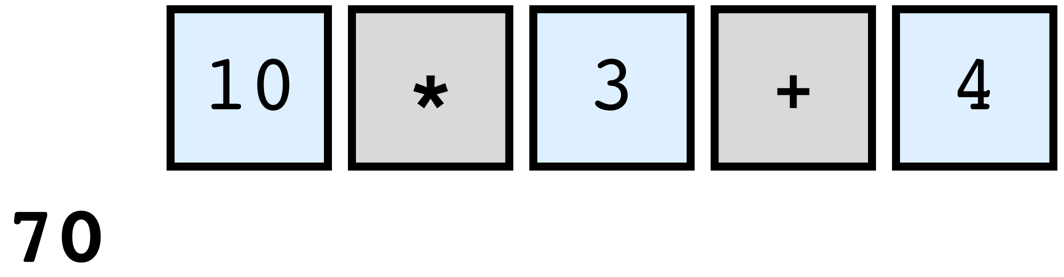
Evaluate right to left



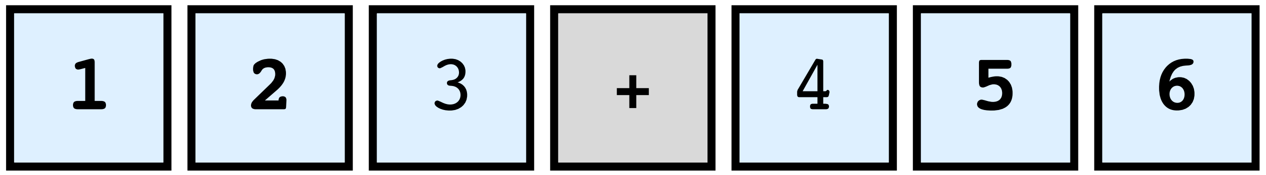
Evaluate right to left



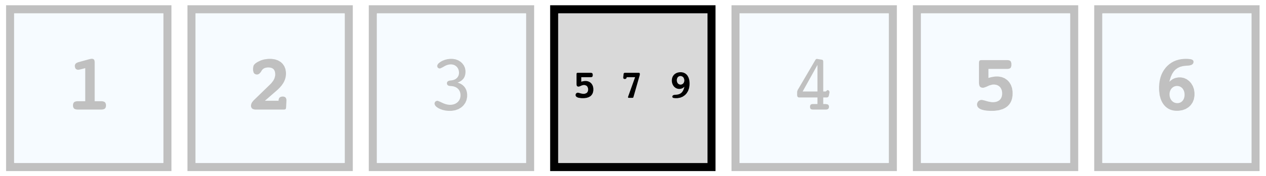
Evaluate right to left



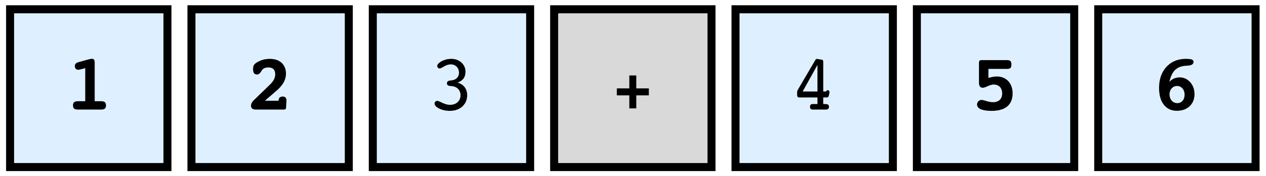
Extends to arrays



Extends to arrays

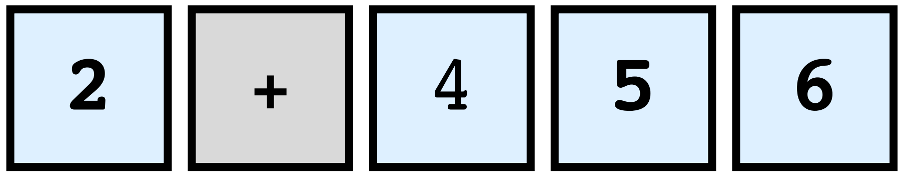


Extends to arrays

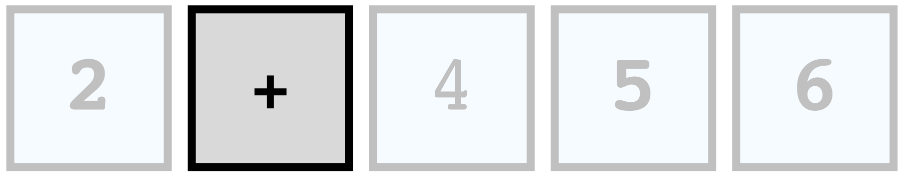


5 7 9

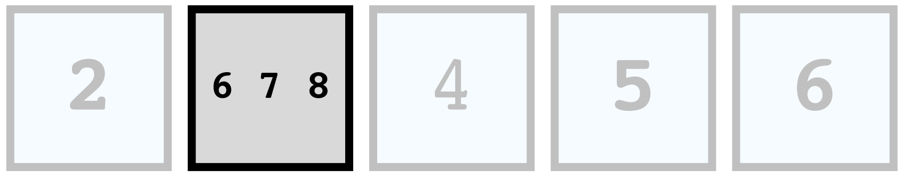
Mix scalars and arrays



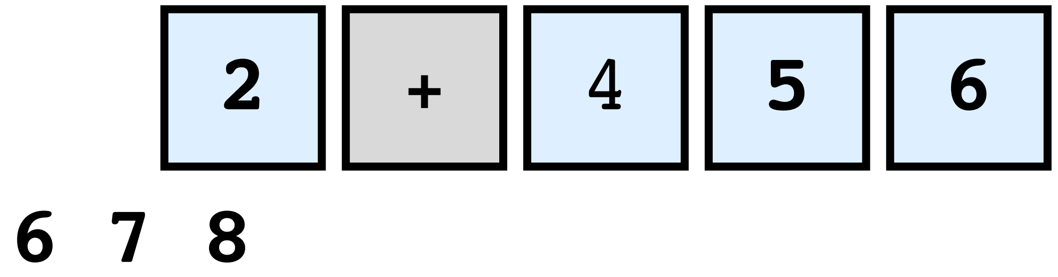
Mix scalars and arrays



Mix scalars and arrays



Mix scalars and arrays



Uniform

Logicals

=	<	>
≤	≥	≠

Arithmetics

+	×	÷
-	|	^
⌈	⌊	⊛

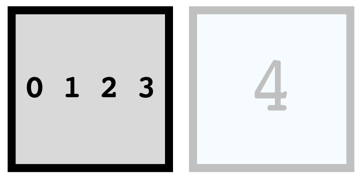
Generate Integers

$\boxed{\iota}\ \boxed{4}$

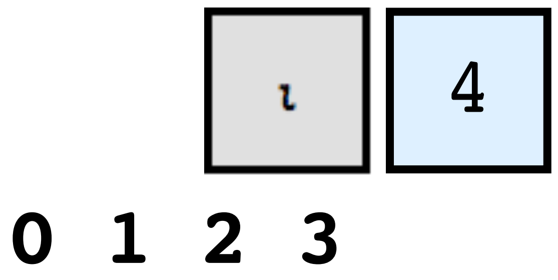
Generate Integers

ι 4

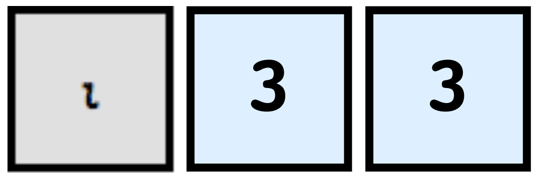
Generate Integers



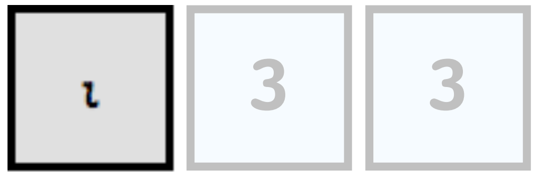
Generate Integers



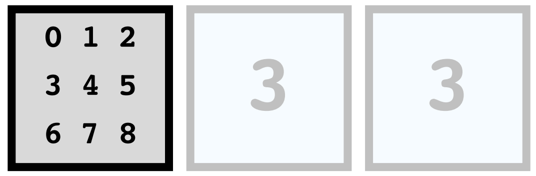
Generate arrays of integers



Generate arrays of integers



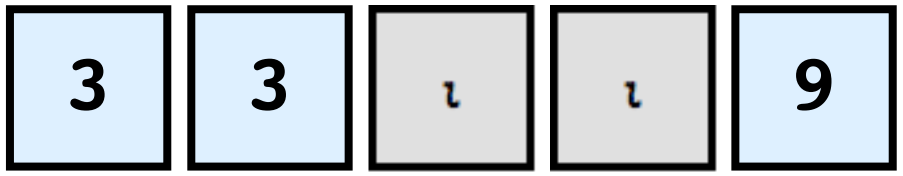
Generate arrays of integers



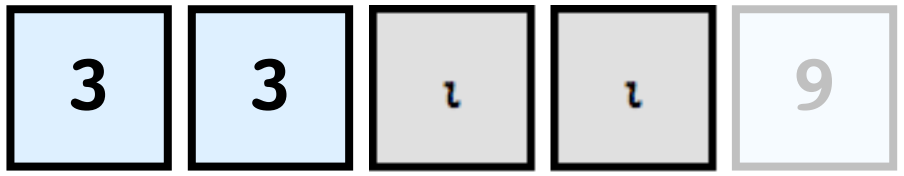
Generate arrays of integers

```
    ι 3 3
0 1 2
3 4 5
6 7 8
```

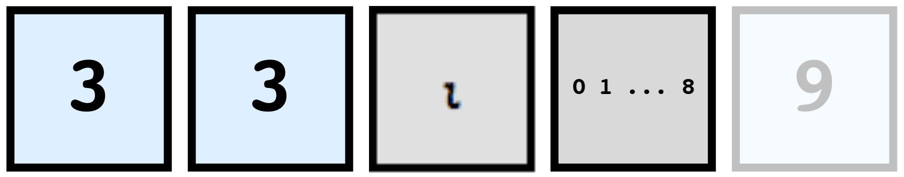
Reshape Arrays



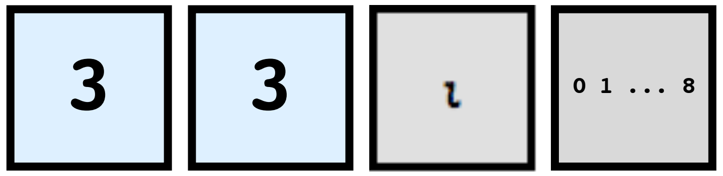
Reshape Arrays



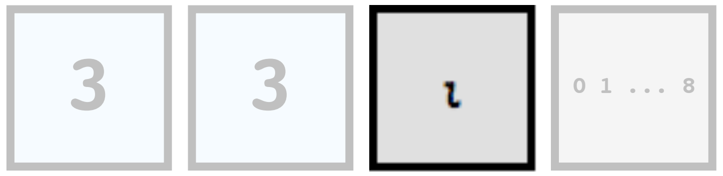
Reshape Arrays



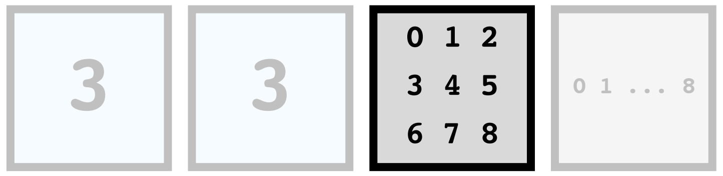
Reshape Arrays



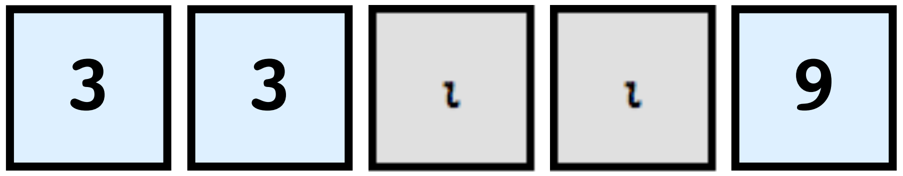
Reshape Arrays



Reshape Arrays



Reshape Arrays

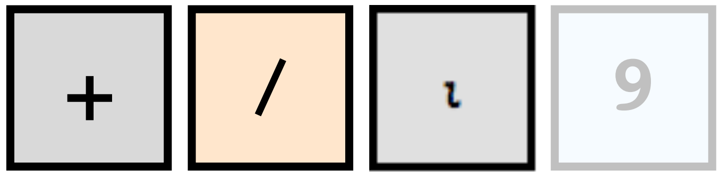


```
0  1  2
3  4  5
6  7  8
```

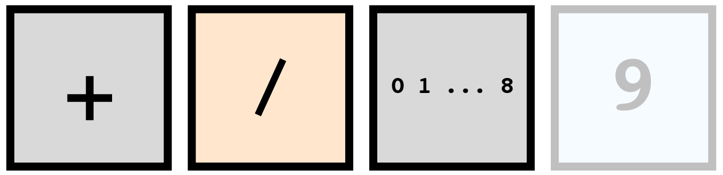
Operators

```
+ / ι 9
```

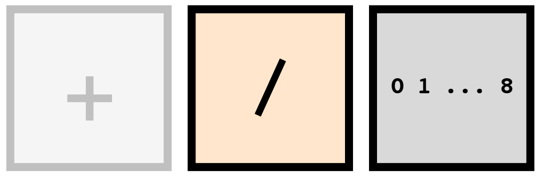
Operators



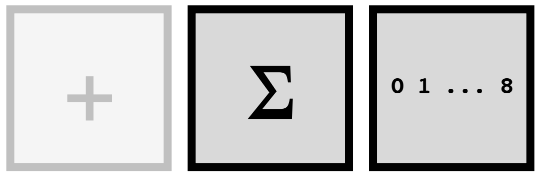
Operators



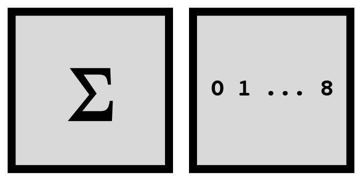
Operators



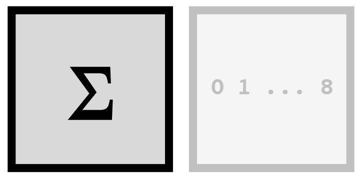
Operators



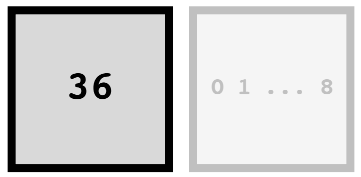
Operators



Operators



Operators



Operators

```
 + / ι 9
36
```

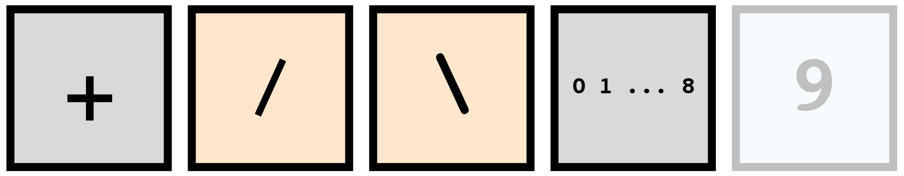
Prefix Scan Operator

```
+ / \ ⍳ 9
```

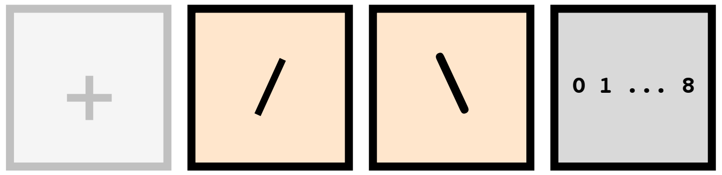
Prefix Scan Operator

+ / \ ι 9

Prefix Scan Operator



Prefix Scan Operator



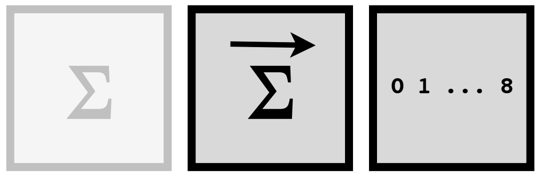
Prefix Scan Operator

$\boxed{+}\ \boxed{\Sigma}\ \boxed{\backslash}\ \boxed{\texttt{0 1 ... 8}}$

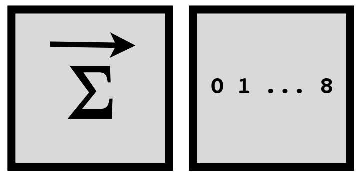
Prefix Scan Operator

Σ \ `0 1 ... 8`

Prefix Scan Operator



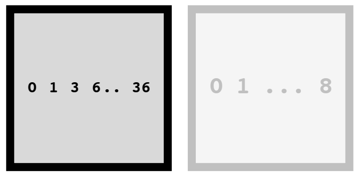
Prefix Scan Operator



Prefix Scan Operator

$\overrightarrow{\Sigma}$ 0 1 ... 8

Prefix Scan Operator



Prefix Scan Operator

```
+ / \ ι 9
```

```
0  1  3  6  10  15  21  28  36
```

Prefix Scan Operator

```
+ / \ ι 9
```

```
0  1  3  6  10  15  21  28  36
```

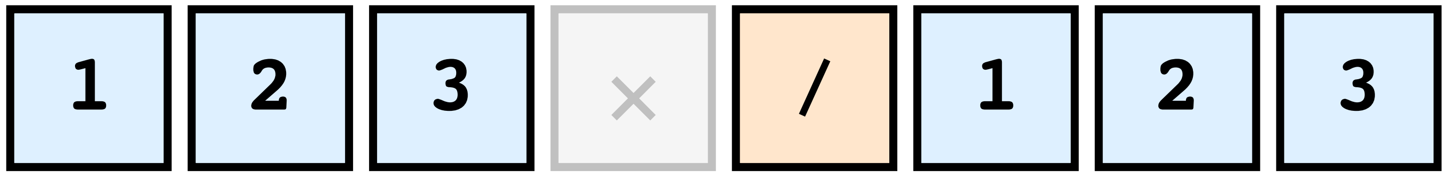
Example

```
x ← 6 7 8
u ← 1 0 1 0 0 1 0
+/\u
1 1 2 2 2 3 3
u * +/\u
1 0 2 0 0 3 0
(u * +/\u)▯ 0,x
6 0 7 0 0 8 0
```

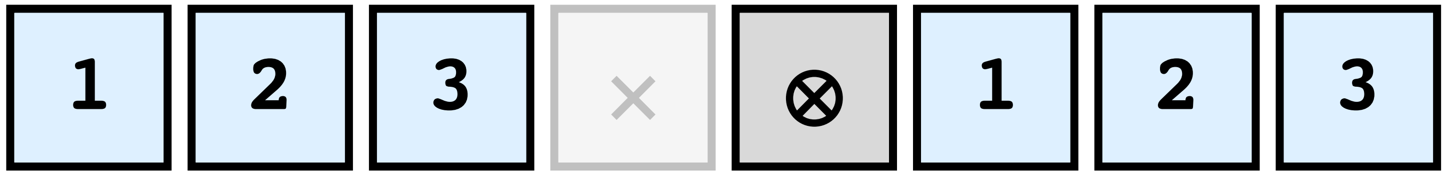
Outer Product Operator

1 2 3 × / 1 2 3

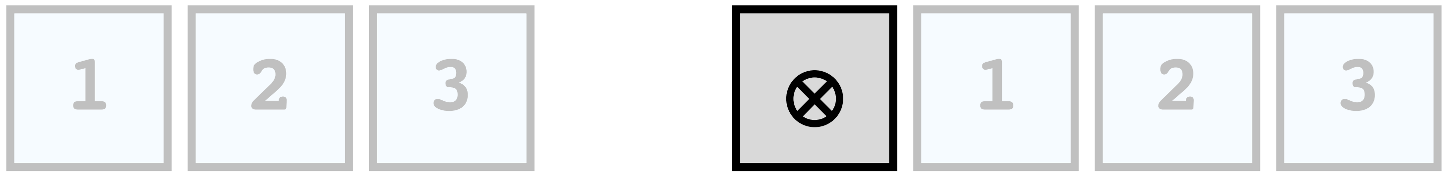
Outer Product Operator



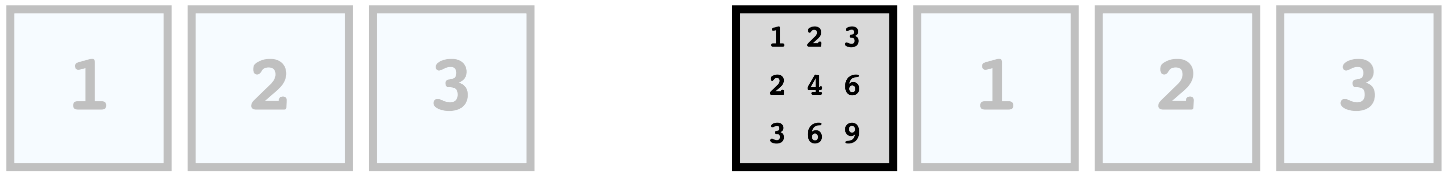
Outer Product Operator



Outer Product Operator



Outer Product Operator

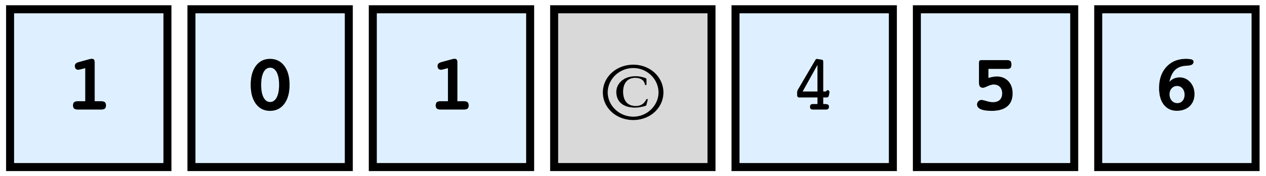


Outer Product Operator

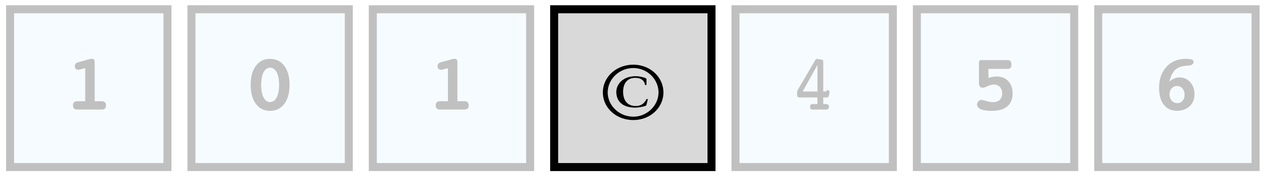
```
1 2 3 × / 1 2 3
```

```
1  2  3
2  4  6
3  6  9
```

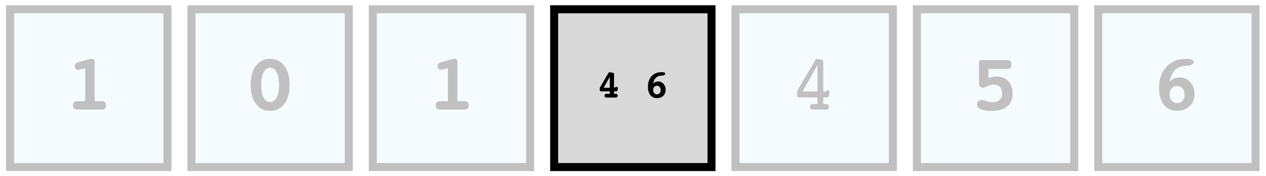
Compression



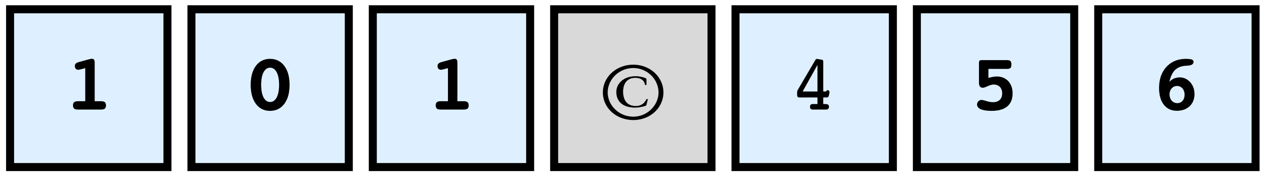
Compression



Compression



Compression

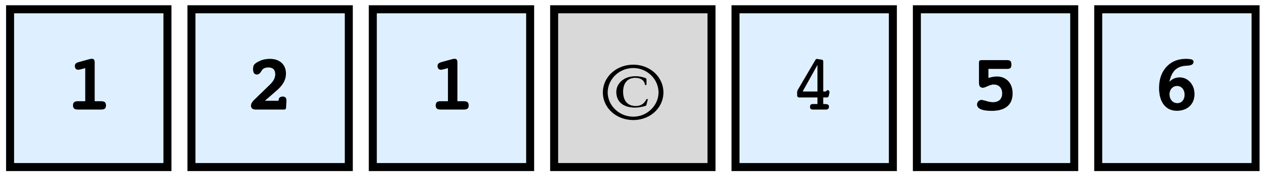


```
4 6
```

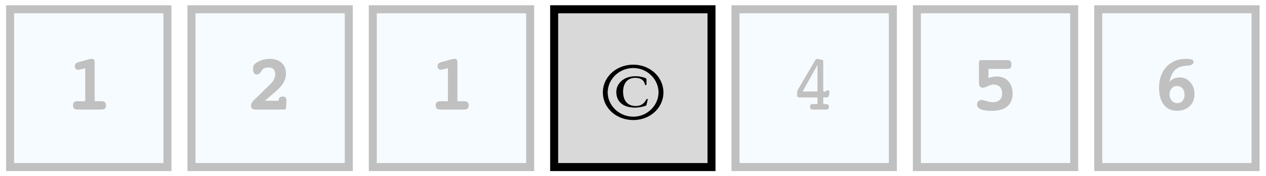
Example

```
(0 = 2 | 4 5 6) © 4 5 6
4 6
```

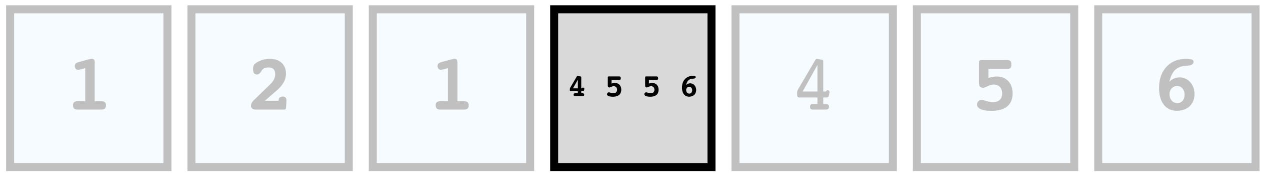
Compression is copy



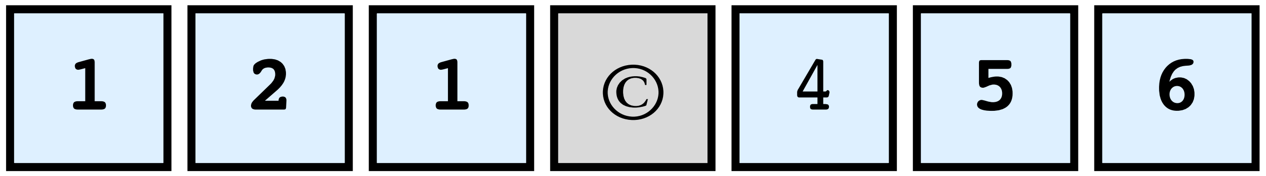
Compression is copy



Compression is copy



Compression is copy

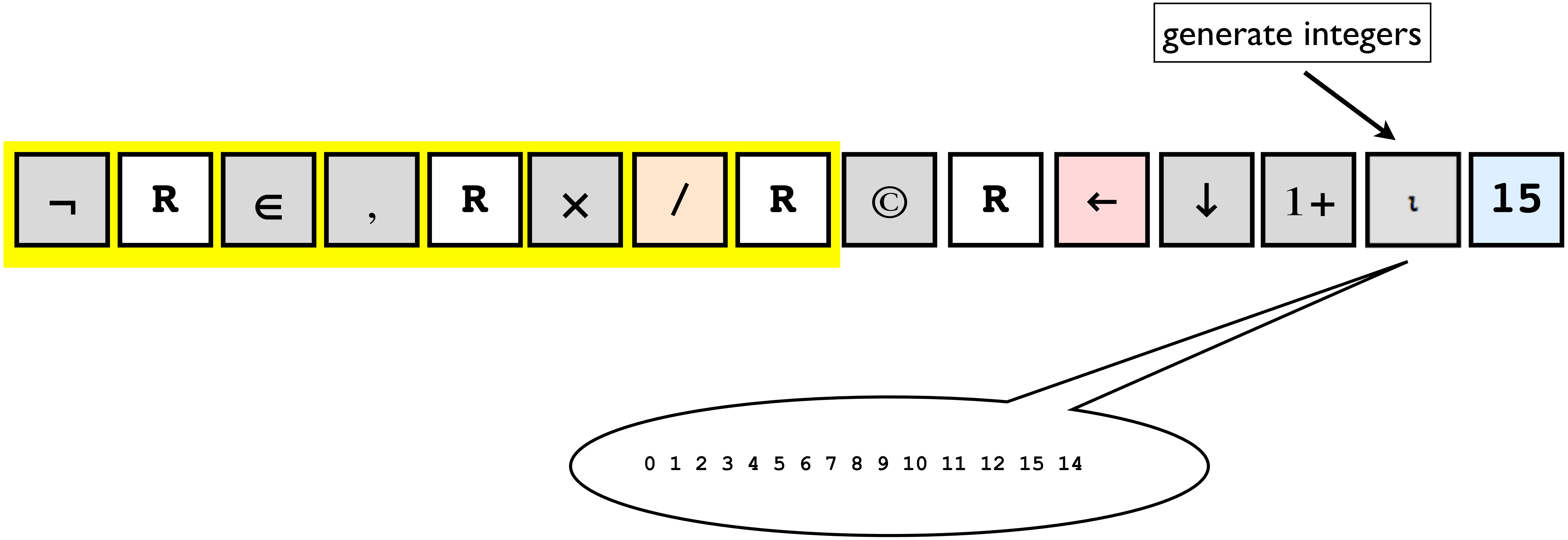


4 5 5 6

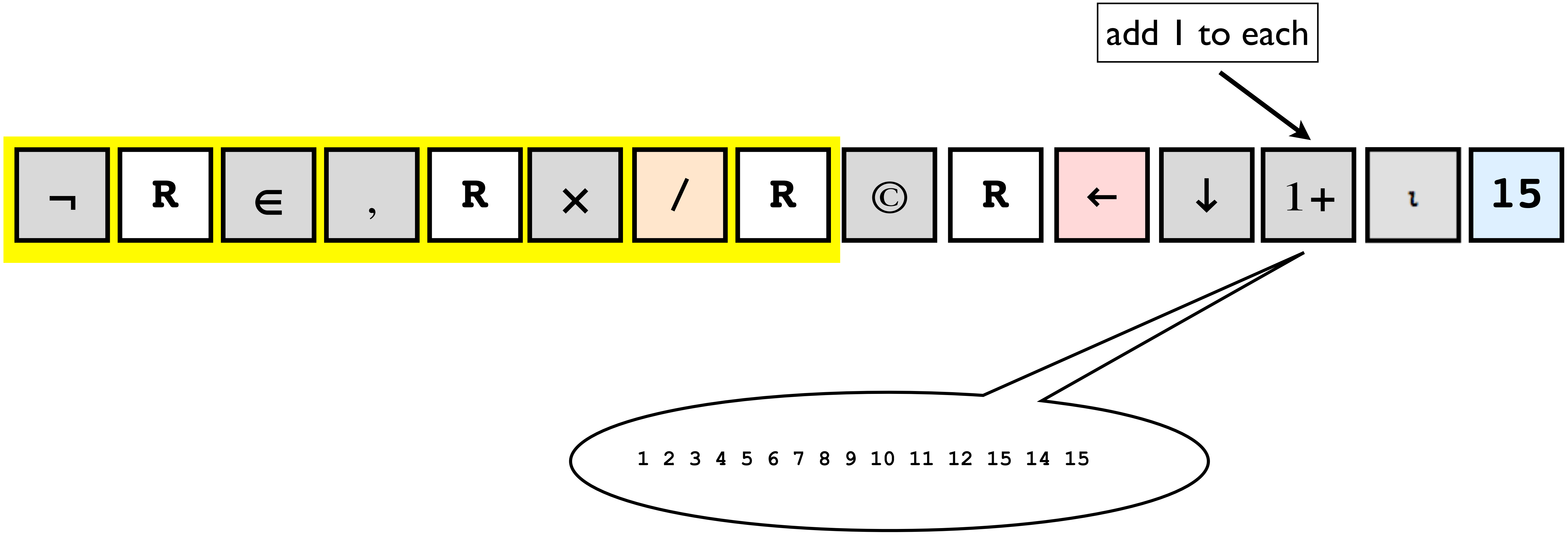
First Primes

```
¬ R ∈ , R × / R © R ← ↓ 1+ ι 15
```

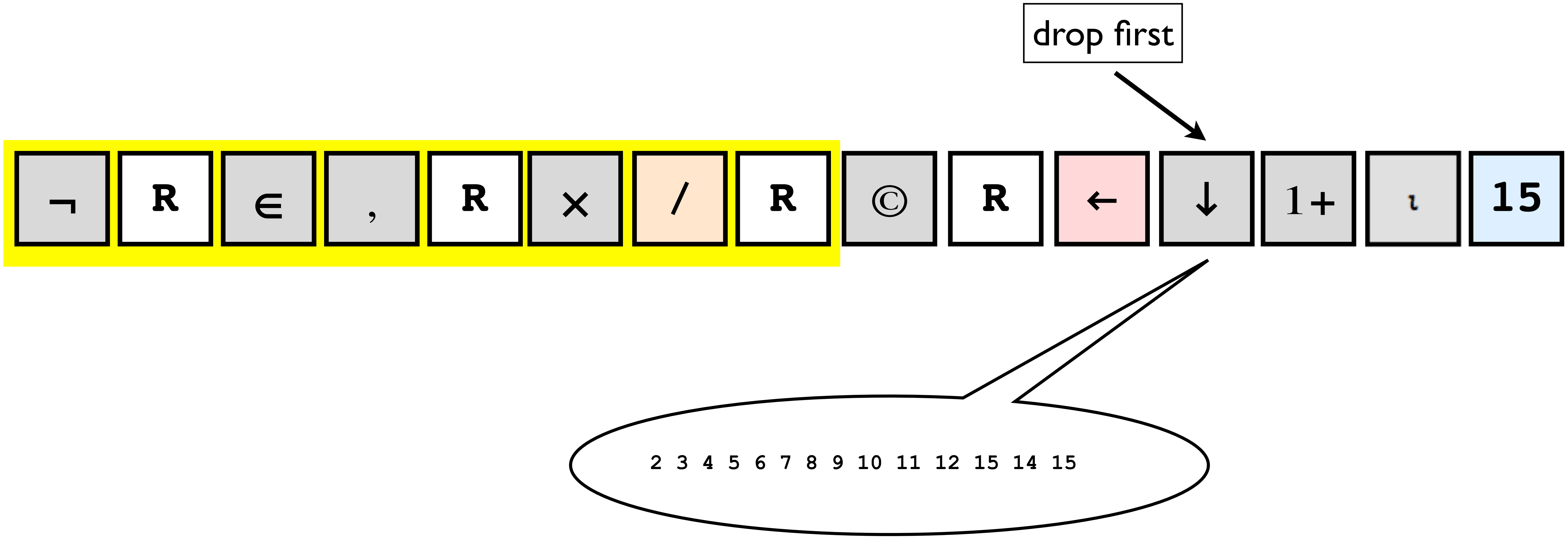
First Primes



First Primes



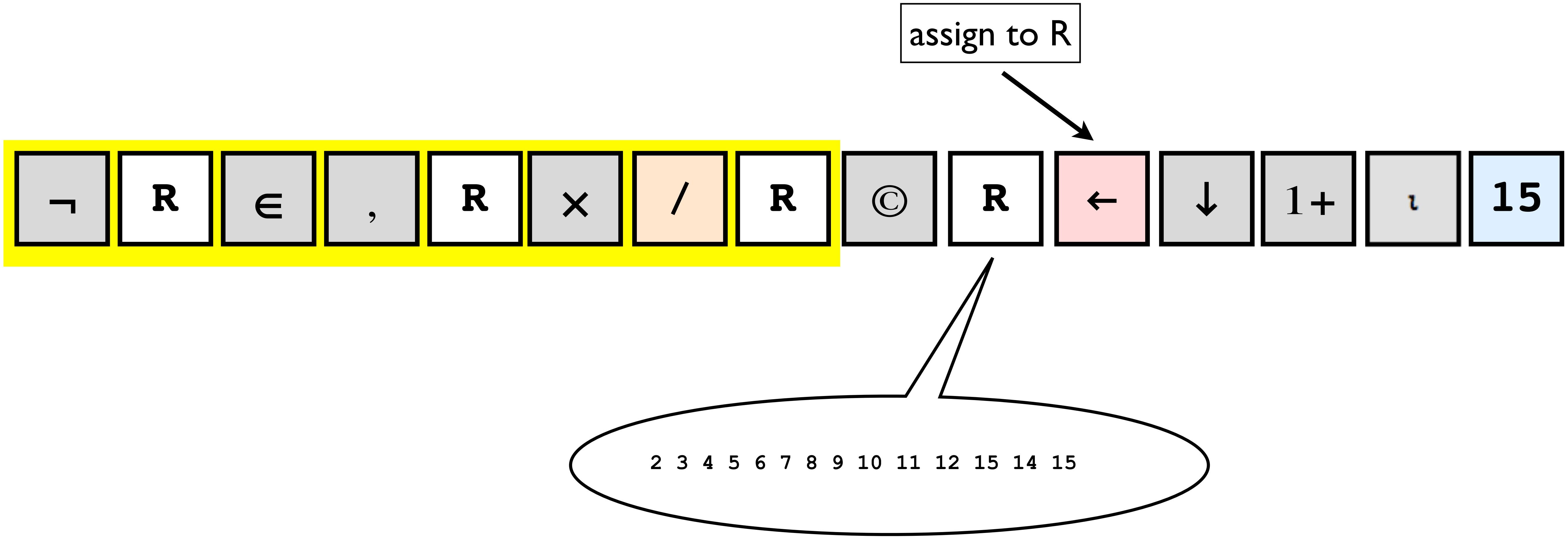
First Primes



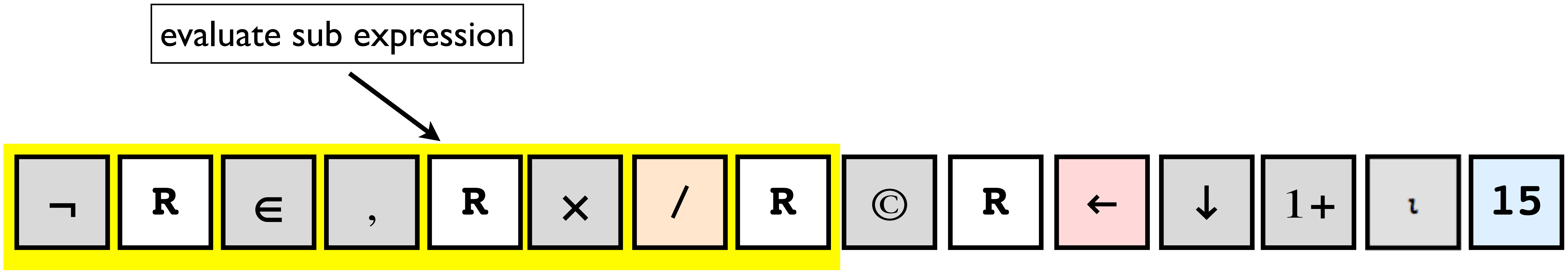
```
¬ R ∈ , R × / R © R ← ↓ 1+ ι 15
```

```
2 3 4 5 6 7 8 9 10 11 12 15 14 15
```

First Primes



First Primes



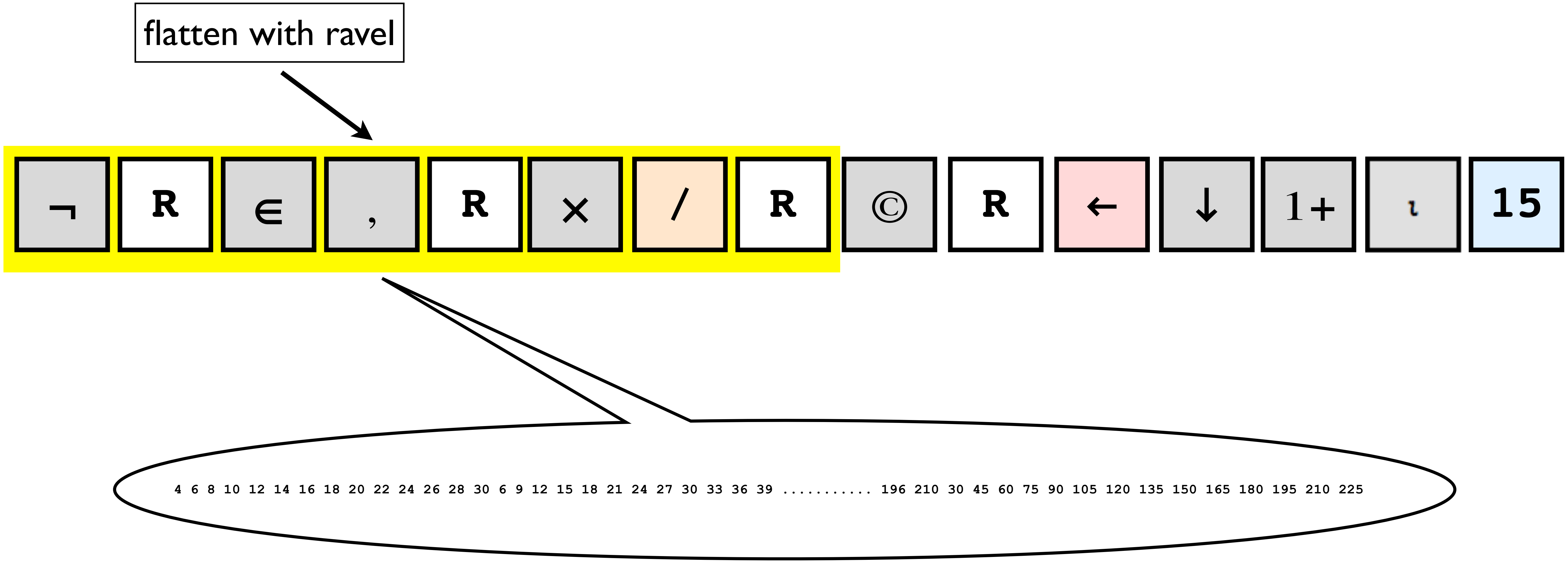
First Primes

up the dimensionality with outer product

```
¬ R ∈ , R × ⊗ / R © R ← ↓ 1+ ι 15
```

```
 4  6  8 10 12  14  16  18  20  22  24  26  28  30
 6  9 12 15 18  21  24  27  30  33  36  39  42  45
 8 12 16 20 24  28  32  36  40  44  48  52  56  60
10 15 20 25 30  35  40  45  50  55  60  65  70  75
12 18 24 30 36  42  48  54  60  66  72  78  84  90
14 21 28 35 42  49  56  63  70  77  84  91  98 105
16 24 32 40 48  56  64  72  80  88  96 104 112 120
18 27 36 45 54  63  72  81  90  99 108 117 126 135
20 30 40 50 60  70  80  90 100 110 120 130 140 150
22 33 44 55 66  77  88  99 110 121 132 143 154 165
24 36 48 60 72  84  96 108 120 132 144 156 168 180
26 39 52 65 78  91 104 117 130 143 156 169 182 195
28 42 56 70 84  98 112 126 140 154 168 182 196 210
30 45 60 75 90 105 120 135 150 165 180 195 210 225
```

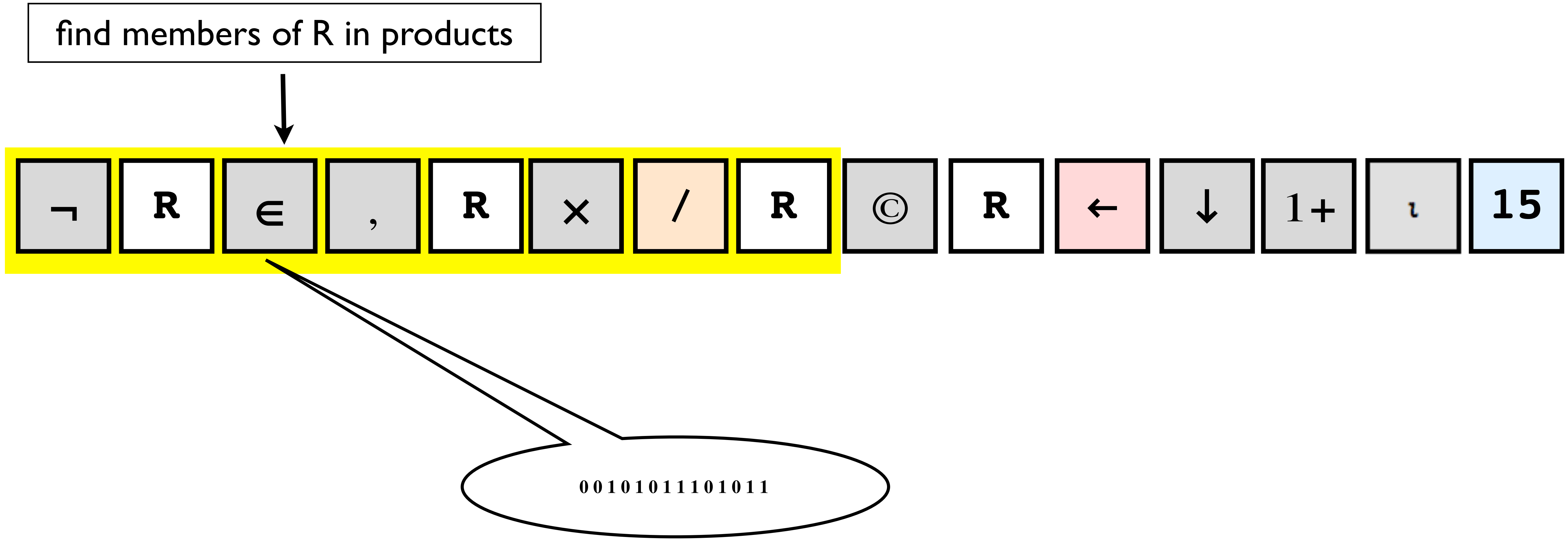
First Primes



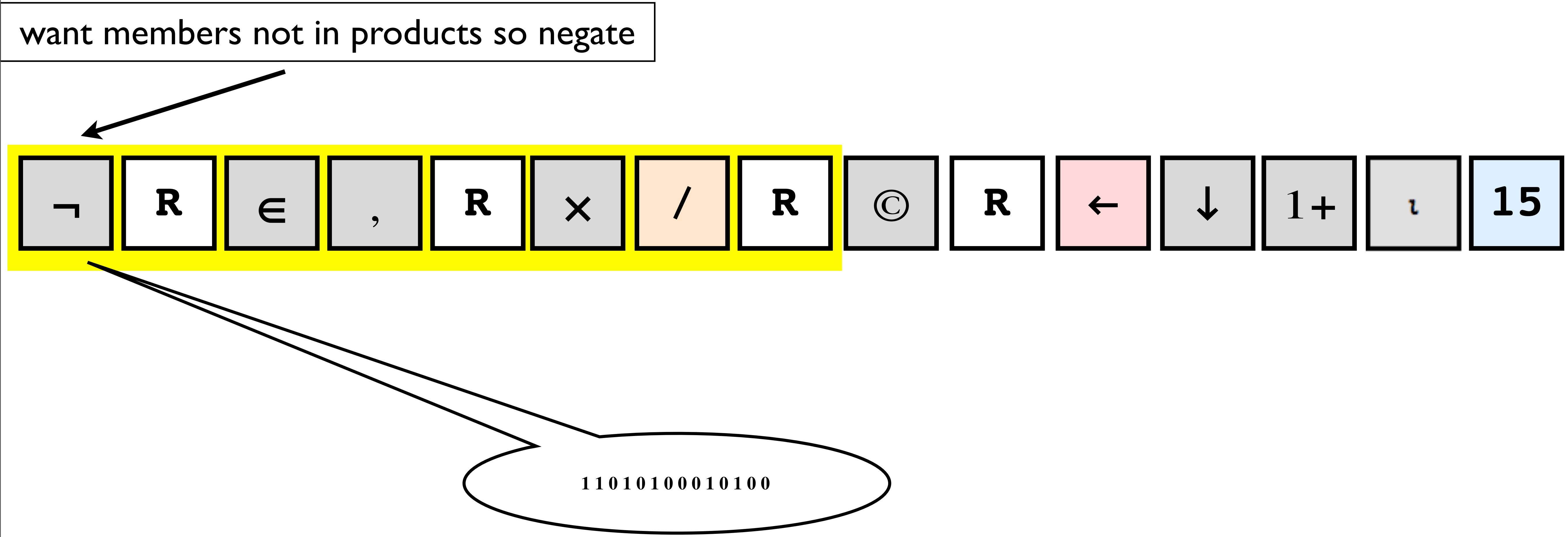
```
¬ R ∈ , R × / R © R ← ↓ 1+ ɩ 15
```

```
4 6 8 10 12 14 16 18 20 22 24 26 28 30 6 9 12 15 18 21 24 27 30 33 36 39 ........... 196 210 30 45 60 75 90 105 120 135 150 165 180 195 210 225
```

First Primes



First Primes

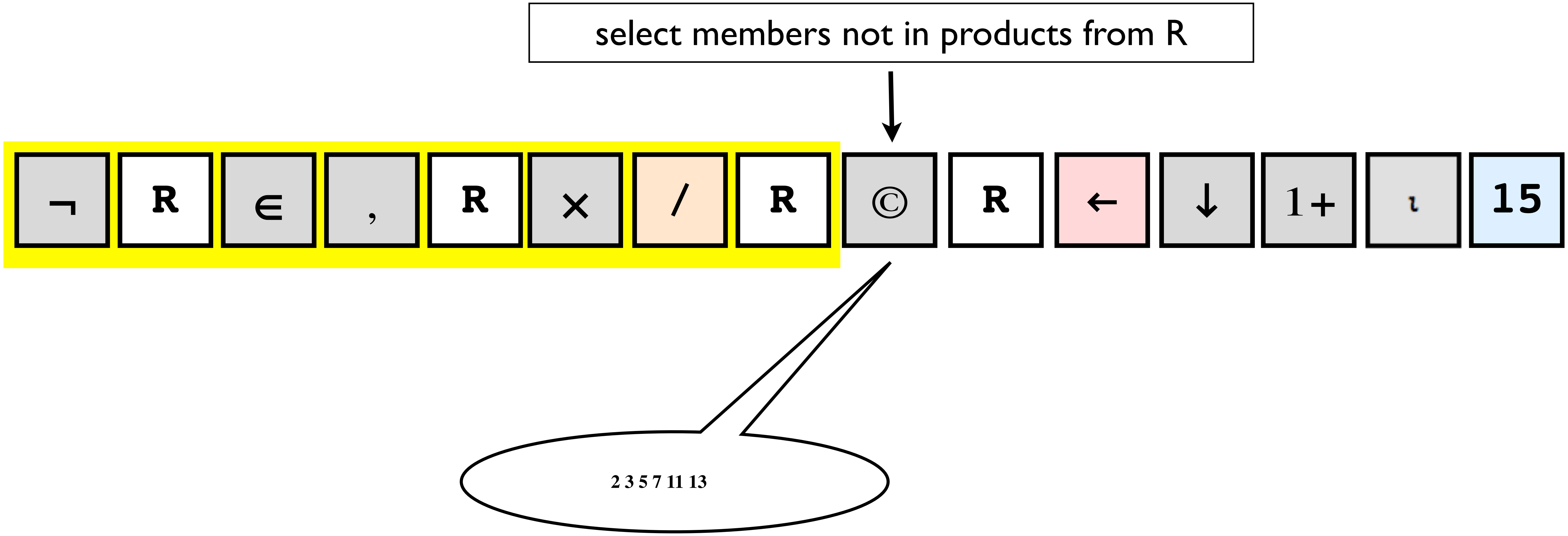


want members not in products so negate

```
¬ R ∈ , R × / R © R ← ↓ 1+ ι 15
```

```
11010100010100
```

First Primes



First Primes

```
¬ R ∈ , R × / R © R ← ↓ 1+ ɩ 15
```

```
2 3 5 7 11 13
```

Examples without variable

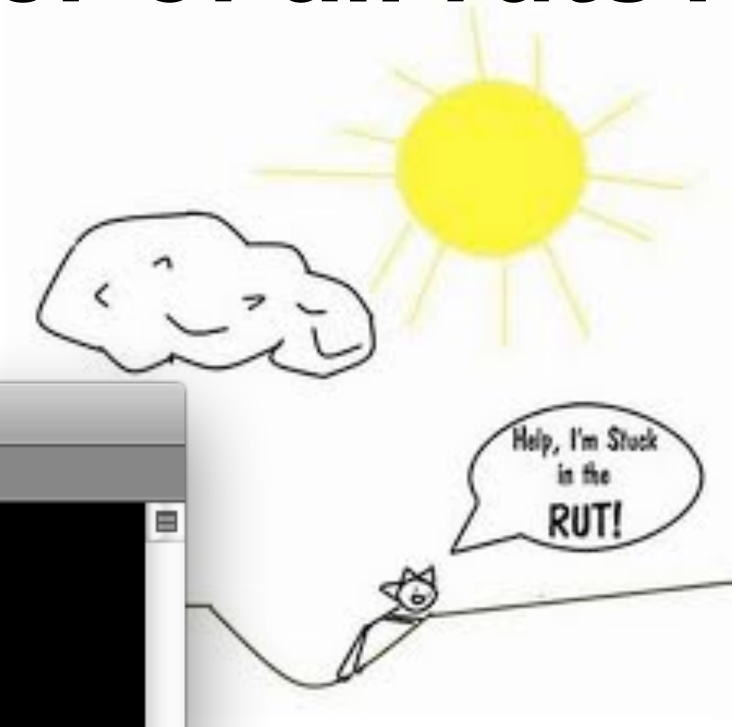
```
((¬∘(⊢ ∈ ,∘(⊢ ×/ ⊢))) © ⊢) ↓ 1+ ɩ 15
or in ascii J:
((-.@:(] e. ,@:(] */ ]))) # ]) }. >: i. 100
```

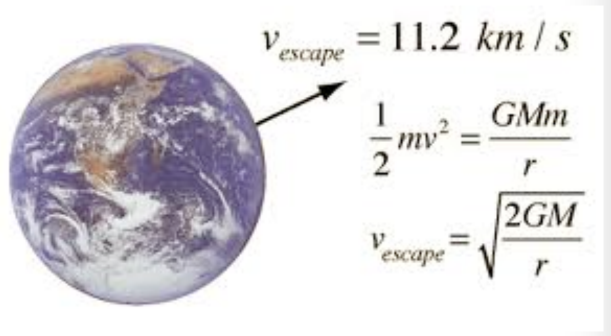
Remember those grooves

I love APL, J and K but...

- It can look a bit “alien”
- Learning it is a bit prickly
- But it really will expand your brain!

But can we escape the mother of all ruts?

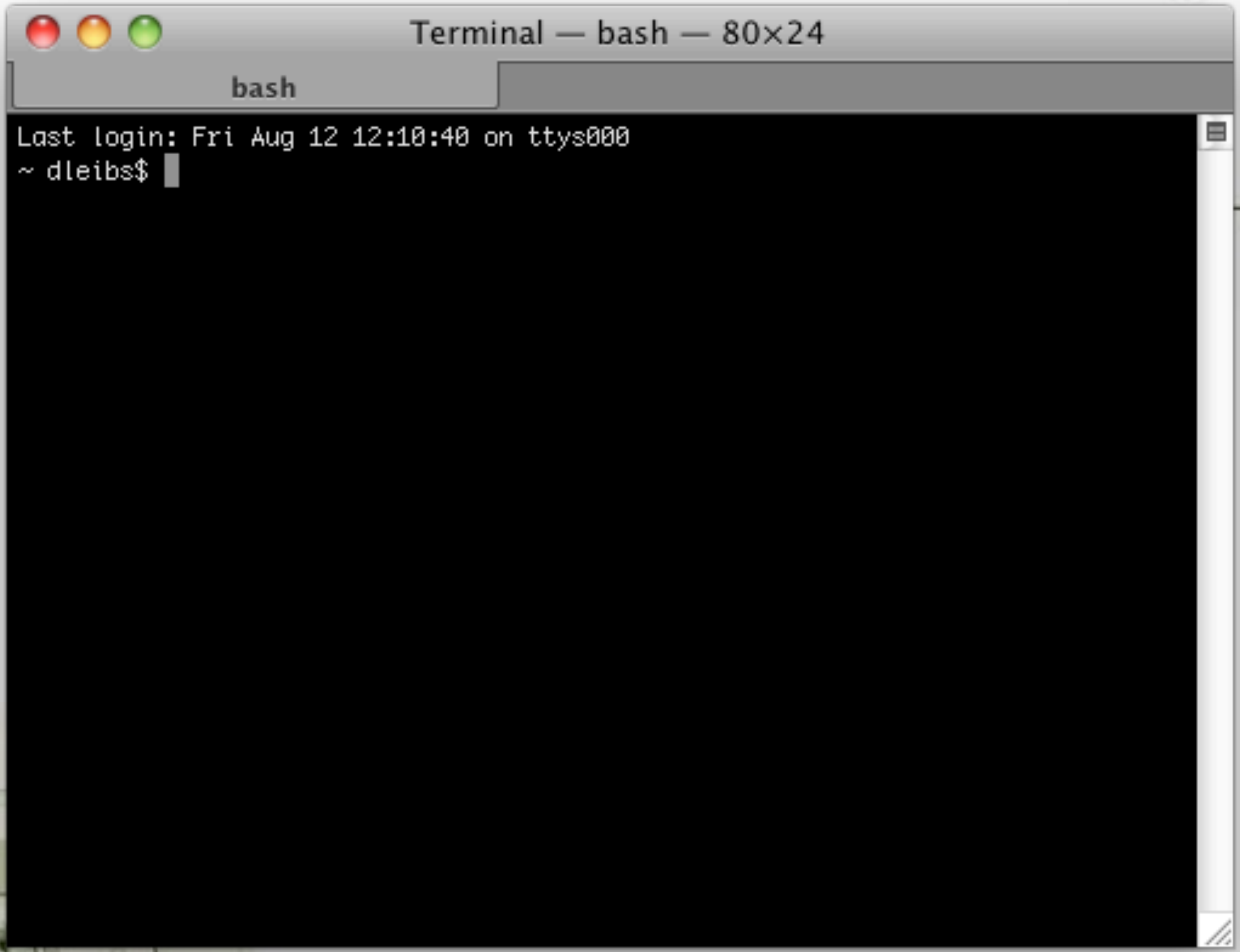




$$v_{escape} = 11.2\ km/s$$

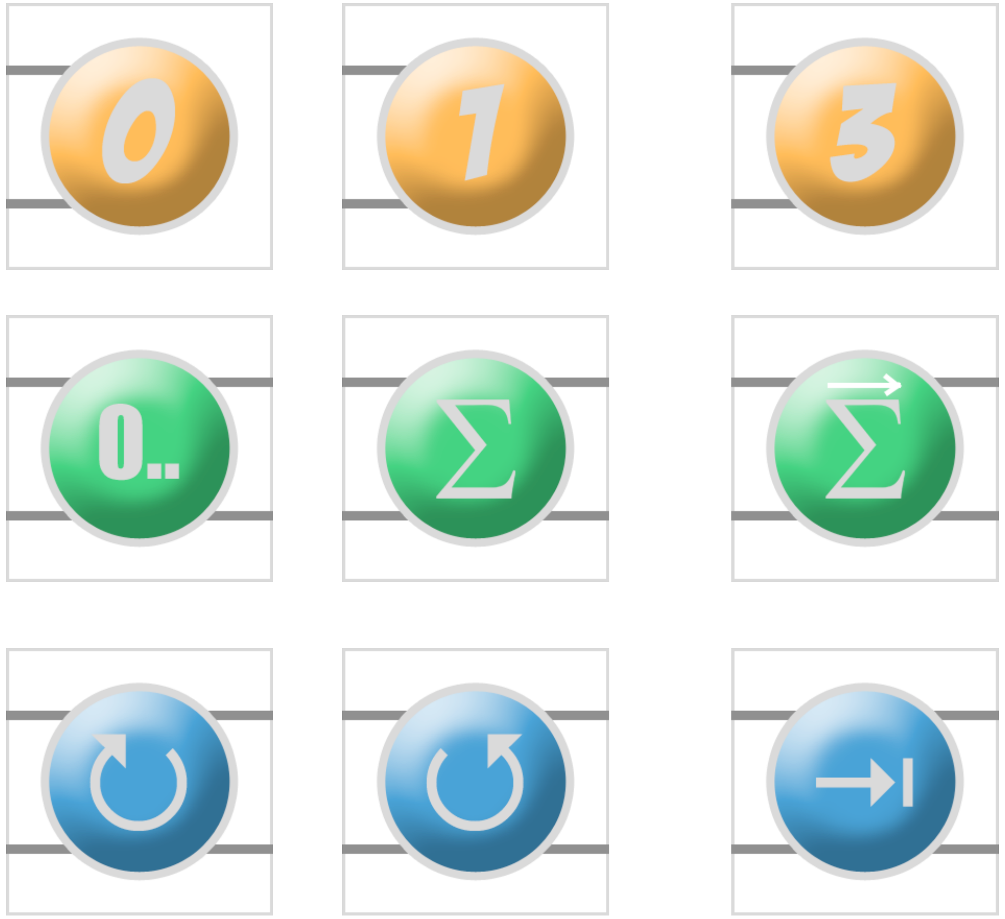
$$\frac{1}{2}mv^2 = \frac{GMm}{r}$$

$$v_{escape} = \sqrt{\frac{2GM}{r}}$$



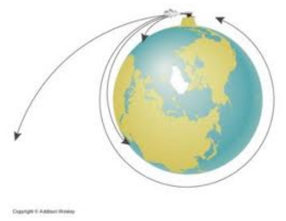
```
Last login: Fri Aug 12 12:10:40 on ttys000
~ dleibs$ 
```

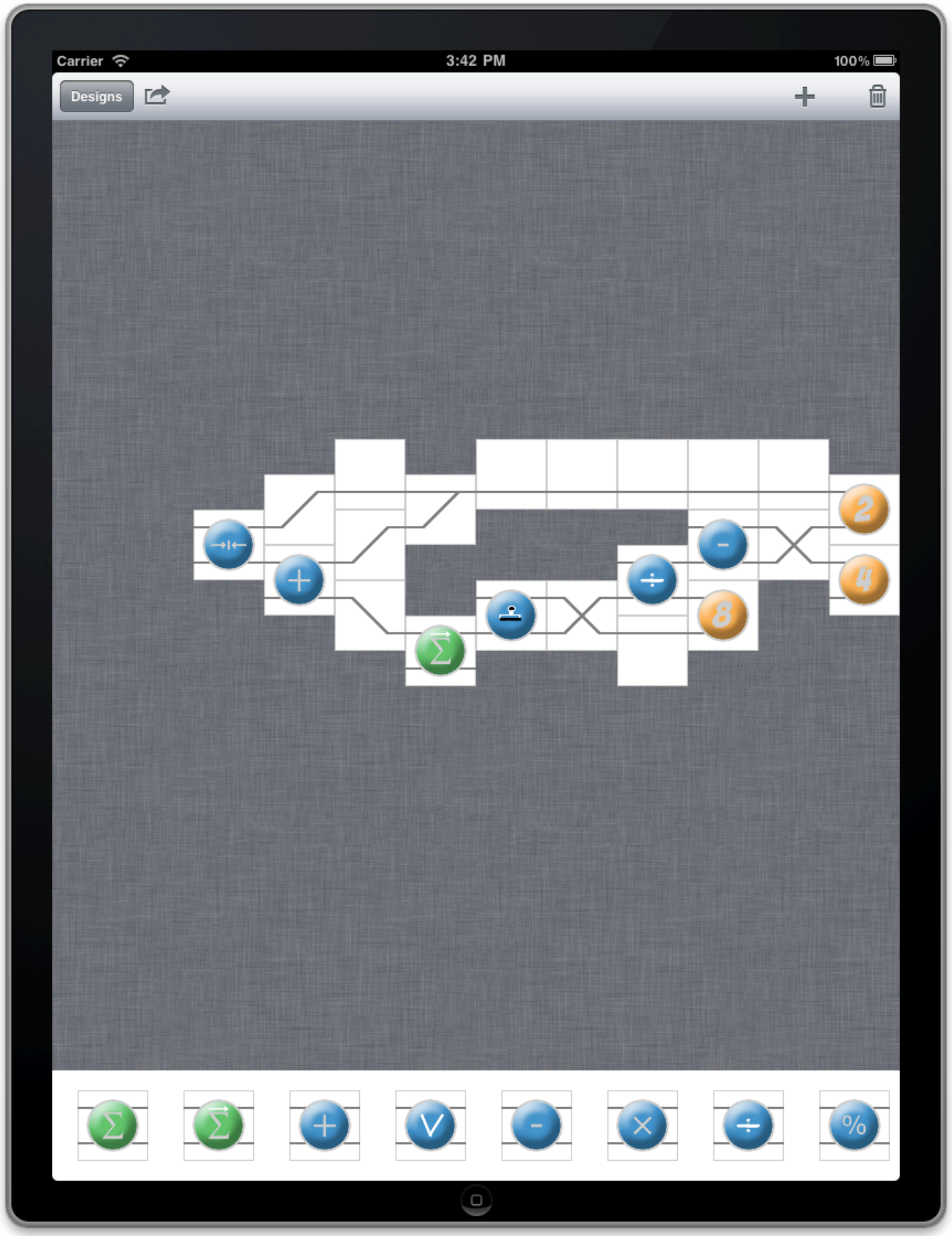
Candy colored tiles?

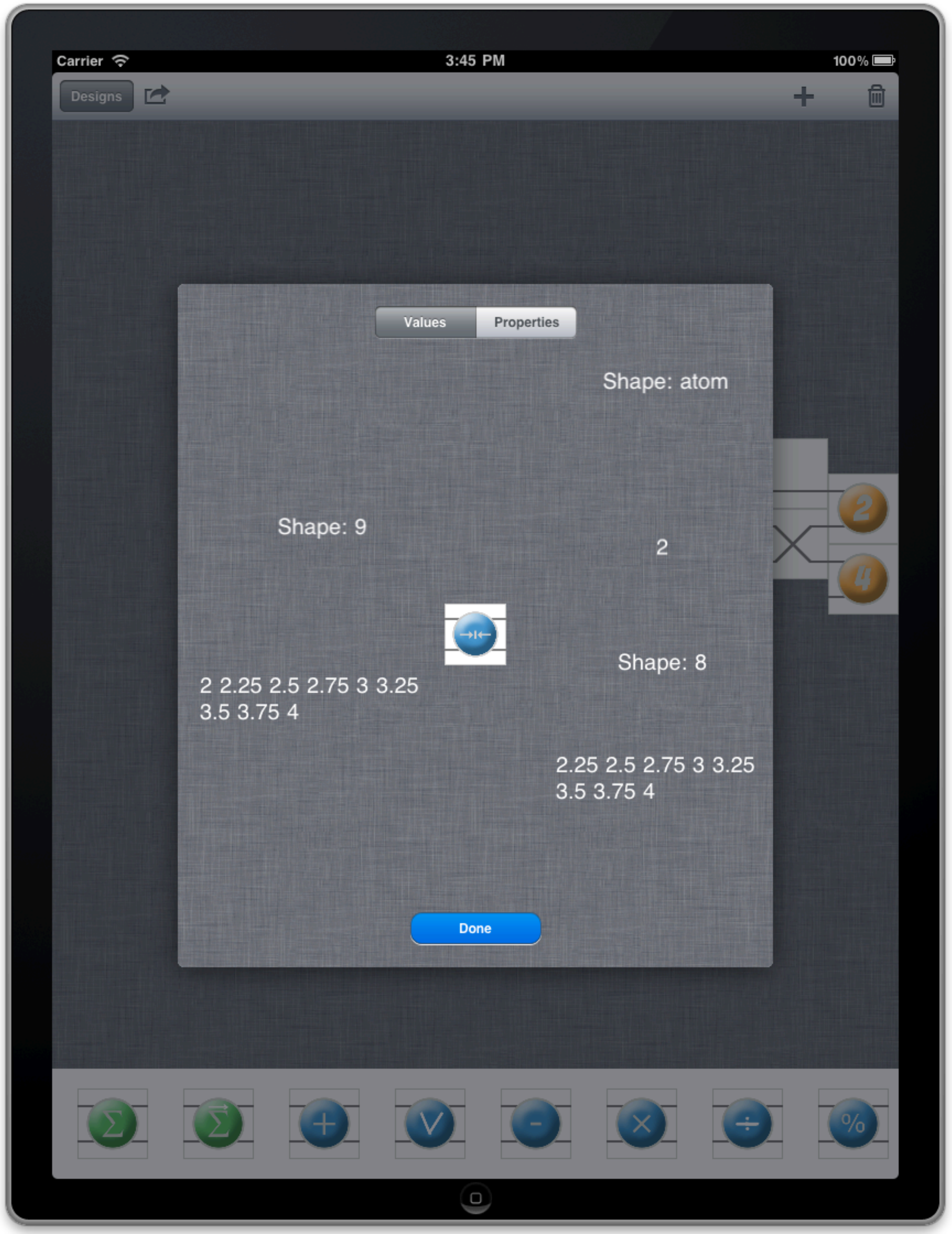


and an iPad App

Data Flow Puzzle Game?







Shape: atom

Shape: 9

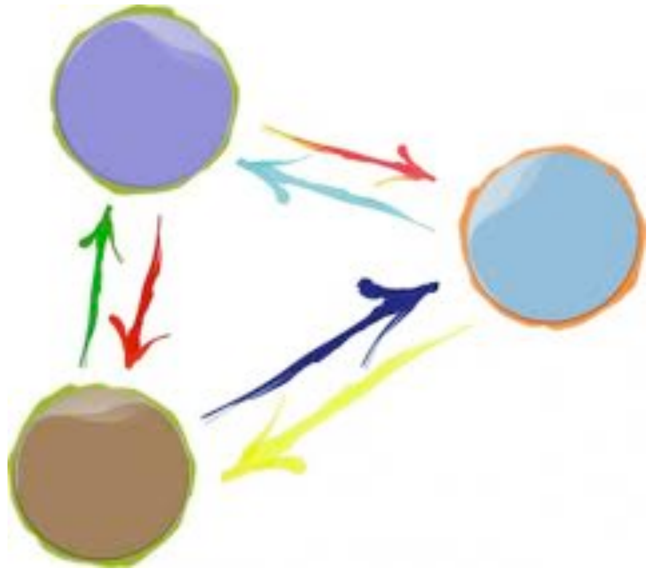
2

Shape: 8

2 2.25 2.5 2.75 3 3.25 3.5 3.75 4

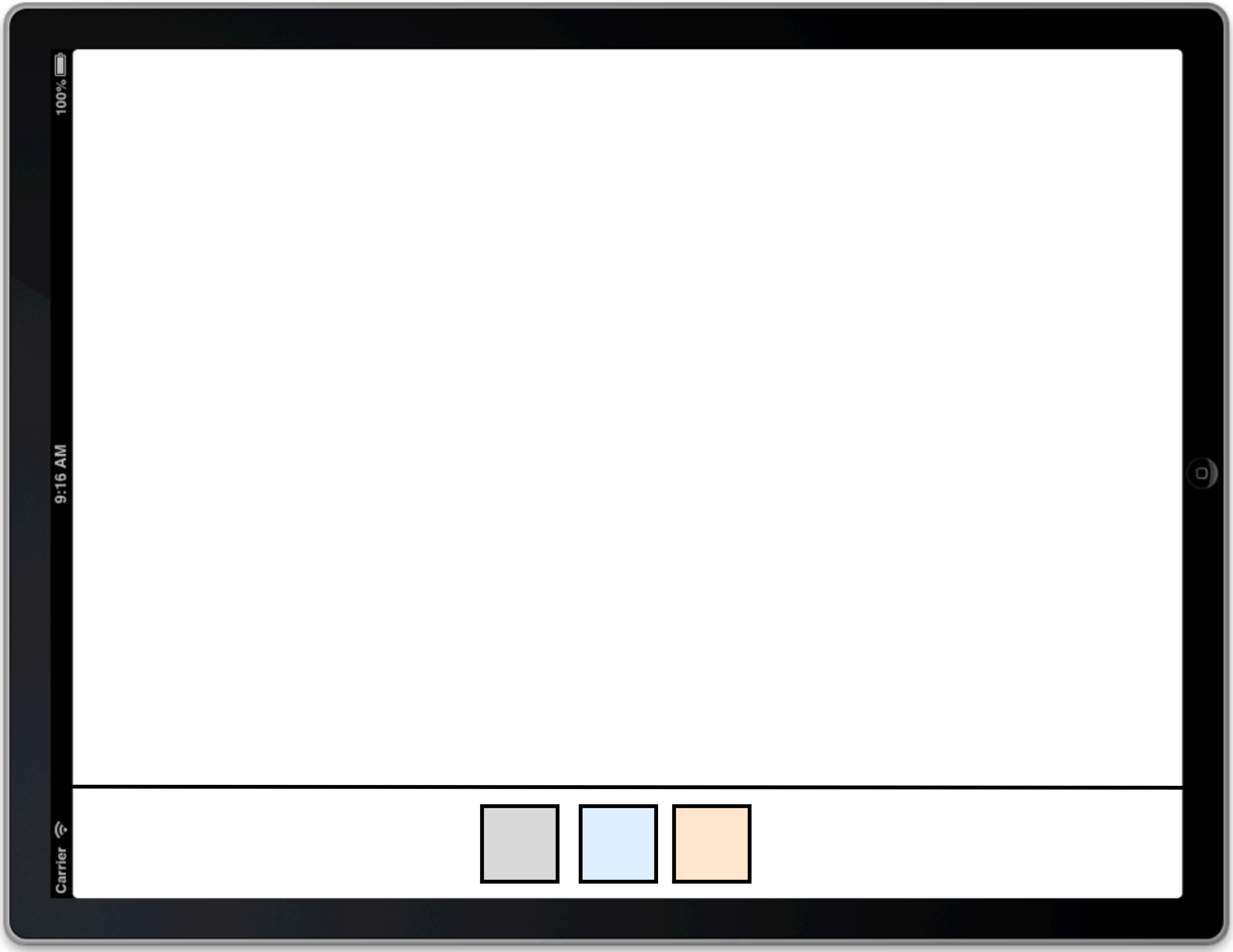
2.25 2.5 2.75 3 3.25 3.5 3.75 4

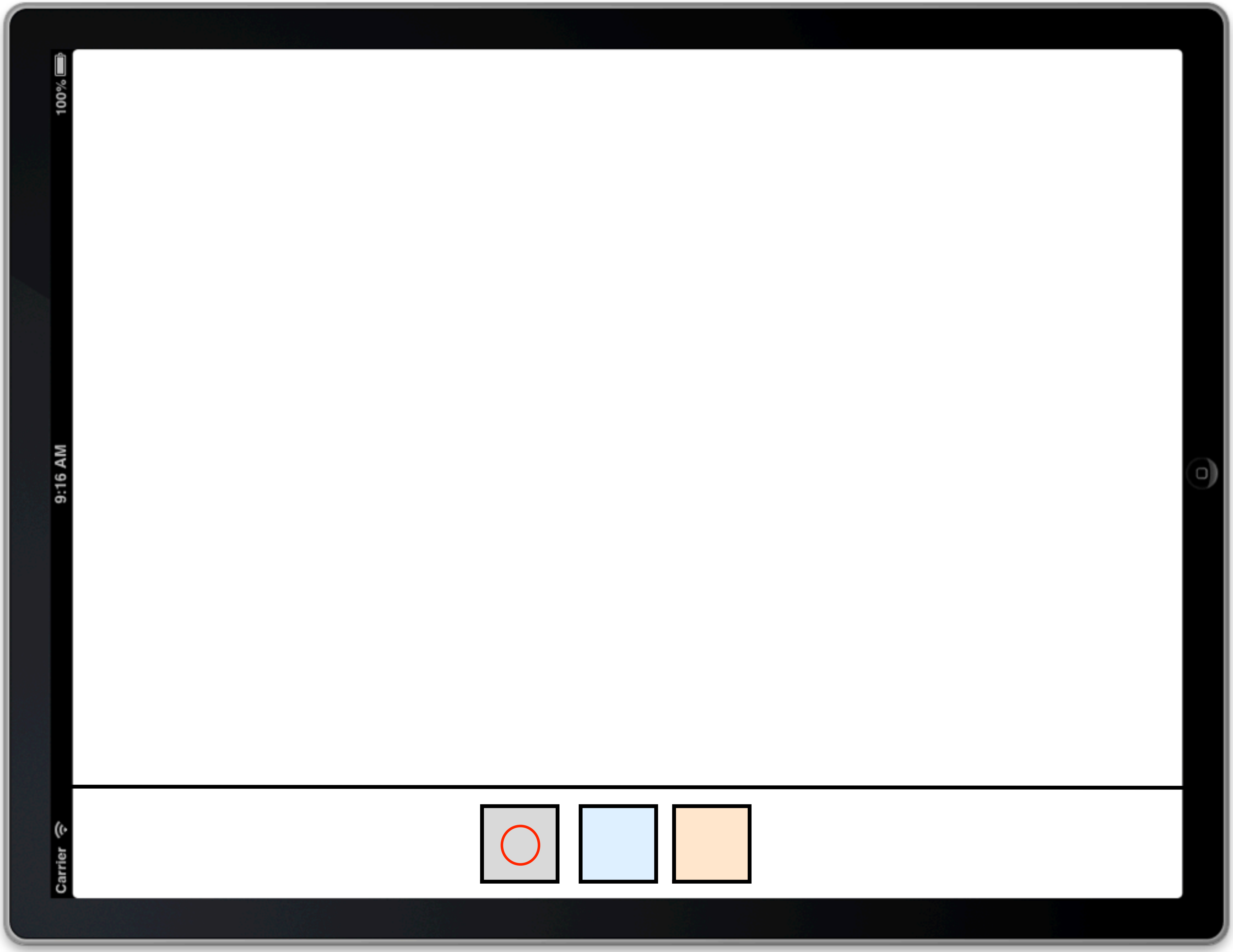
Some Interaction Ideas

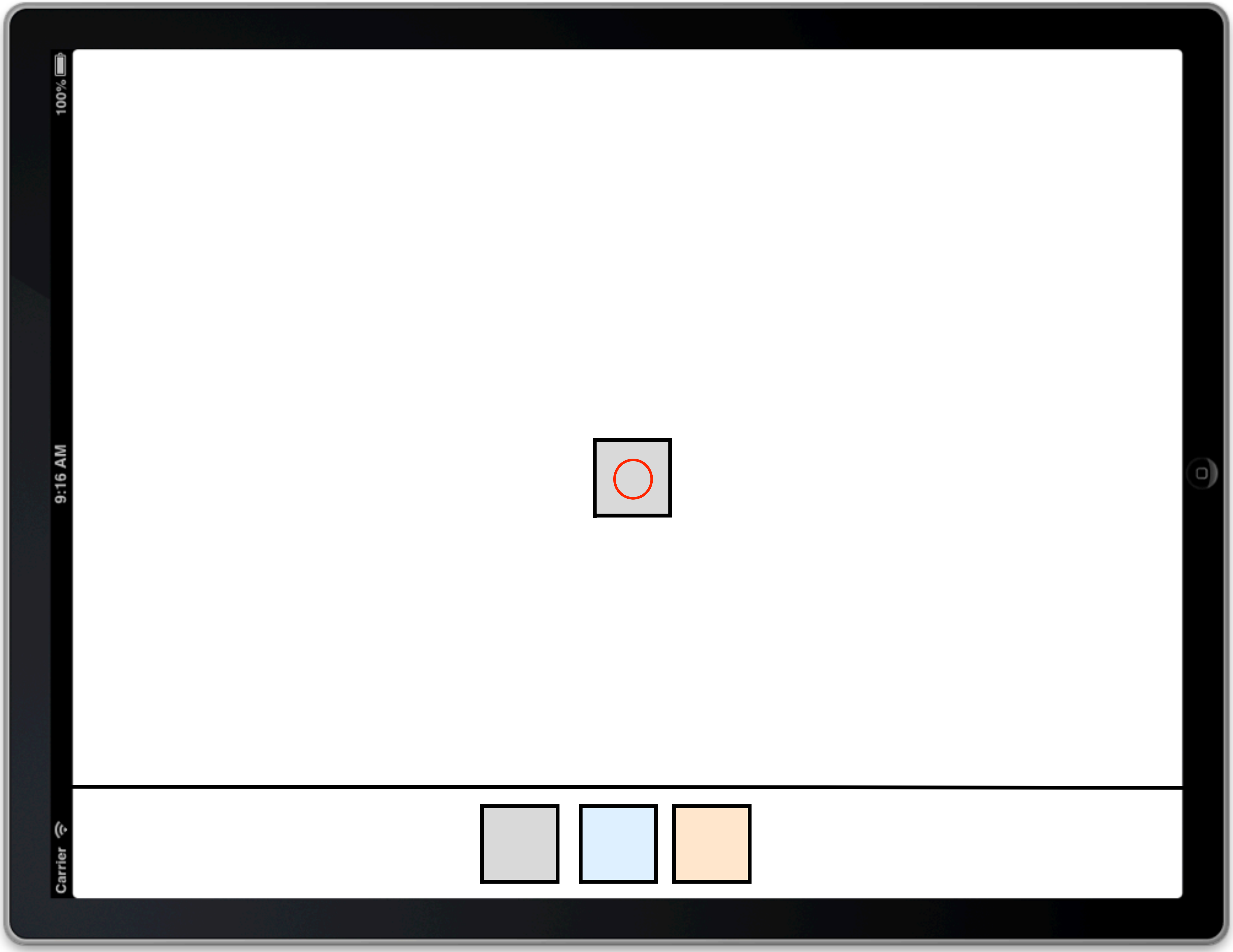


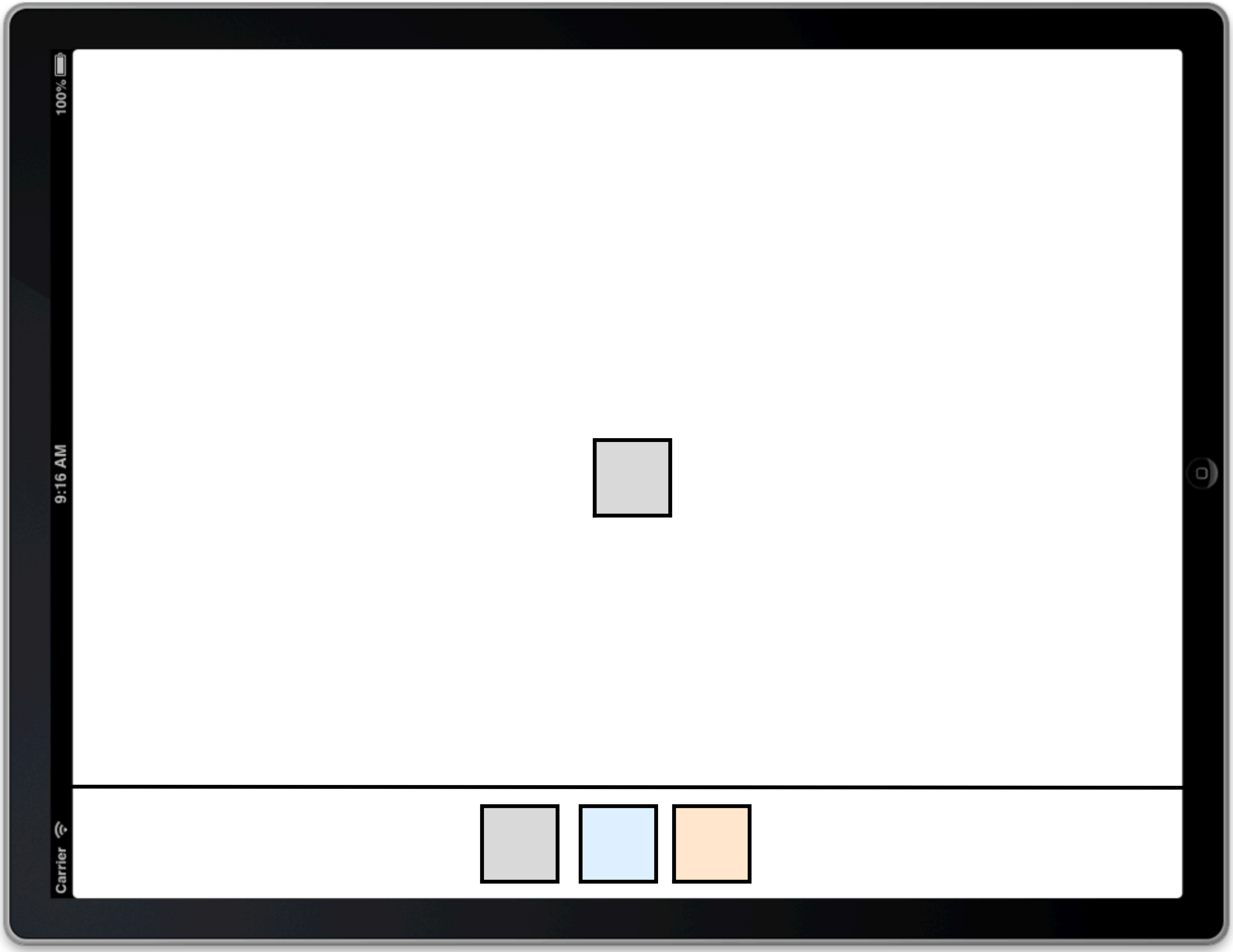
Pretend this is an iPad

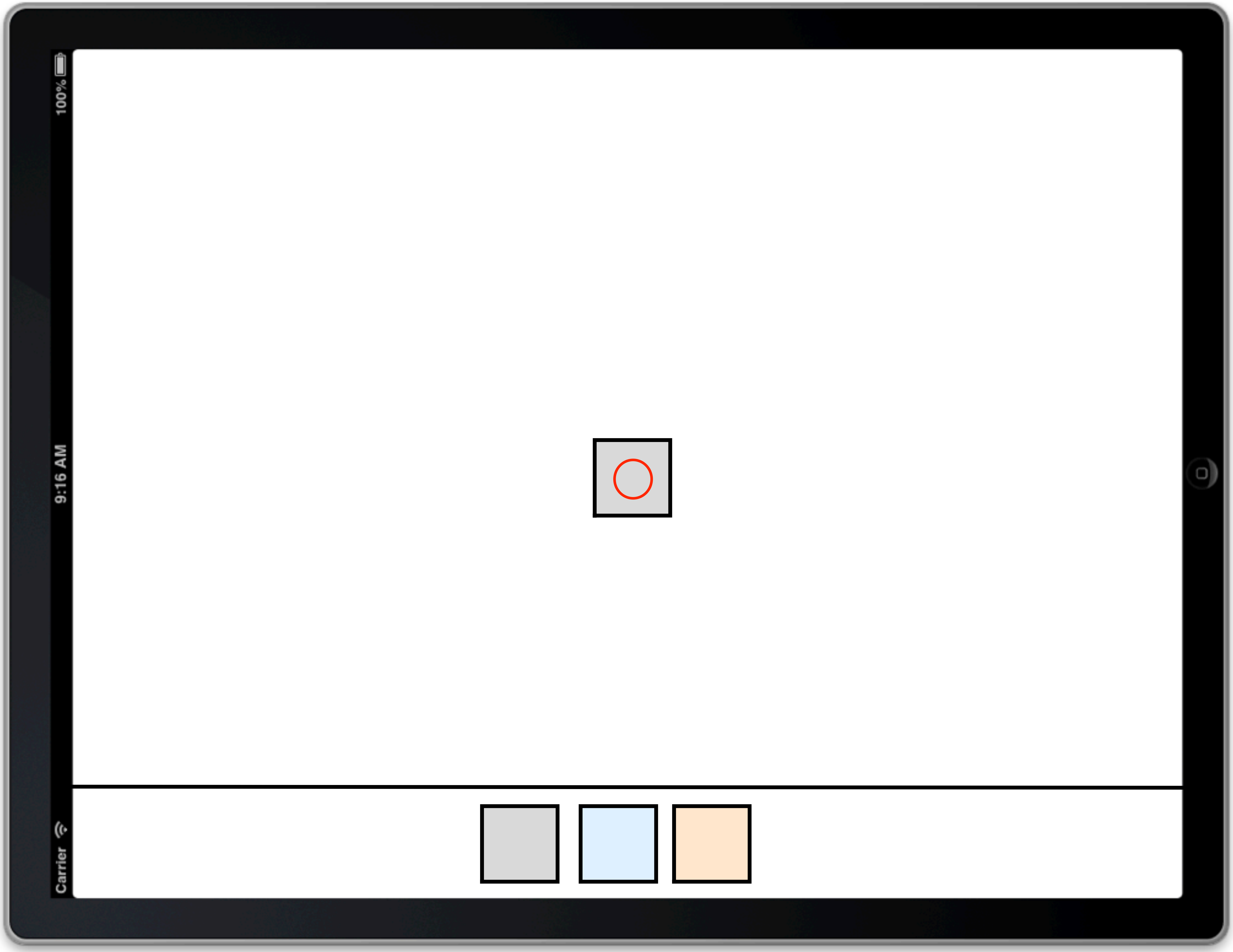
1. René Magritte, La Condition humaine, 1933.

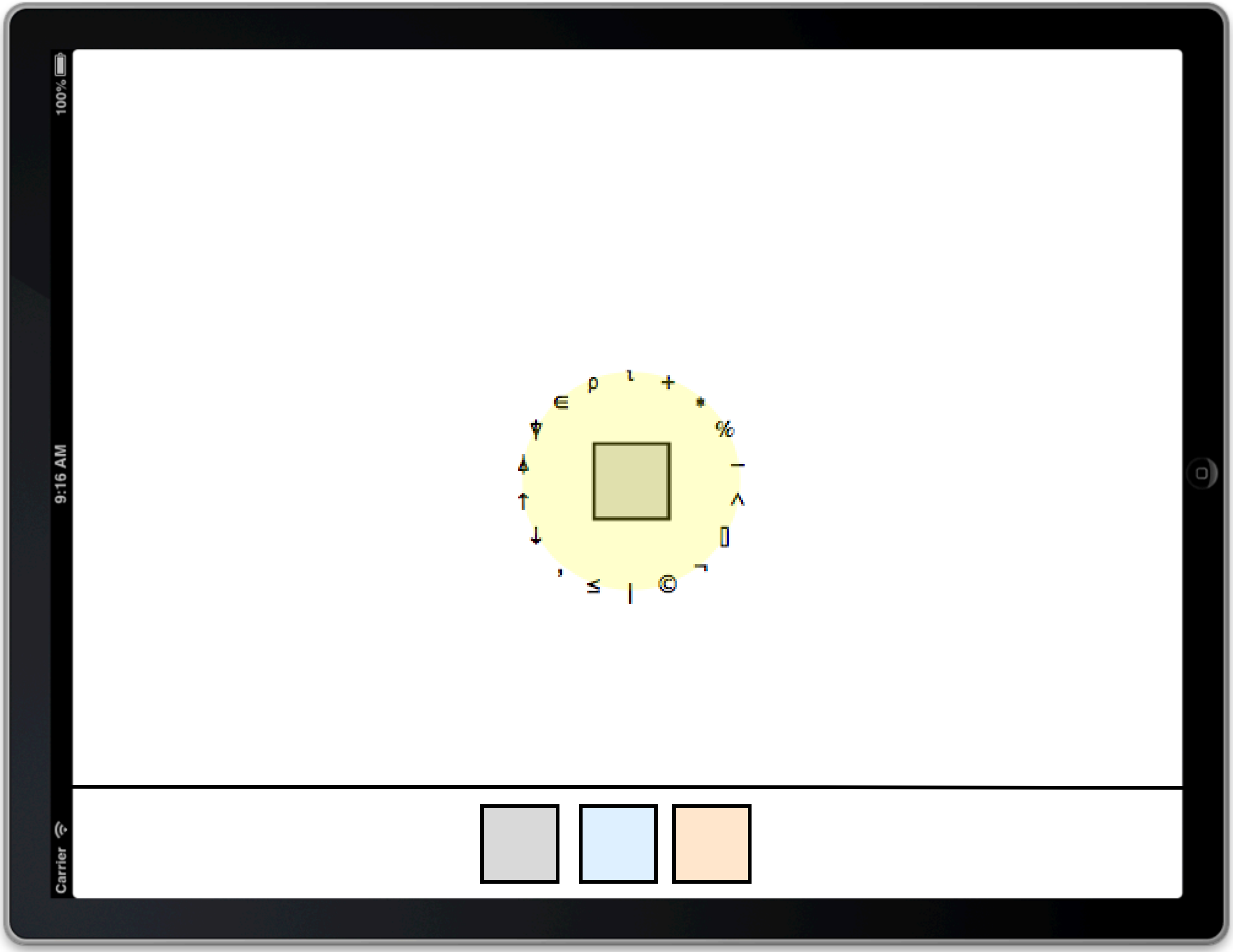

100%
9:16 AM
Carrier

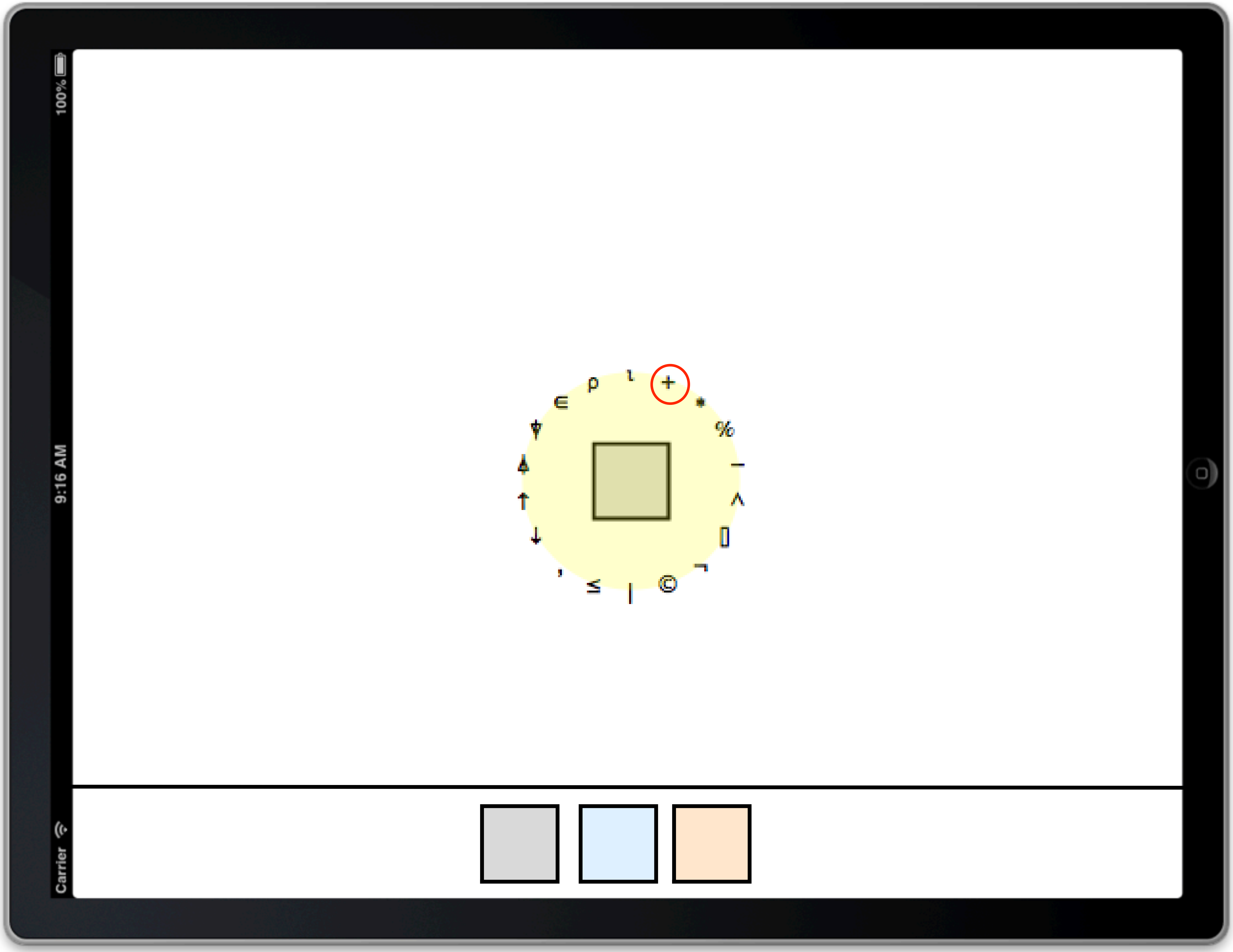

9:16 AM
100%
Carrier


100%
9:16 AM
Carrier

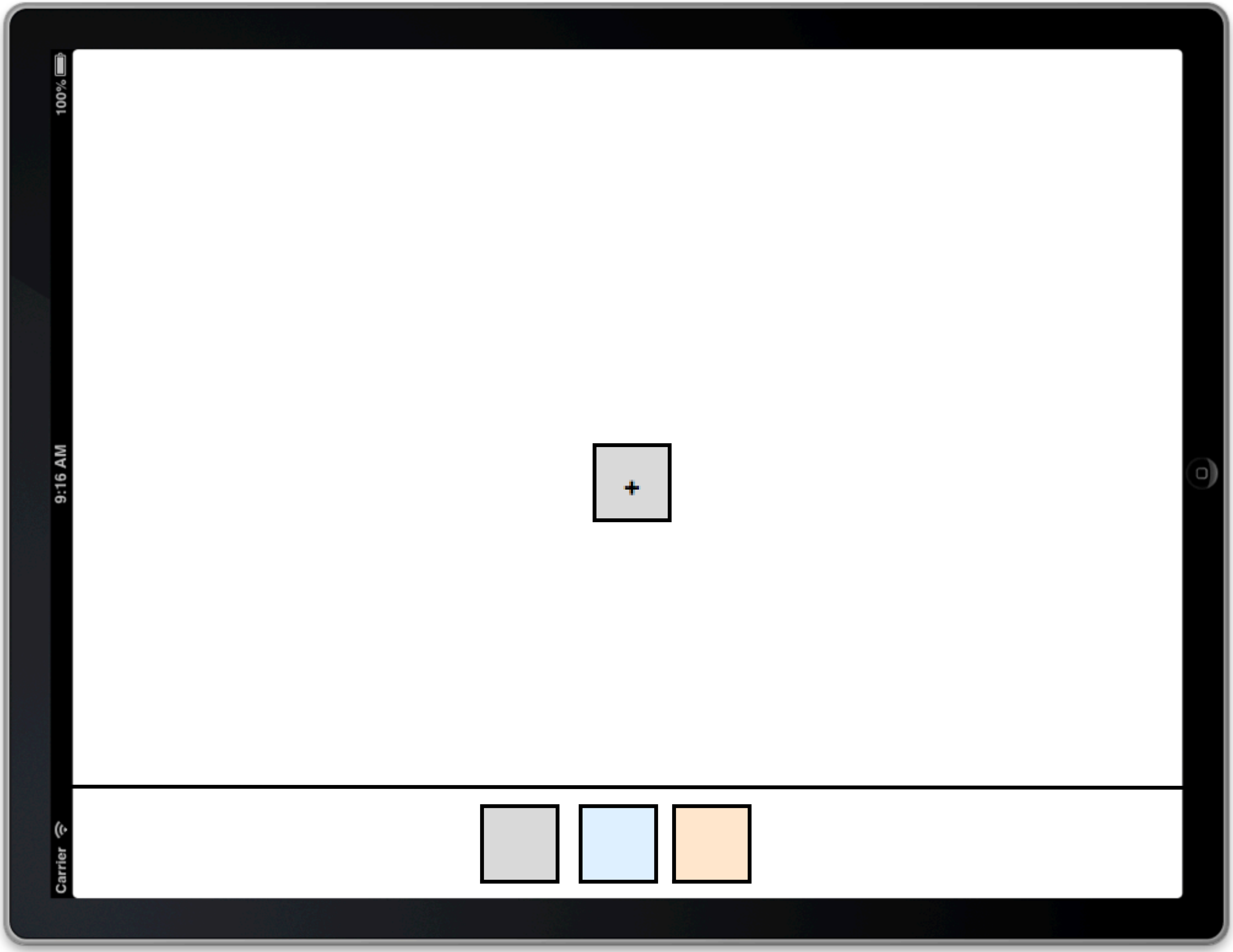

100%
9:16 AM
Carrier

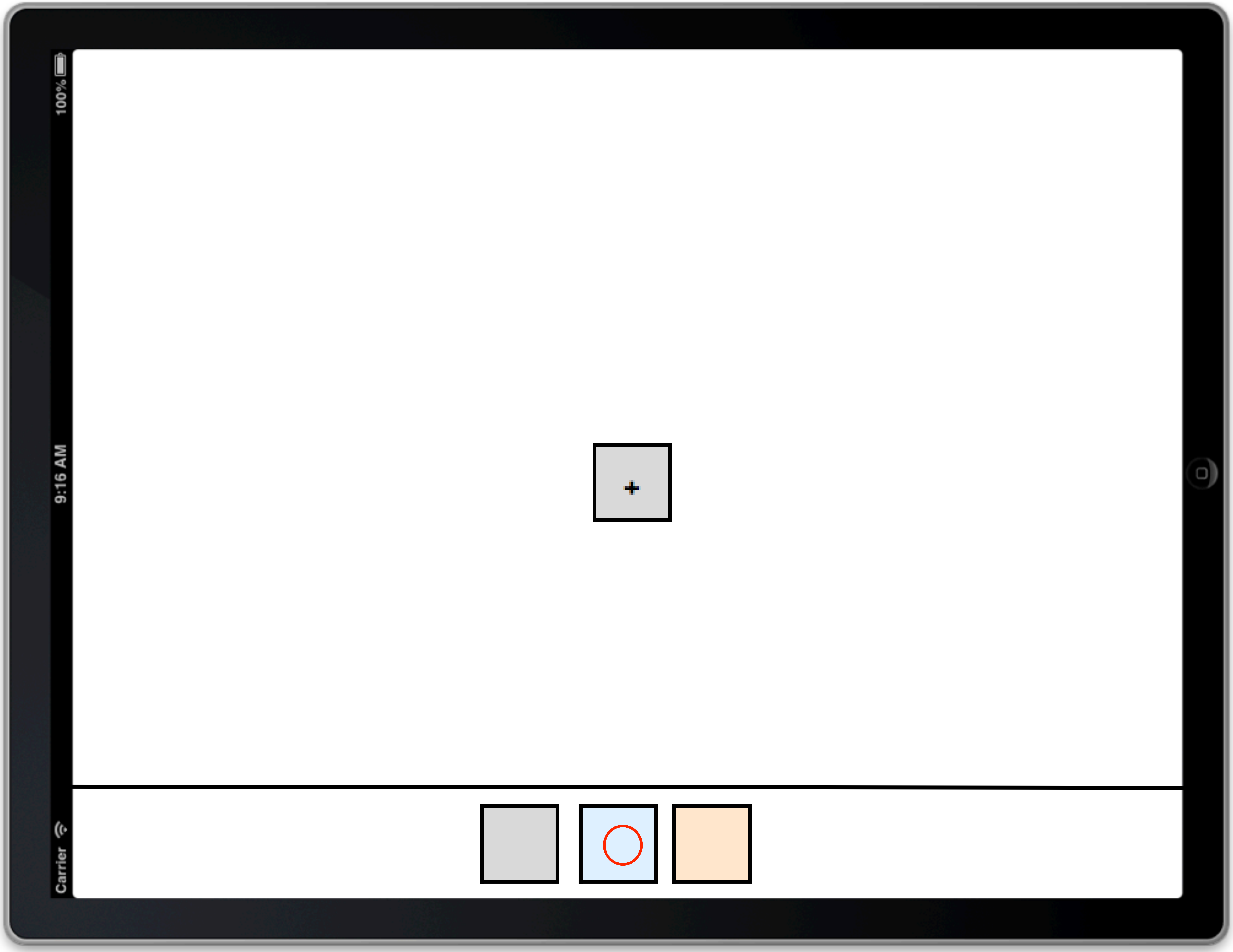

Carrier
9:16 AM
100%

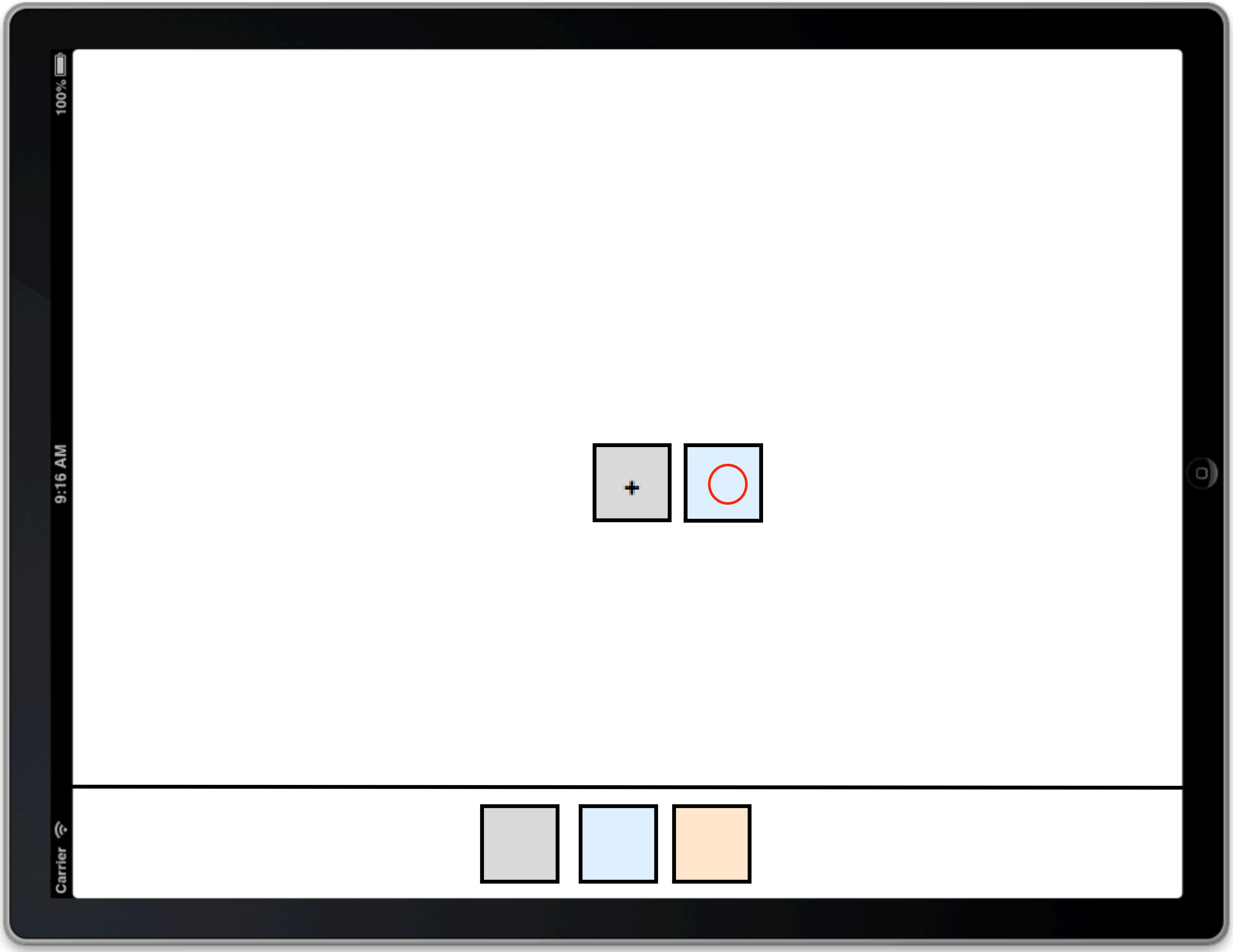

100%
9:16 AM
Carrier
ρ
ι
+
∈
*
⍋
%
⍒
−
↑
^
↓
⌷
,
¬
≤
|
©

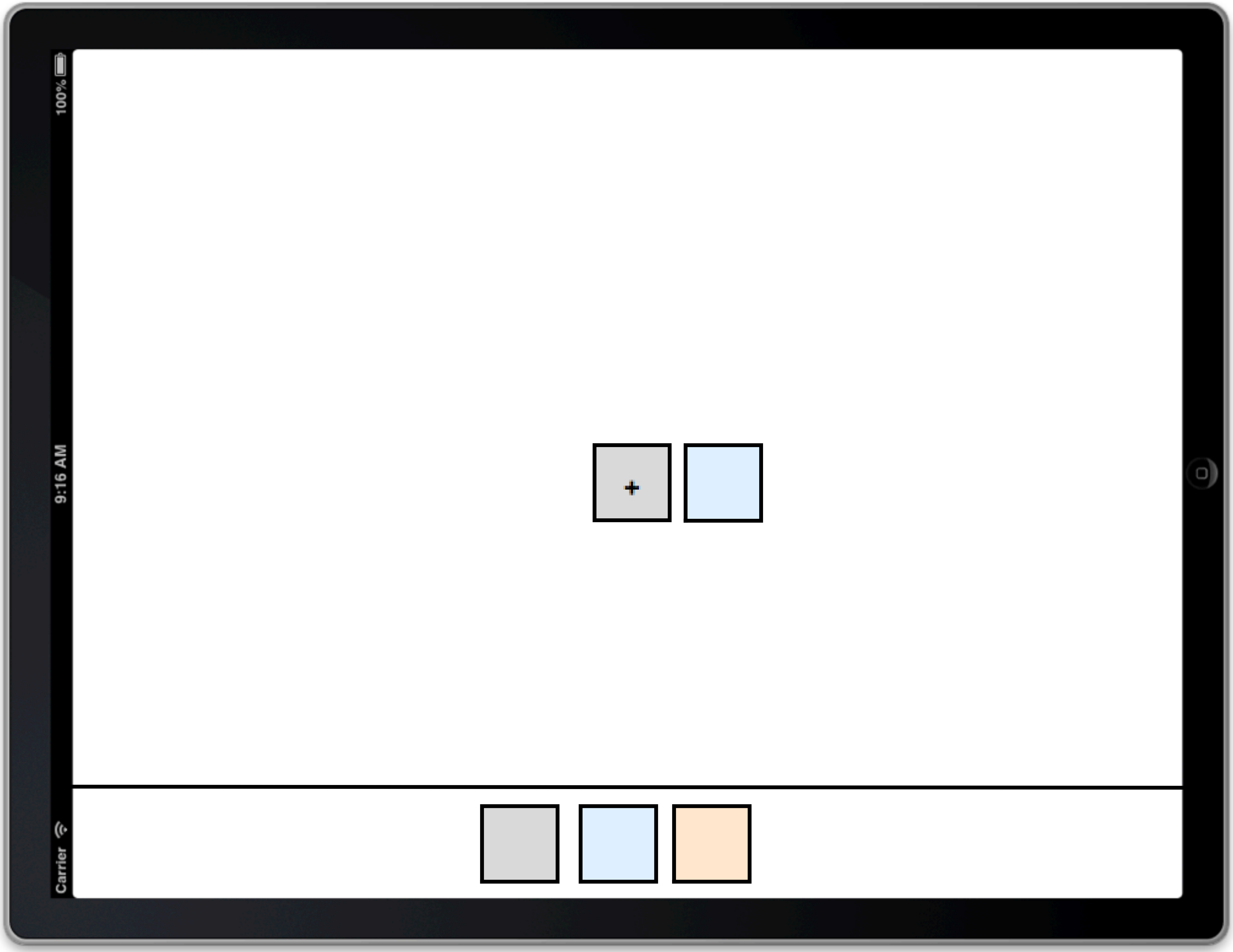


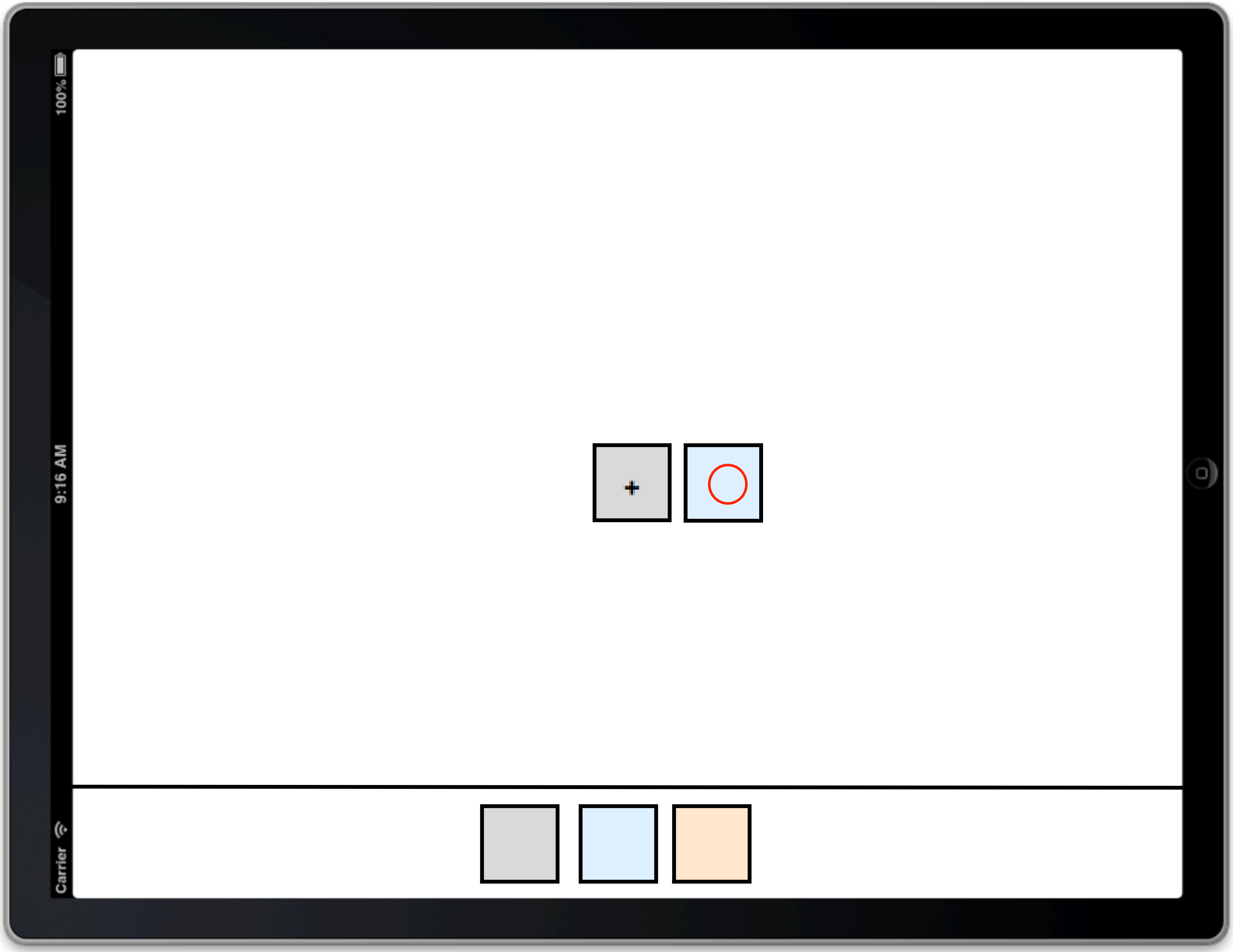
100%
9:16 AM
Carrier
ρ
ι
+
∈
%
⍋
⍒
↑
↓
,
≤
|
©
¬

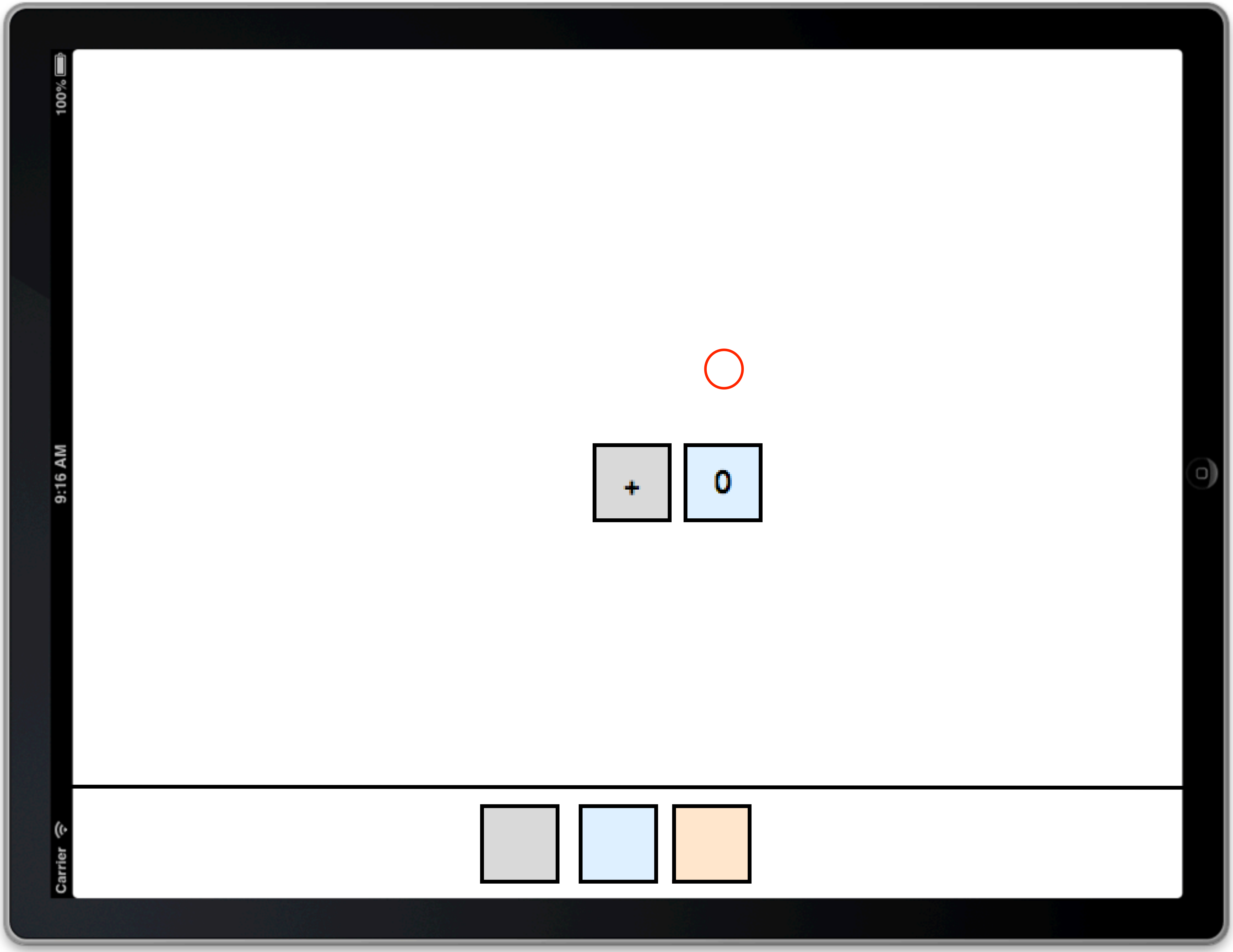

100%
9:16 AM
Carrier
+

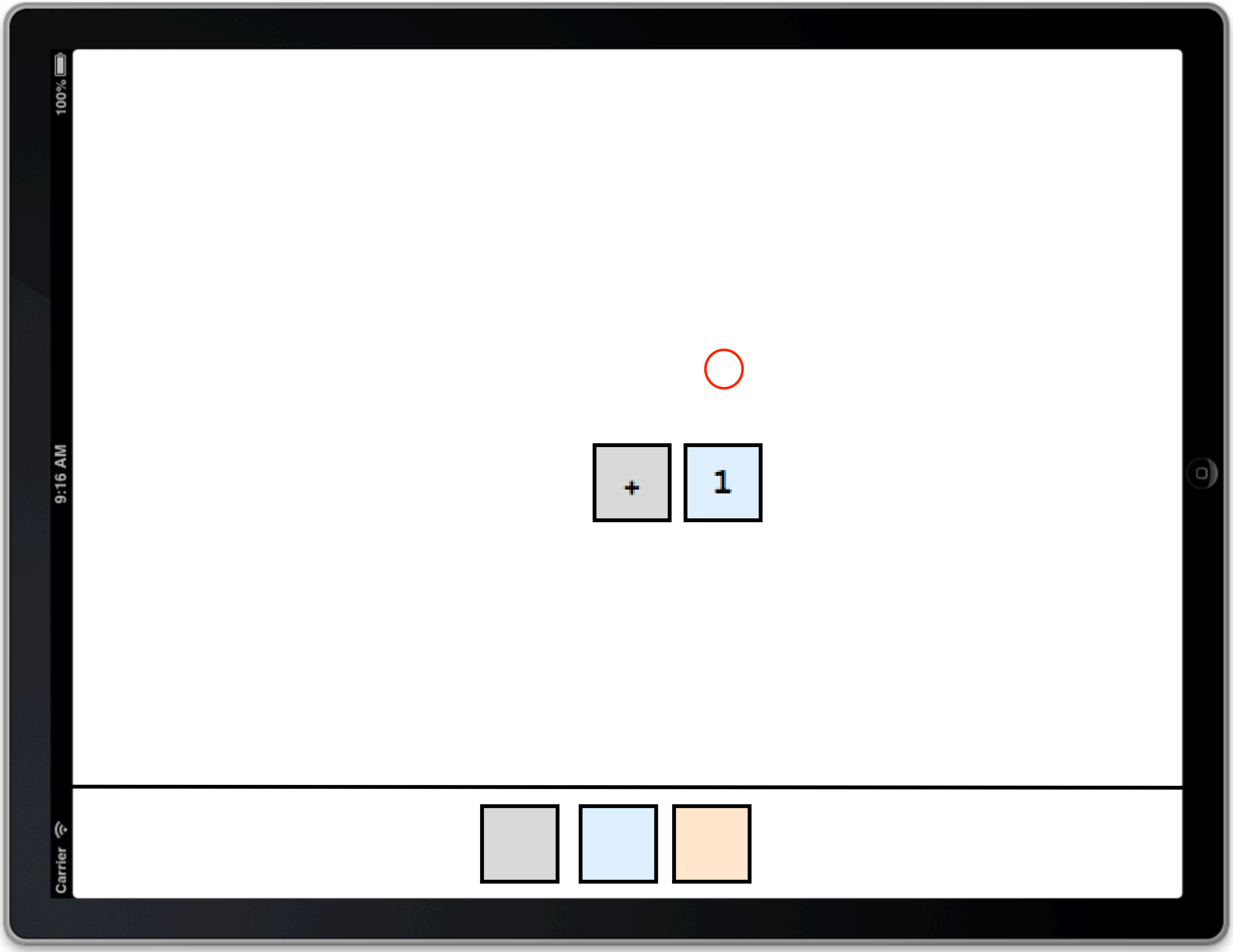

Carrier
9:16 AM
100%
+

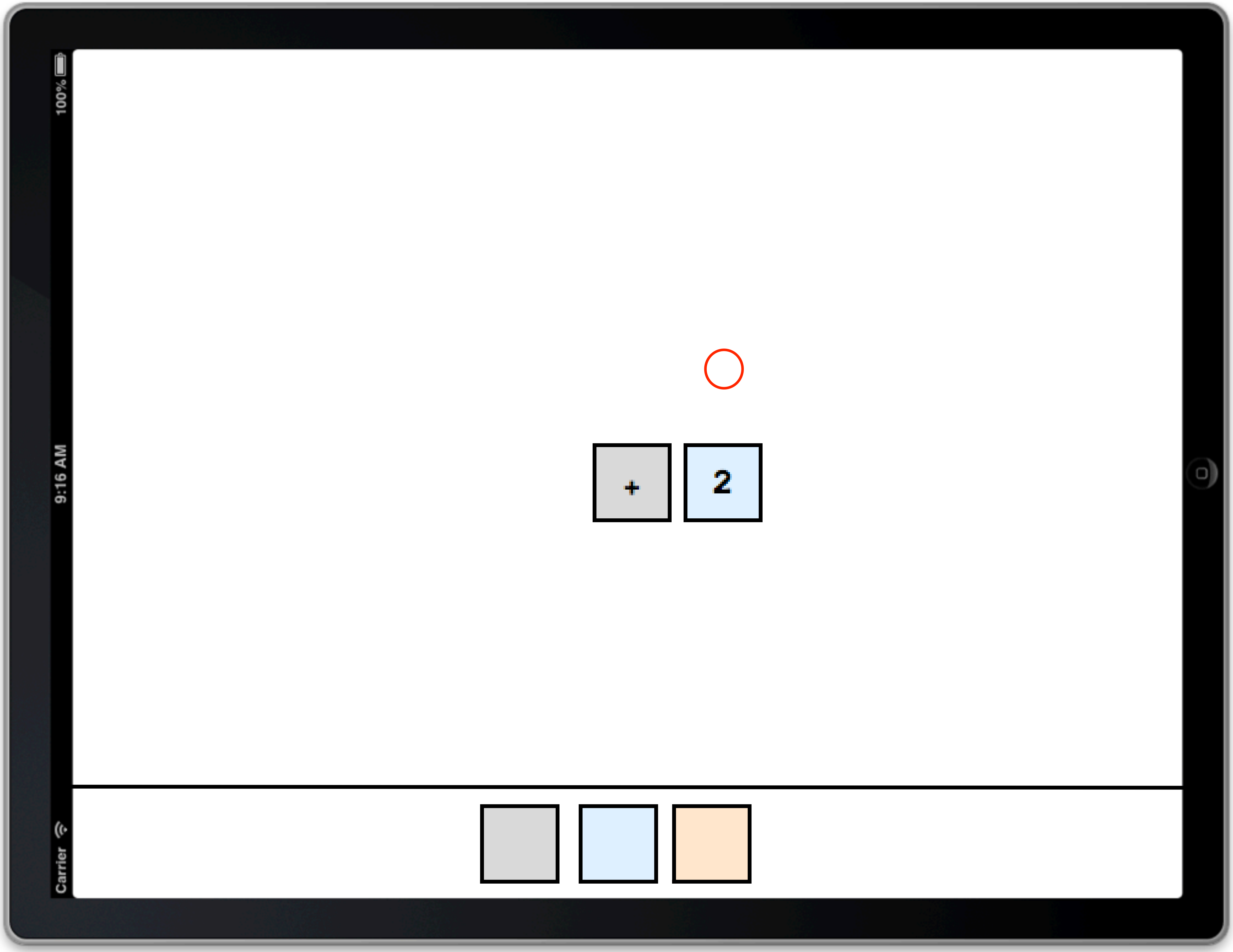

100%
9:16 AM
Carrier
+

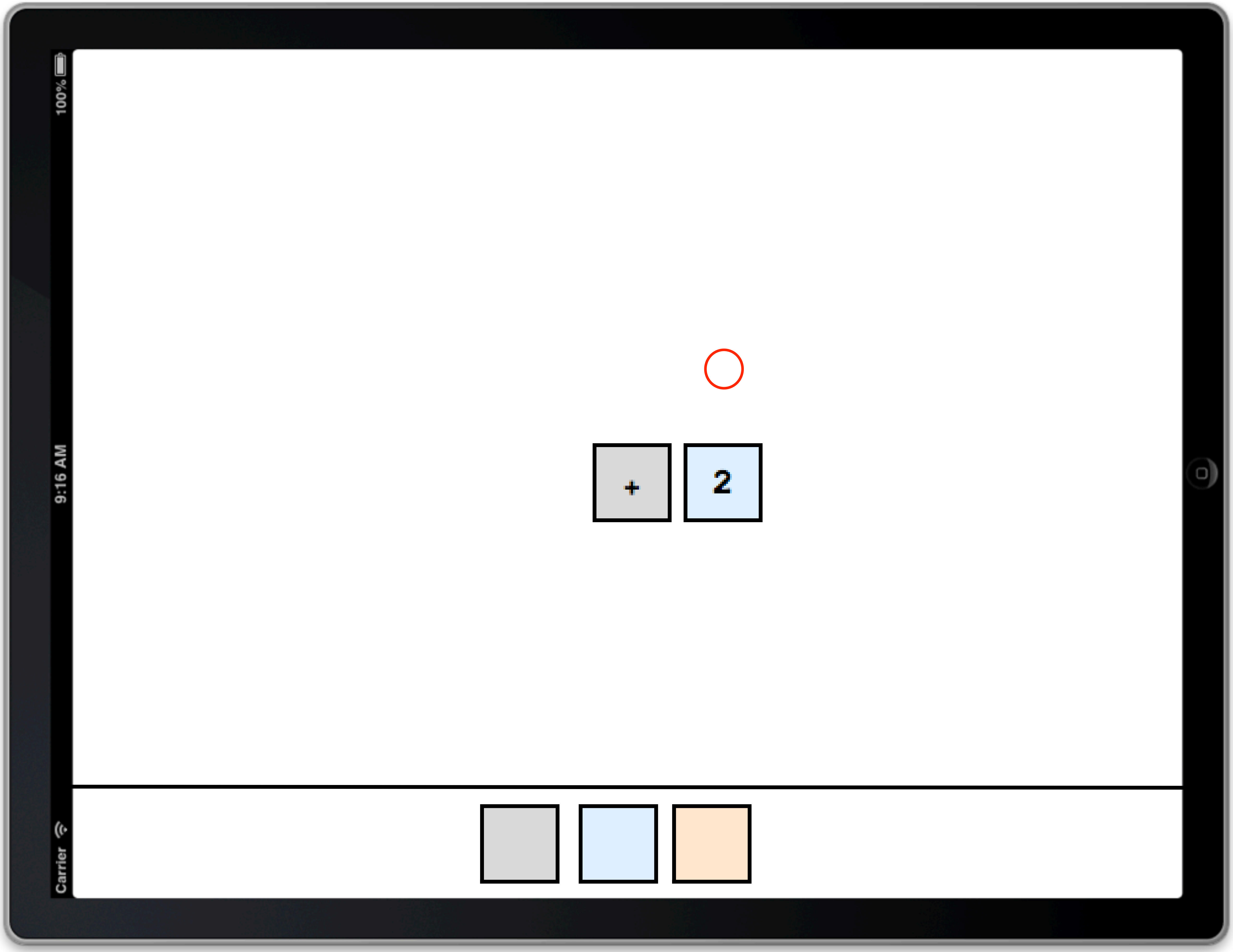

100%
9:16 AM
Carrier
+

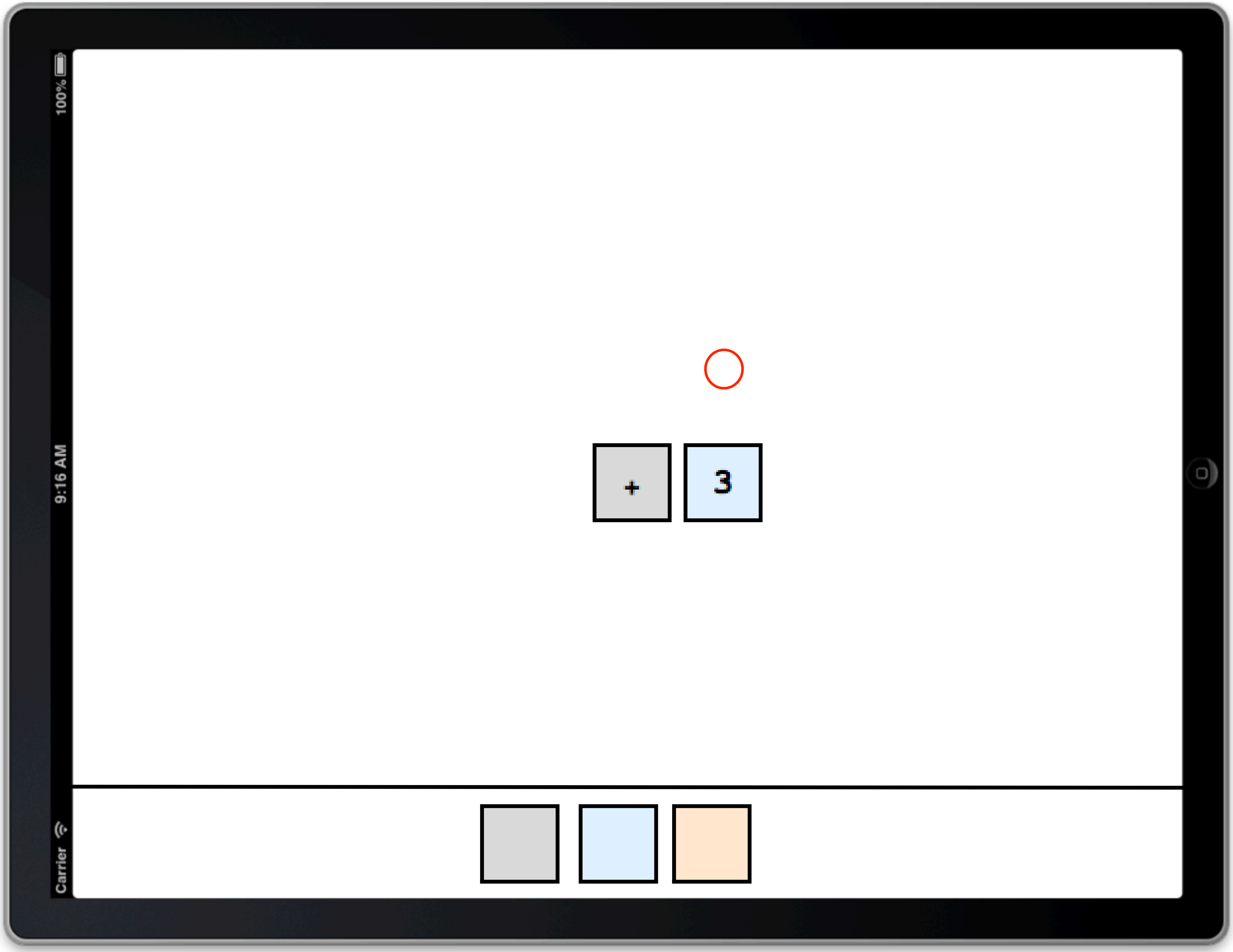

Carrier
9:16 AM
100%
+


100%
9:16 AM
Carrier
+
0

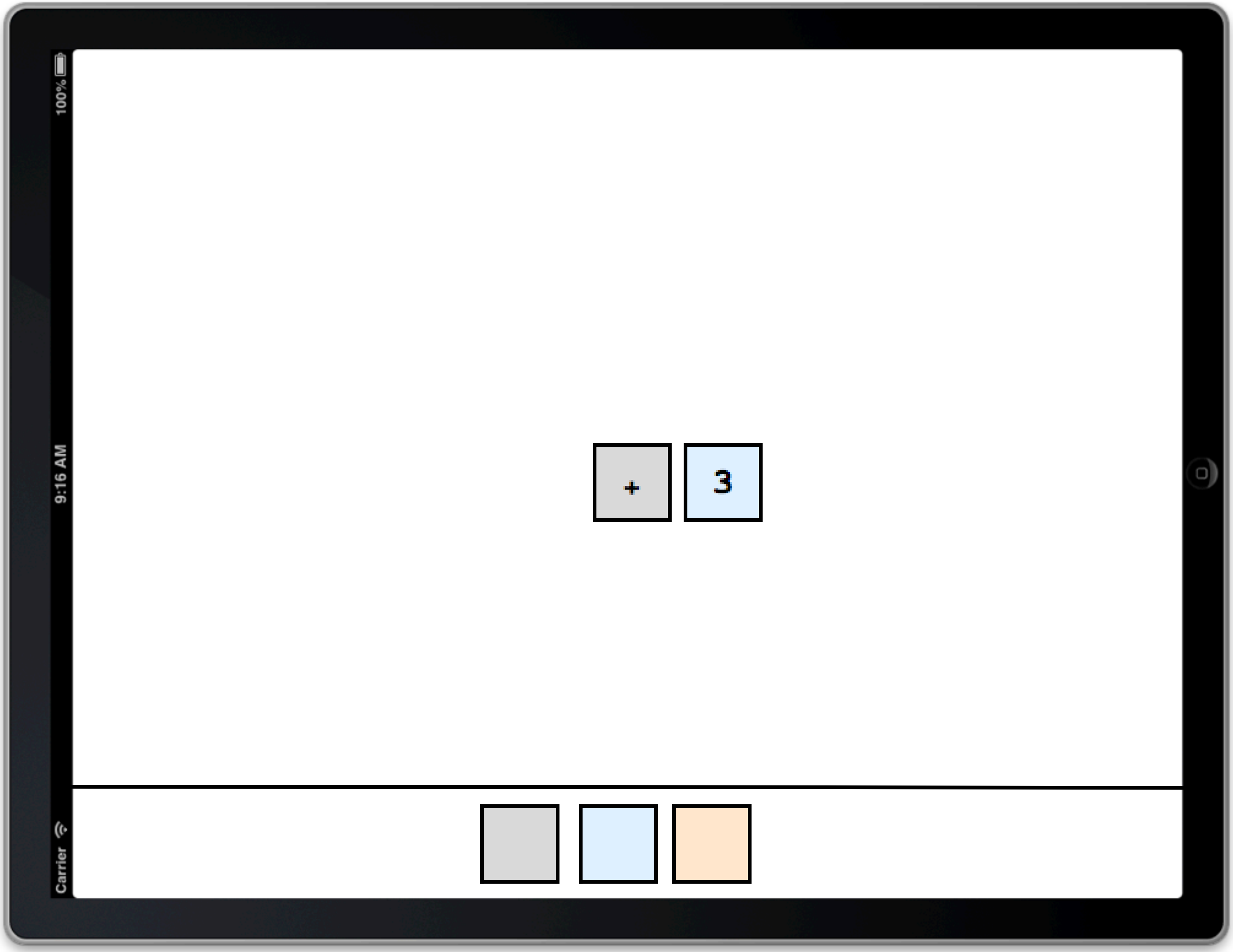

100%
9:16 AM
Carrier
+
1

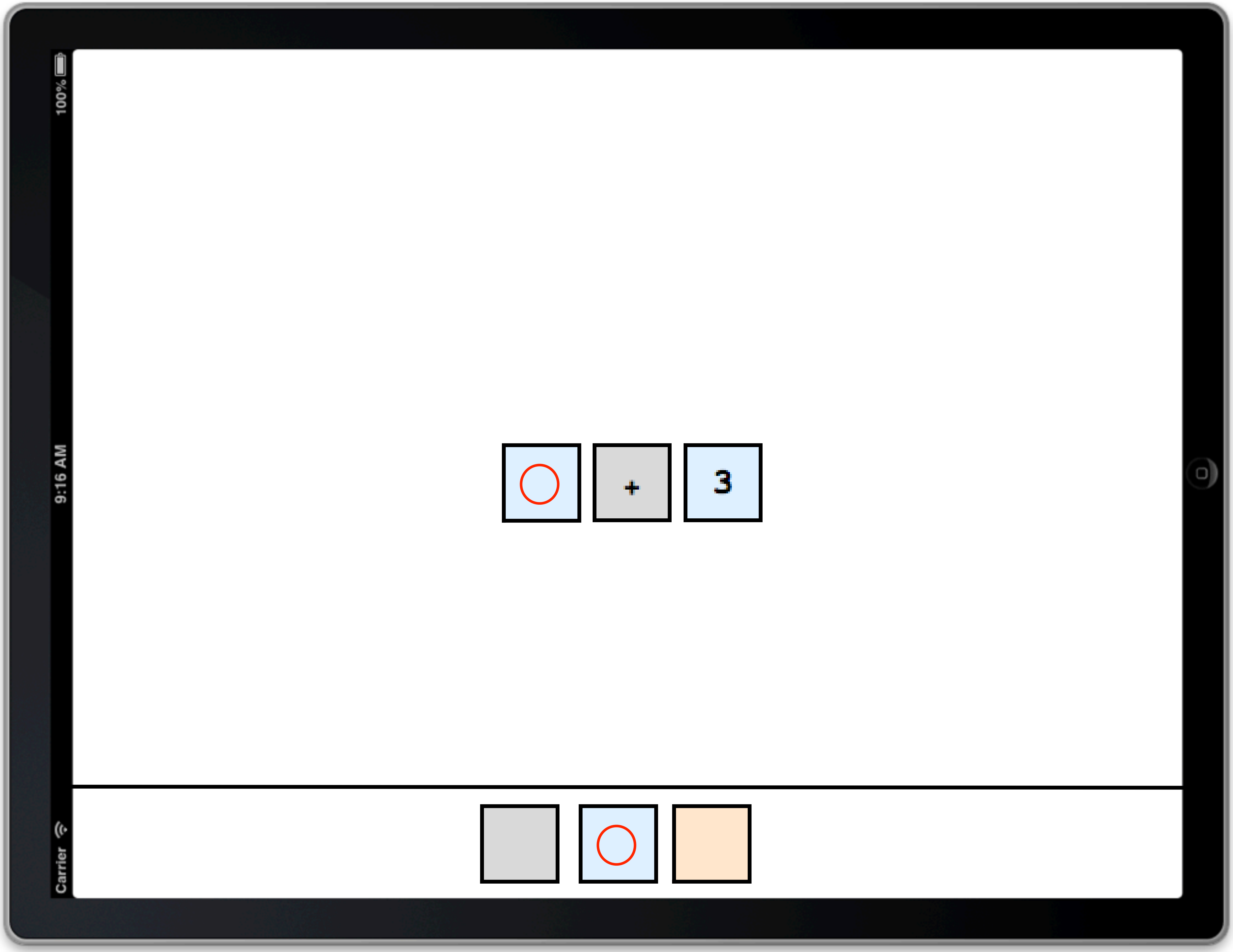

100%
9:16 AM
Carrier
+
2

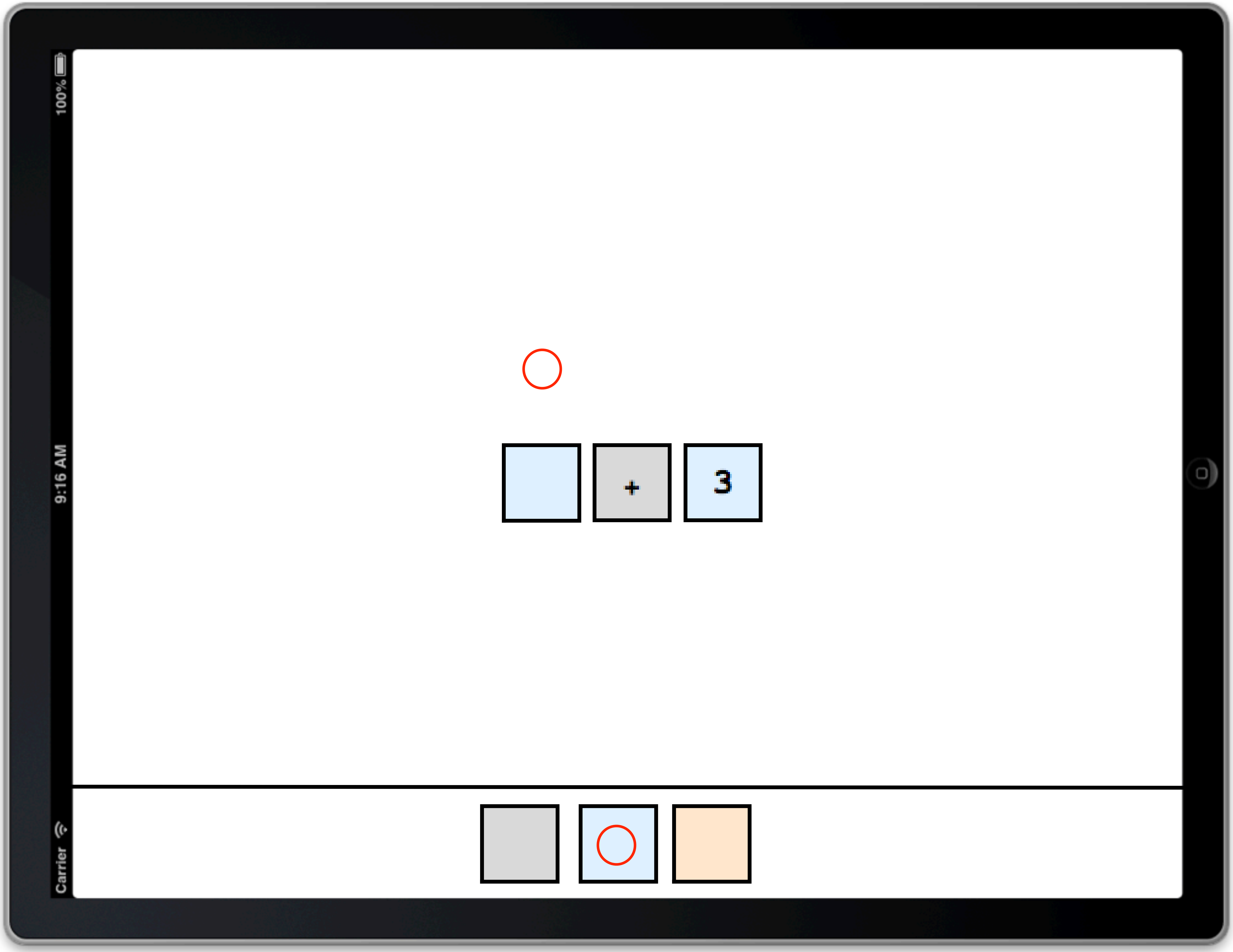

100%
9:16 AM
Carrier
+
2

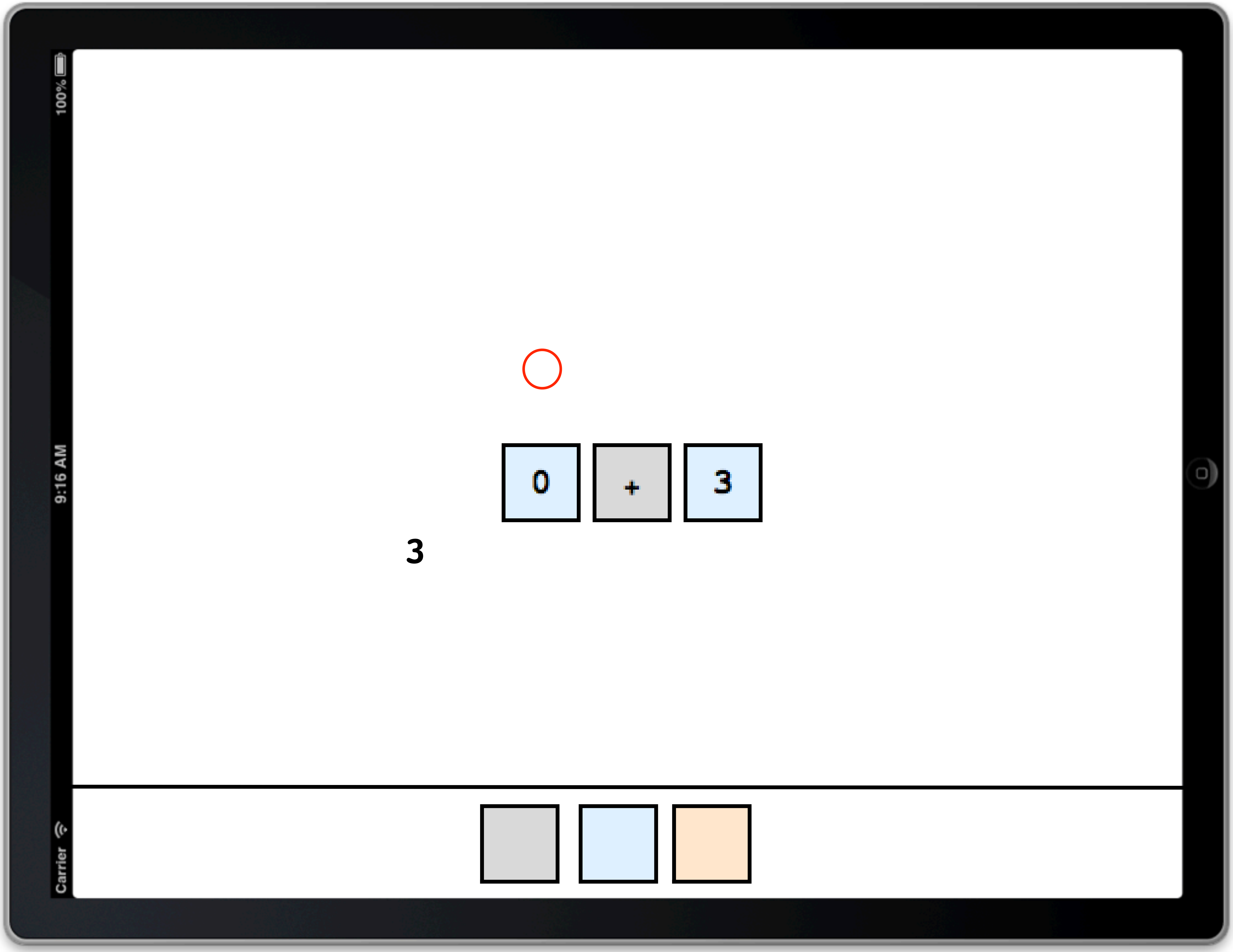


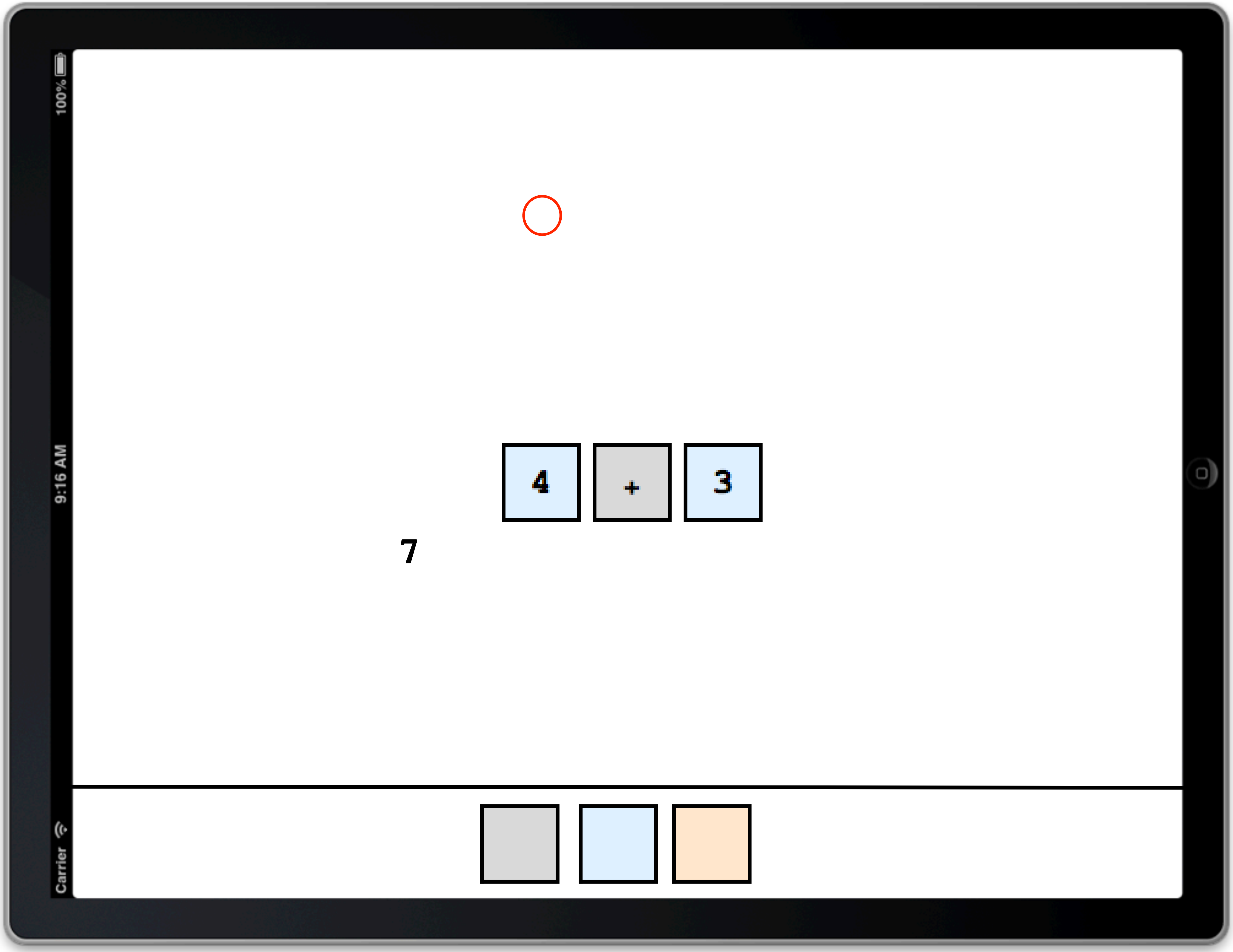
Monday, October 1, 12

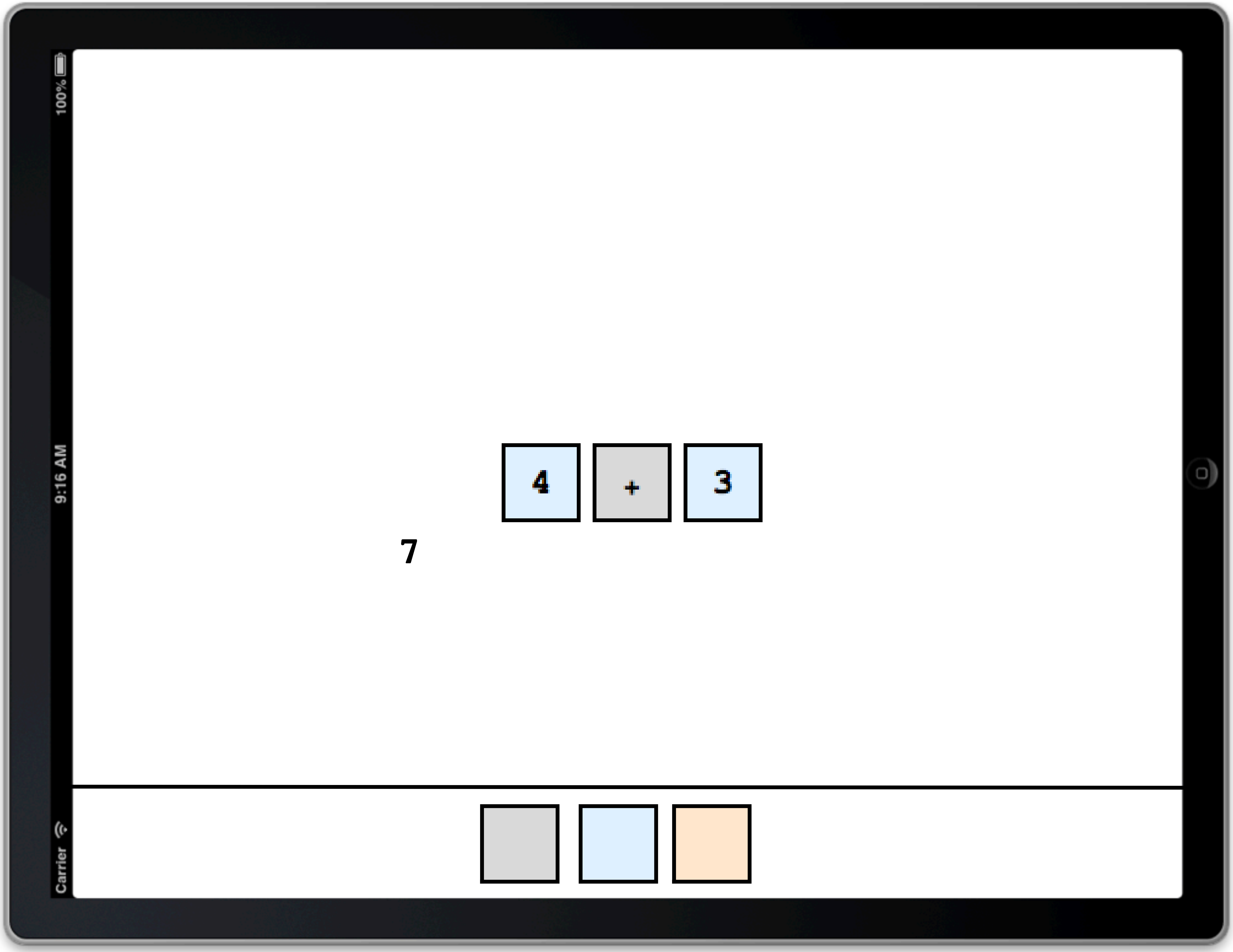

100%
9:16 AM
Carrier
+
3

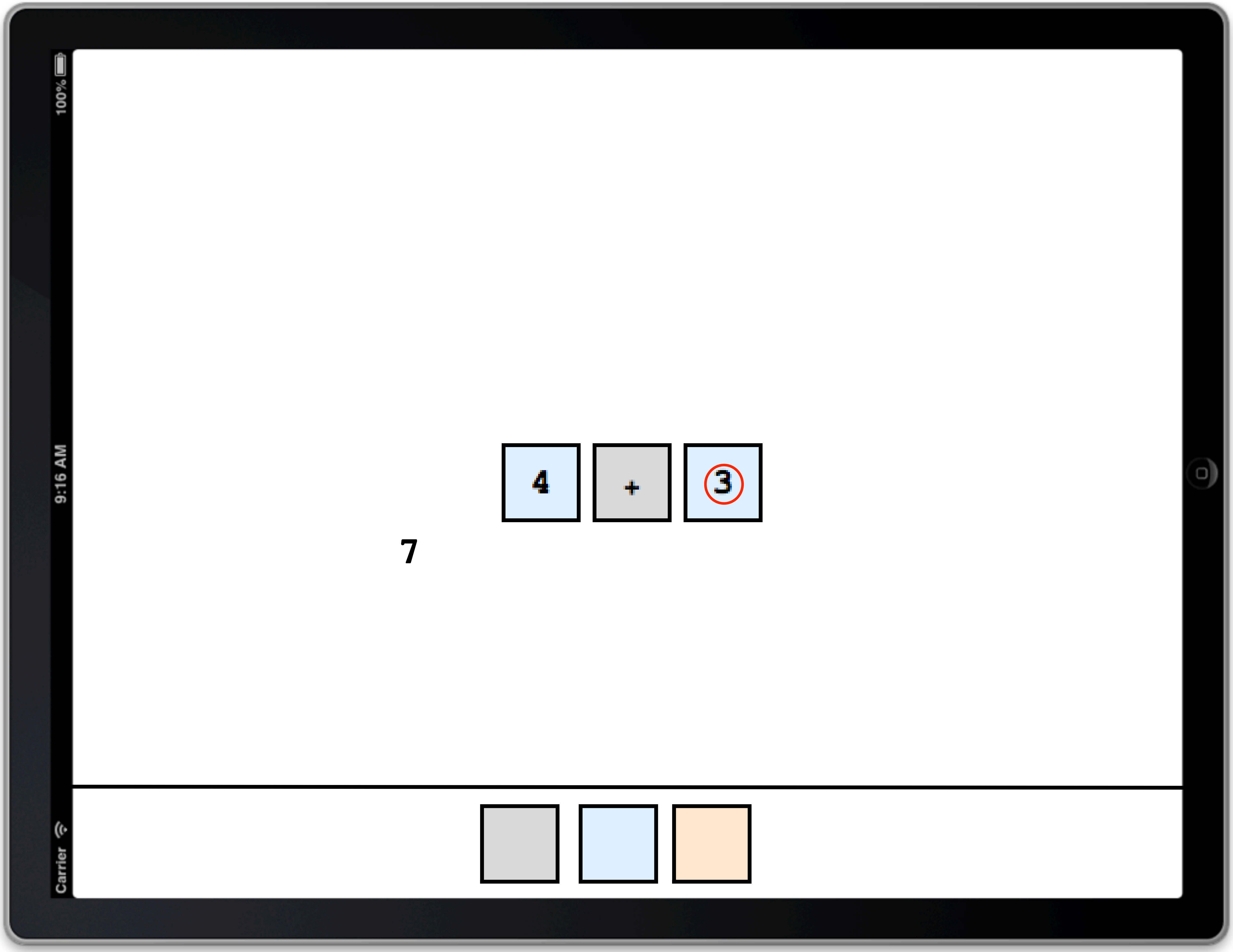

Carrier
9:16 AM
100%
+
3

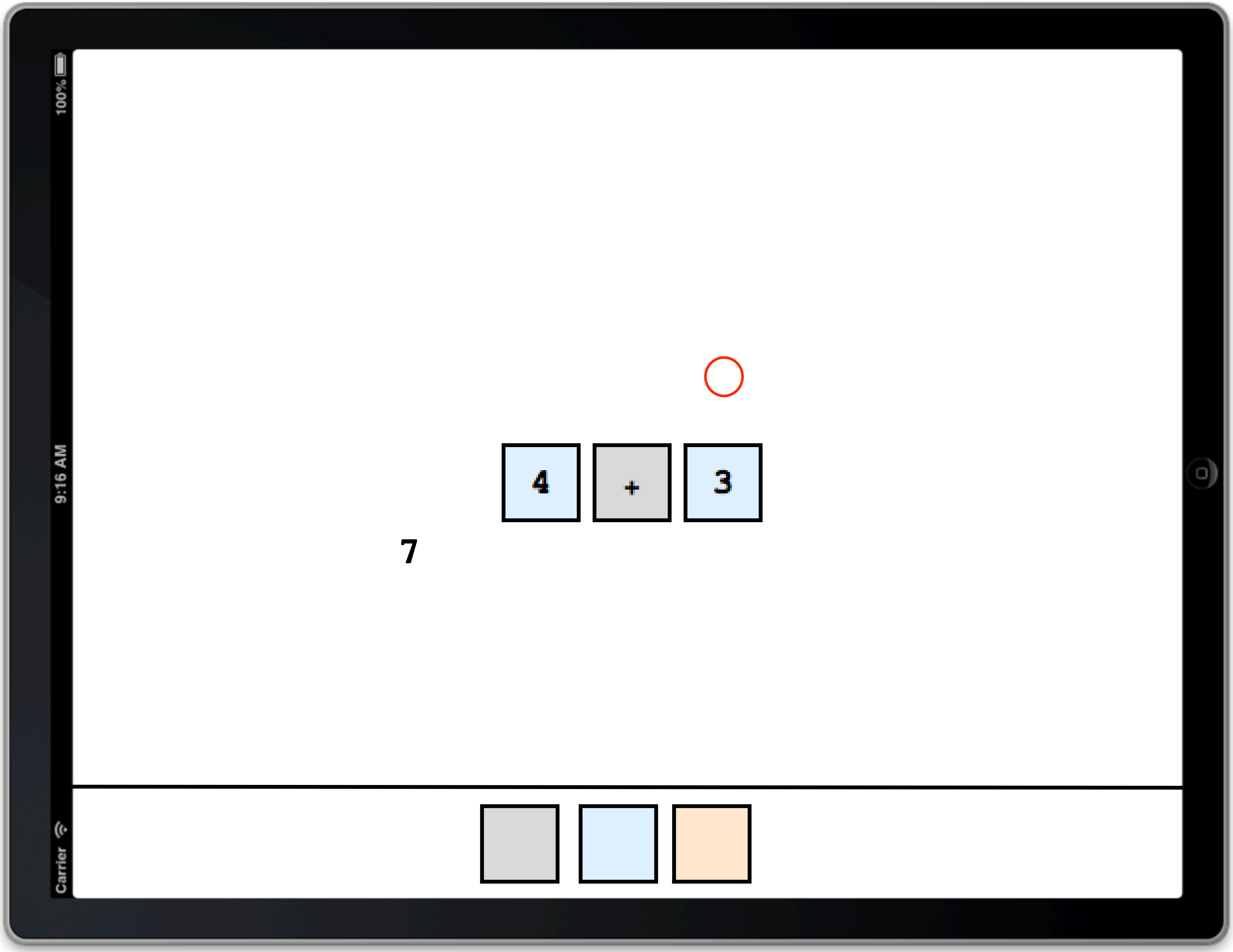

100%
9:16 AM
Carrier
+
3

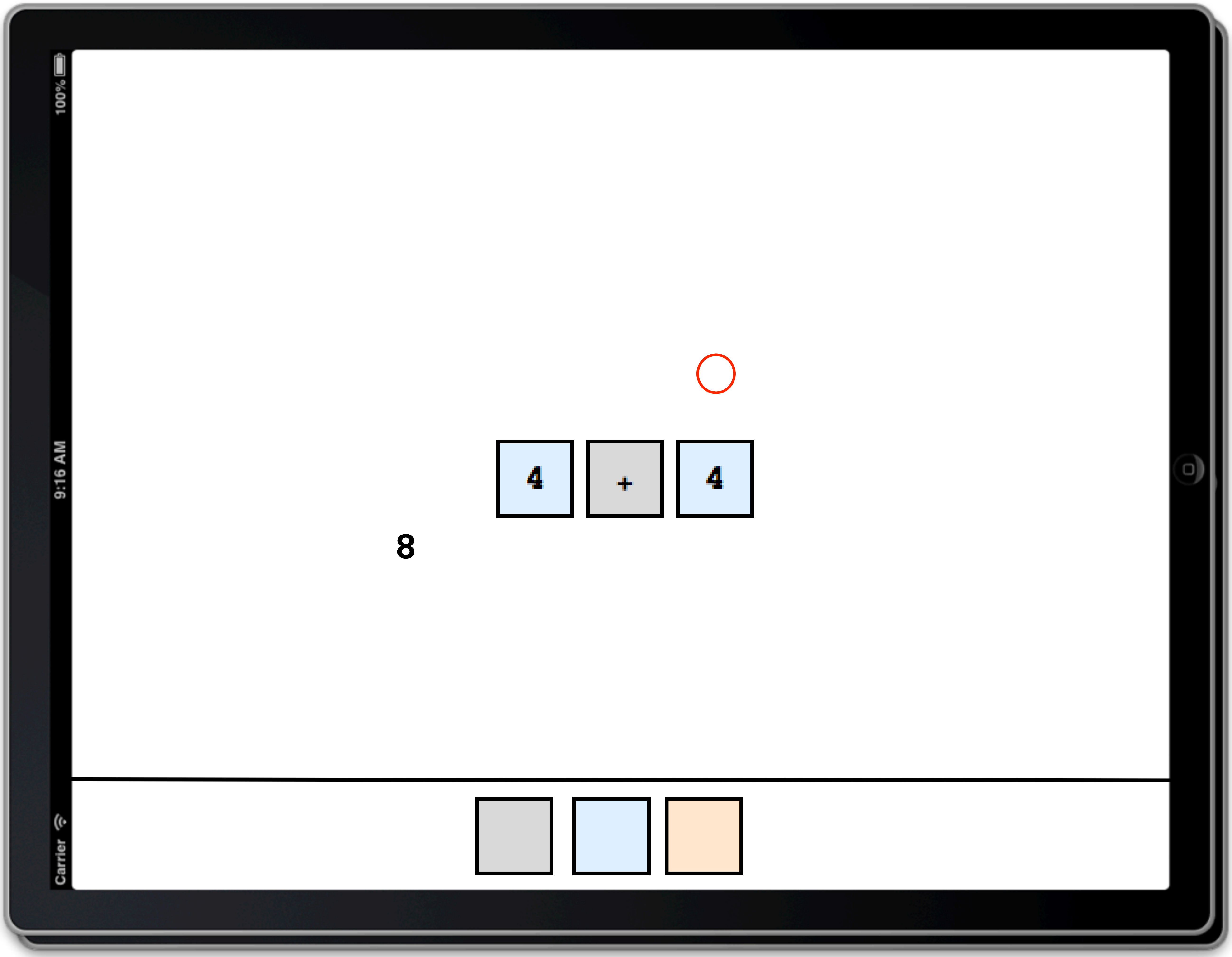

100%
9:16 AM
Carrier
0
+
3
3

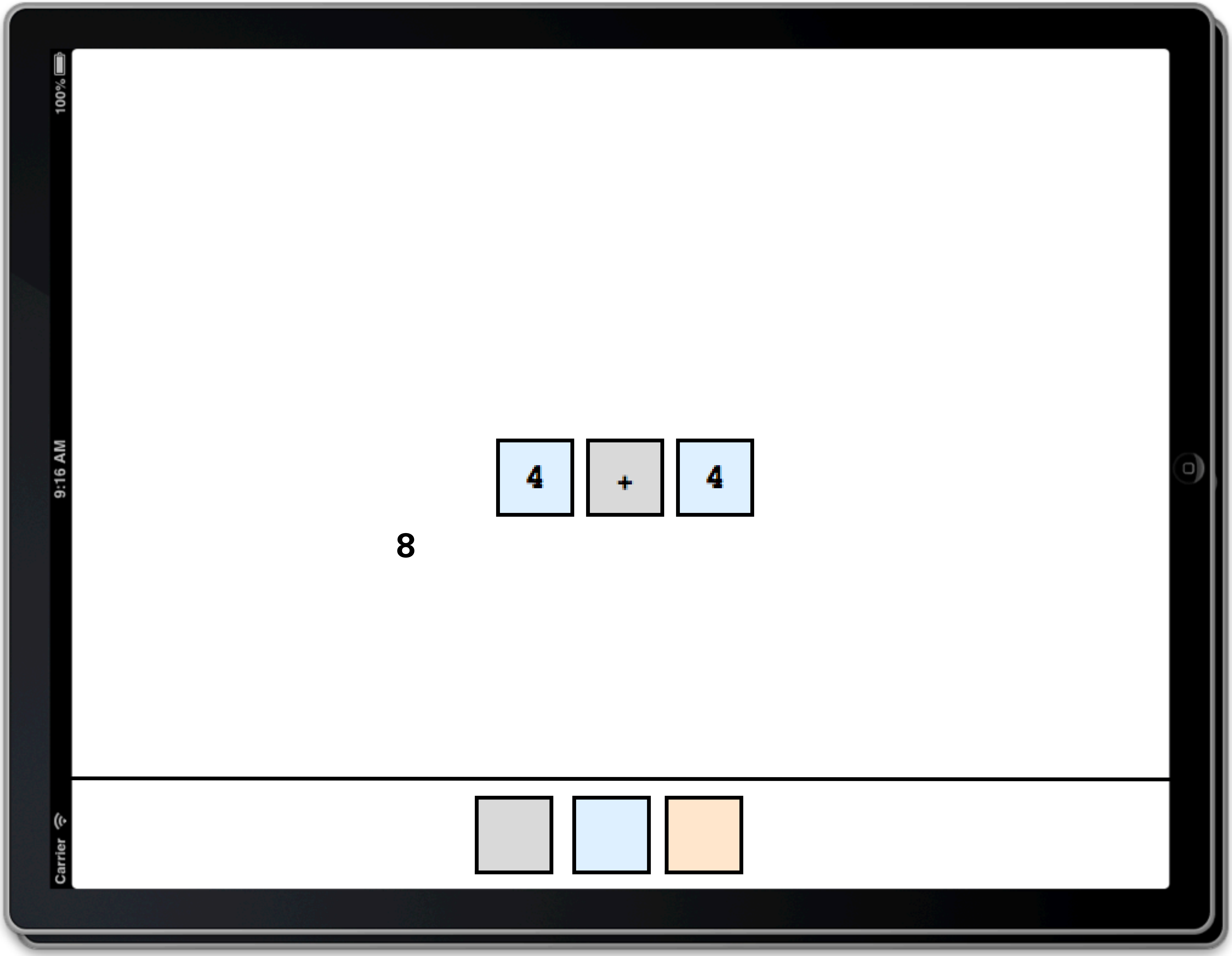

100%
9:16 AM
Carrier
4
+
3
7

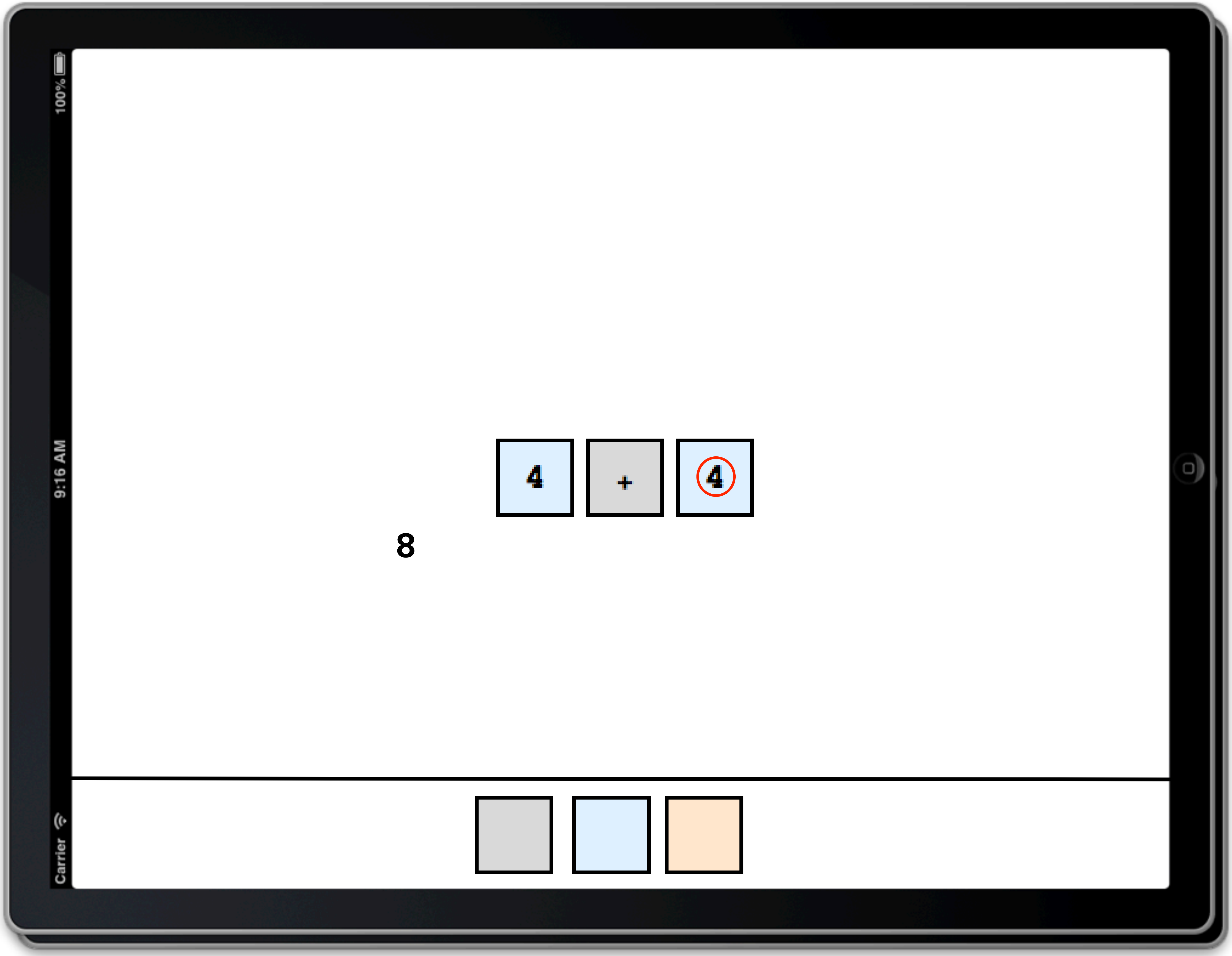

100%
9:16 AM
Carrier
4
+
3
7

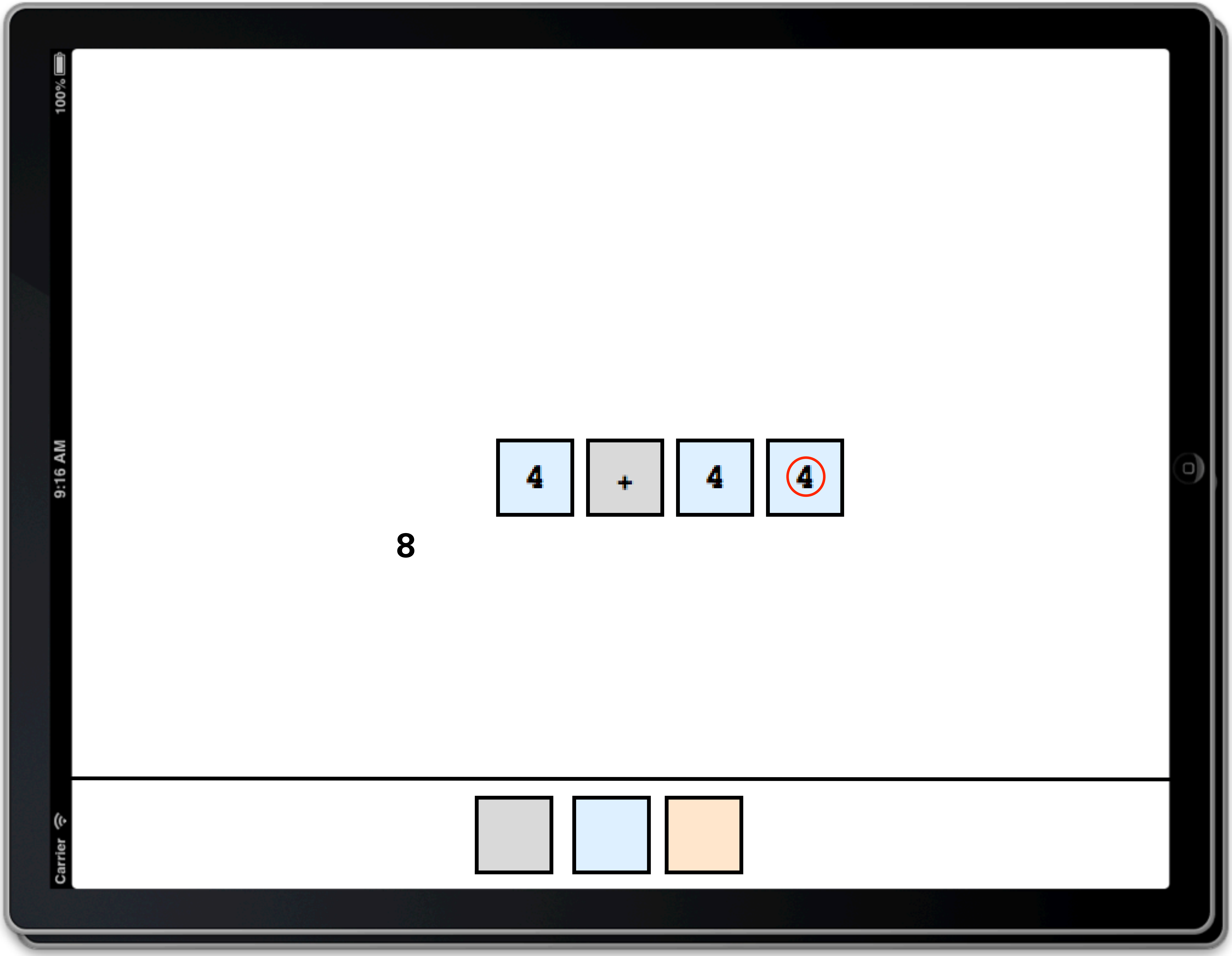

100%
9:16 AM
Carrier
4
+
3
7

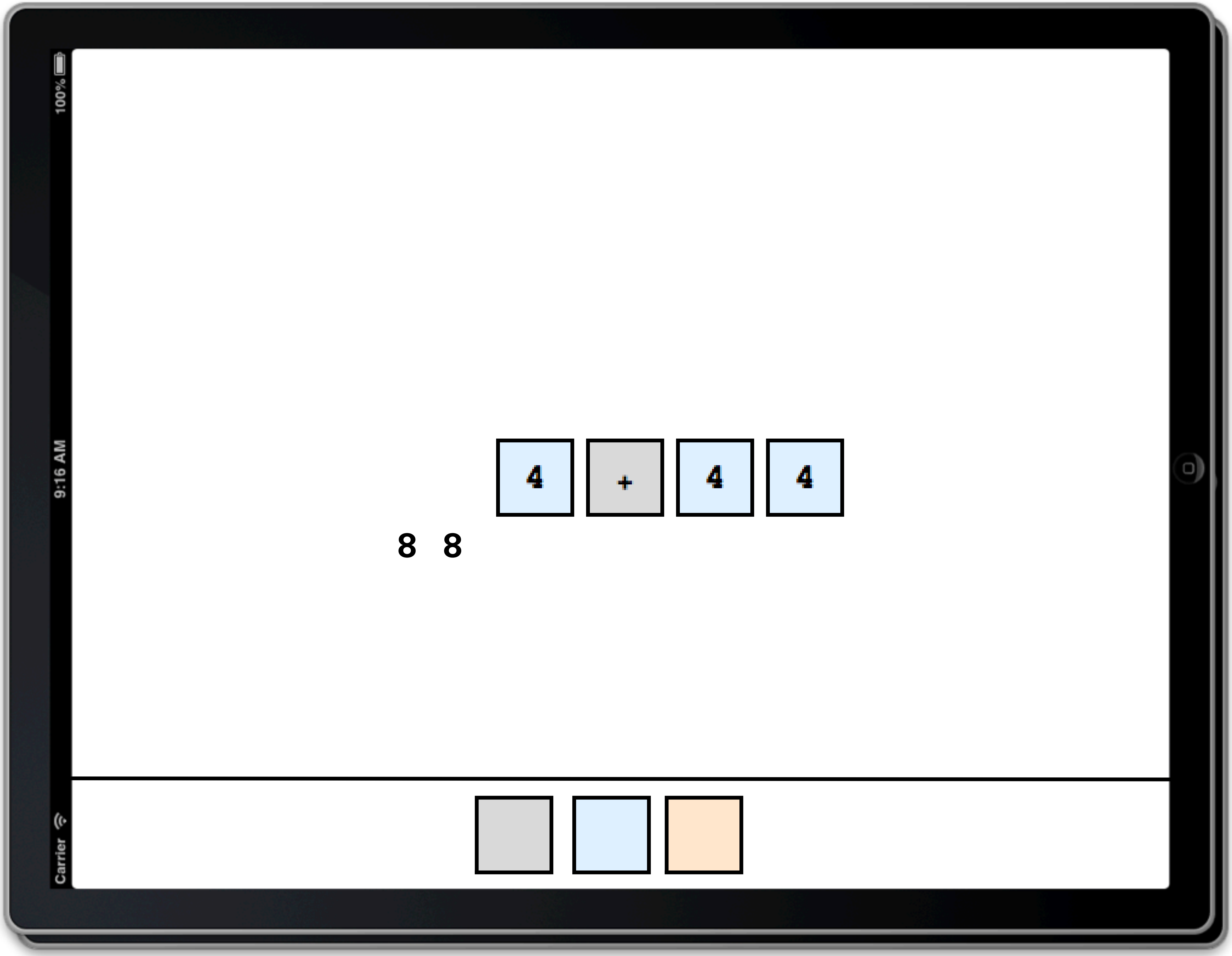

100%
9:16 AM
Carrier
4
+
3
7

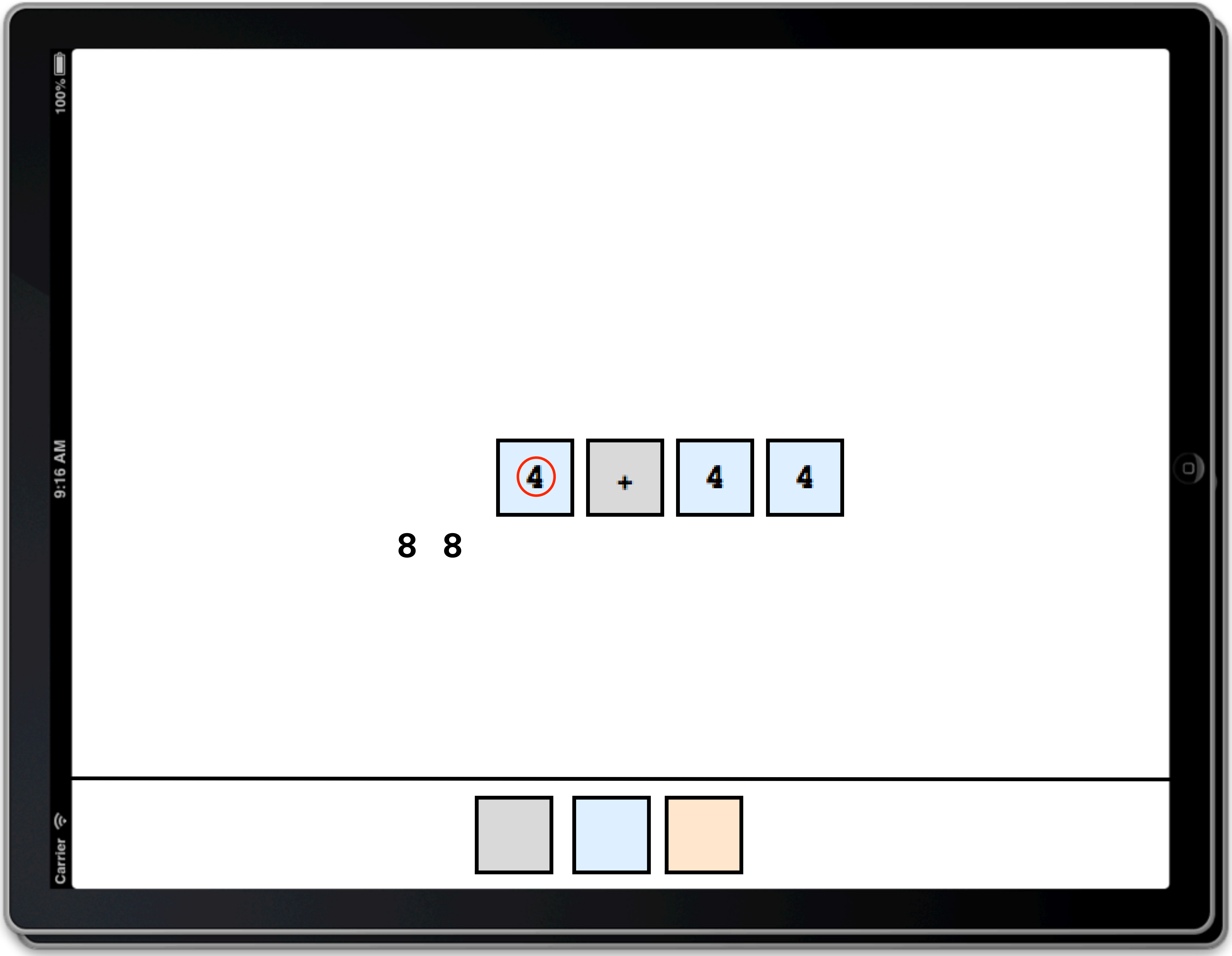

100%
9:16 AM
Carrier
4
+
4
8


100%
9:16 AM
Carrier
4
+
4
8

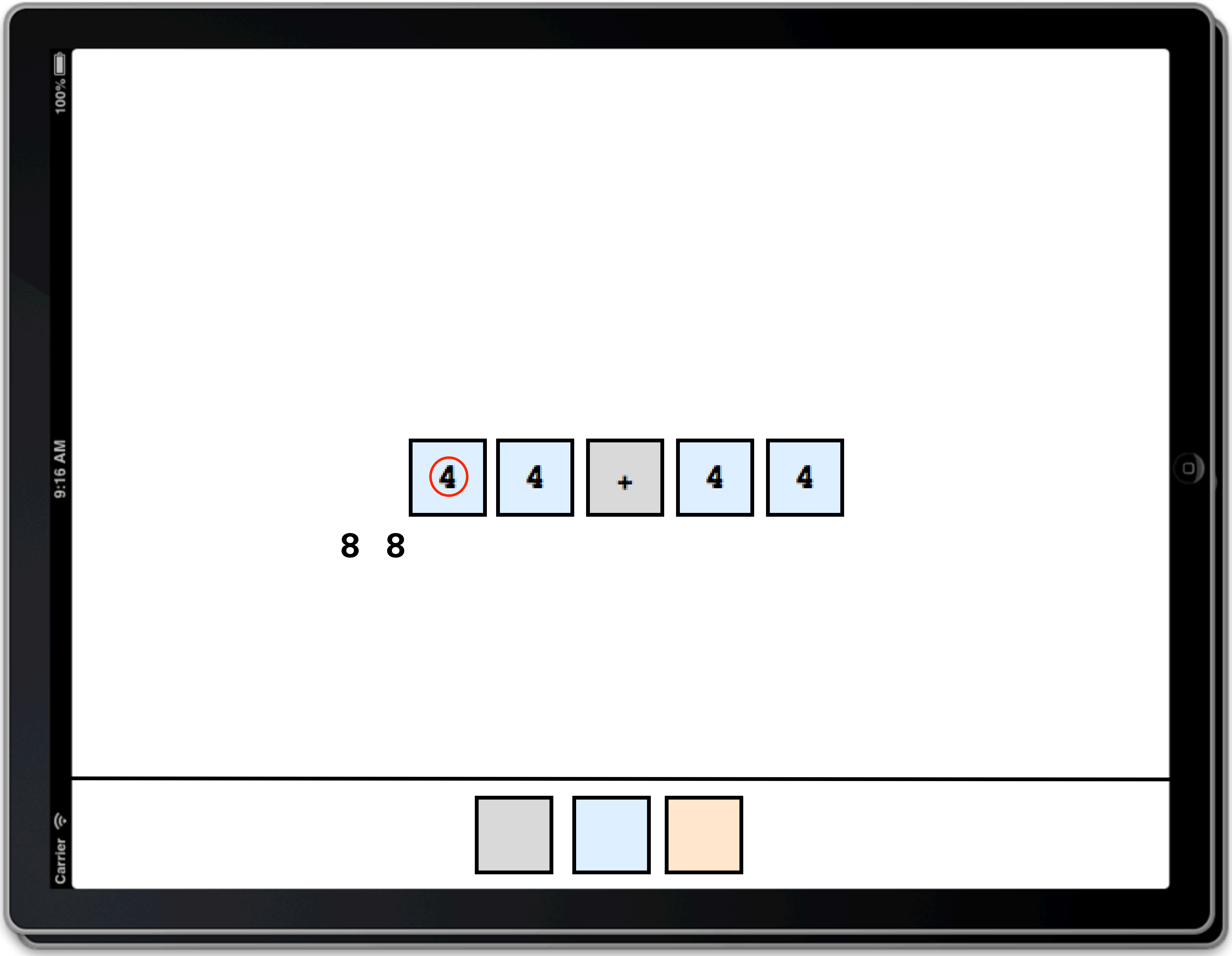

100%
9:16 AM
Carrier
4
+
4
8

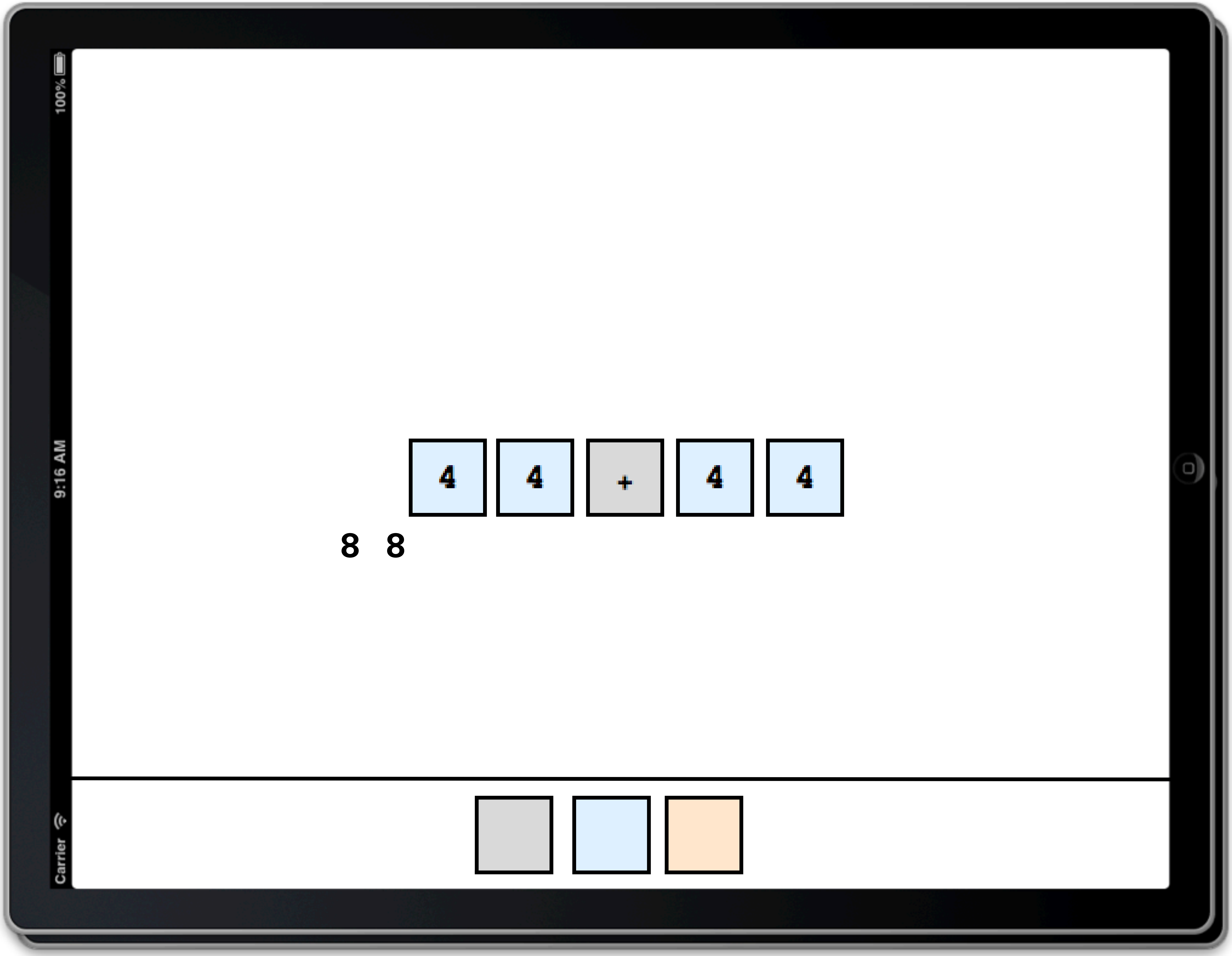

100%
9:16 AM
Carrier
4
+
4
4
8

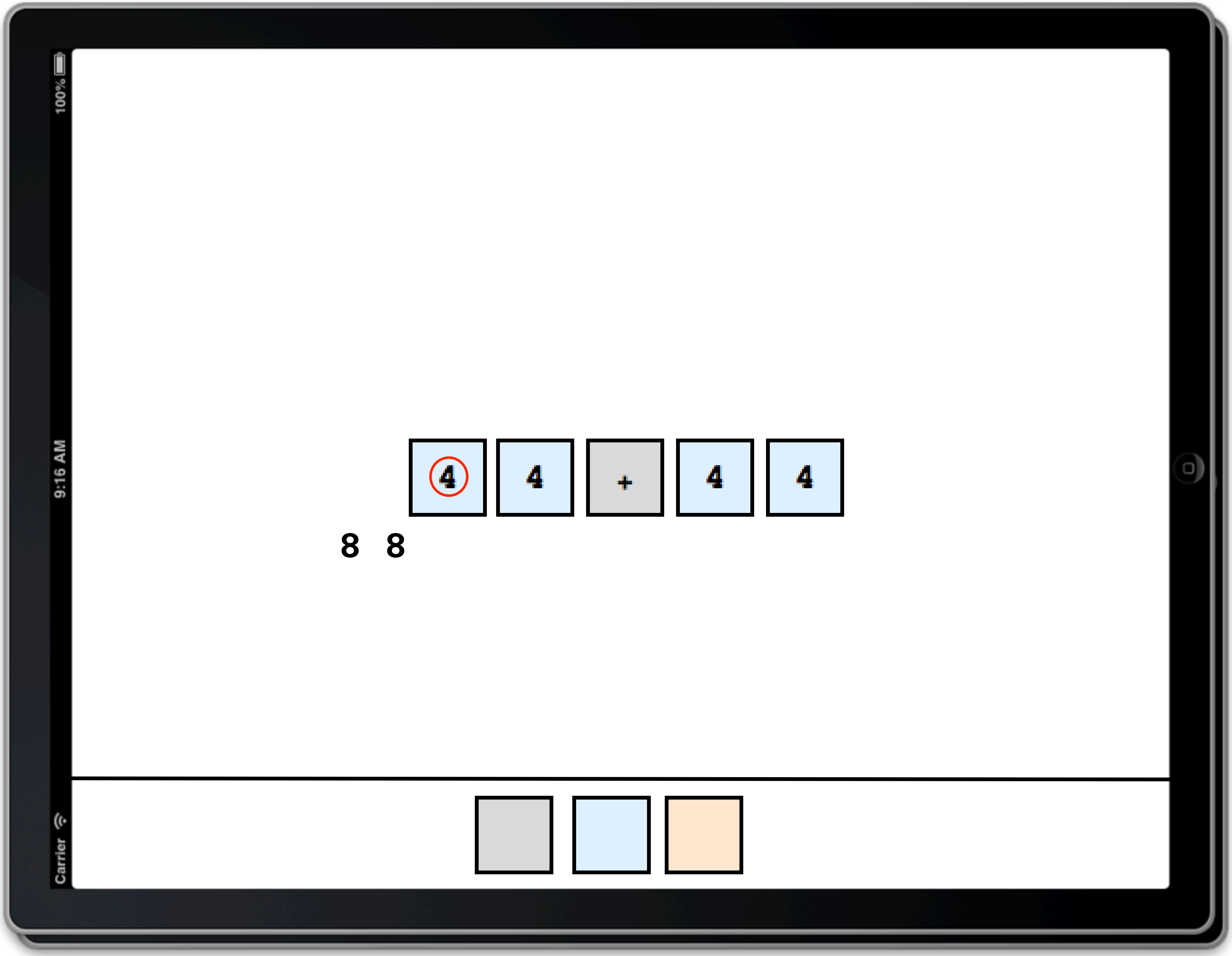

100%
9:16 AM
Carrier
4
+
4
4
8 8



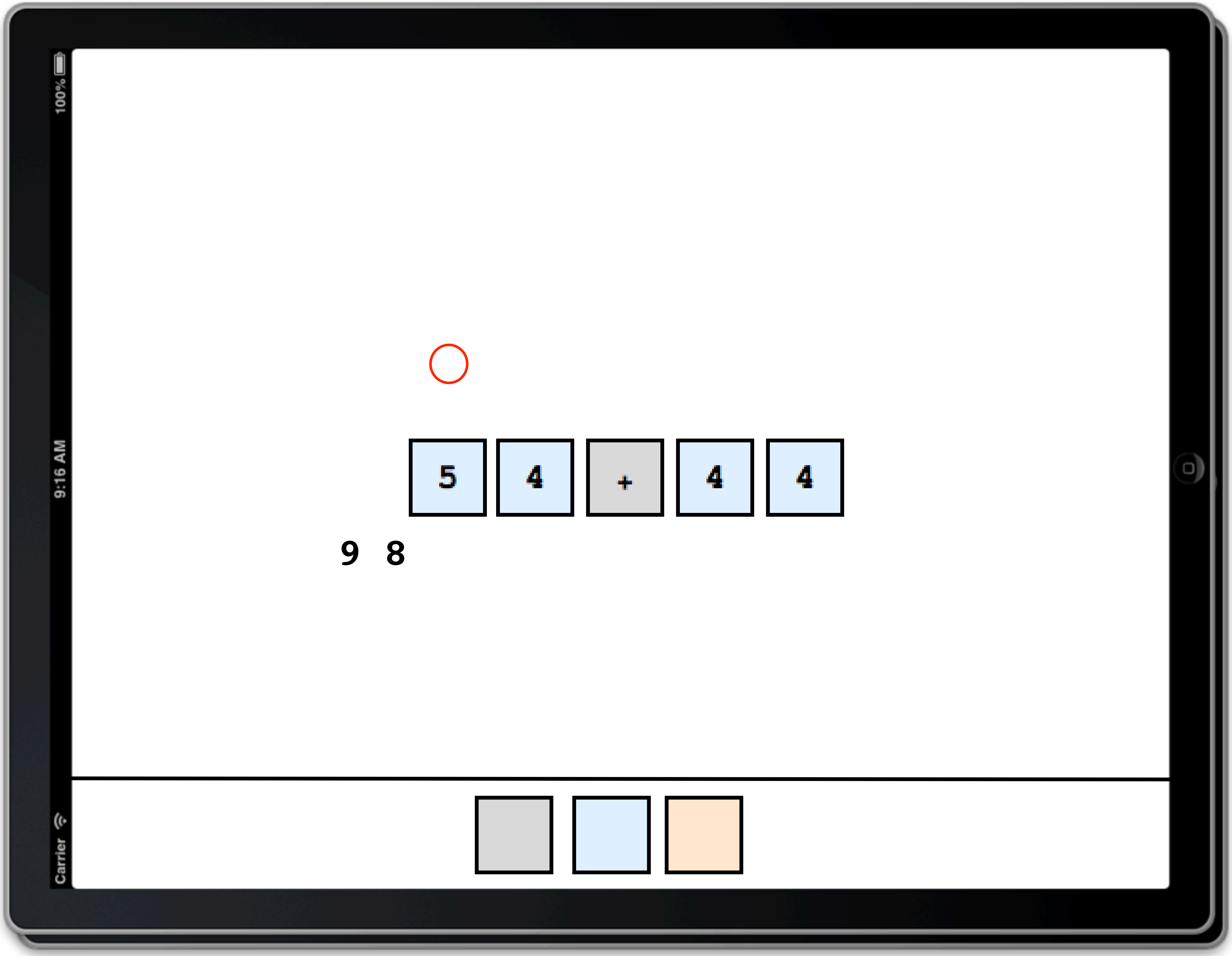
Monday, October 1, 12

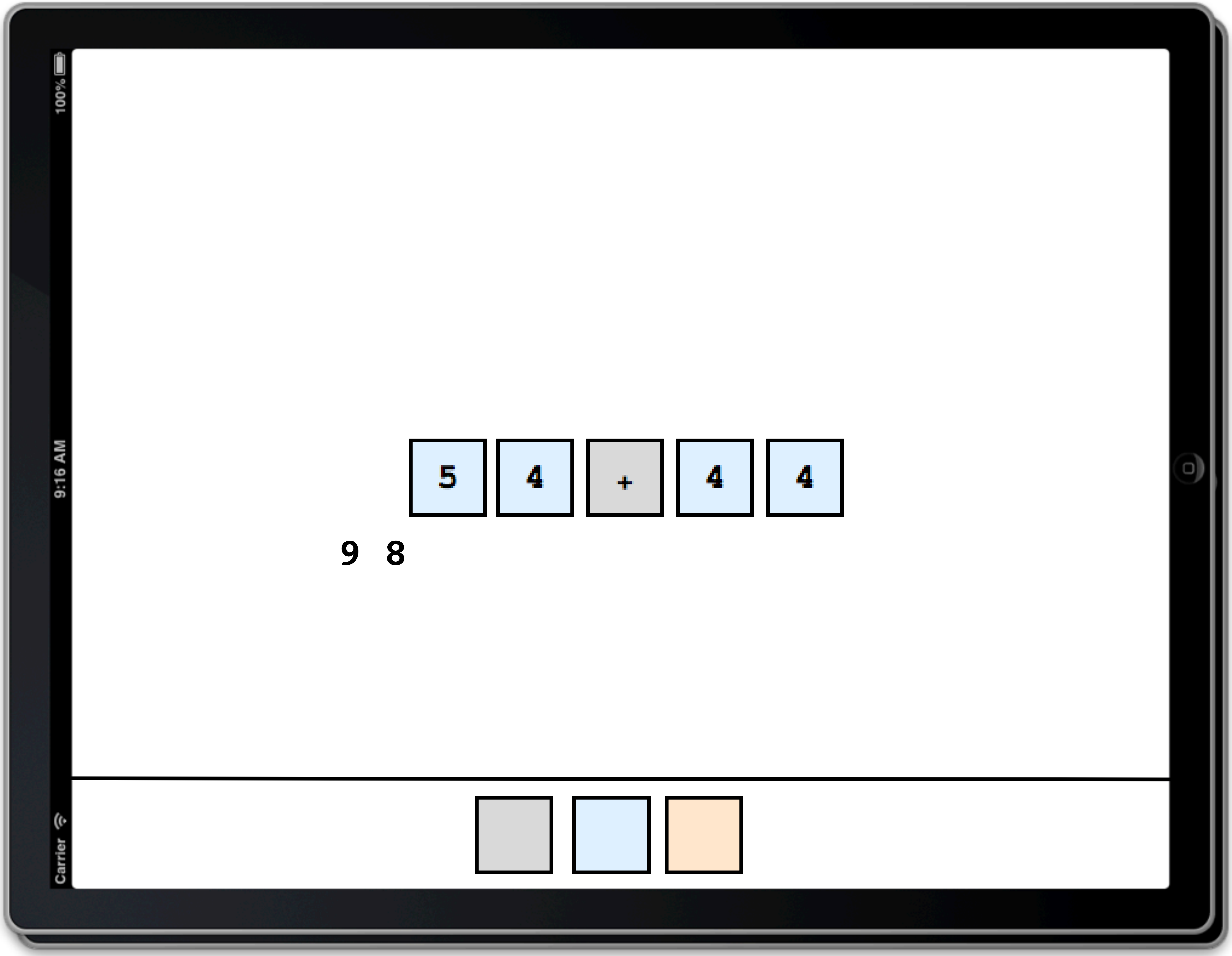

100%
9:16 AM
Carrier
4
4
+
4
4
8
8

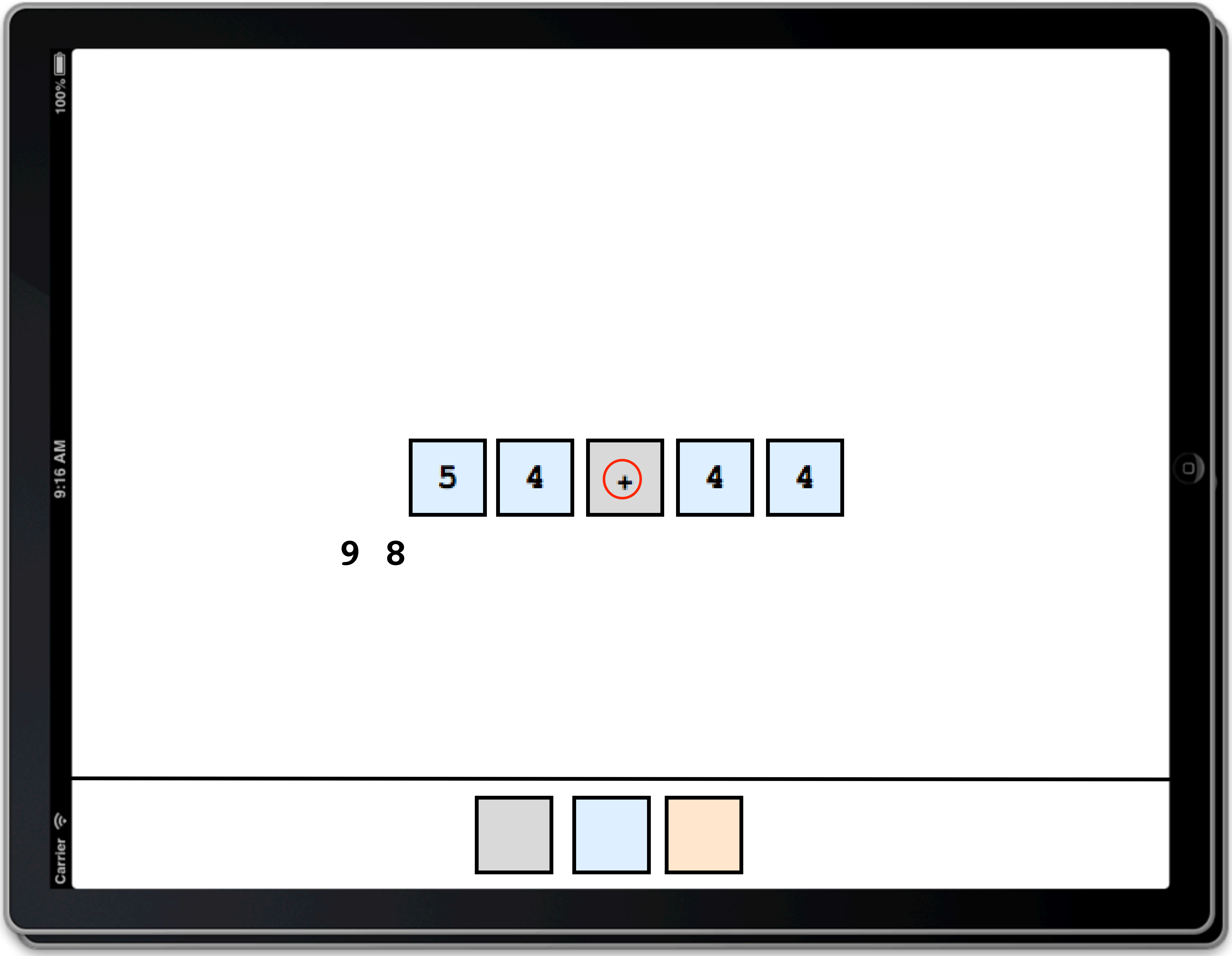

100%
9:16 AM
Carrier
4
4
+
4
4
8
8

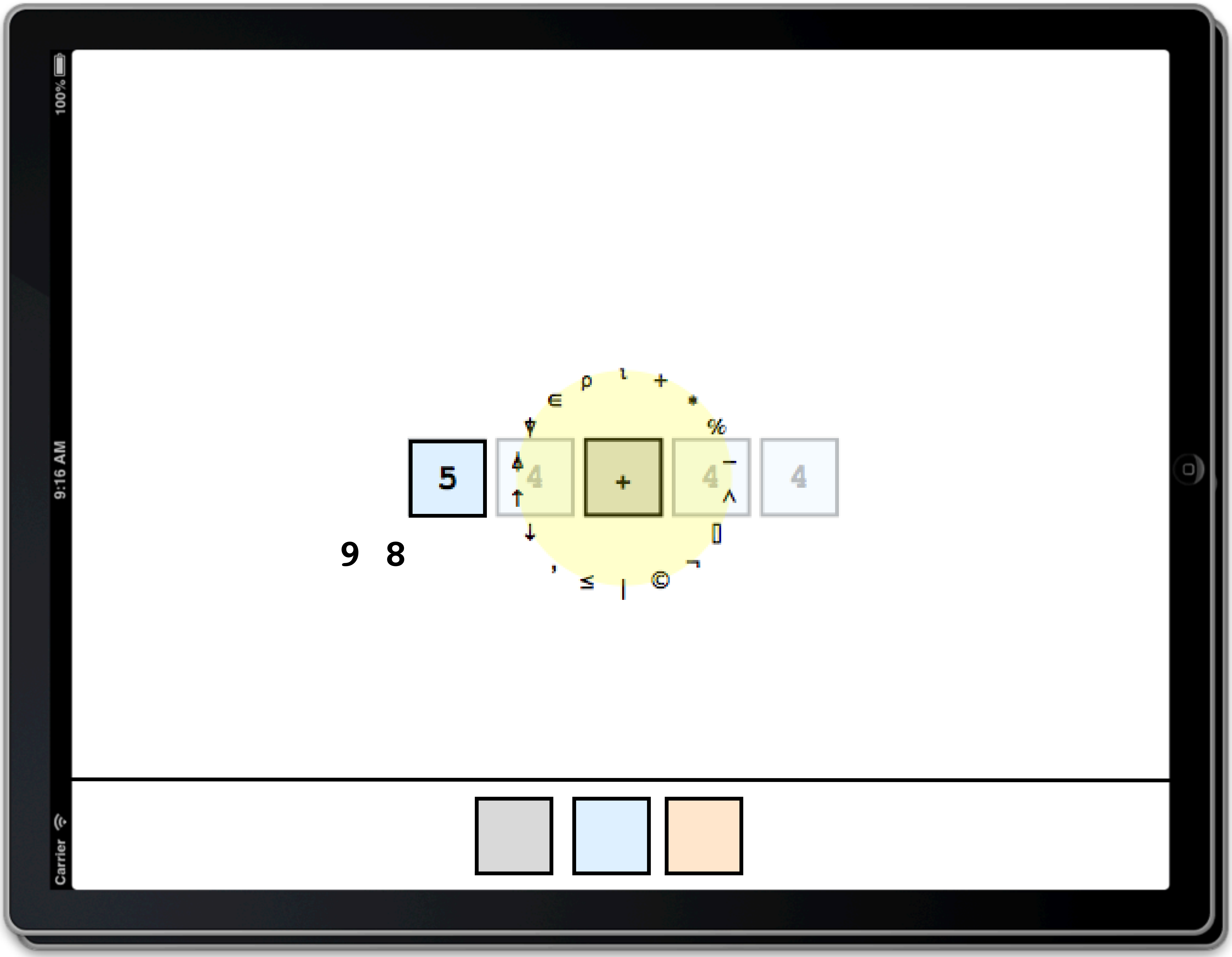


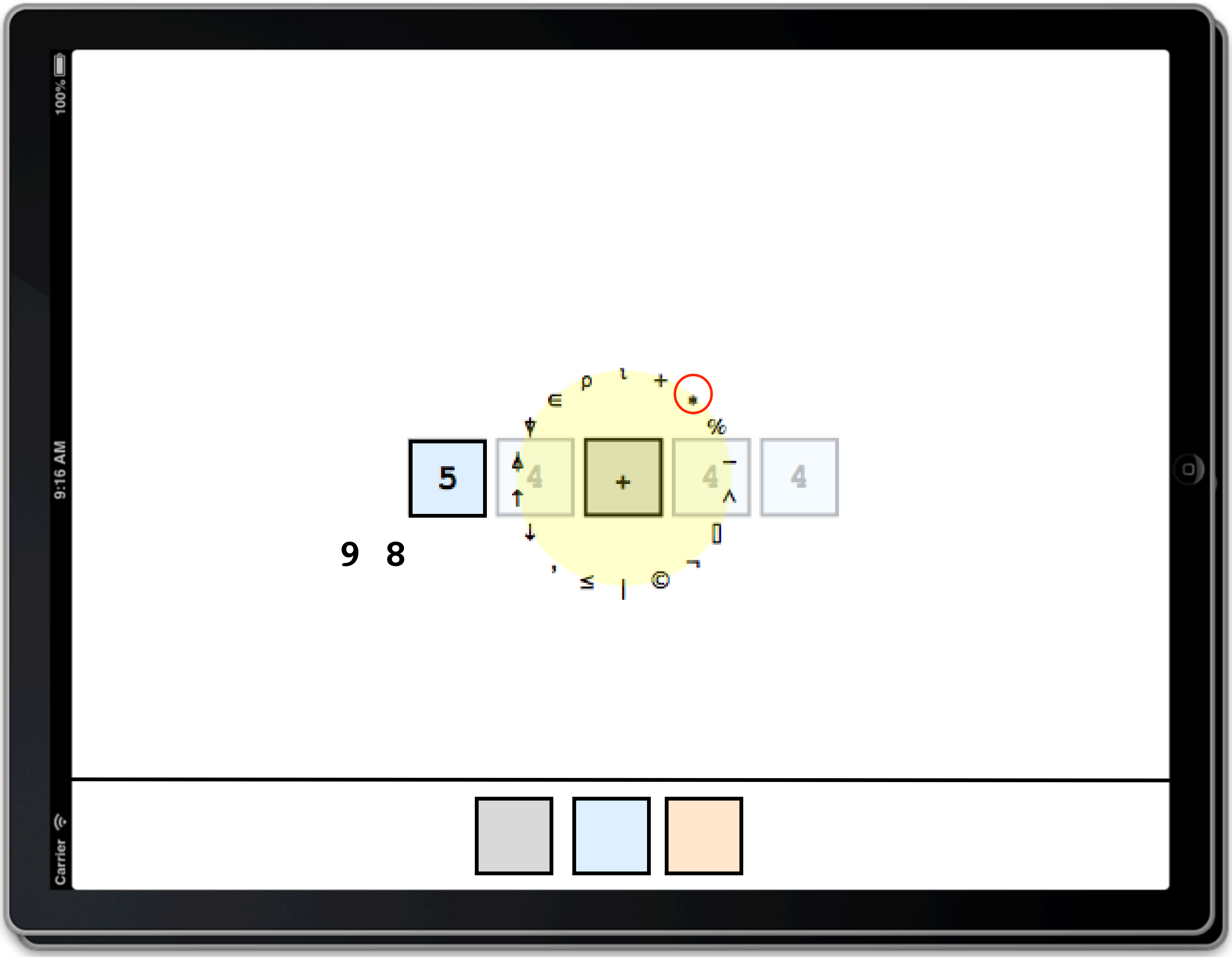
100%
9:16 AM
Carrier
4
4
+
4
4
8 8

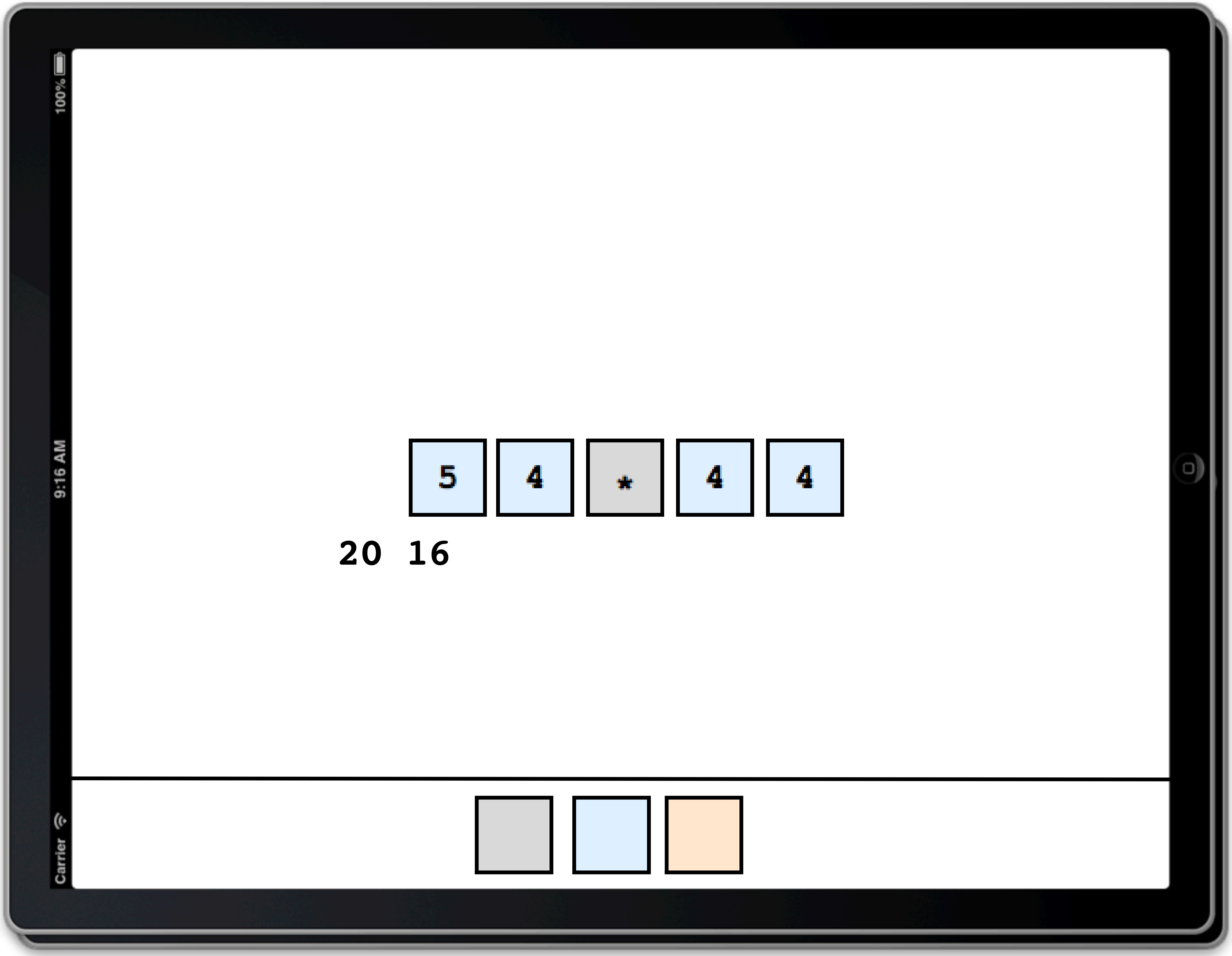

100%
9:16 AM
Carrier
5
4
+
4
4
9
8

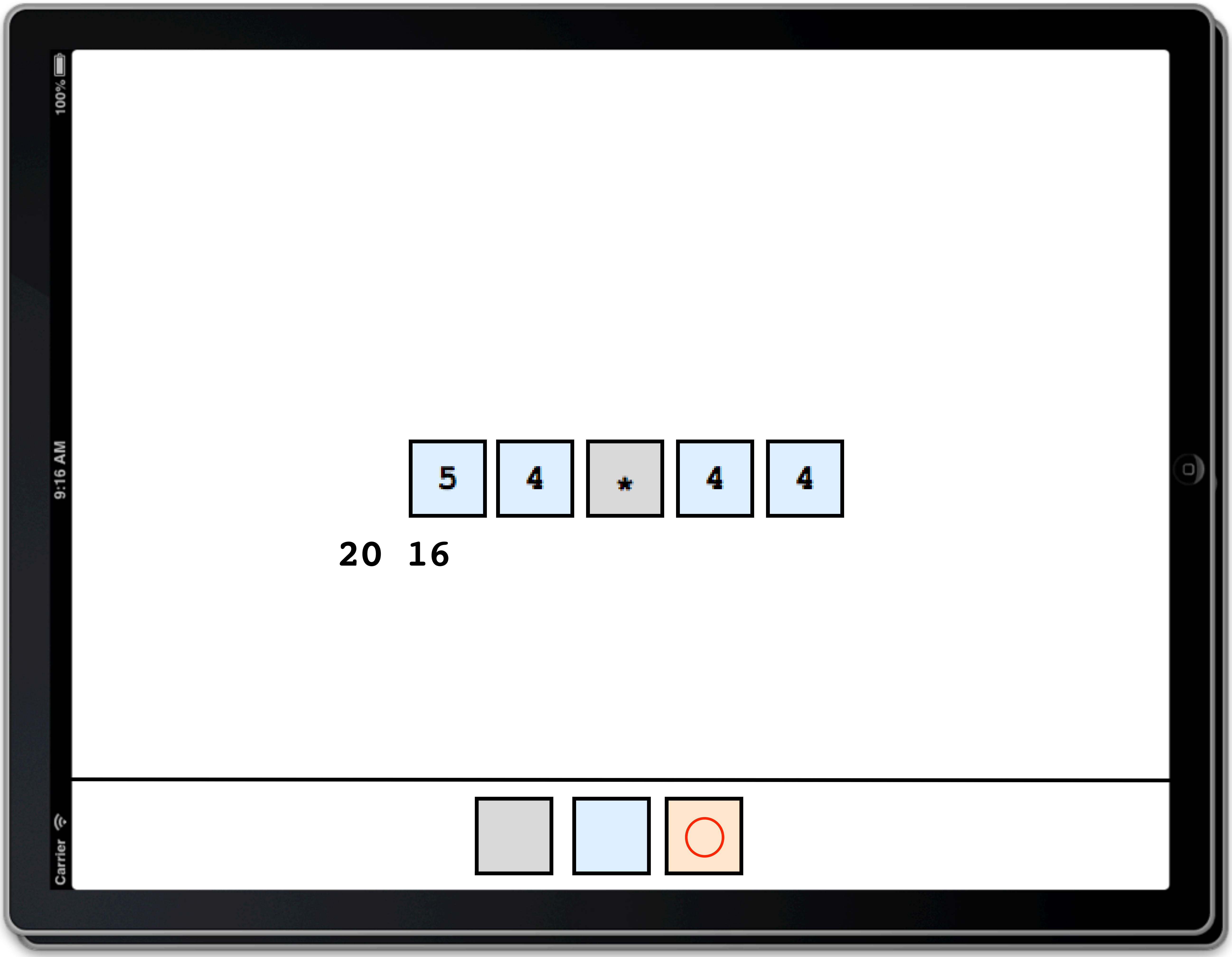

100%
9:16 AM
Carrier
5
4
+
4
4
9
8


100%
9:16 AM
Carrier
5
4
+
4
4
9
8

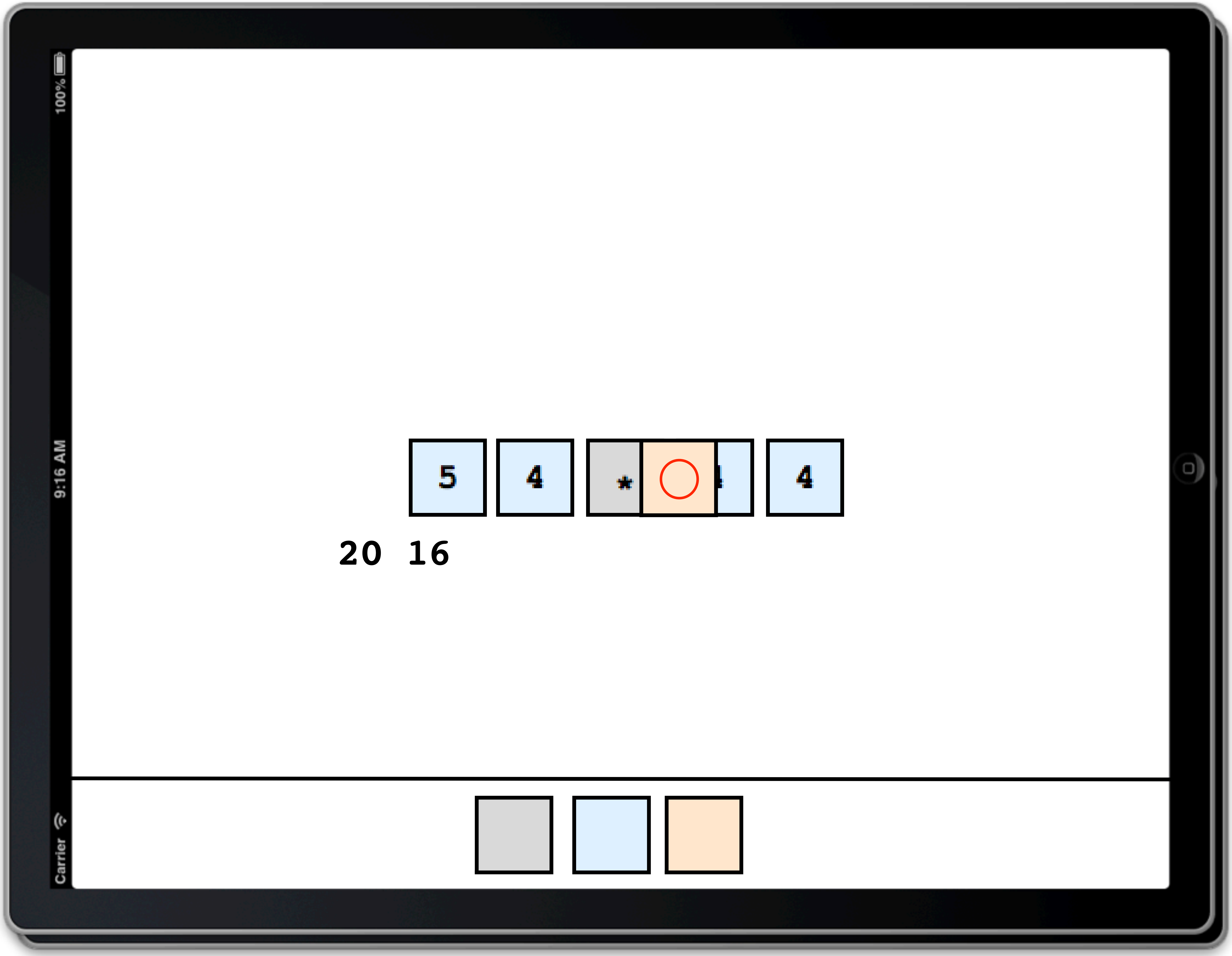

100%
9:16 AM
Carrier
5
4
+
4
4
9
8
∊
ρ
⍳
+
*
%
⍋
⍒
↑
↓
−
^
⌷
,
≤
|
©
¬

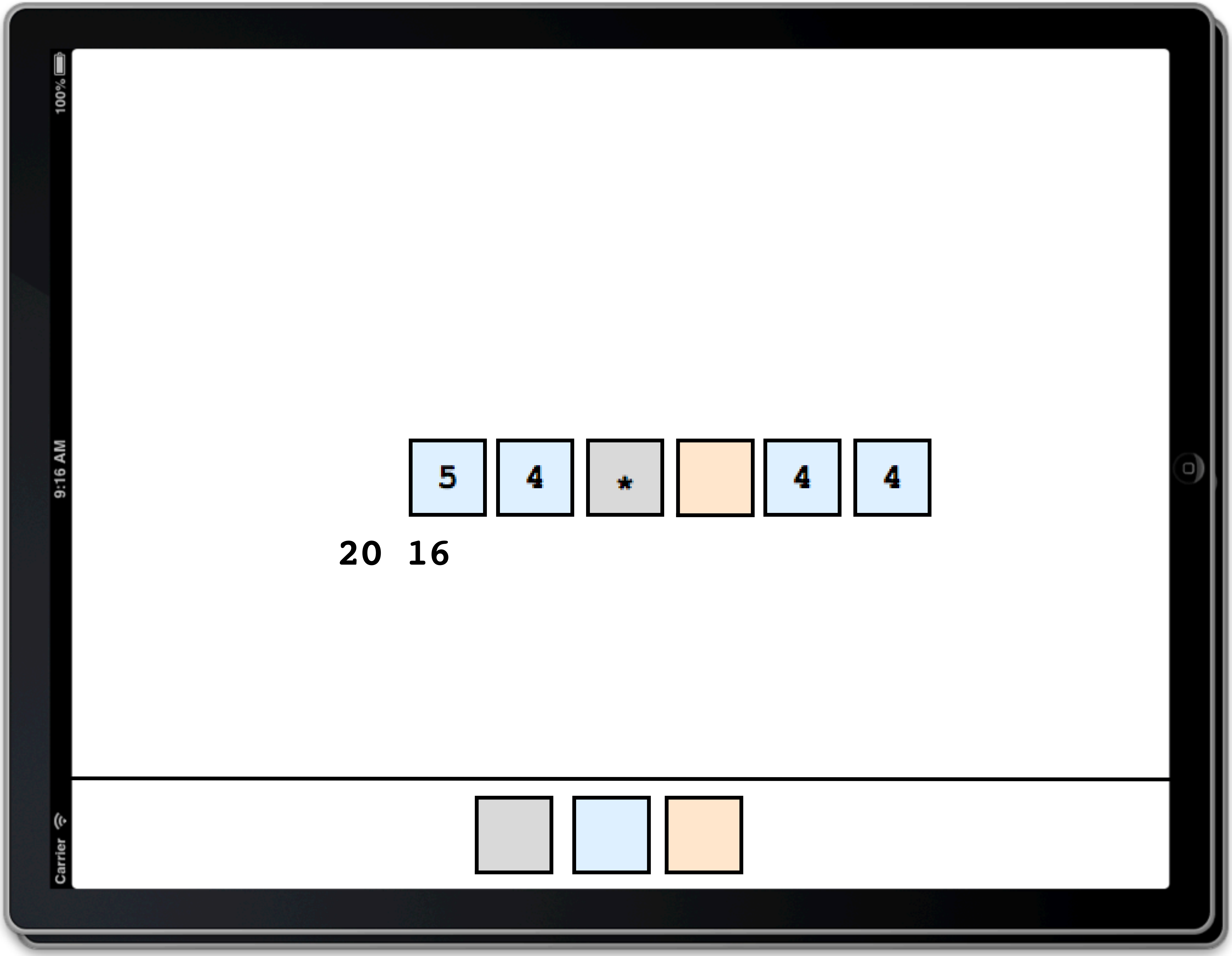

100%
9:16 AM
Carrier
5
4
+
4
4
9 8
∊ ρ ɩ + * % − ^ ⌷ ¬ © | ≤ , ↓ ↑ ⍋ ⍒

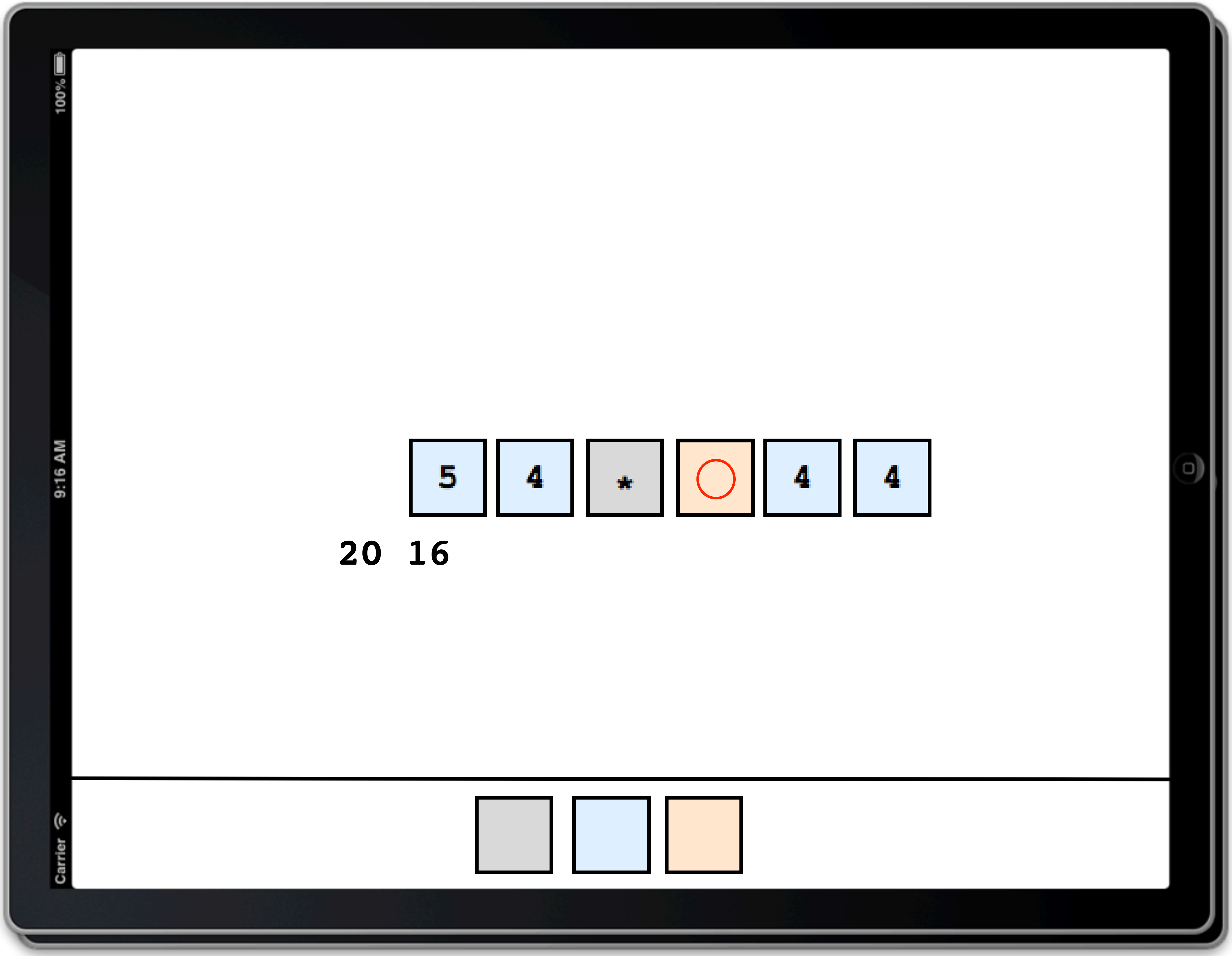

Carrier
9:16 AM
100%
5
4
*
4
4
20
16

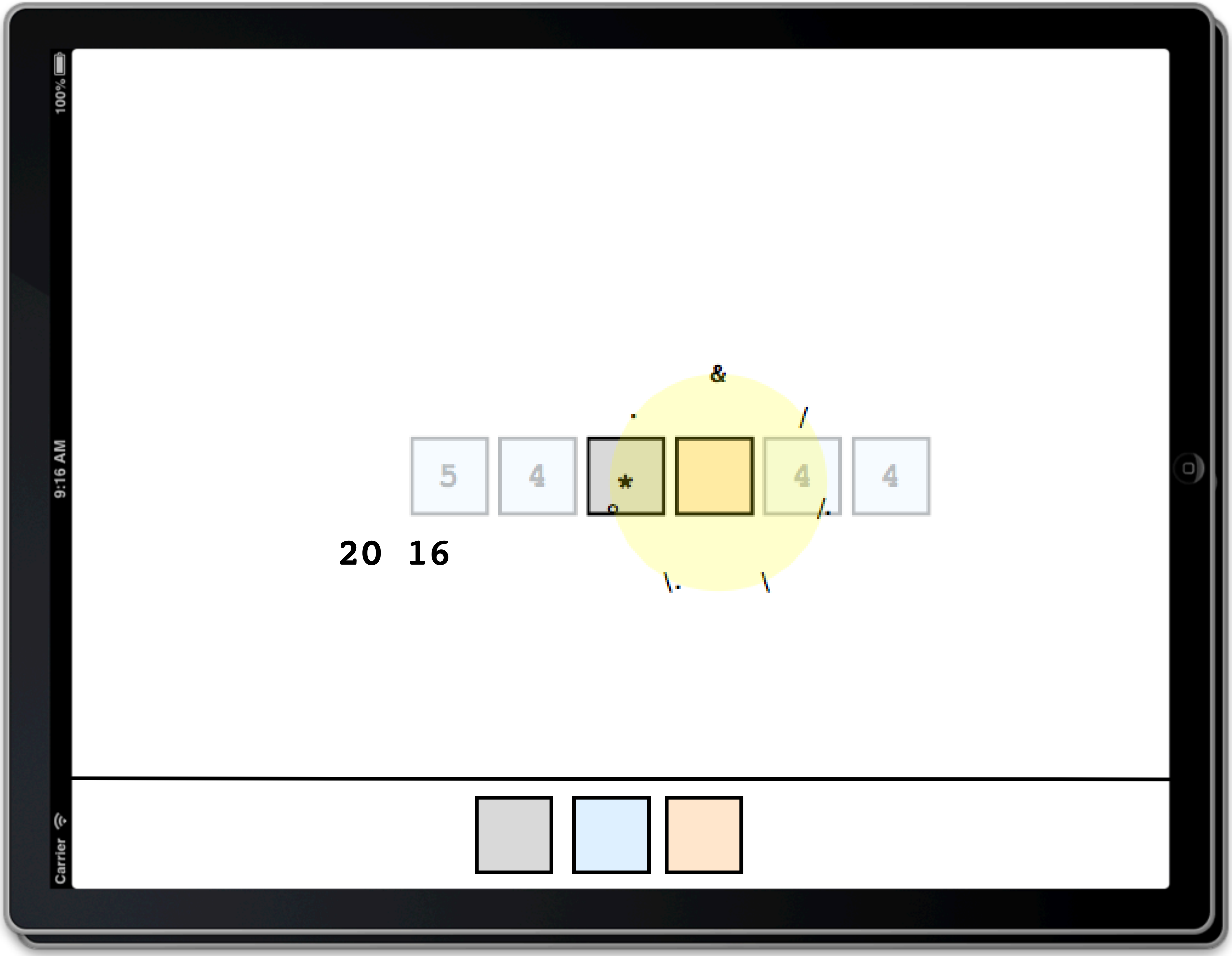


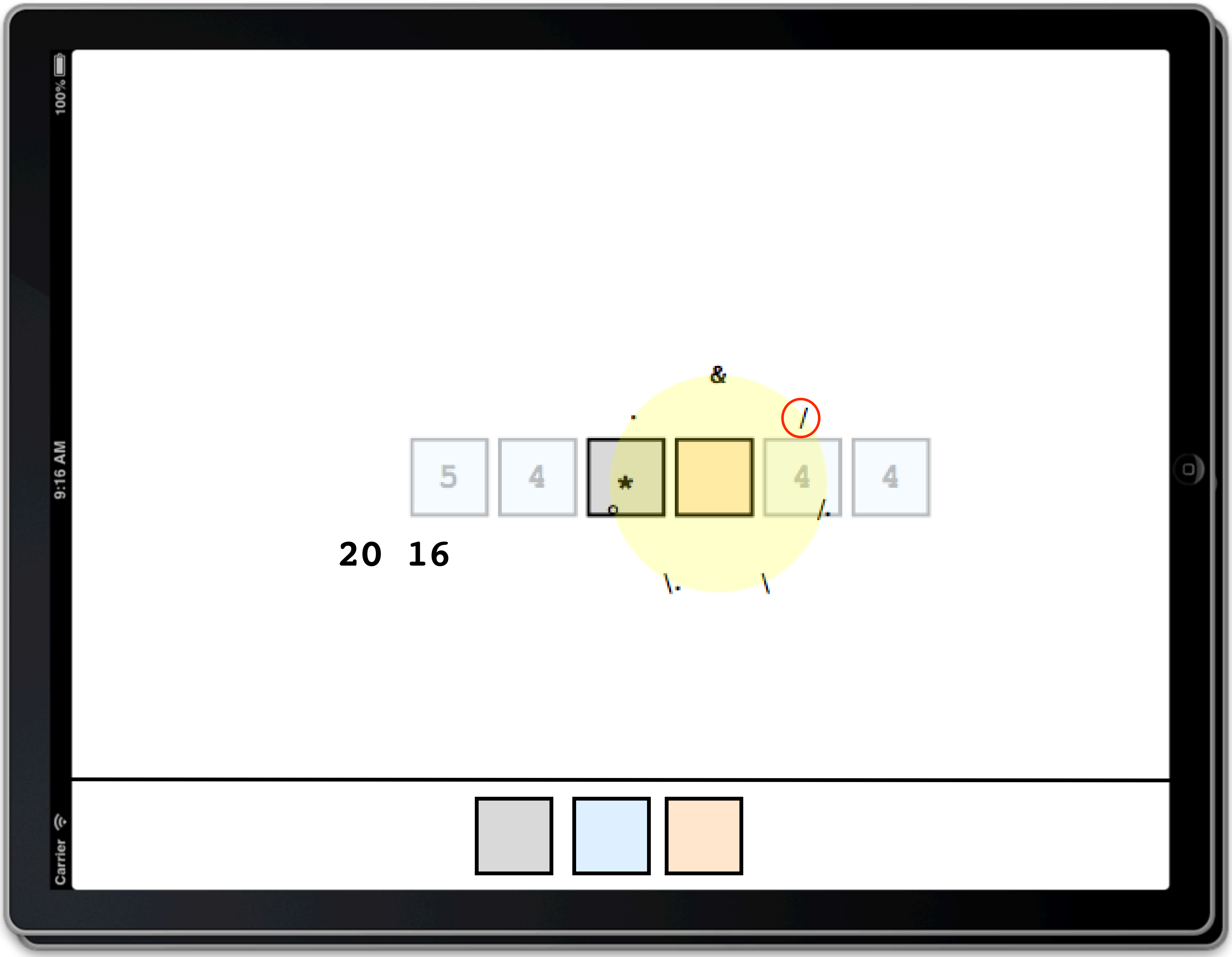
Monday, October 1, 12

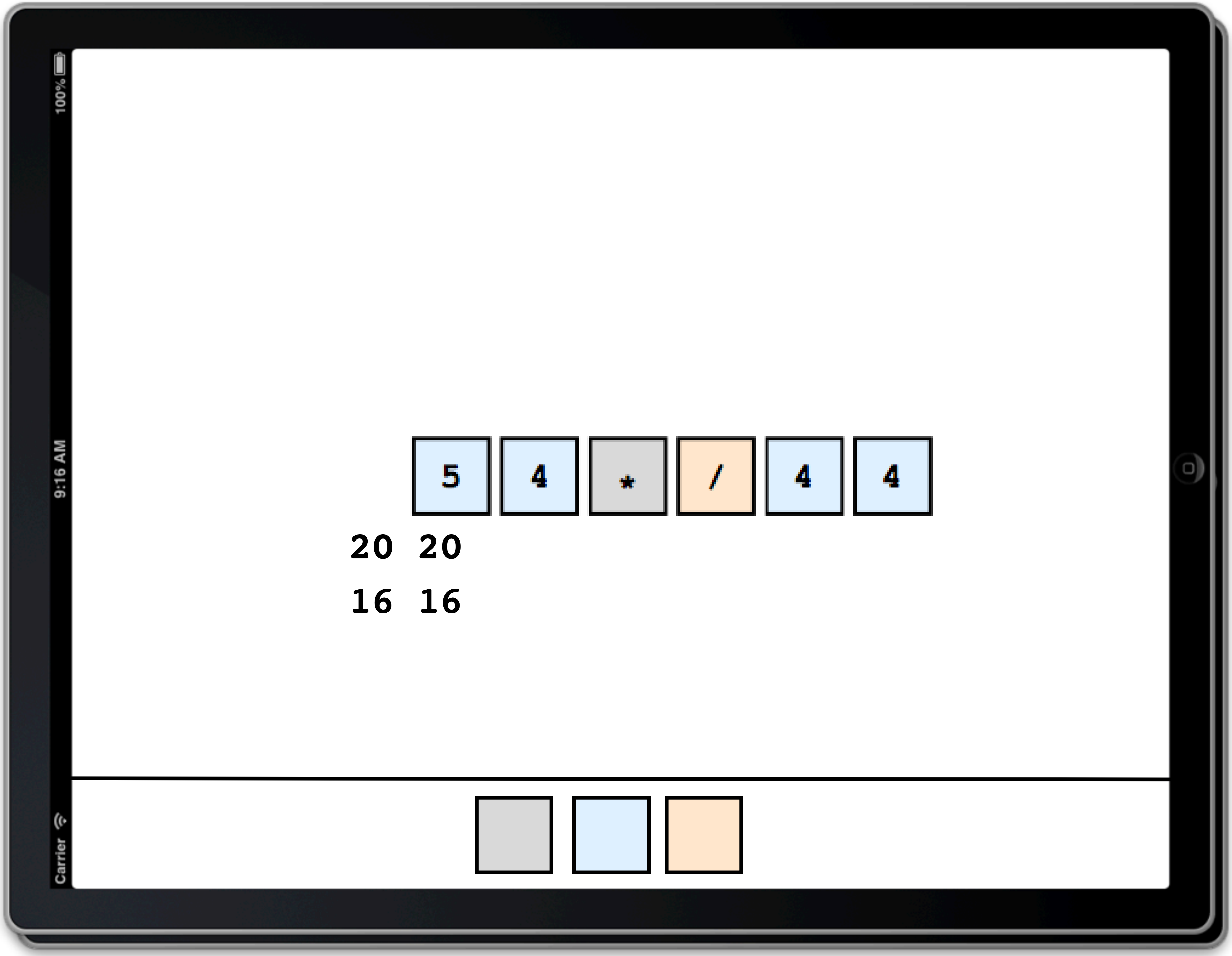

100%
9:16 AM
Carrier
5
4
*
4
4
20
16

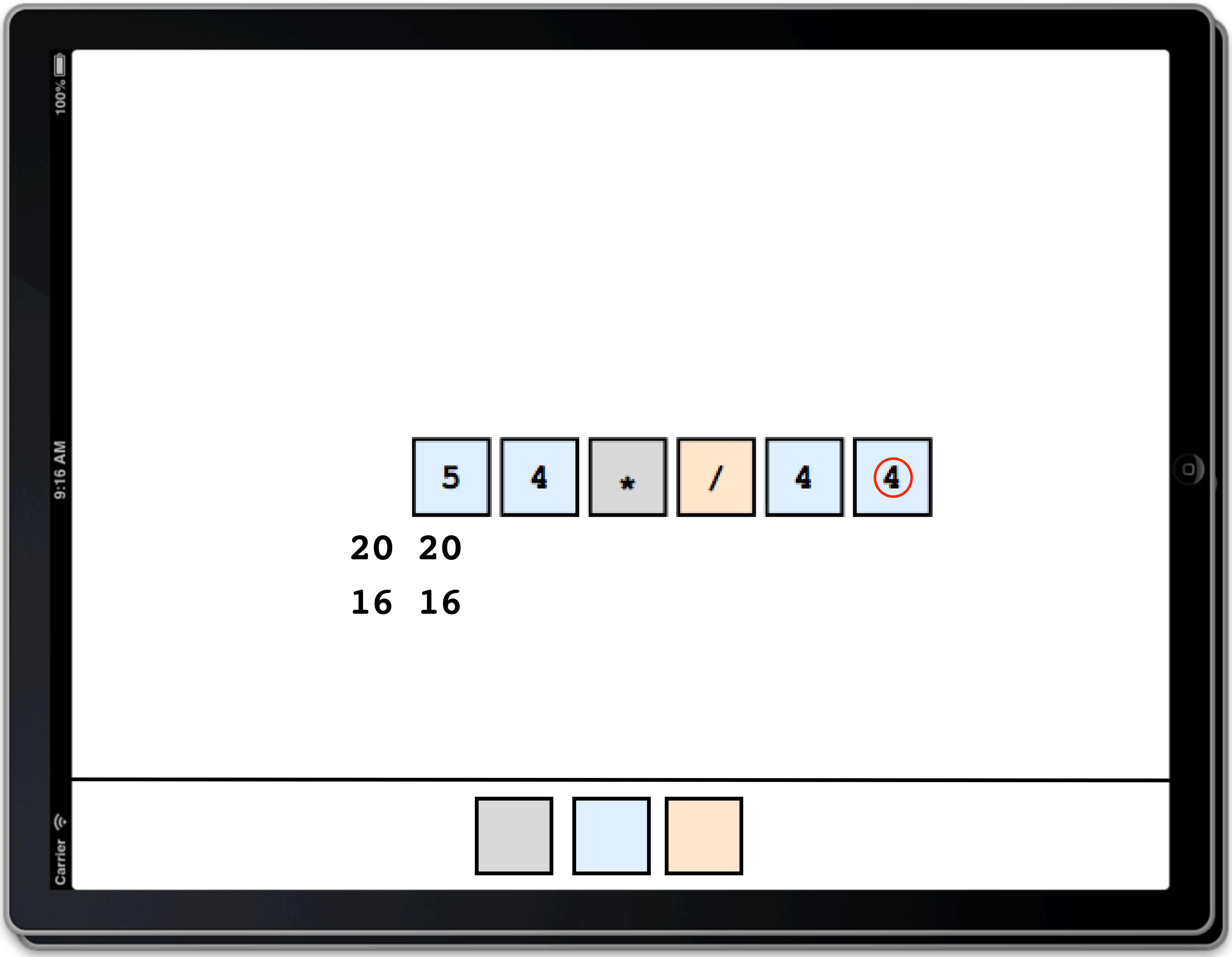

100%
9:16 AM
Carrier
5
4
*
4
4
20 16


100%
9:16 AM
Carrier
5
4
*
4
4
20
16

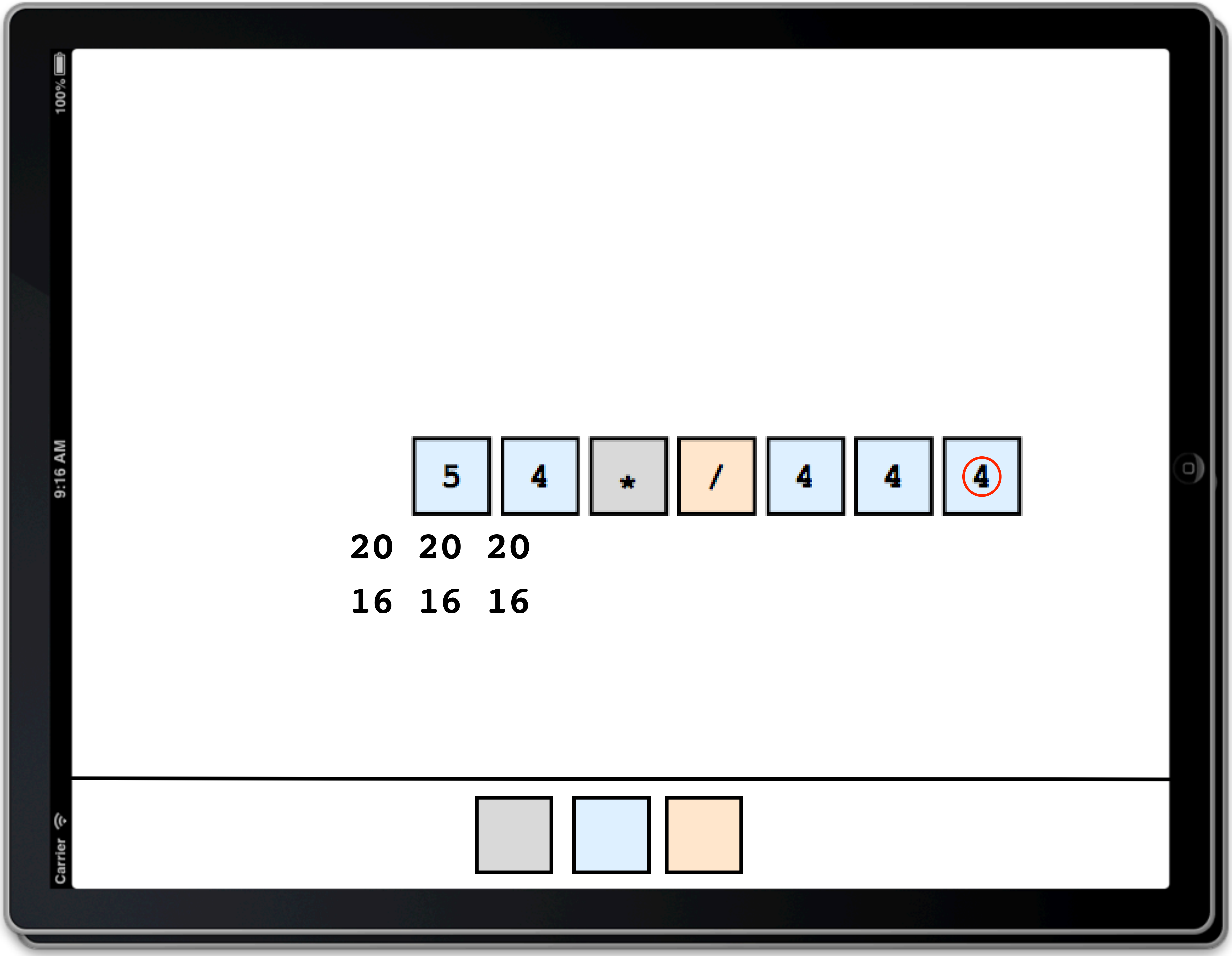

100%
9:16 AM
Carrier
&
5
4
*
4
4
20 16


100%
9:16 AM
Carrier
&
/
5
4
*
4
4
/.
\.
\
20 16


100%
9:16 AM
Carrier
5
4
*
/
4
4
20 20
16 16

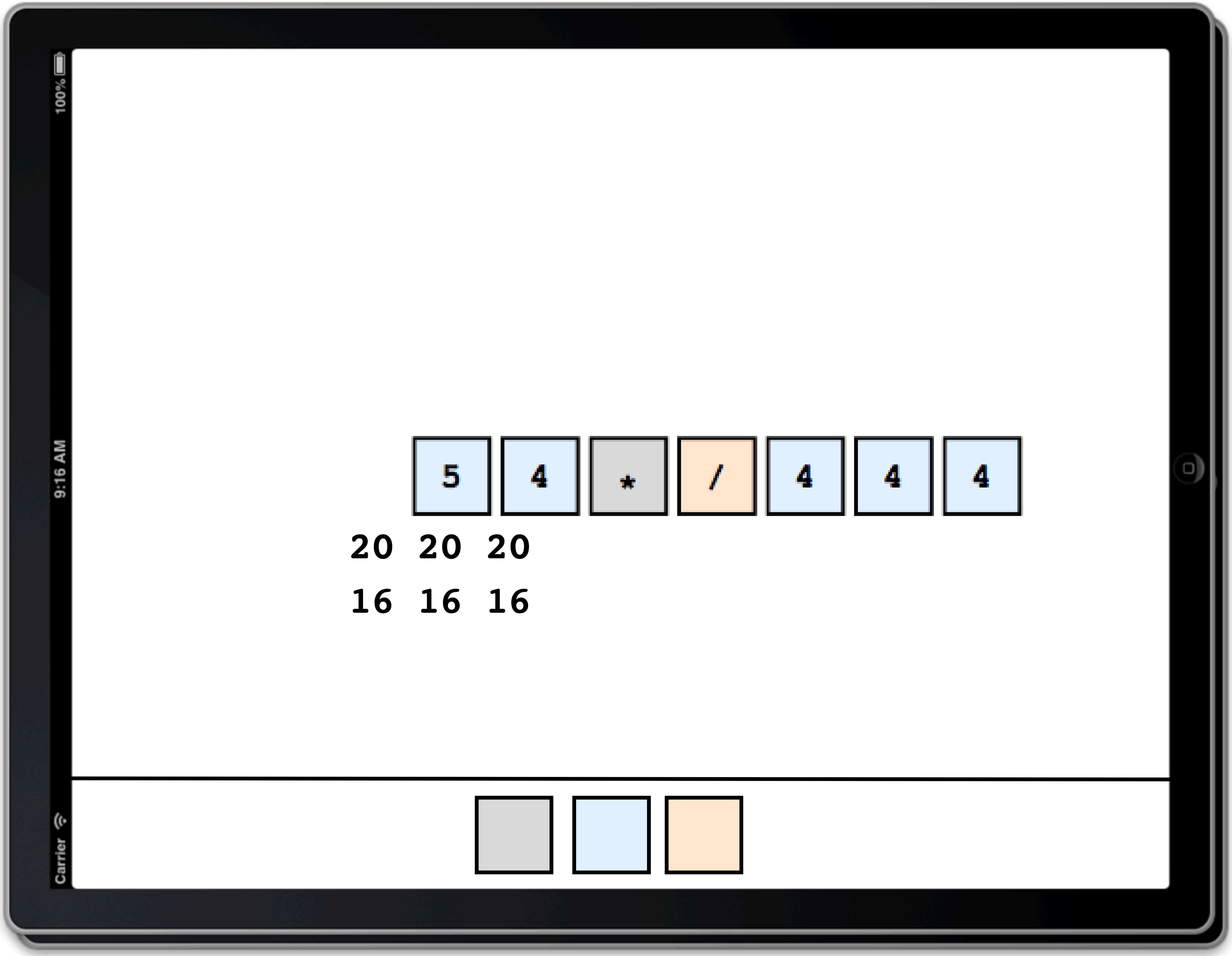


100%
9:16 AM
Carrier
5
4
*
/
4
4
20 20
16 16



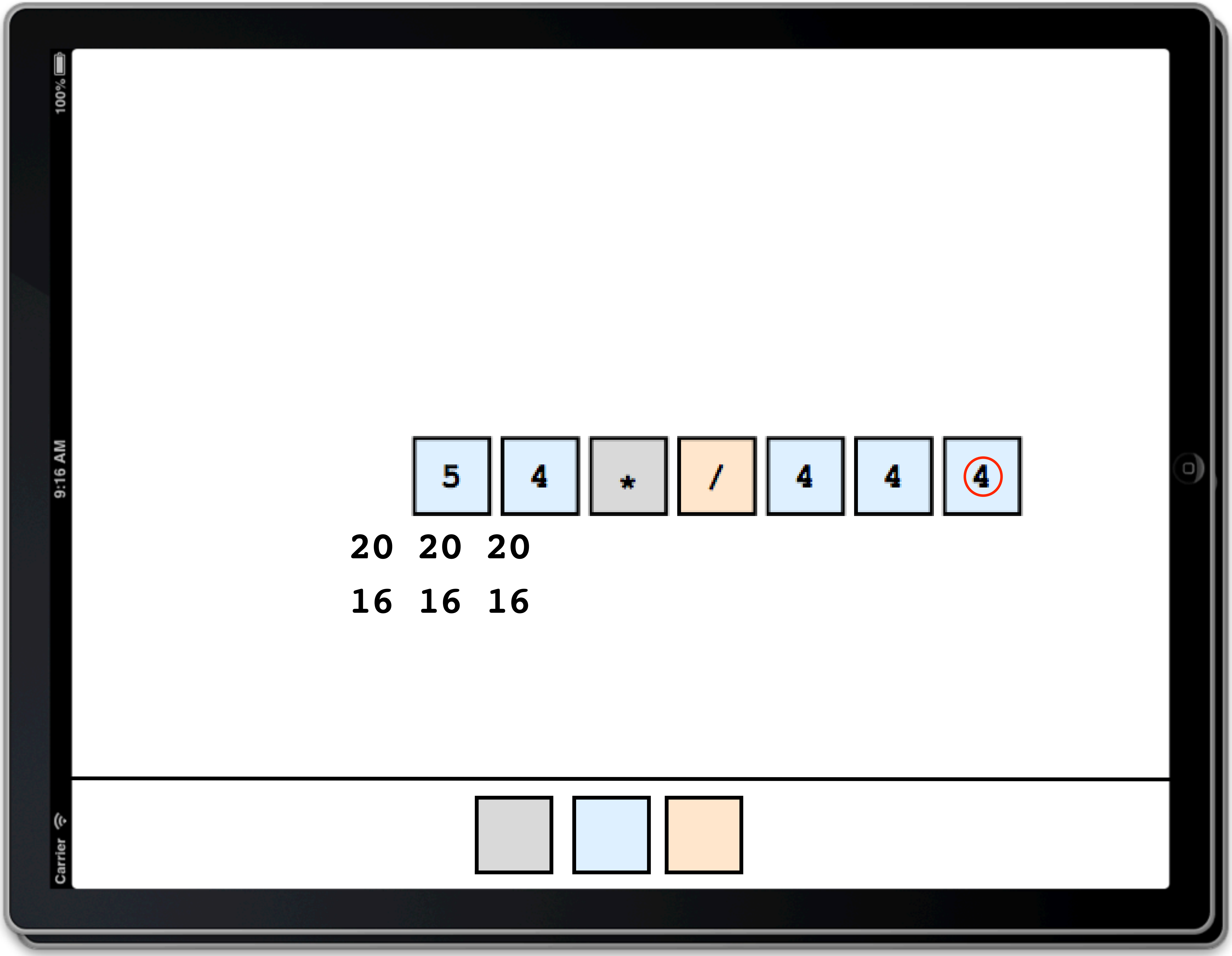
20 20 20

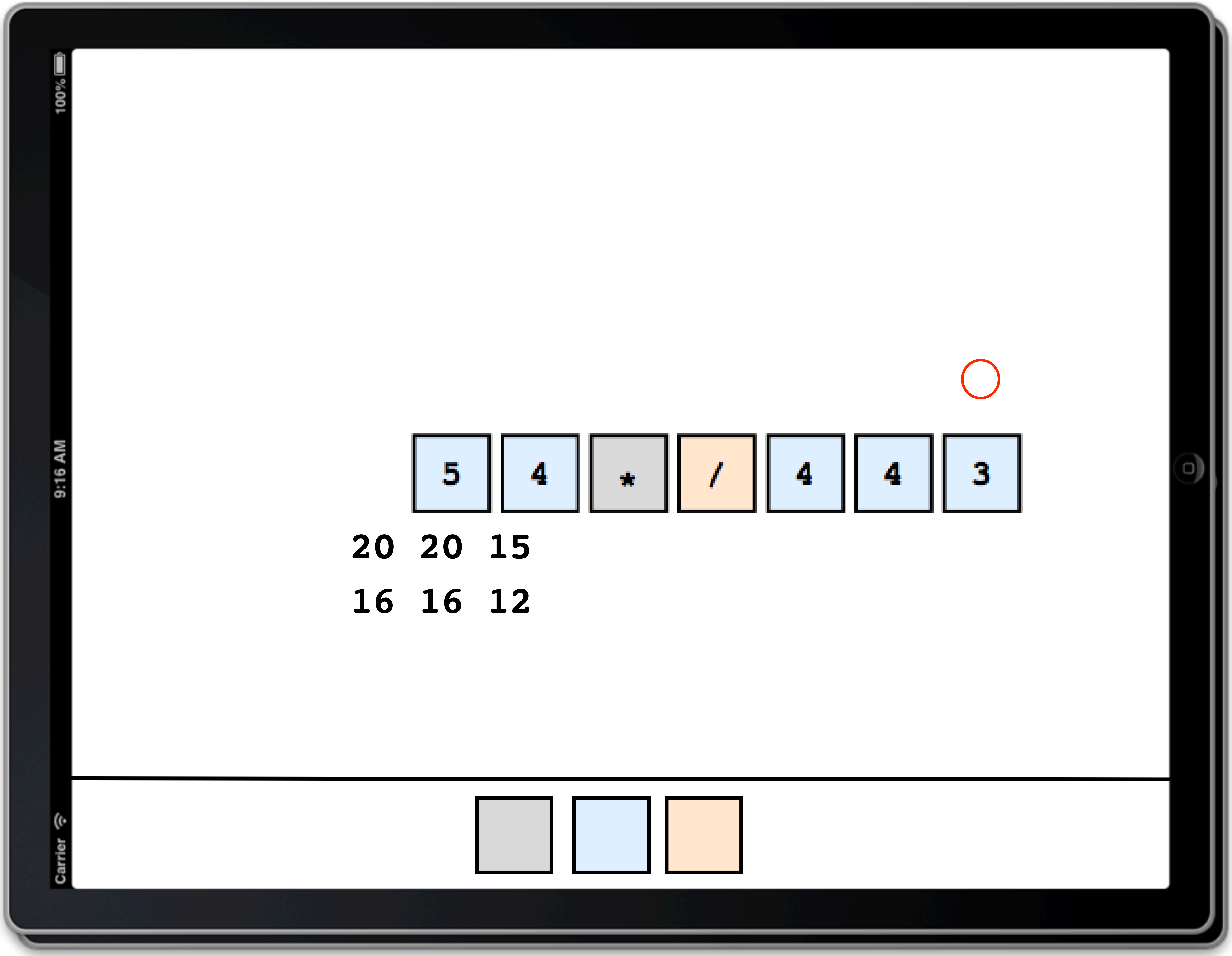
16 16 16



20 20 20

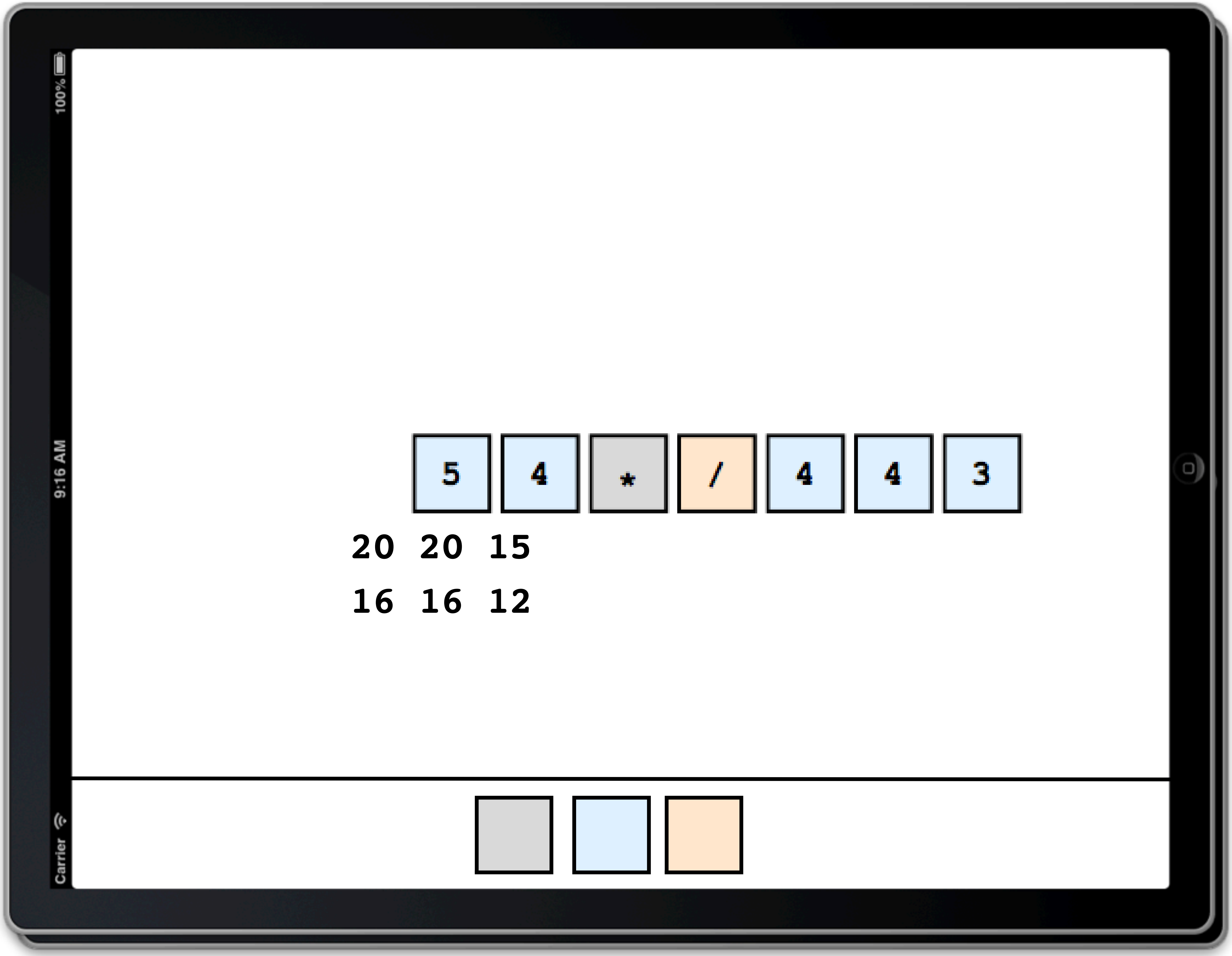
16 16 16


100%
9:16 AM
Carrier
5
4
*
/
4
4
4
20 20 20
16 16 16



20 20 15

16 16 12



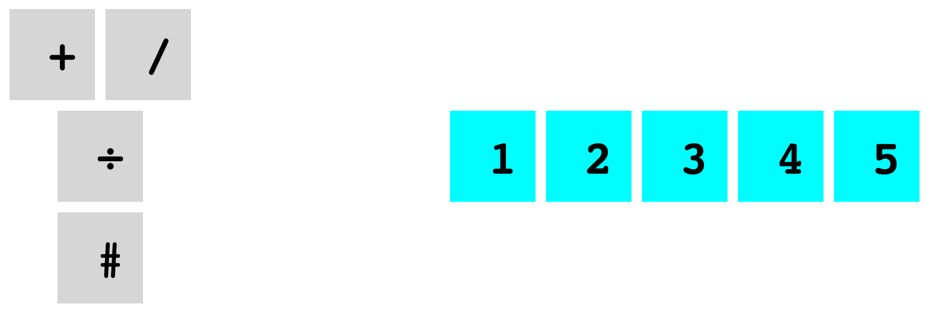
5 4 * / 4 4 3

20 20 15

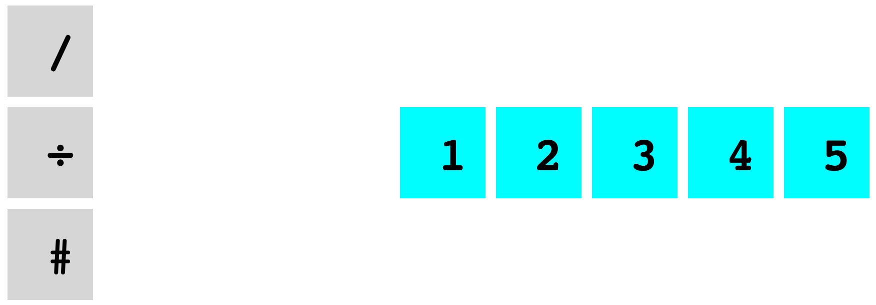
16 16 12

Animation Ideas

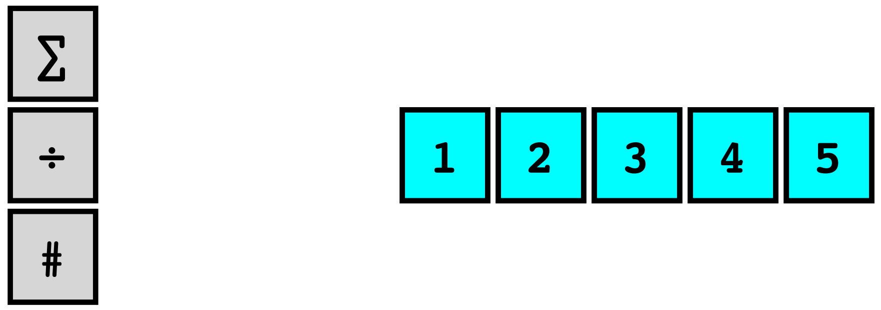
Arithmetic Mean



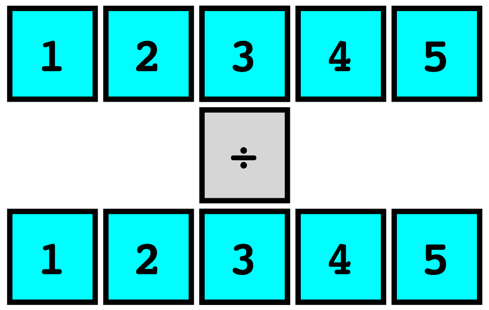
Arithmetic Mean



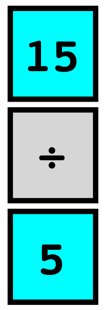
Arithmetic Mean



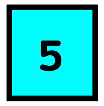
Arithmetic Mean



Arithmetic Mean



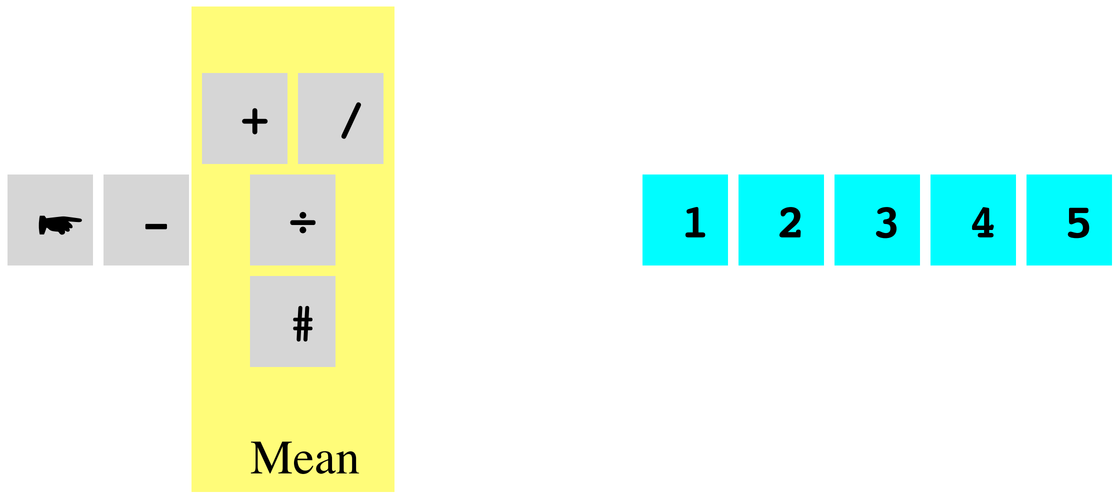
Arithmetic Mean



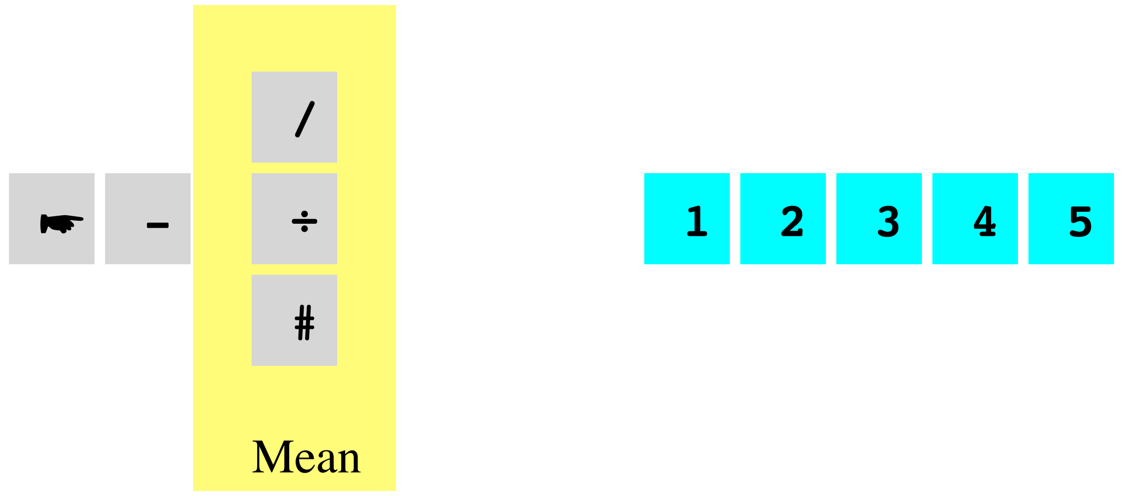
Arithmetic Mean

$\boxed{3}$

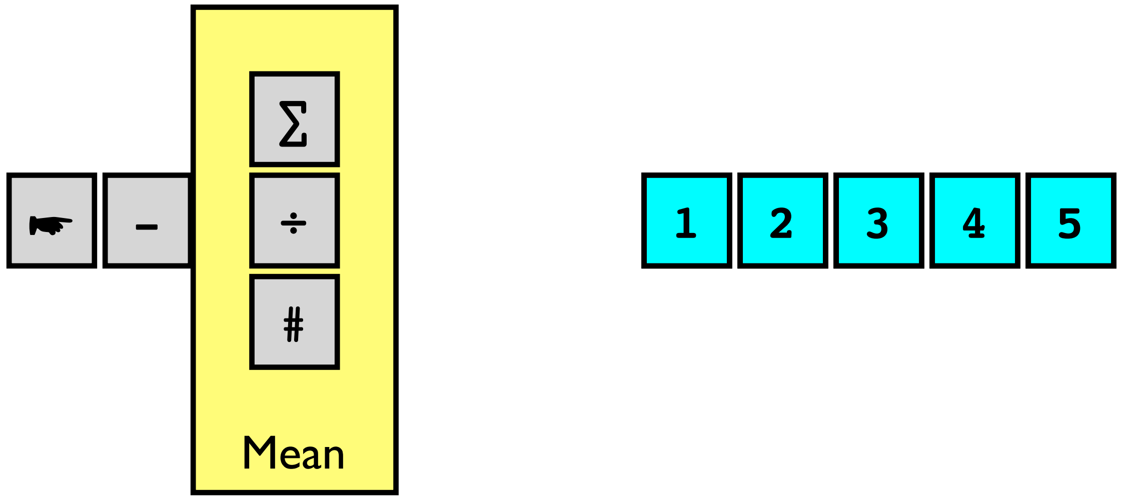
Deviation



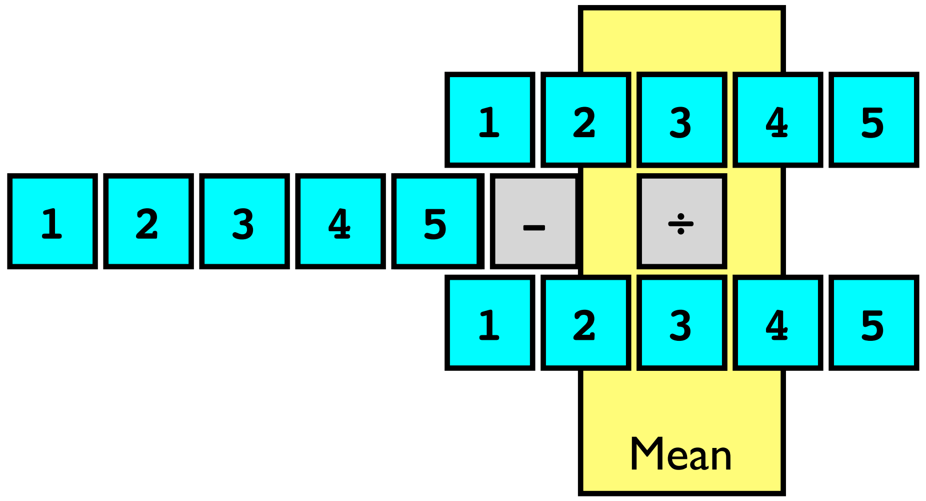
Deviation



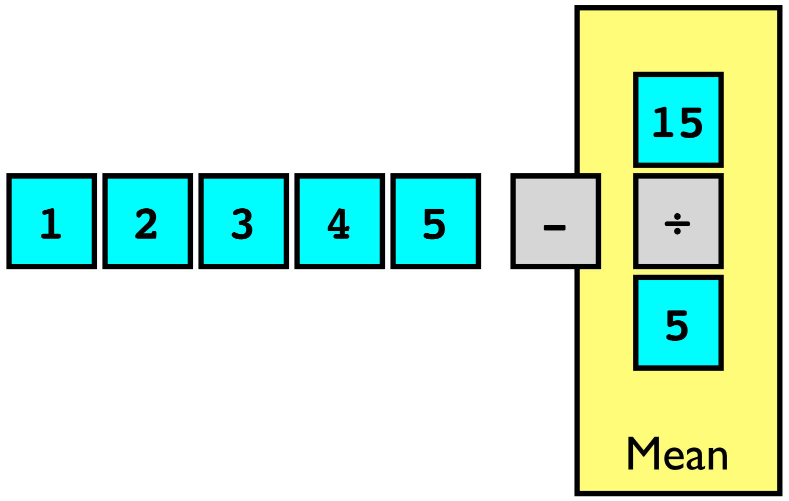
Deviation



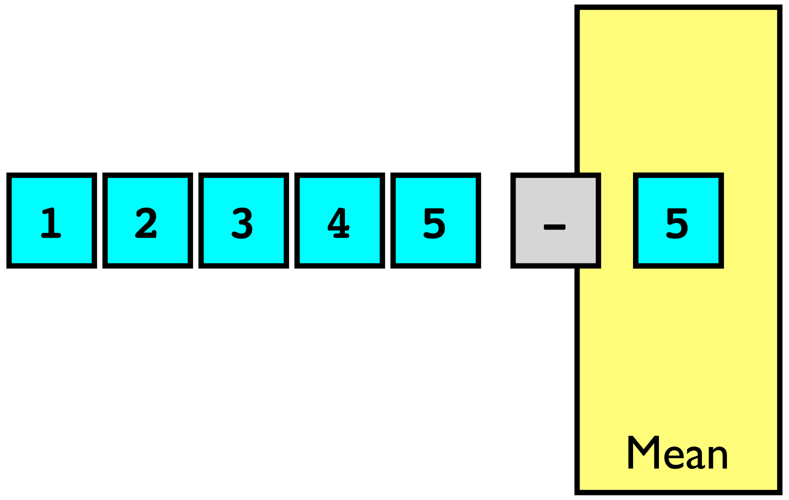
Deviation



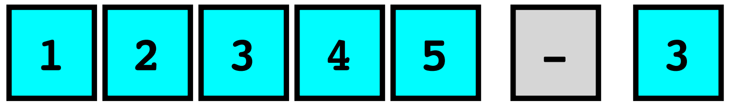
Deviation



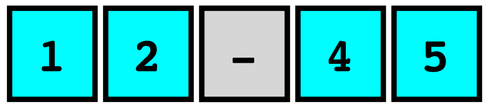
Deviation



Deviation



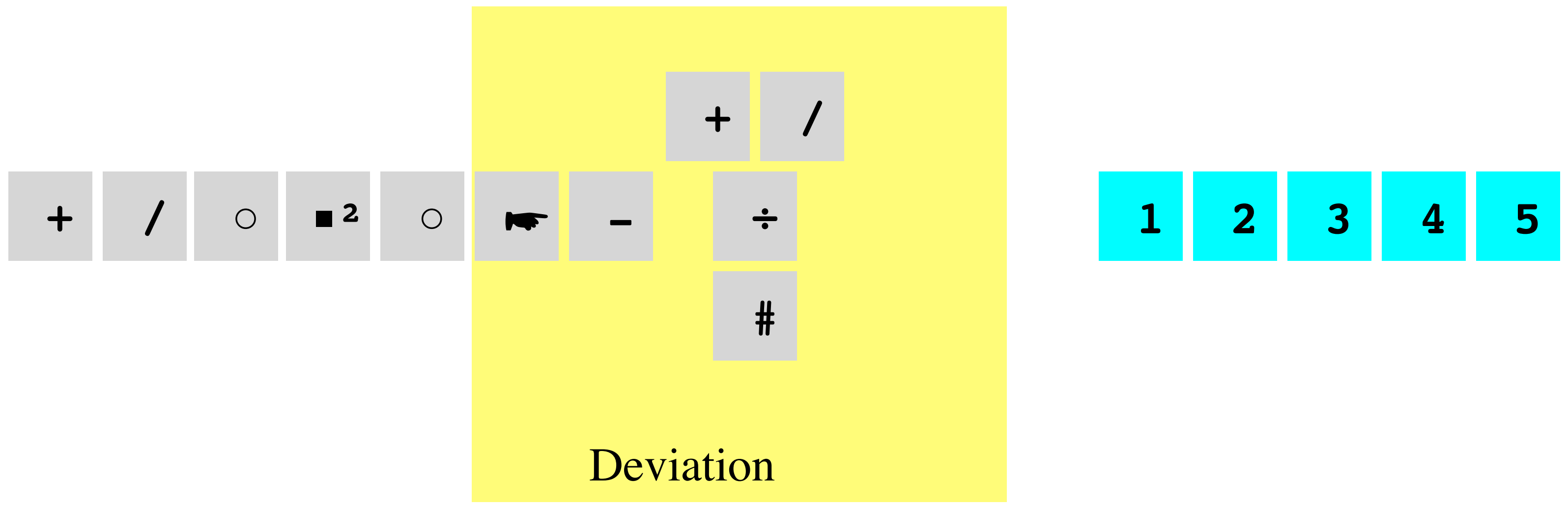
Deviation



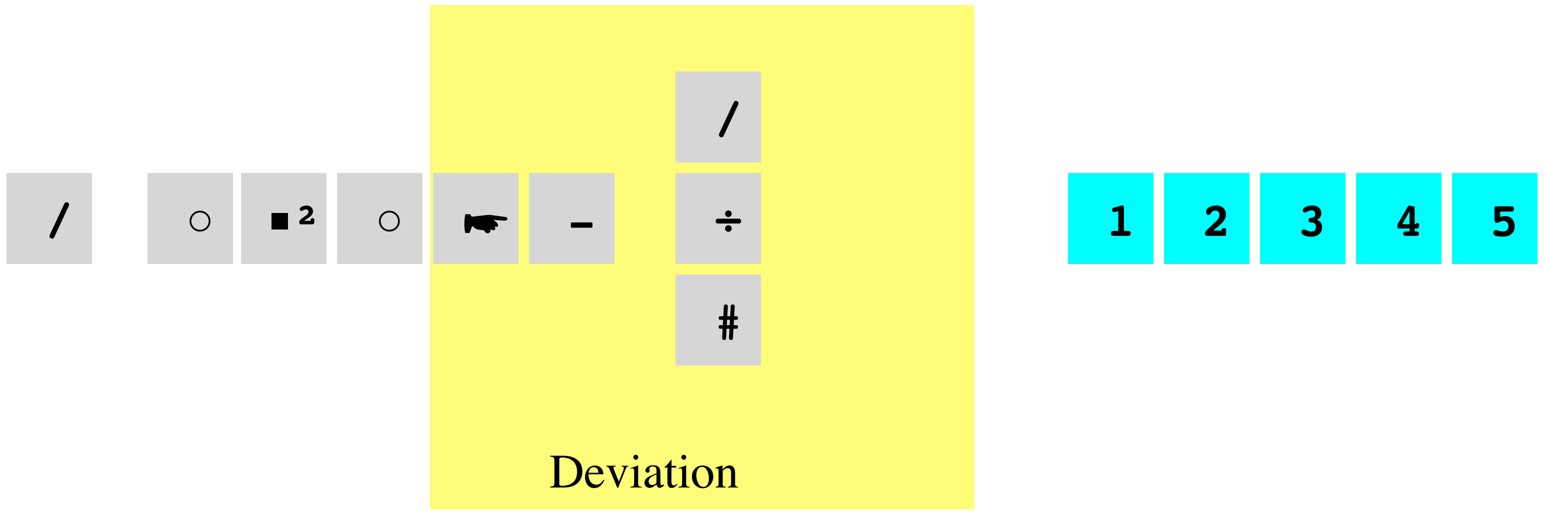
Deviation

−2	−1	0	1	2

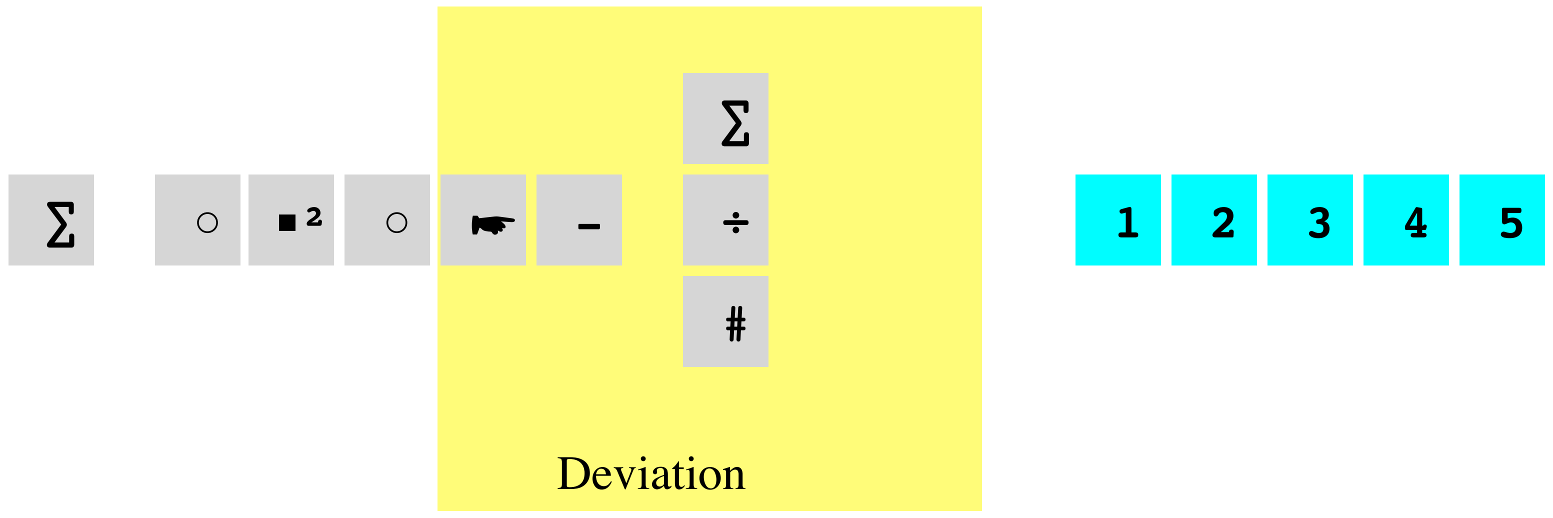
Sum of Squared Deviation



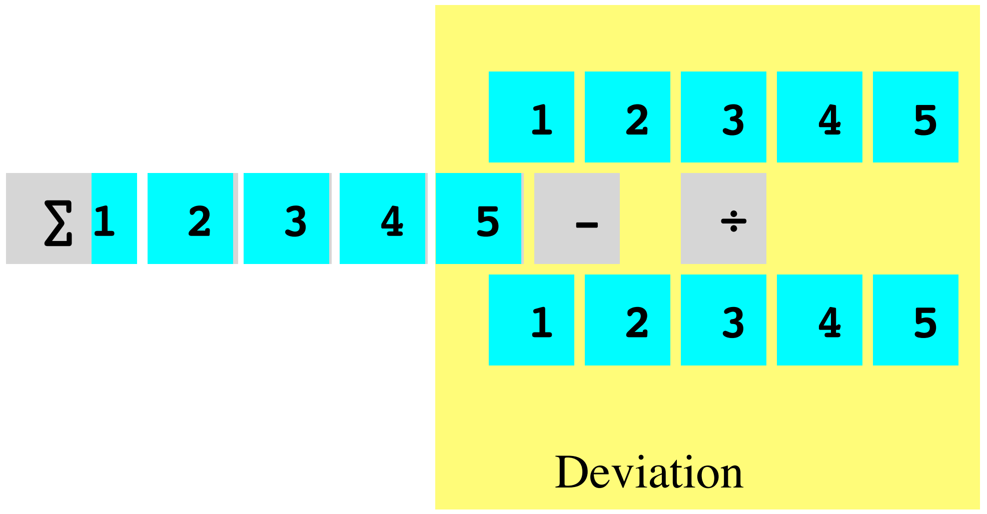
Sum of Squared Deviation



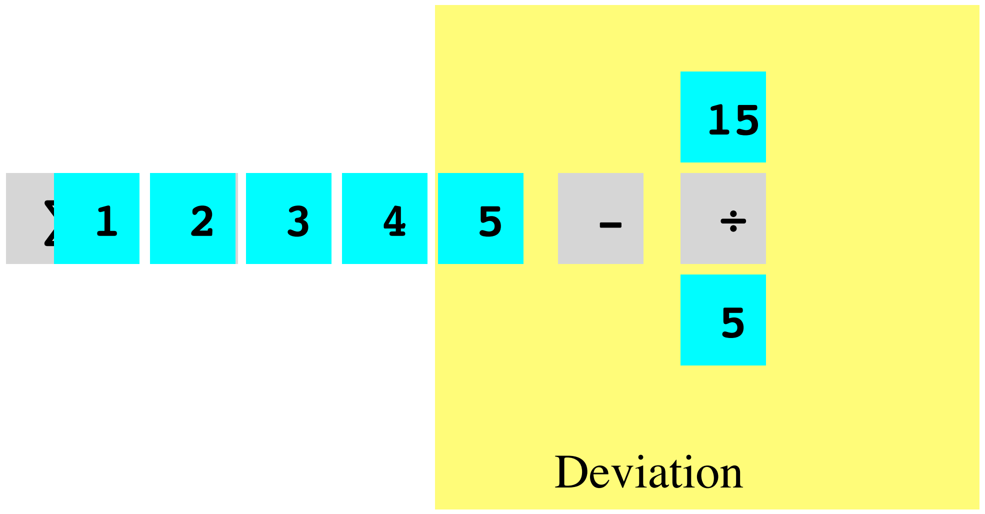
Sum of Squared Deviation



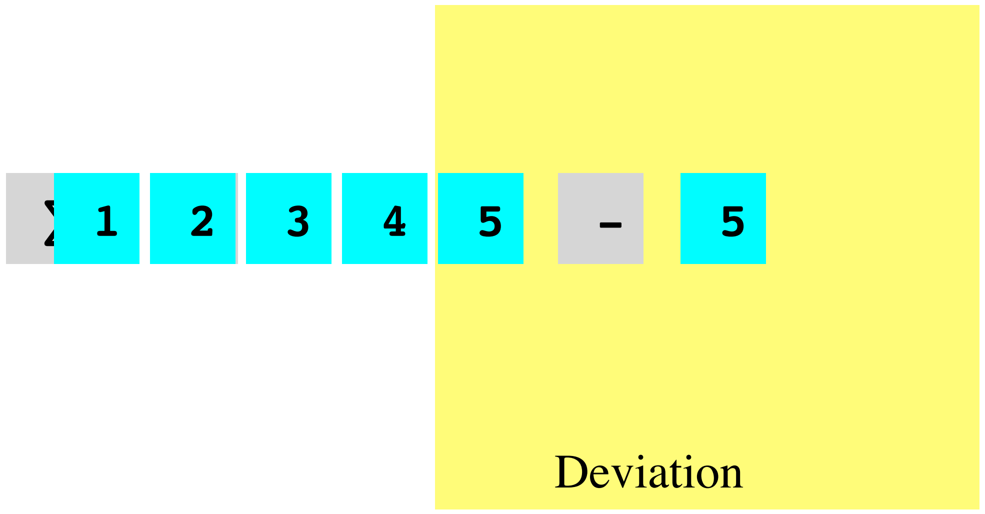
Sum of Squared Deviation



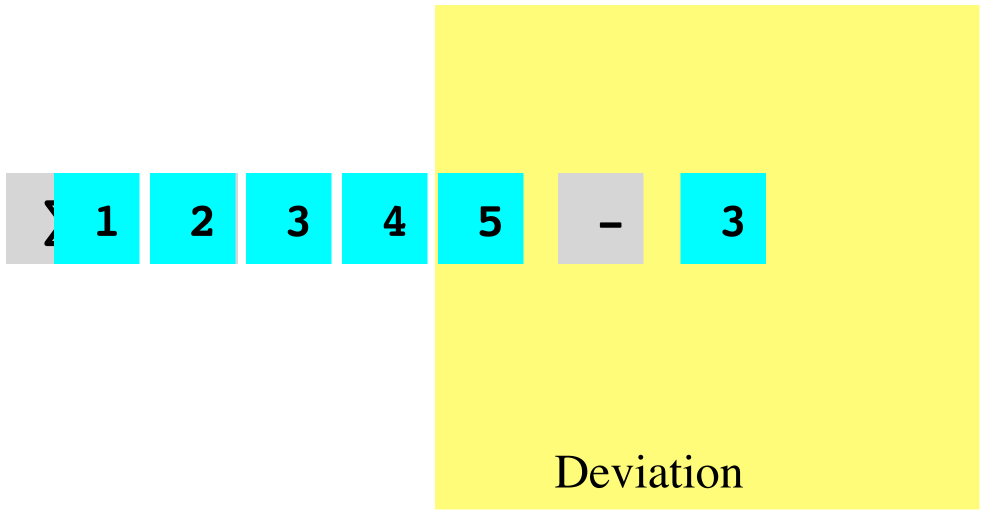
Sum of Squared Deviation



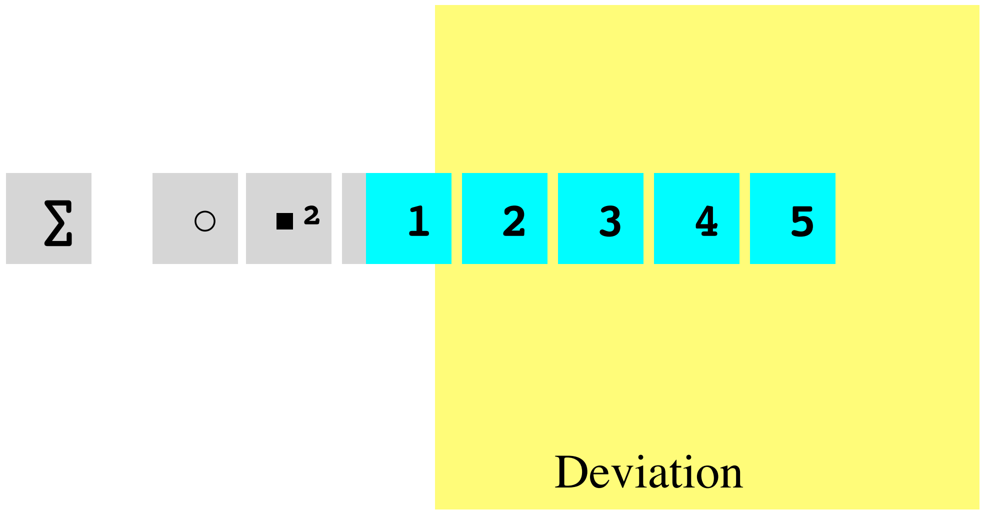
Sum of Squared Deviation



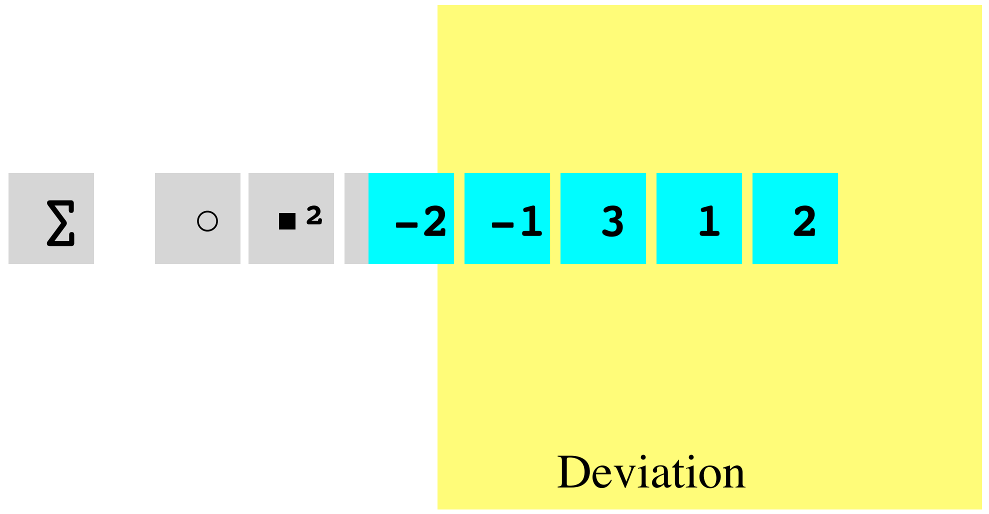
Sum of Squared Deviation



Sum of Squared Deviation



Sum of Squared Deviation



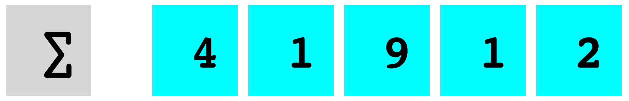
Sum of Squared Deviation

∑	○	■ ²	−2	−1	3	1	2

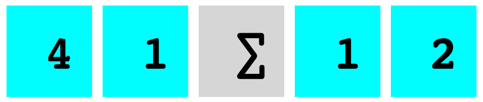
Sum of Squared Deviation

Σ	−2	−1	3	1	2

Sum of Squared Deviation



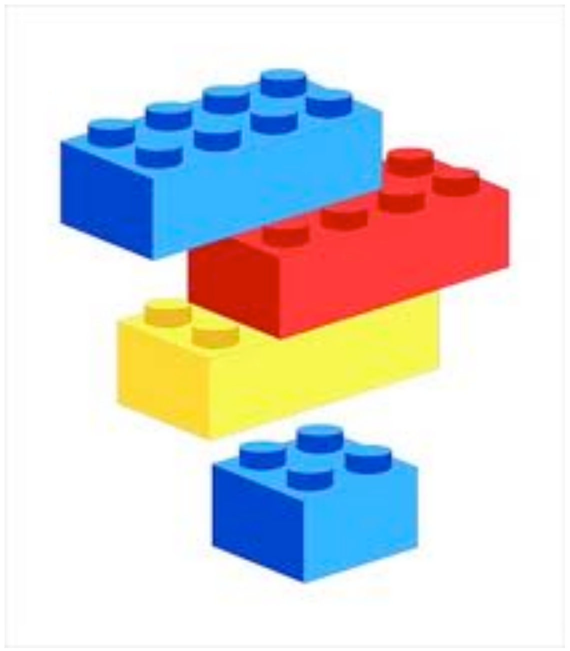
Sum of Squared Deviation



Sum of Squared Deviation

15

A tangible futuristic idea



We start with tangibles

Carving a new groove

Getting off the computer and down on the floor

Lifting to the digital domain

Lifting to the digital domain

Lifted

Resources

J

http://www.jsoftware.com
http://www.jsoftware.com/help/jforc/contents.htm

Notation as a Tool of Thought
http://www.jsoftware.com/papers/tot.htm

K and Q
http://kx.com
http://code.kx.com/wiki/JB:QforMortals2/contents

APL
http://www.dyalog.com
http://www.dyalog.com/MasteringDyalogAPL/MasteringDyalogAPL.pdf

The Cow
http://www.visionarts.ca/photoillusion.htm