The game of TV-Tennis

Daniel Gehrman
Maxim Nonkin
Damir Turtugeshev
Group: 23931

Novosibirsk • 2024

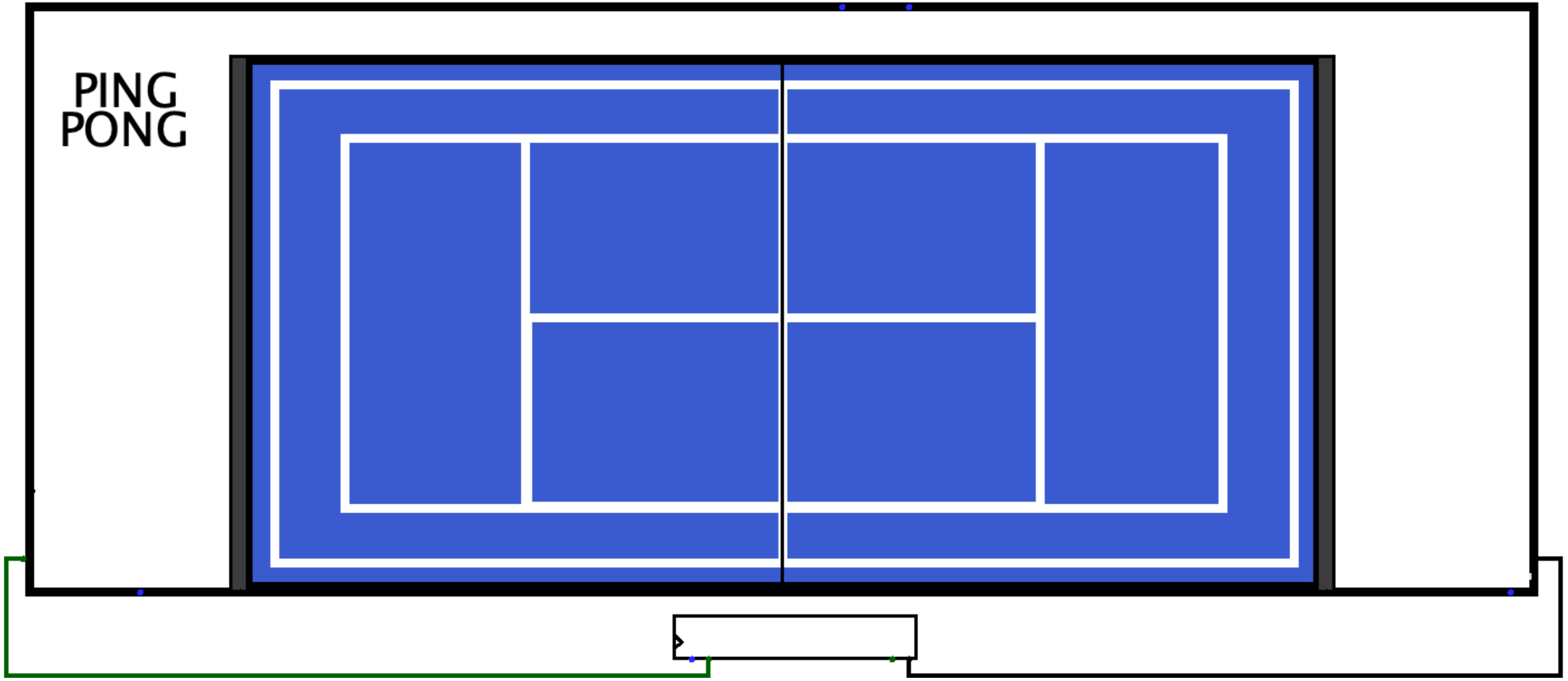
Pong

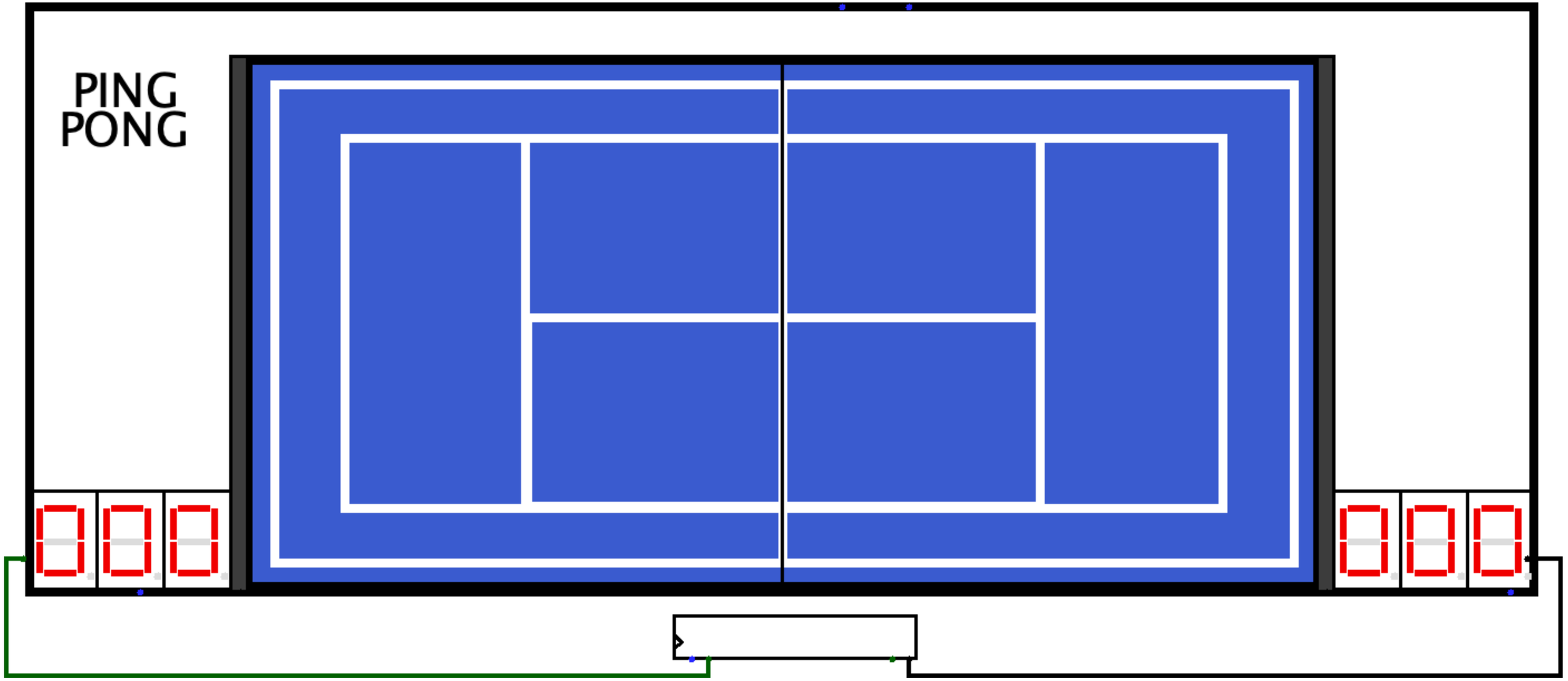
A video game classic

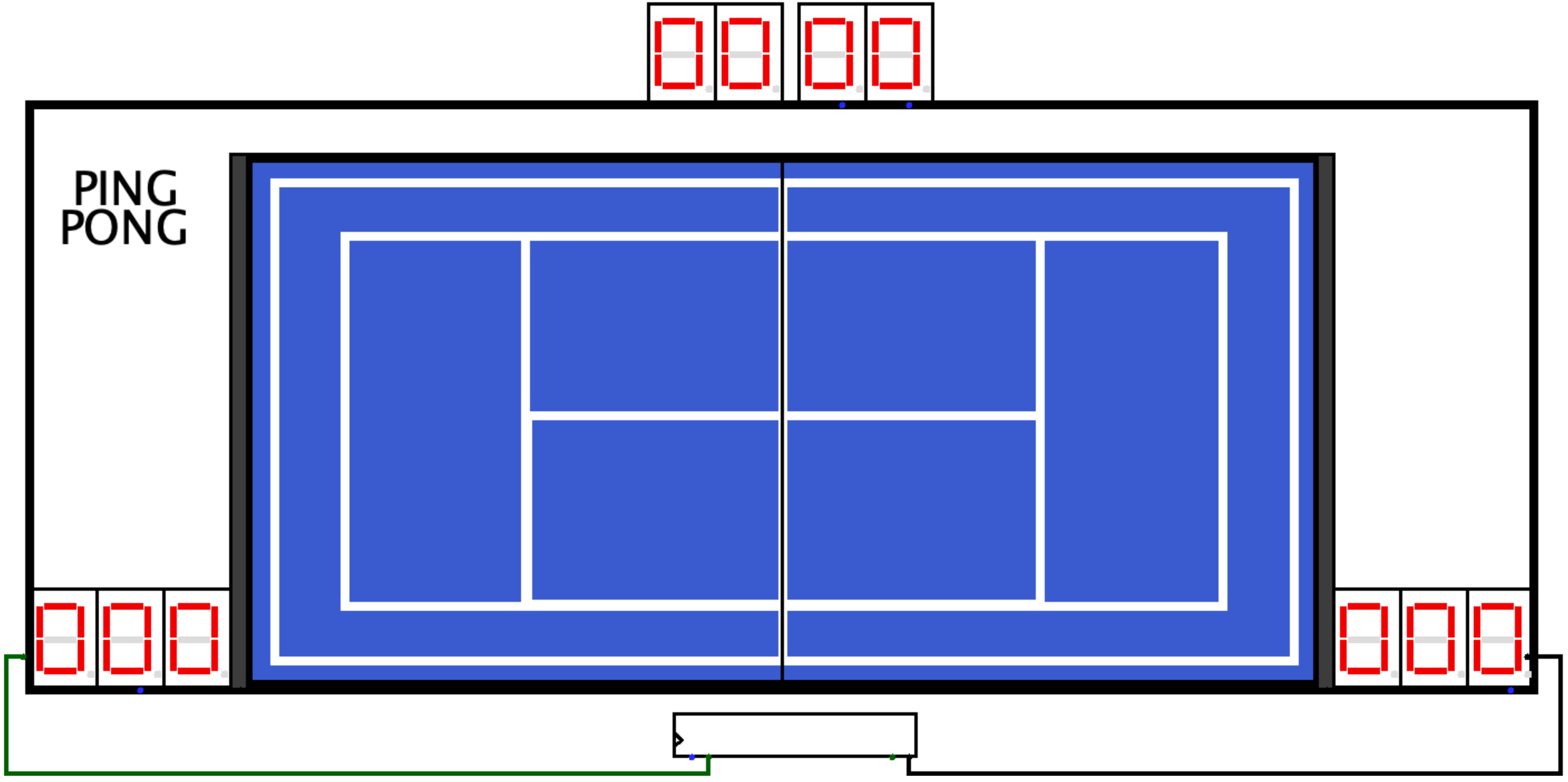
Simple

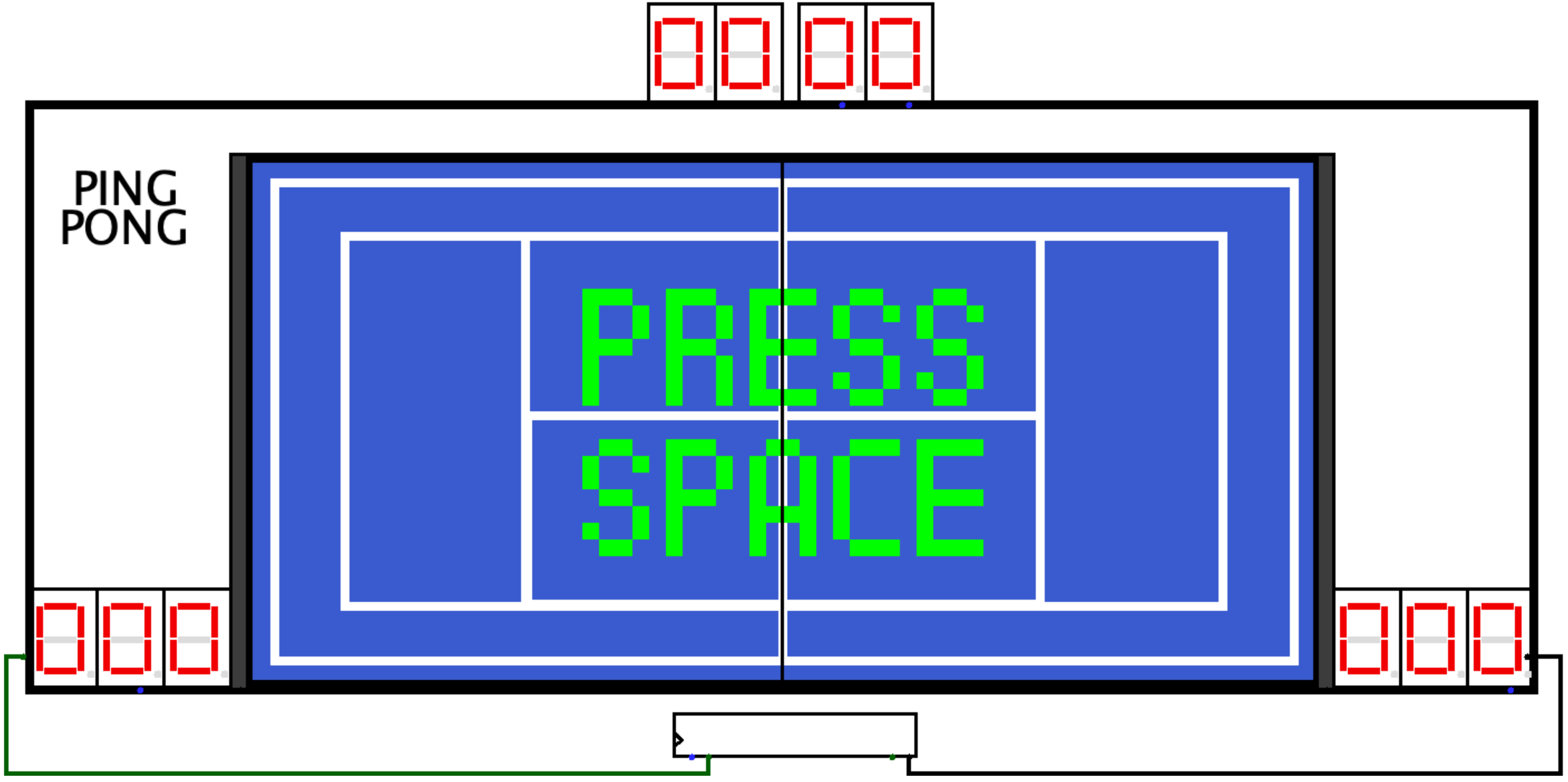
Elegant

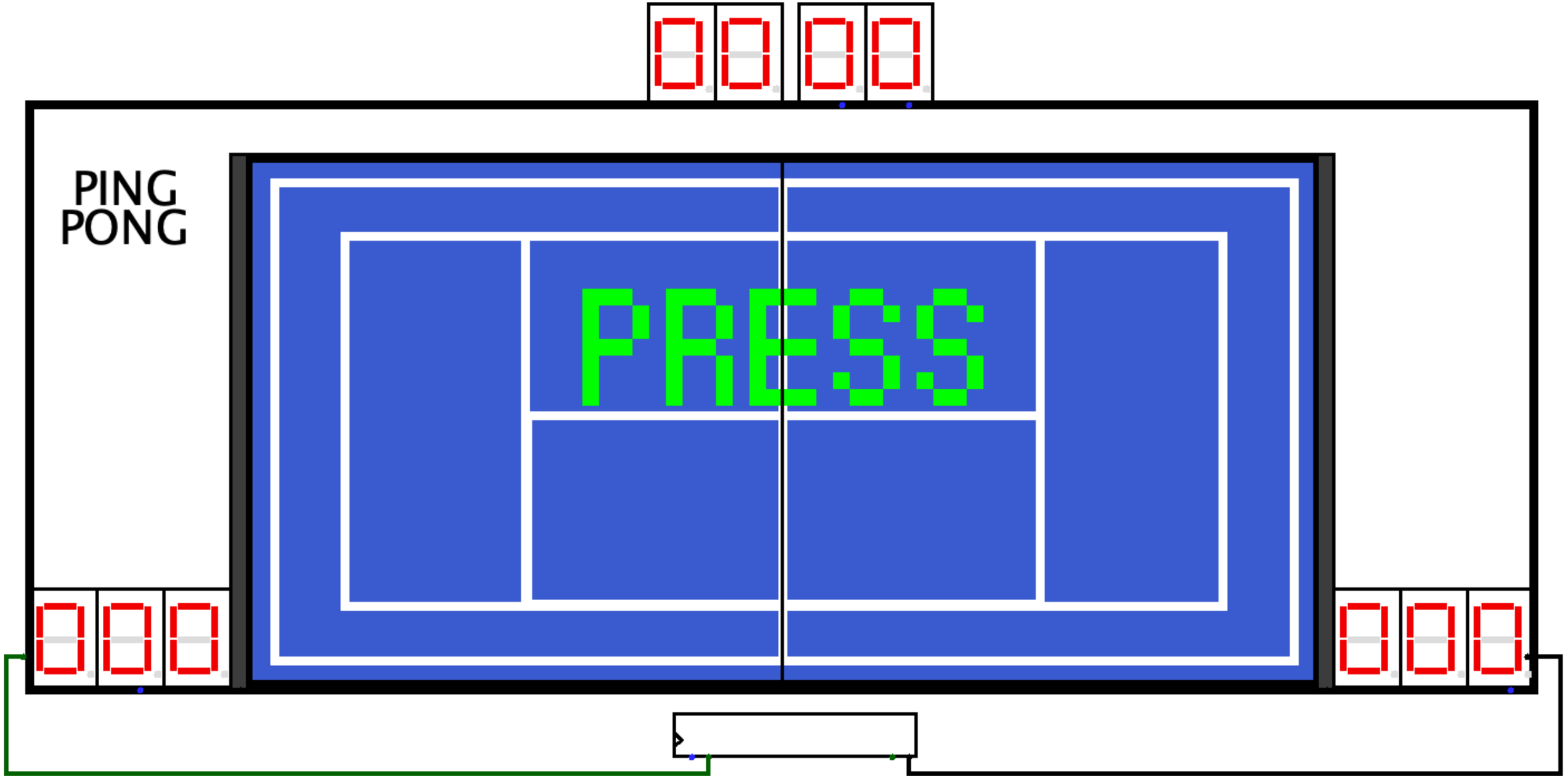
1970’s household staple

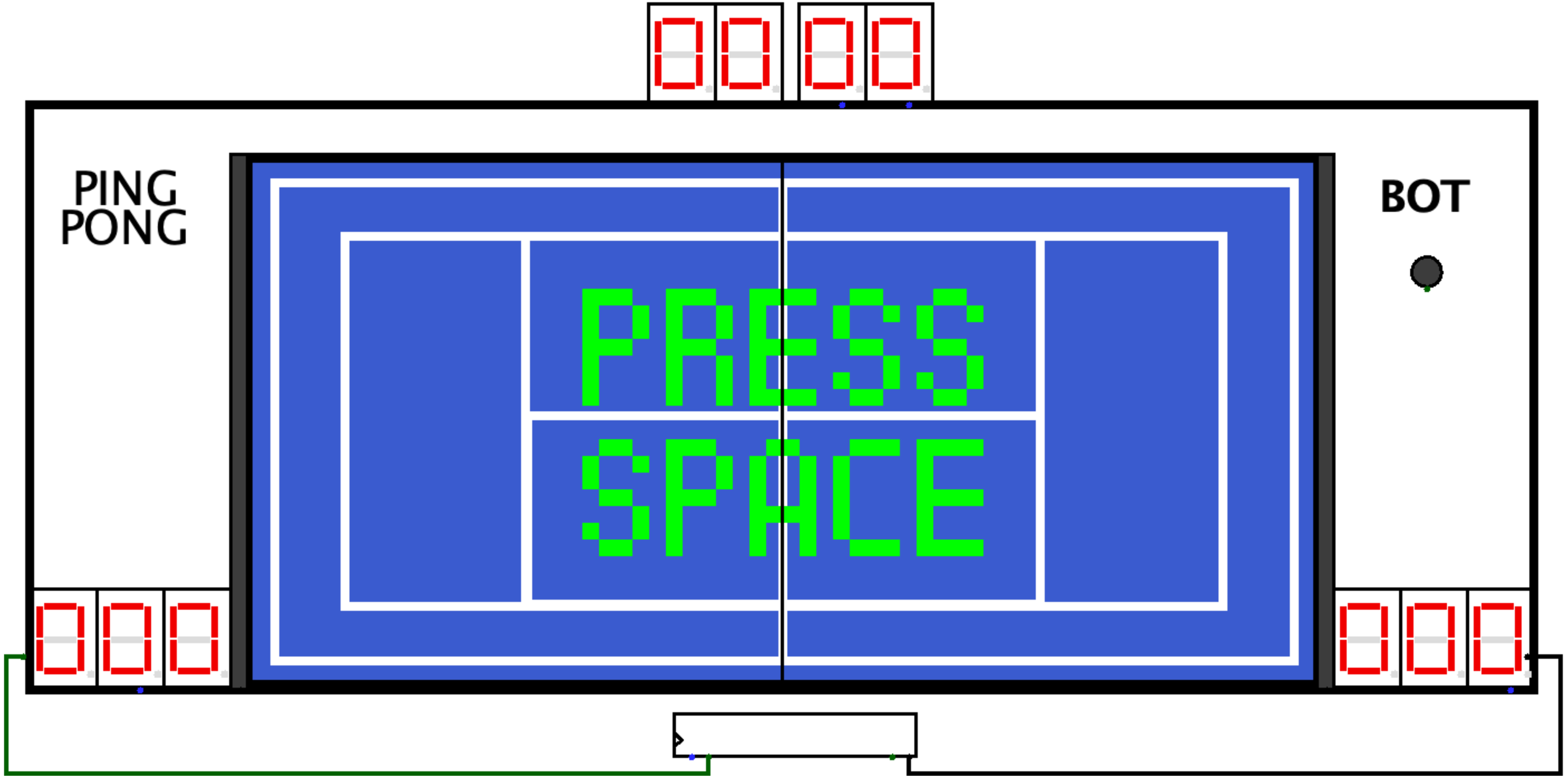

PING
PONG

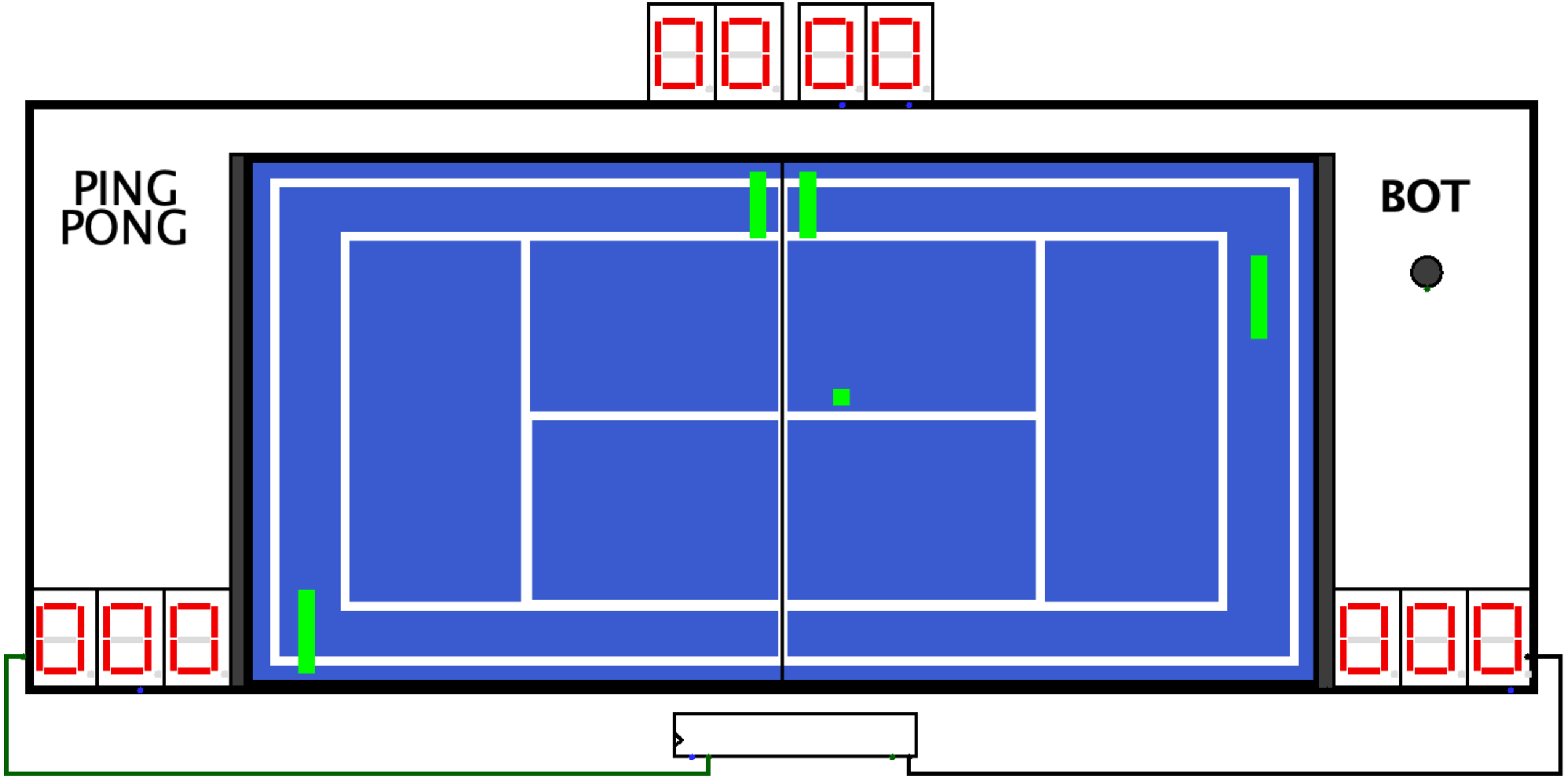

PING
PONG

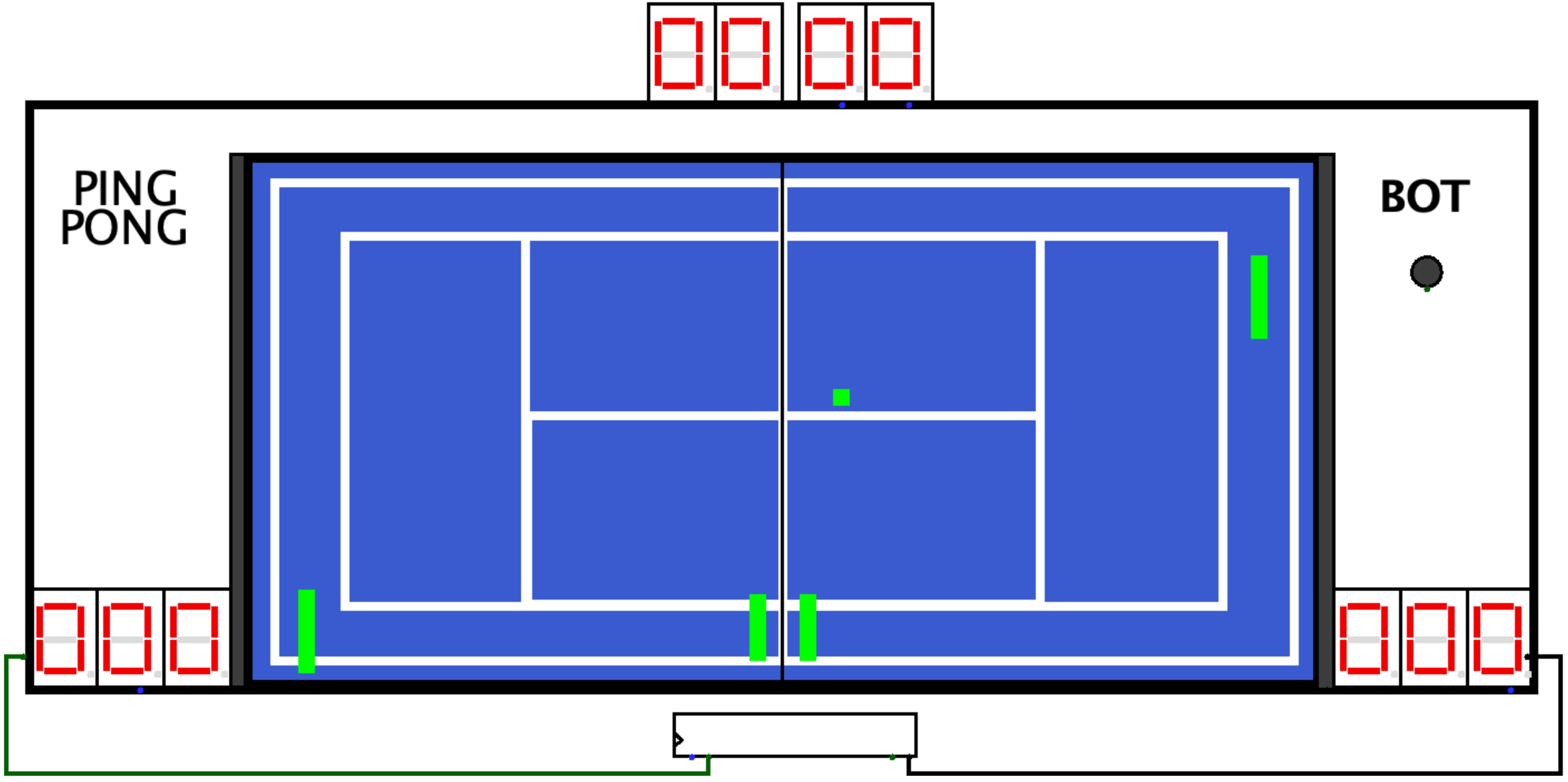

PING
PONG

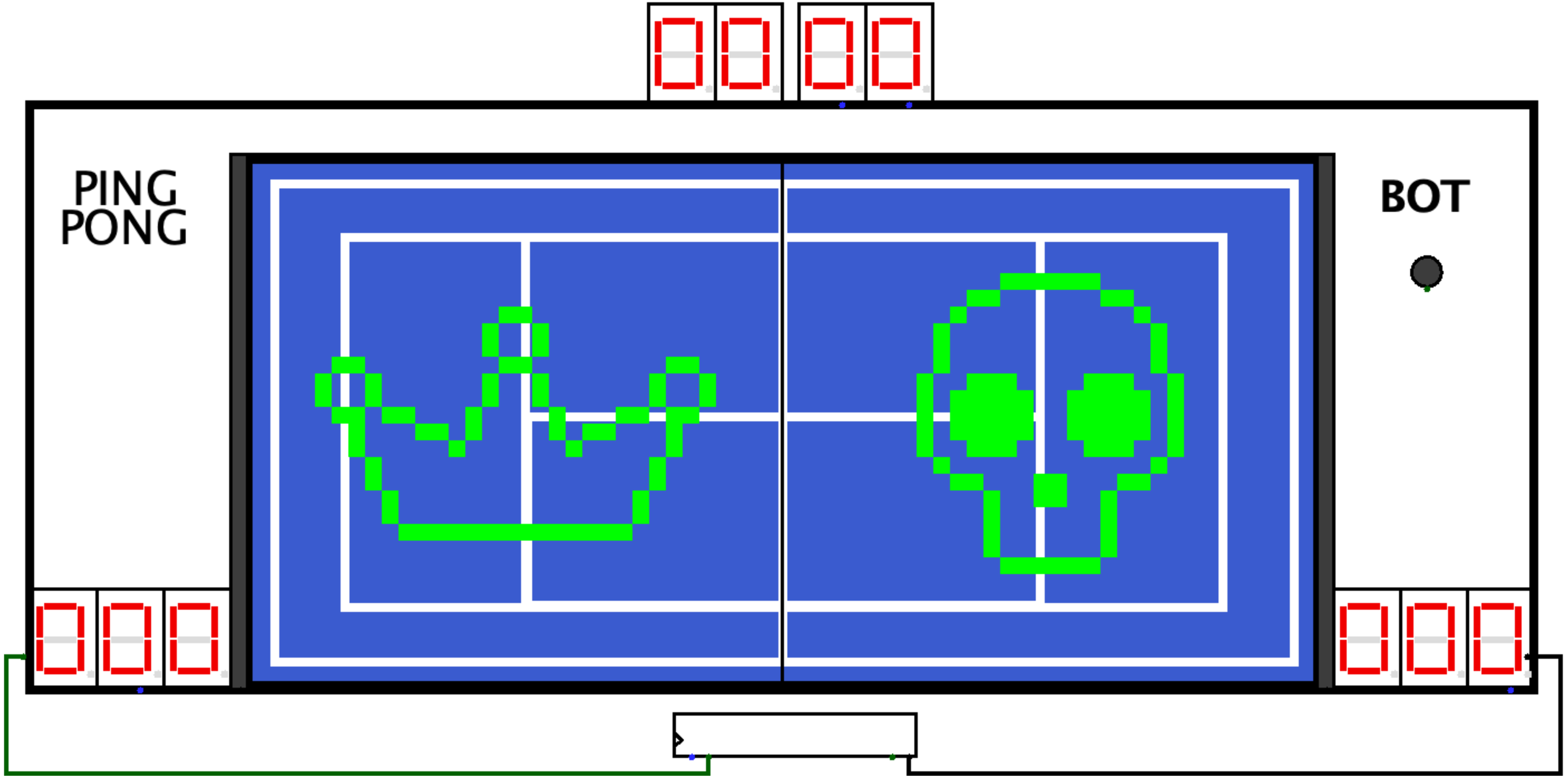

PING
PONG
PRESS
SPACE


PING
PONG
PRESS


PING
PONG
BOT
PRESS
SPACE


PING
PONG
BOT

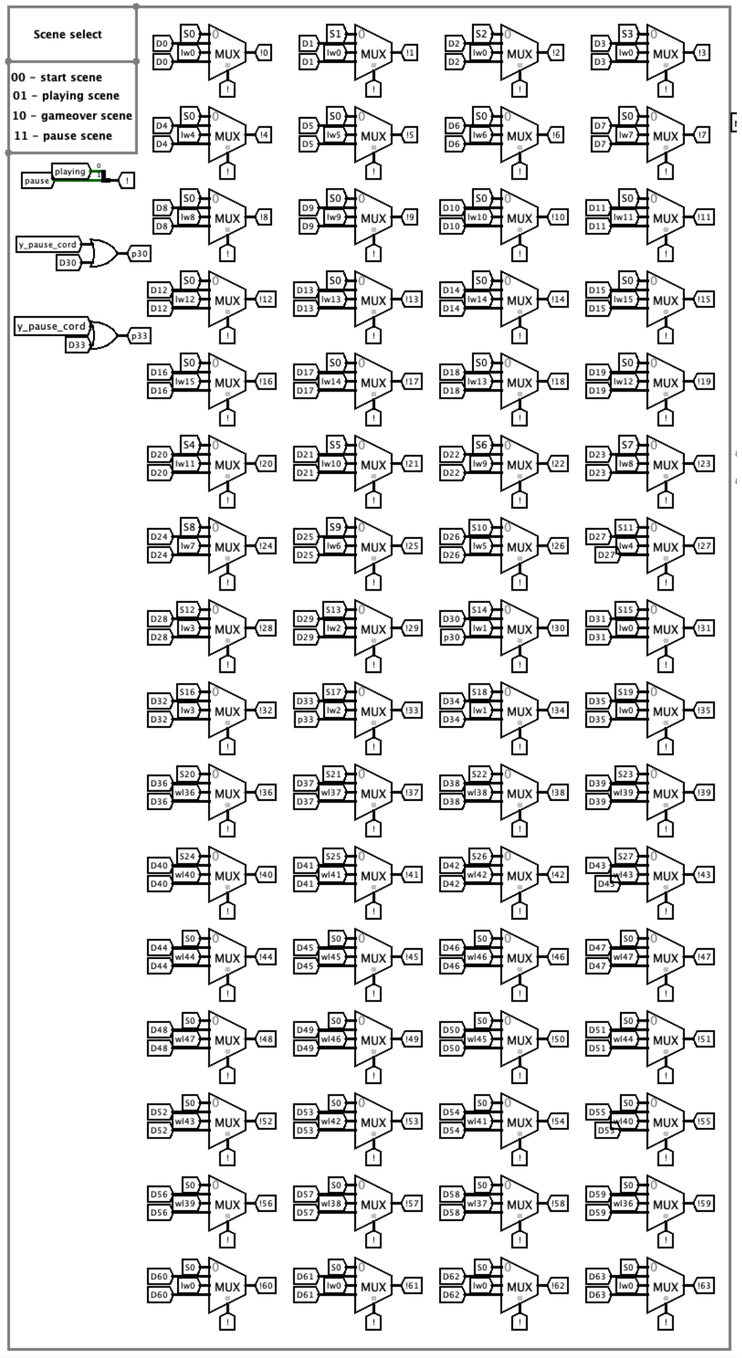

PING
PONG
BOT

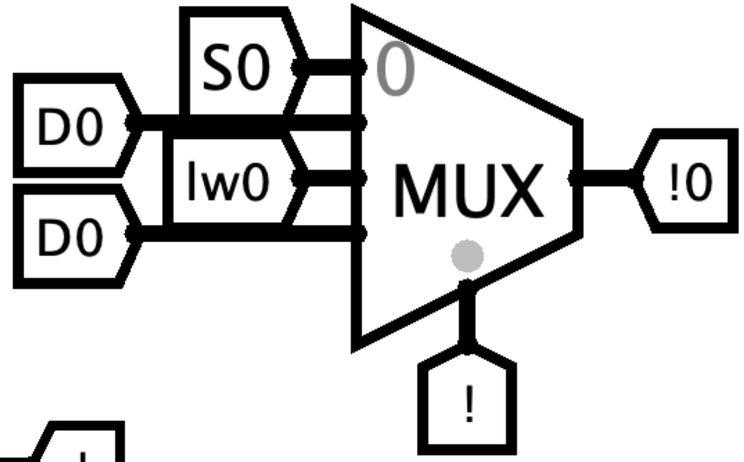

PING
PONG
BOT

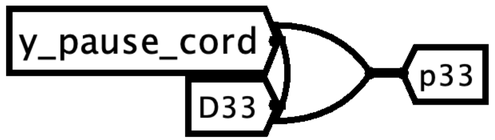
Logisim

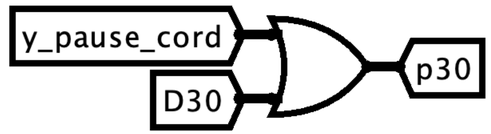
How do we choose which screen to display?

Start screen – 00
Playing screen – 01
Gameover screen – 10
Pause screen – 11

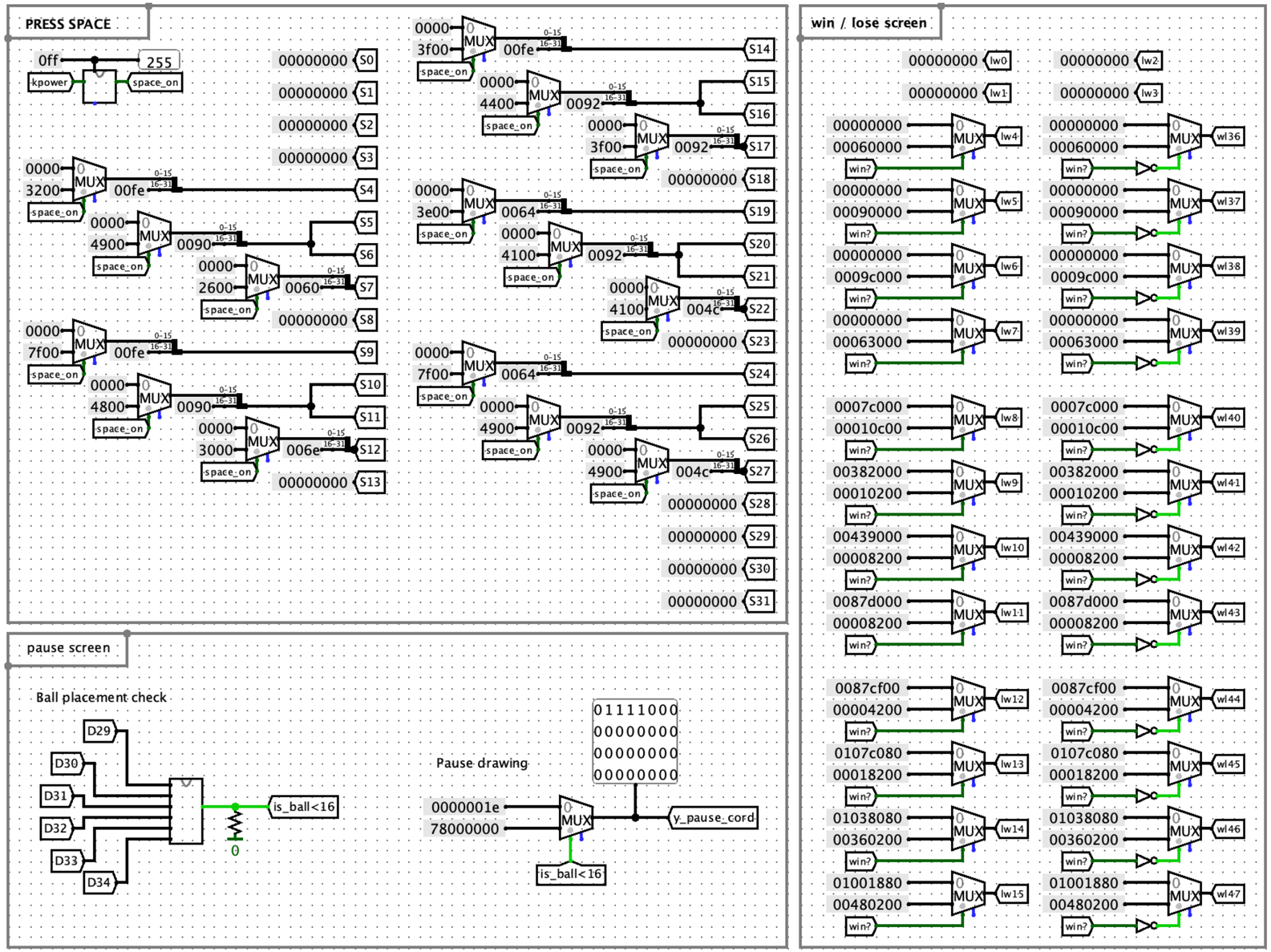




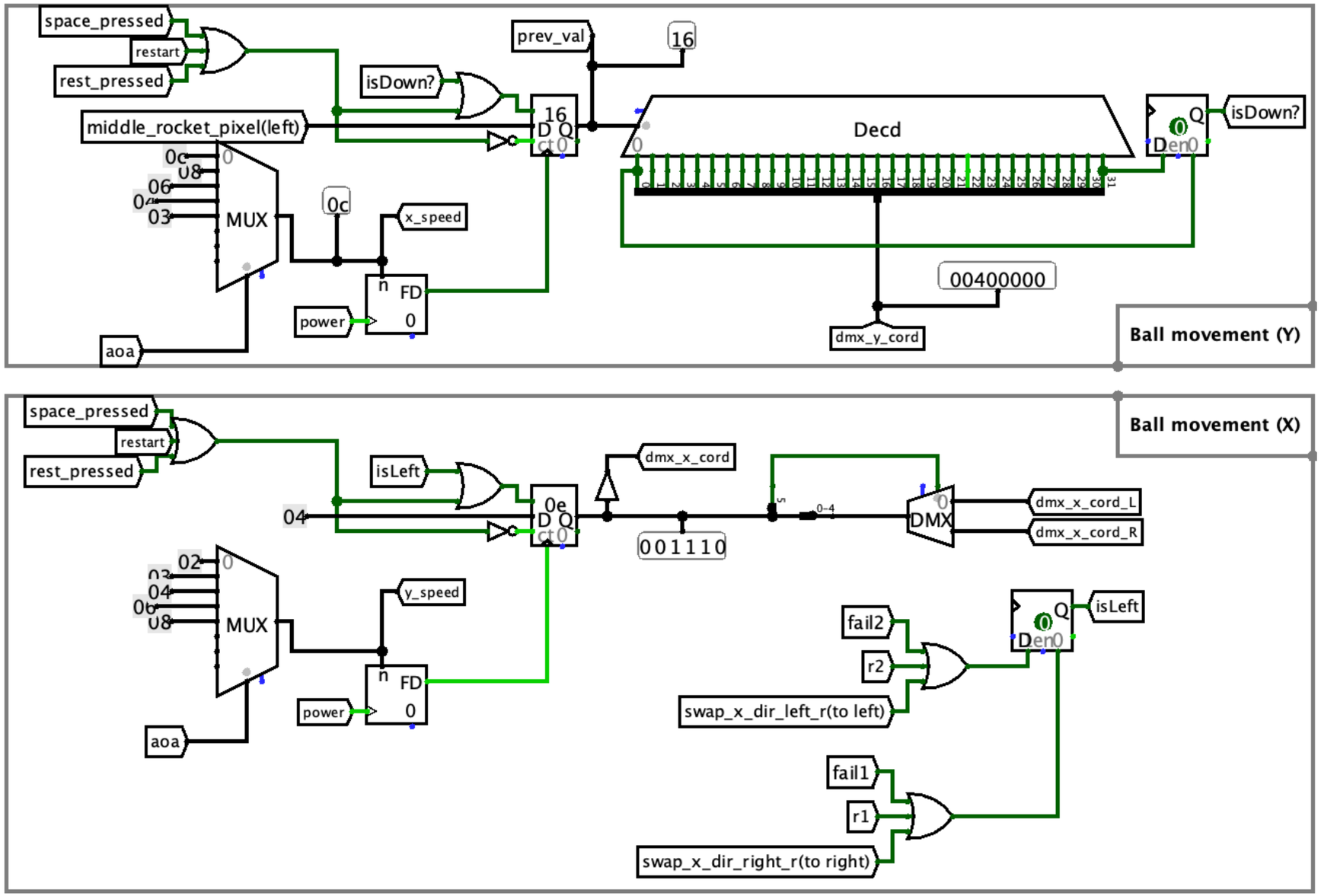




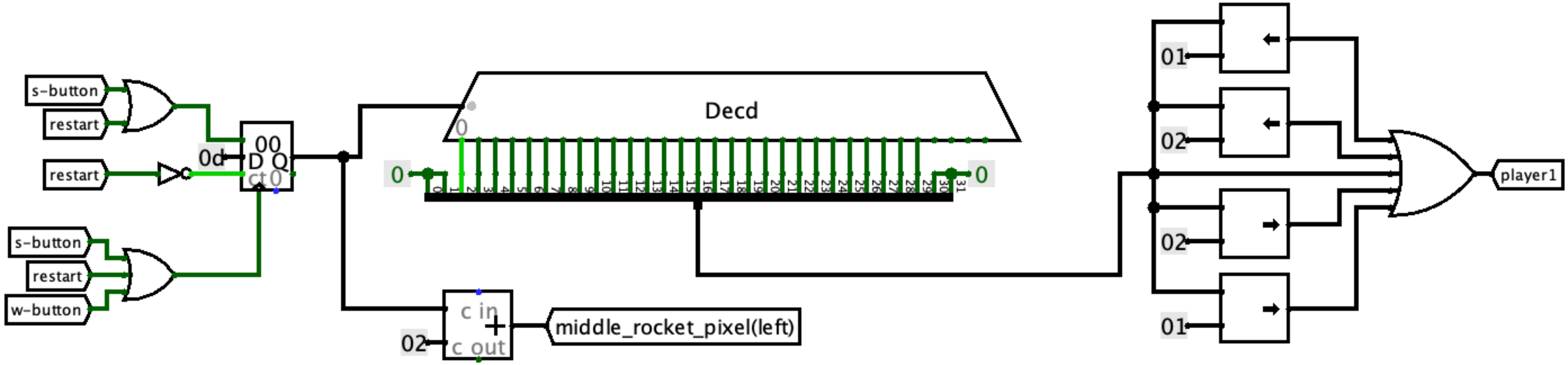
How do we choose the values for these screens?



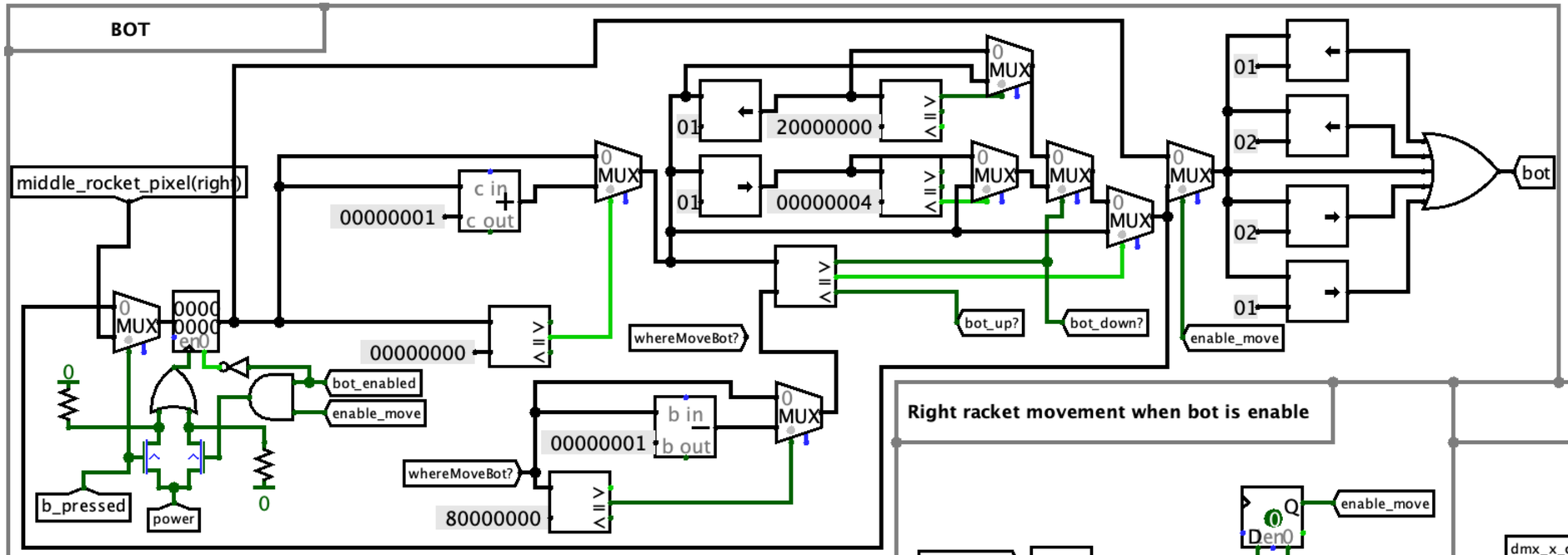
The ball movement

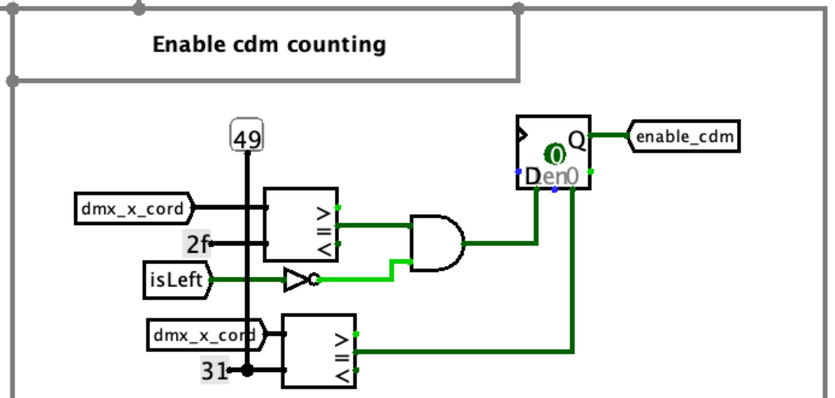


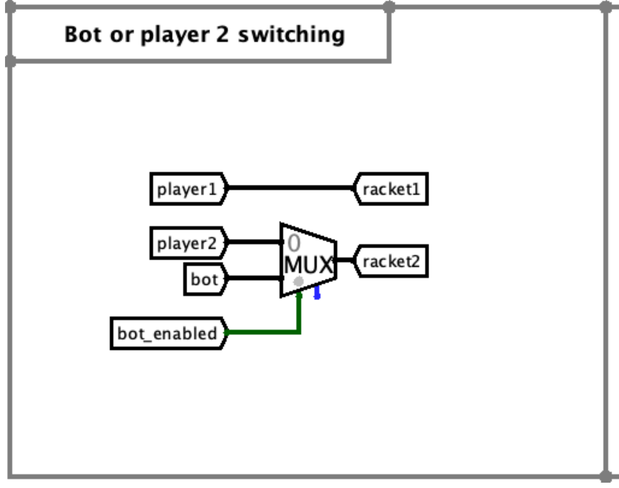
Player movement

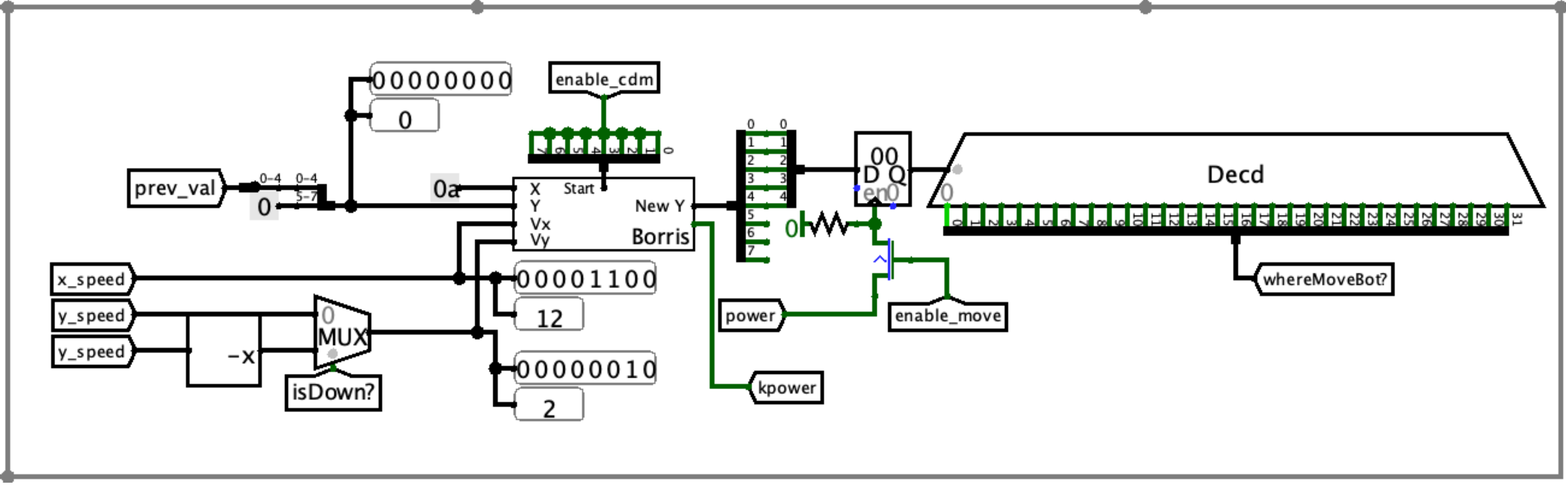


Bot movement

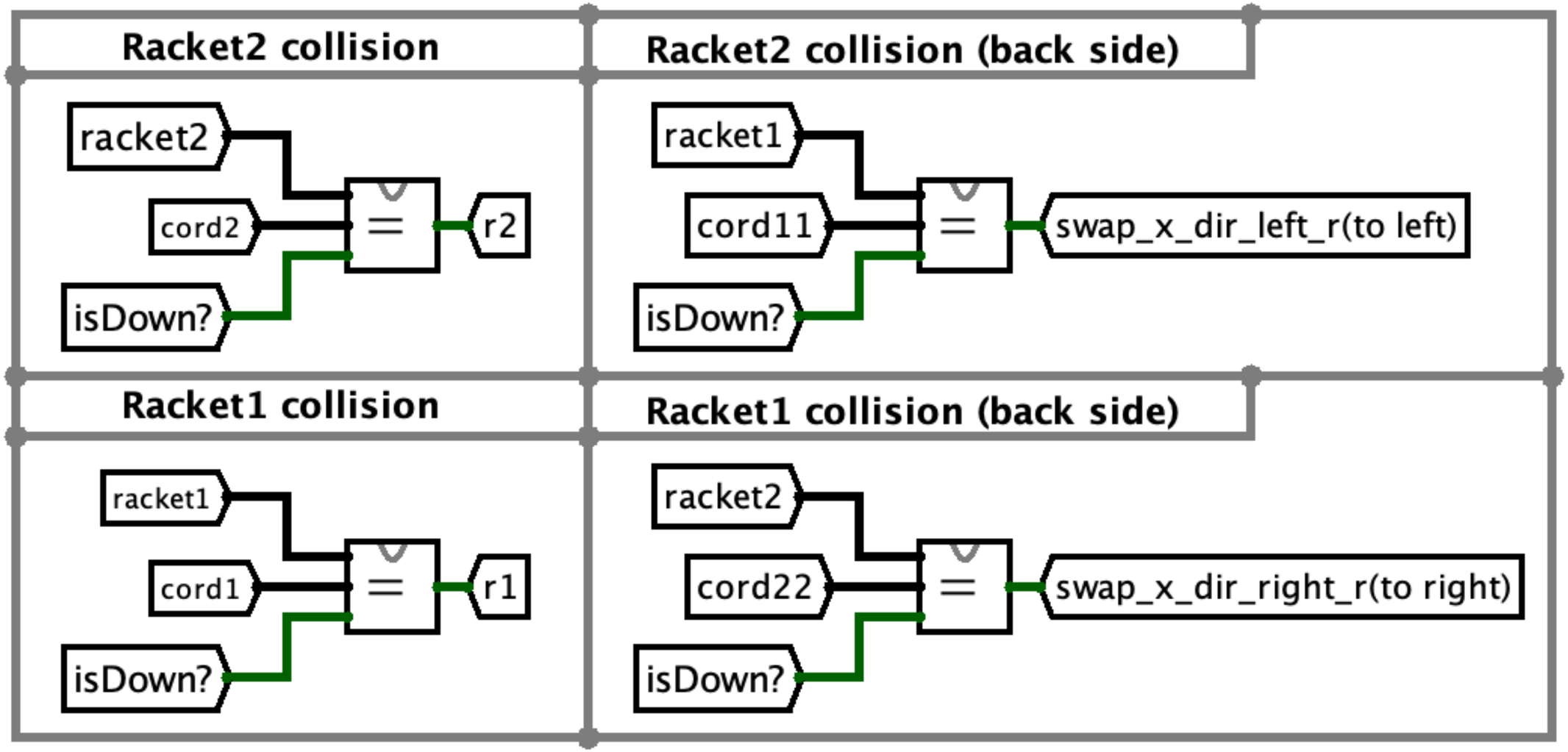




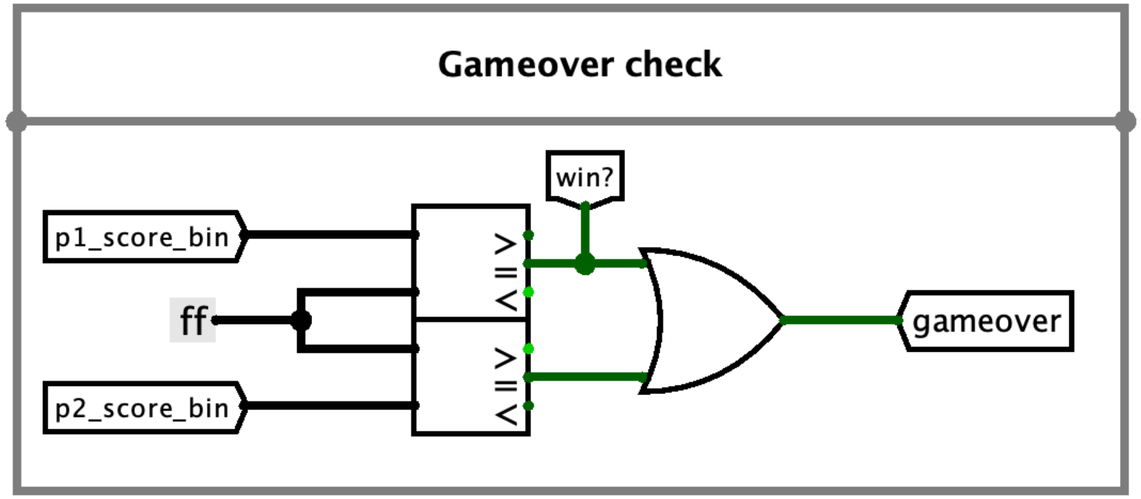


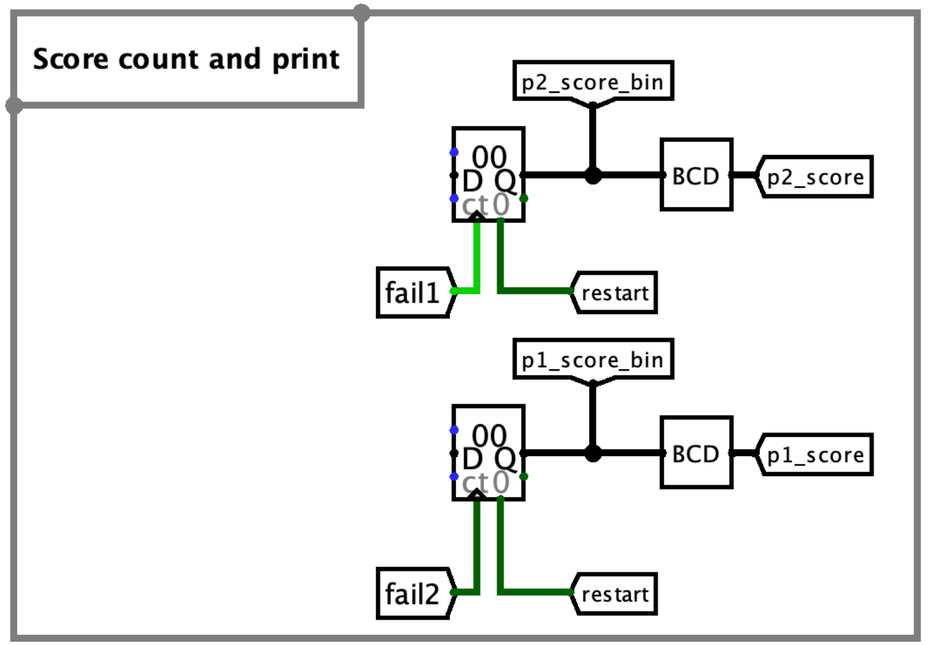


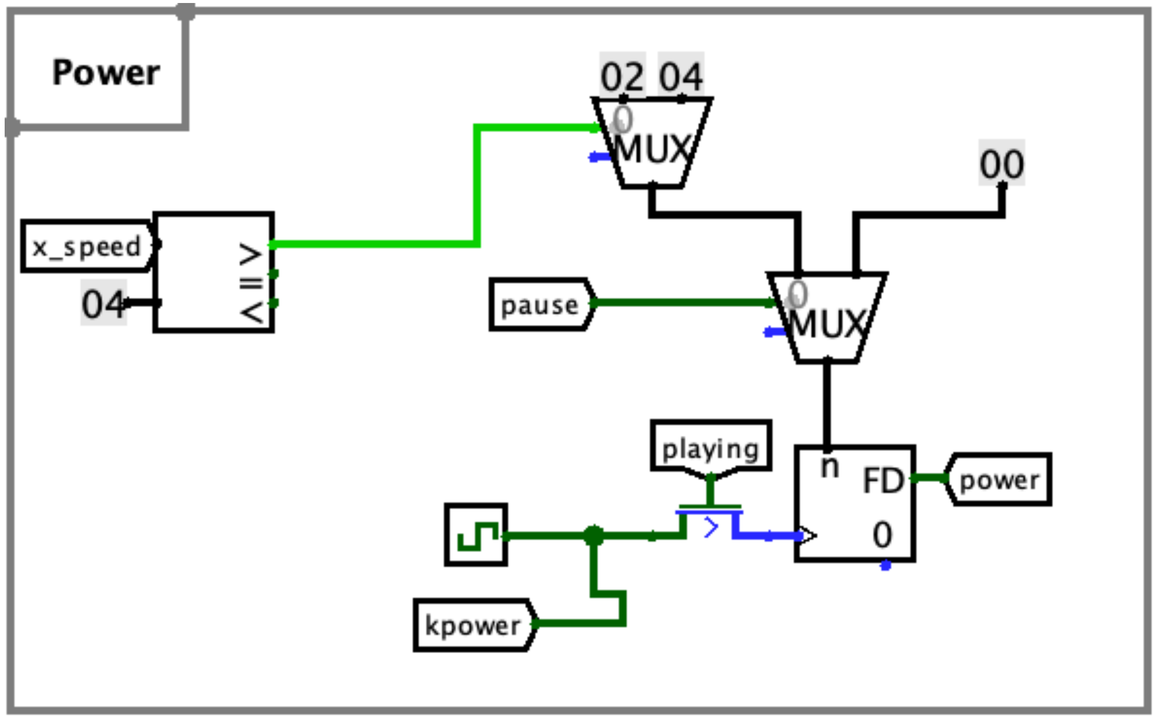
Ball and racket collision test

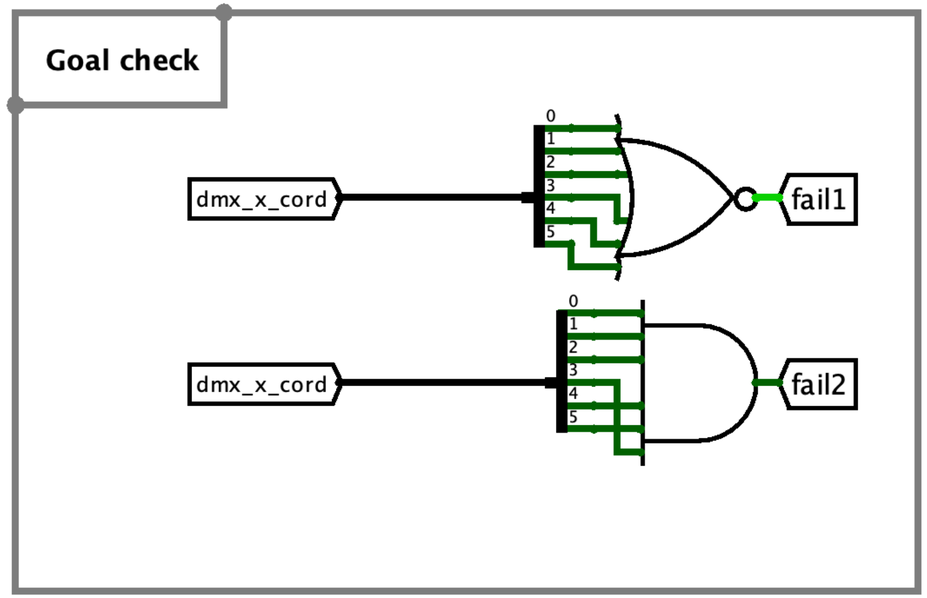


Misc


Gameover check
win?
p1_score_bin
ff
p2_score_bin
gameover


Score count and print
p2_score_bin
00
D Q
ct 0
BCD
p2_score
fail1
restart
p1_score_bin
00
D Q
ct 0
BCD
p1_score
fail2
restart

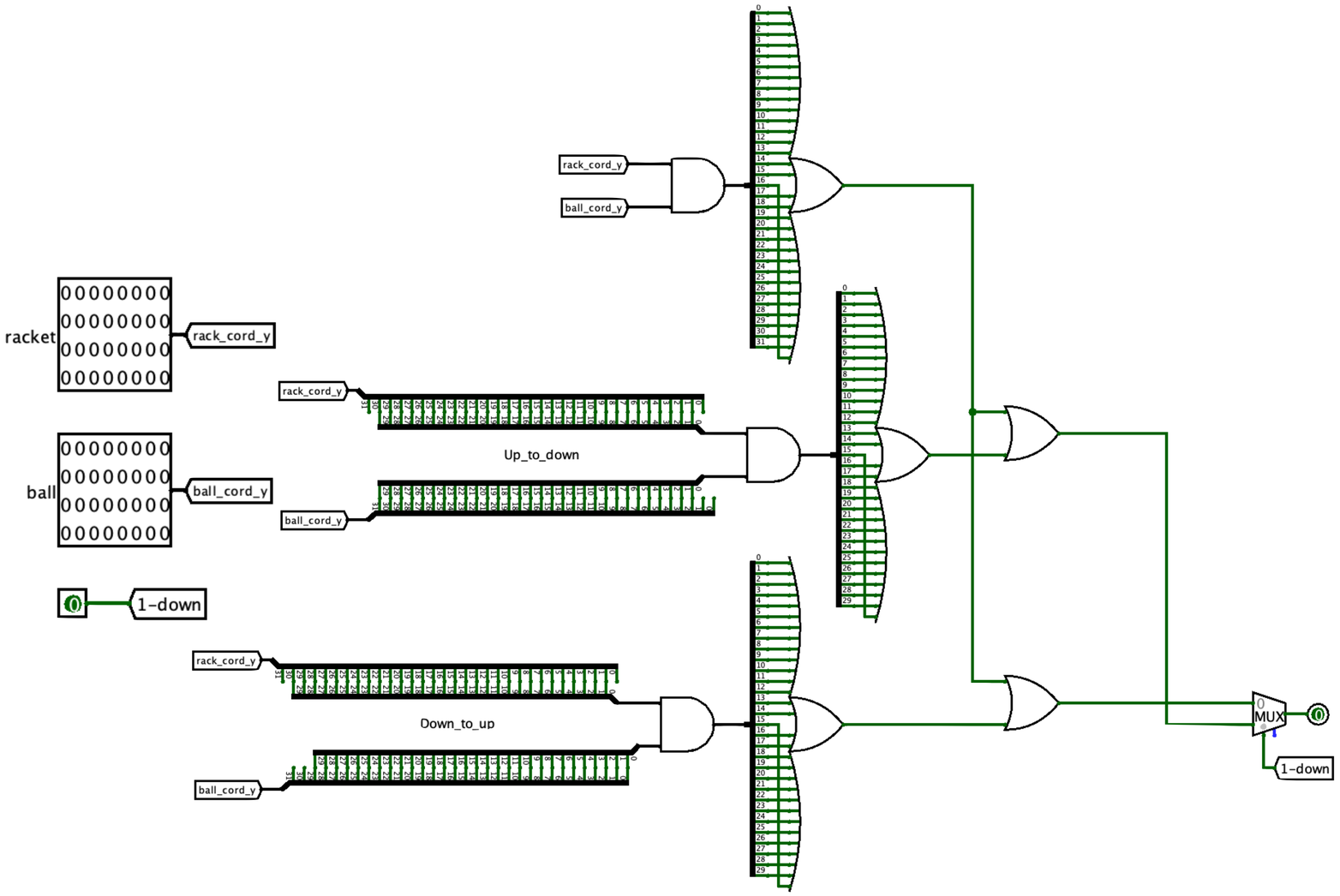

Power
02 04
0
MUX
00
x_speed
04
pause
0
MUX
playing
n
FD
0
power
kpower


Goal check
0
1
2
3
4
5
dmx_x_cord
fail1
dmx_x_cord
fail2

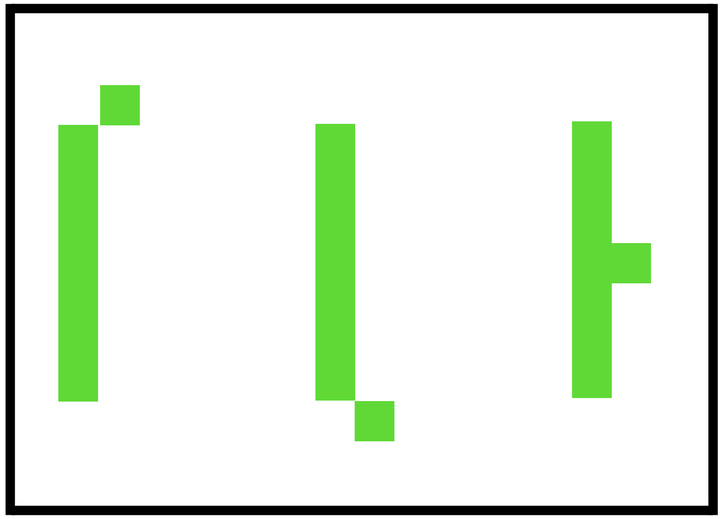
Subcircuits

Comp-y-coord

Purpose: checks for ball and racket collision.



attack variations:

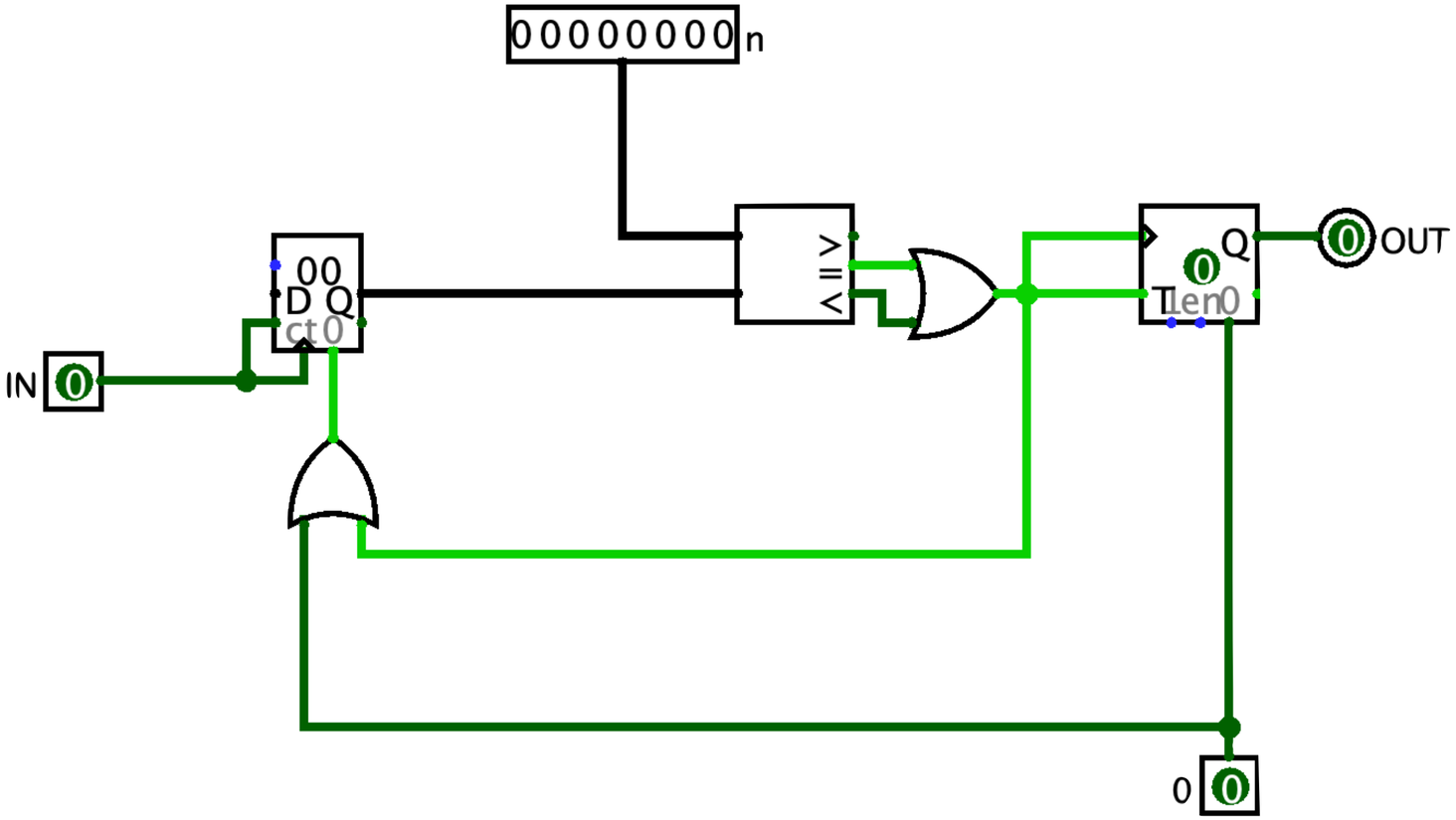


Frequency divider

Purpose: added control of the clock rate.

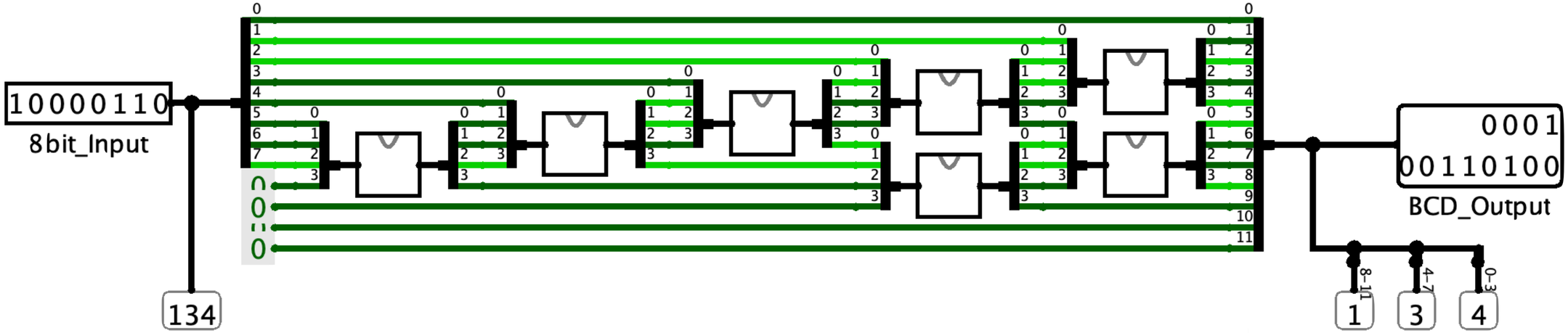
Use cases:

- timer
- flashing “SPACE” text
- ball movement
- register the last button press

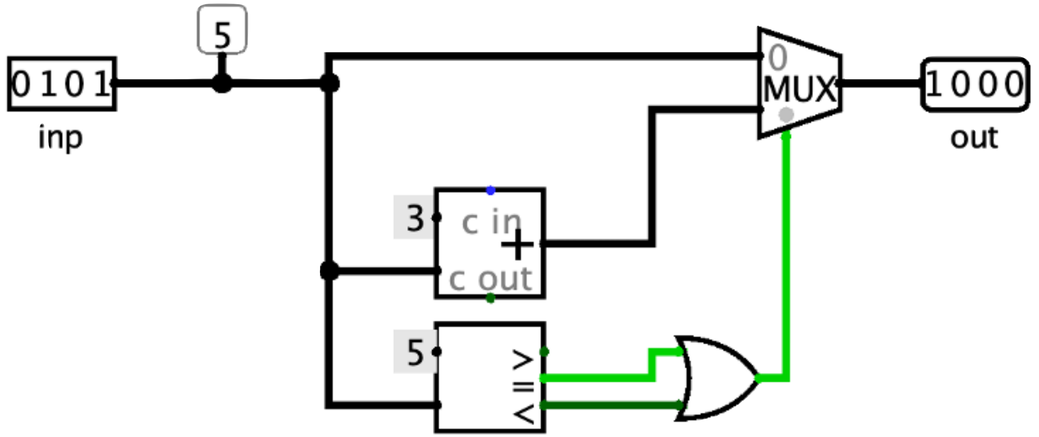


Binary Coded Decimal

Purpose: convert 8bit number into 12 bit BCD representation



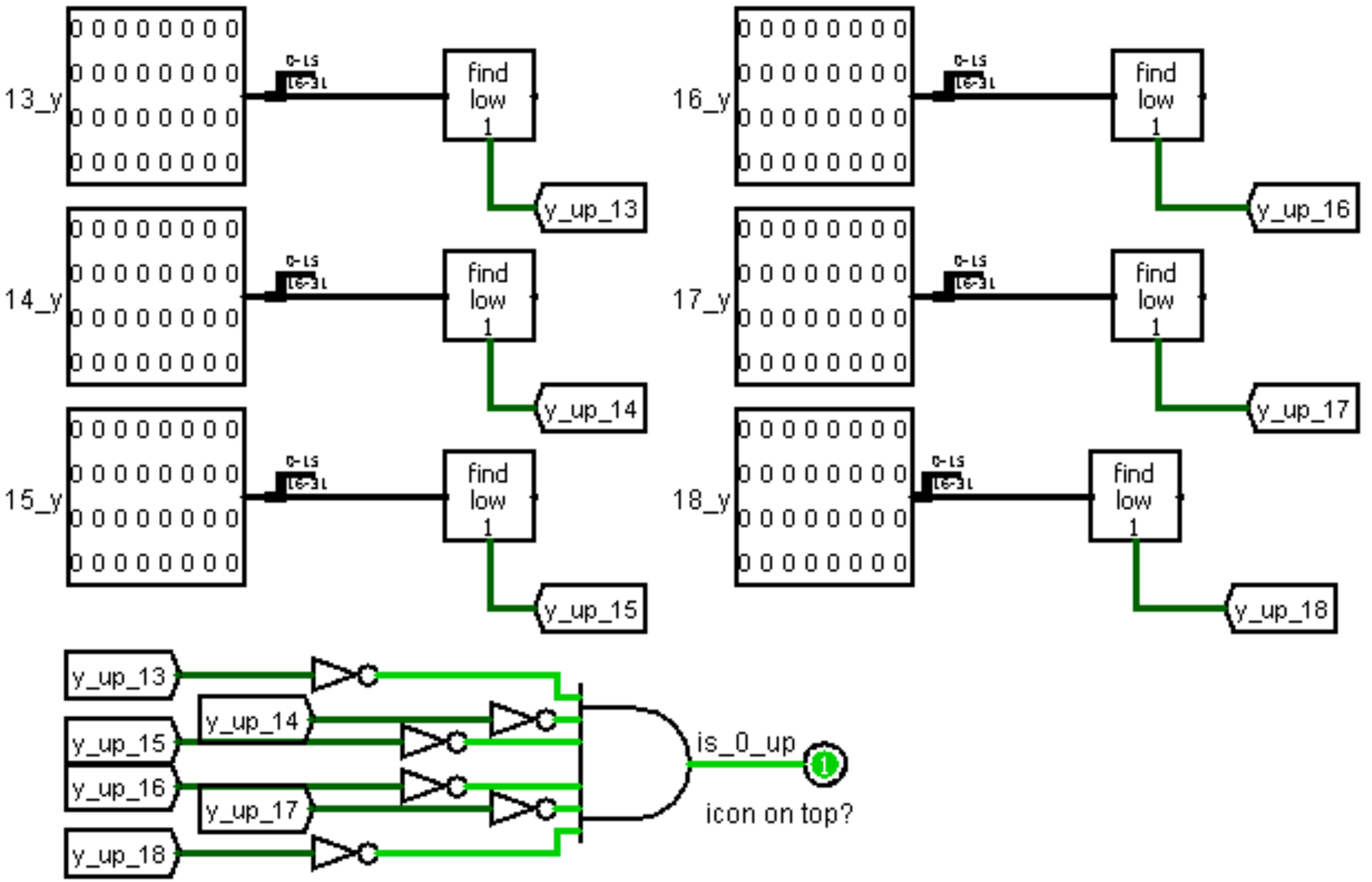
BCD HELPER



Pause

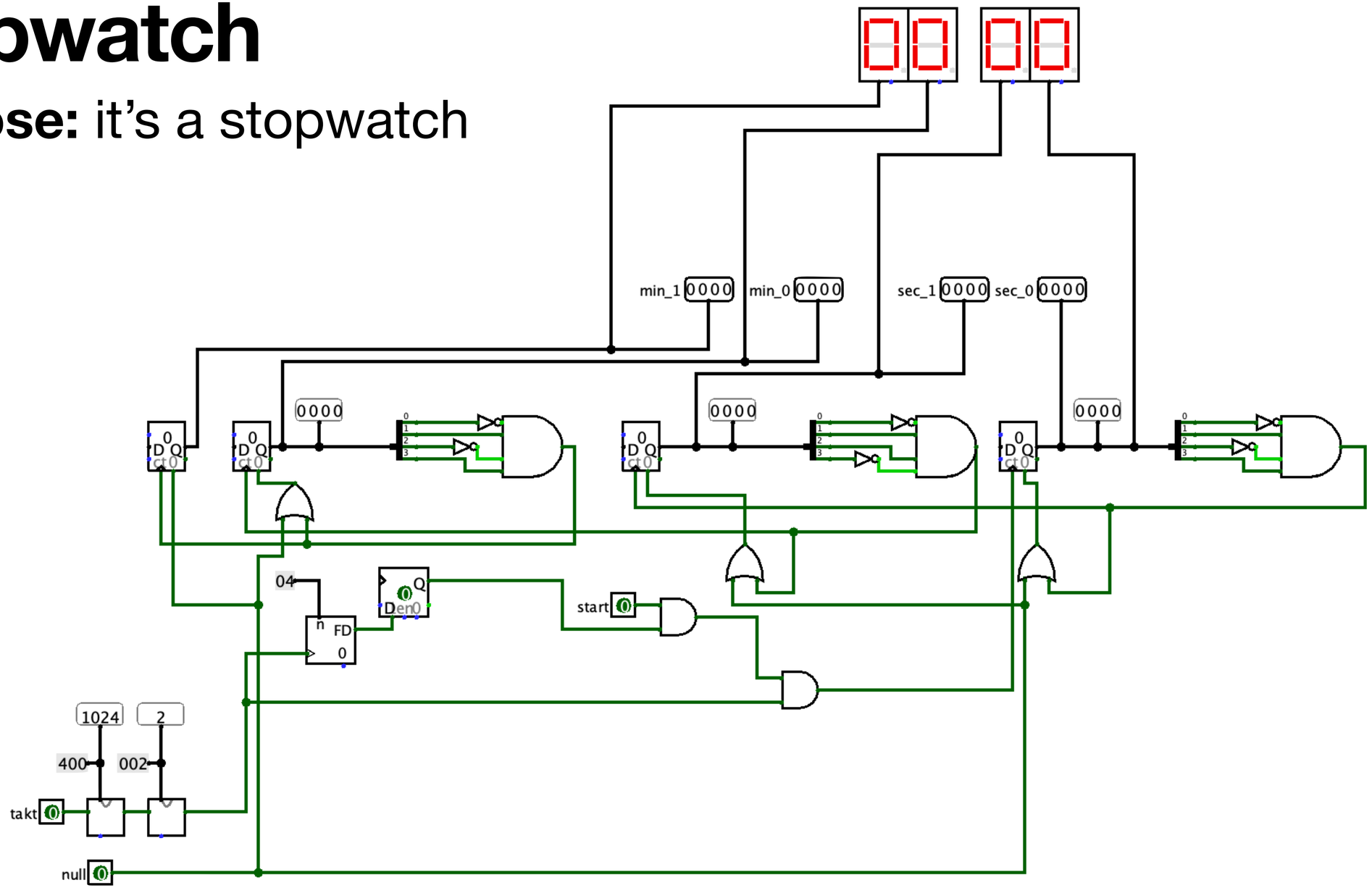
Purpose:

position the pause icon such that it does not overlap with the ball



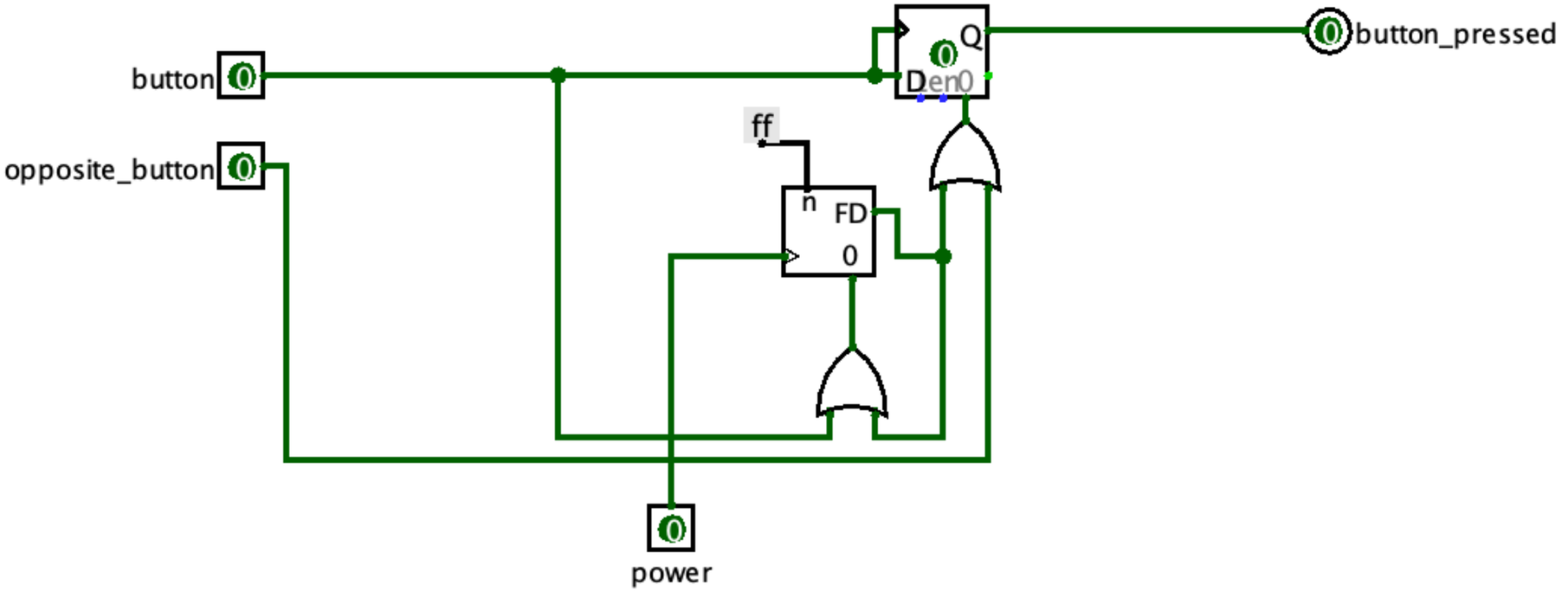
Stopwatch

Purpose: it’s a stopwatch



Last button pressed

Purpose: extend the signal in the raised state

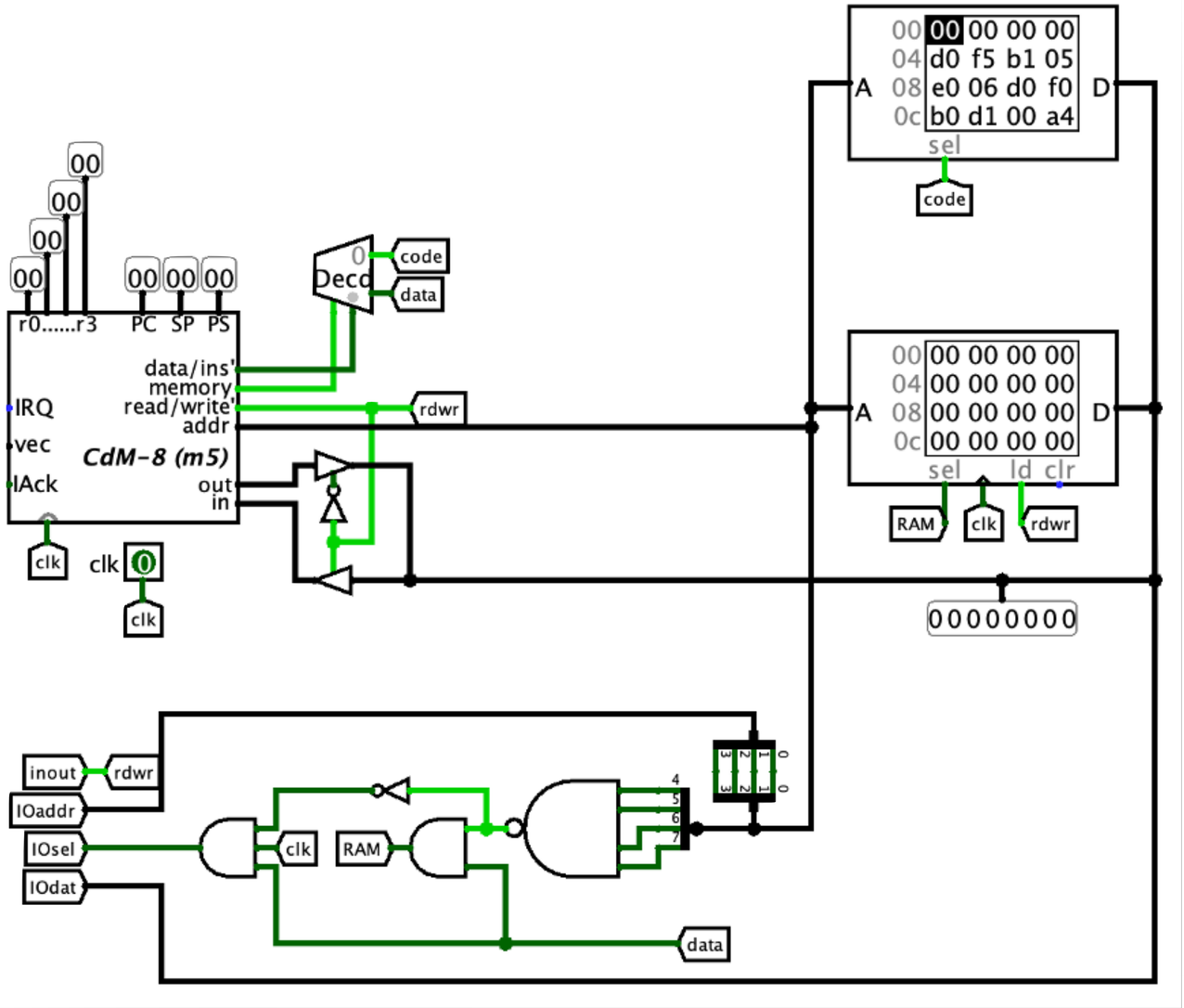


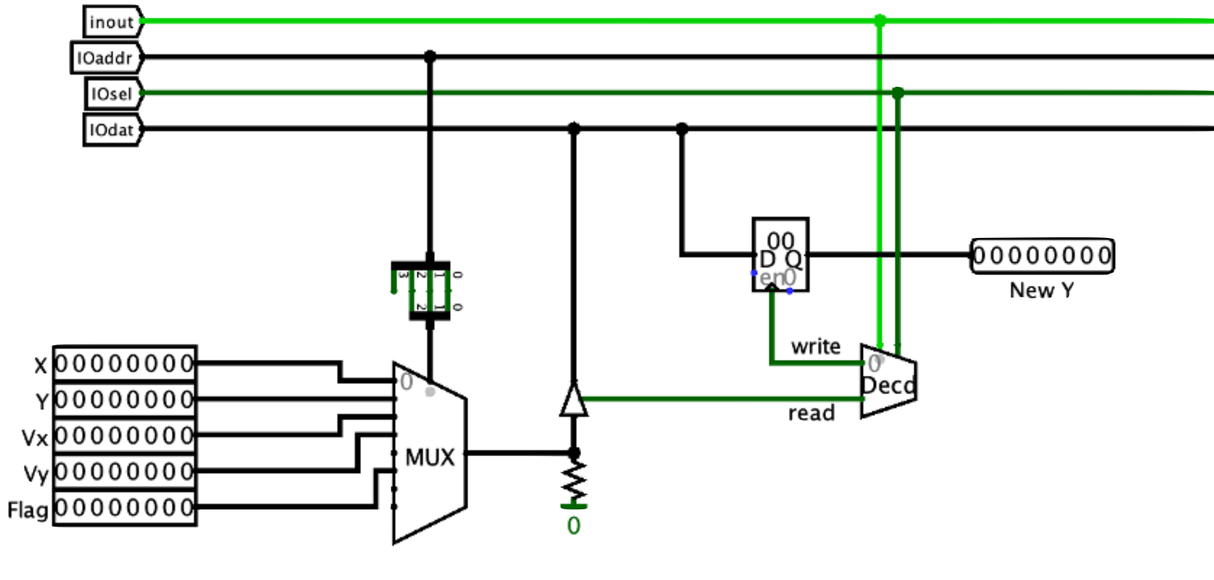
CdM8

Bot Borris

Purpose: bot for single player mode

- Harvard architecture
- 0xF0 — 0xFF reserved for I/O

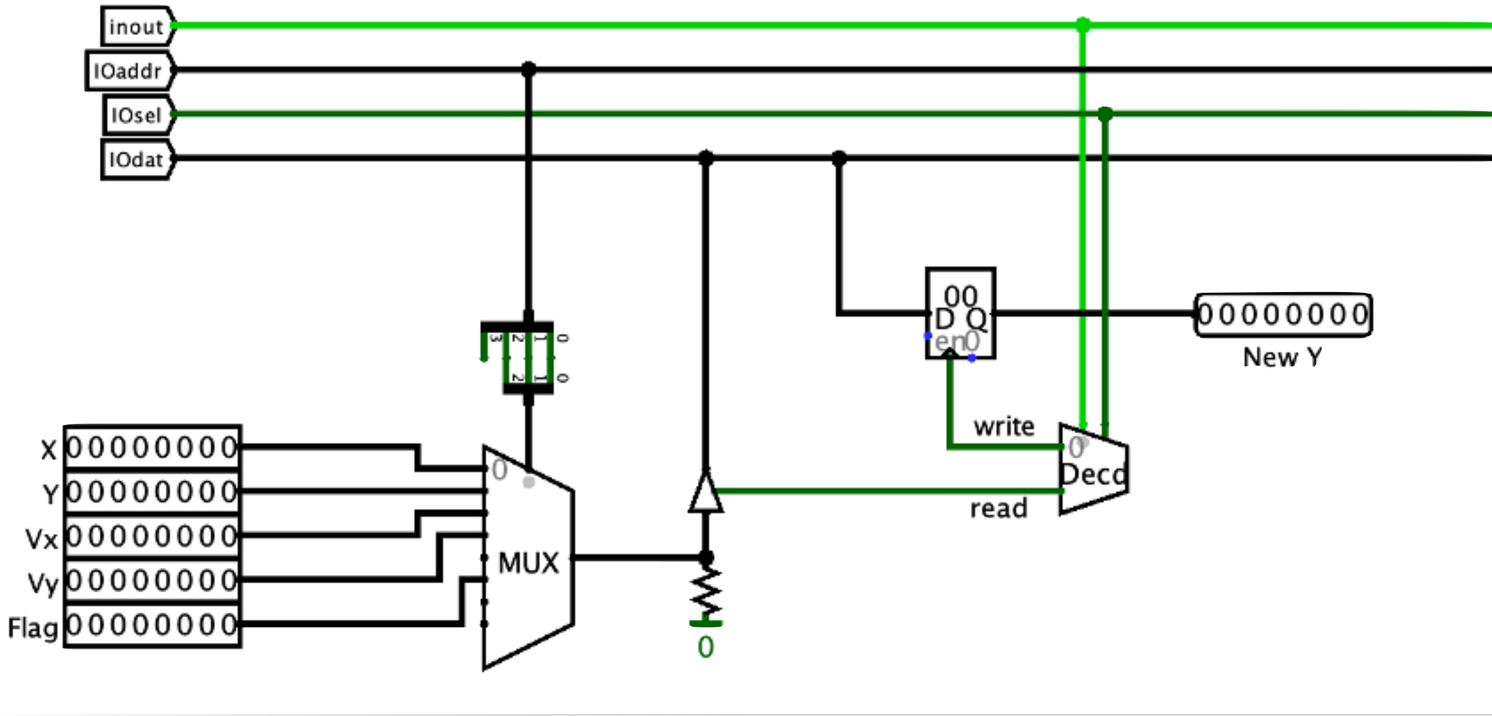




Bot Borris

$Y_{NEW} = (Sy + (Vy * Sx) / Vx) \% 32$

1. $V_Y = VY * SX$
2. $t = VY / Vx$
3. $SY = SY (+/-) t$
4. $SY = SY \% 32$



```
68  # Begin: Vy = (Vy * Sx)
69      # Vy is in r0
70      ldi r1, 0x00  # Sx
71      ld r1, r1
72      ldi r2, 0     # res
73
74      while
75          tst r1
76      stays nz
77          if
78              shr r1
79          is cs
80              add r0, r2
81          fi
82          shl r0
83      wend
84  # End: Vy = (Vy * Sx)
```

```
# Begin: t = (Vy / Vx)
    # Vy is loaded in r2
    move r2, r0   # Vy

    ldi r1, 0x02  # Vx
    ld r1, r1

    ldi r2, 0x02  # temp (also Vx)
    ld r2, r2

    ldi r3, 0     # result

    # Stage 1
    while
        cmp r0, r1    # a - b
    stays pl          # not negative
        tst r1
        shl r1
    wend
    shr r1

    # Stage 2 and 3
    while
        cmp r1, r2        # bnew - b
    stays pl              # not negative
        if
            cmp r0, r1
        is pl
            neg r1
            add r1, r0
            neg r1

            inc r3
            shl r3
            shr r1
        else
            if
                cmp r1, r2
            is le
                tst r3
                shl r3
                break
            fi
            tst r1
            shr r1        # shift b right
            shl r3        # shift ans right
        fi
    wend
    shr r3
# End: t = (Vy / Vx)
```

```
141 # Begin: Sy = Sy (+/-) t
142     # t is loaded into r3
143
144     ldi r0, 0x01  # Sy
145     ld r0, r0
146
147     ldi r1, 0xe0  # sign of Vy
148     ld r1, r1
149
150     if
151         tst r1
152     is nz
153         neg r3
154     fi
155     add r3, r0
156 # End: Yo = Yo (+/-) t
157     # Ynew (Yo or Sy) is loaded into r0
```

```
160 # Begin: % 32
161     # r2 - holds the sign
162     # r3 - holds the 6th bit
163     if
164         tst r0
165     is mi
166         neg r0
167         ldi r2, 1
168     else
169         ldi r2, 0
170     fi
171
172     ldi r1, 31
173     if
174         cmp r1, r0    # 31 - Ynew
175     is  mi            # negative
176         tst r0
177         shl r0
178         tst r0
179         shl r0
180
181         if
182             shl r0
183         is cs
184             ldi r3, 1 # sign bit for 6th bit
185         else
186             ldi r3, 0 # sign bit for 6th bit
187         fi
188
189         tst r0
190         shr r0
191         tst r0
192         shr r0
193         tst r0
194         shr r0
195
196         xor r3, r2
197
198         if
199             tst r2
200         is nz
201             # y = 31 - (y % 32)
202             # r1 already contains 31 (see line 172
203             neg r0
204             add r1, r0
205         fi
206     fi
207 # End: % 32
208
```

Conclusion

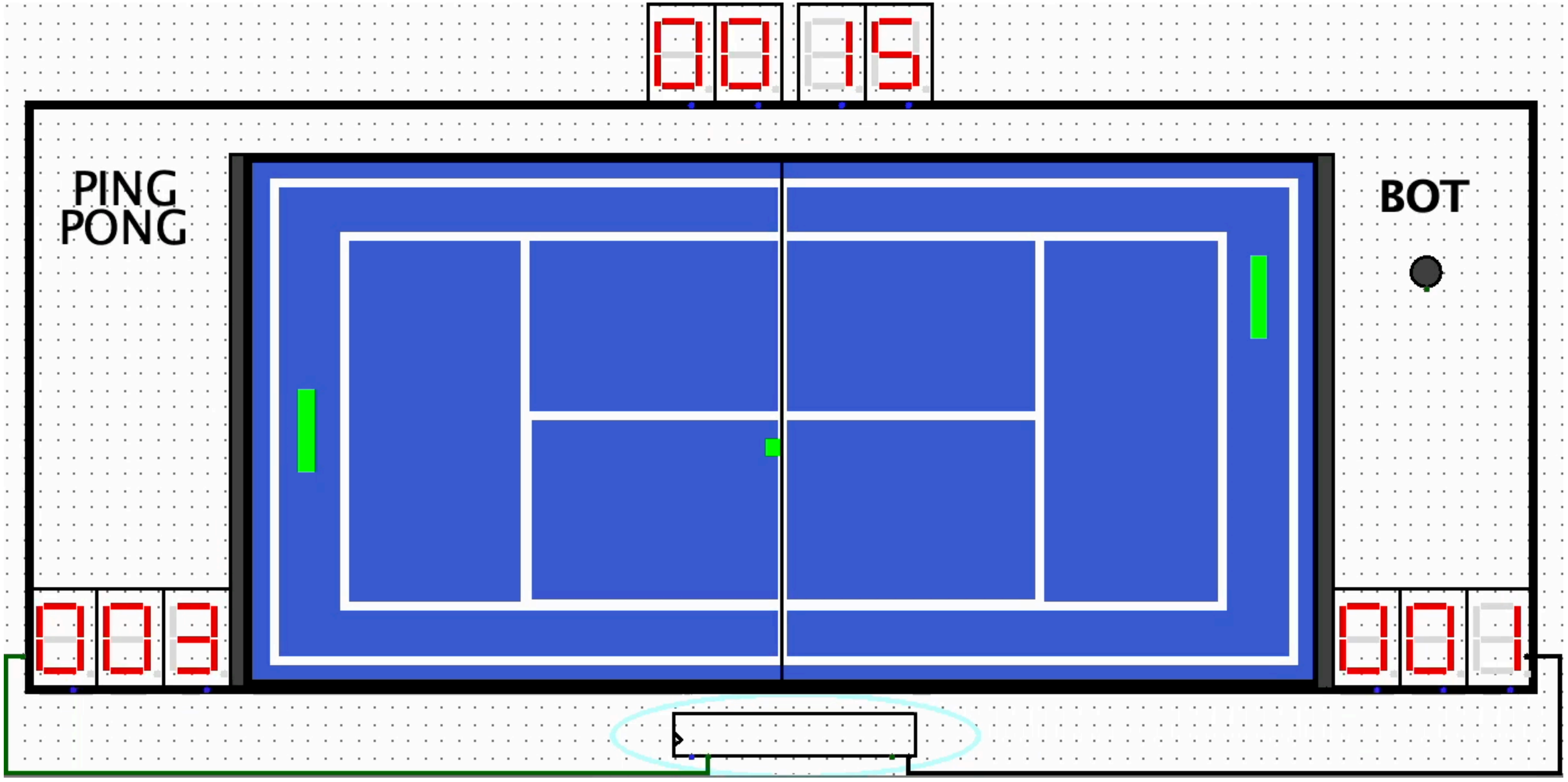
We were able to successfully create a working version of Pong in Logisim

Our improvements

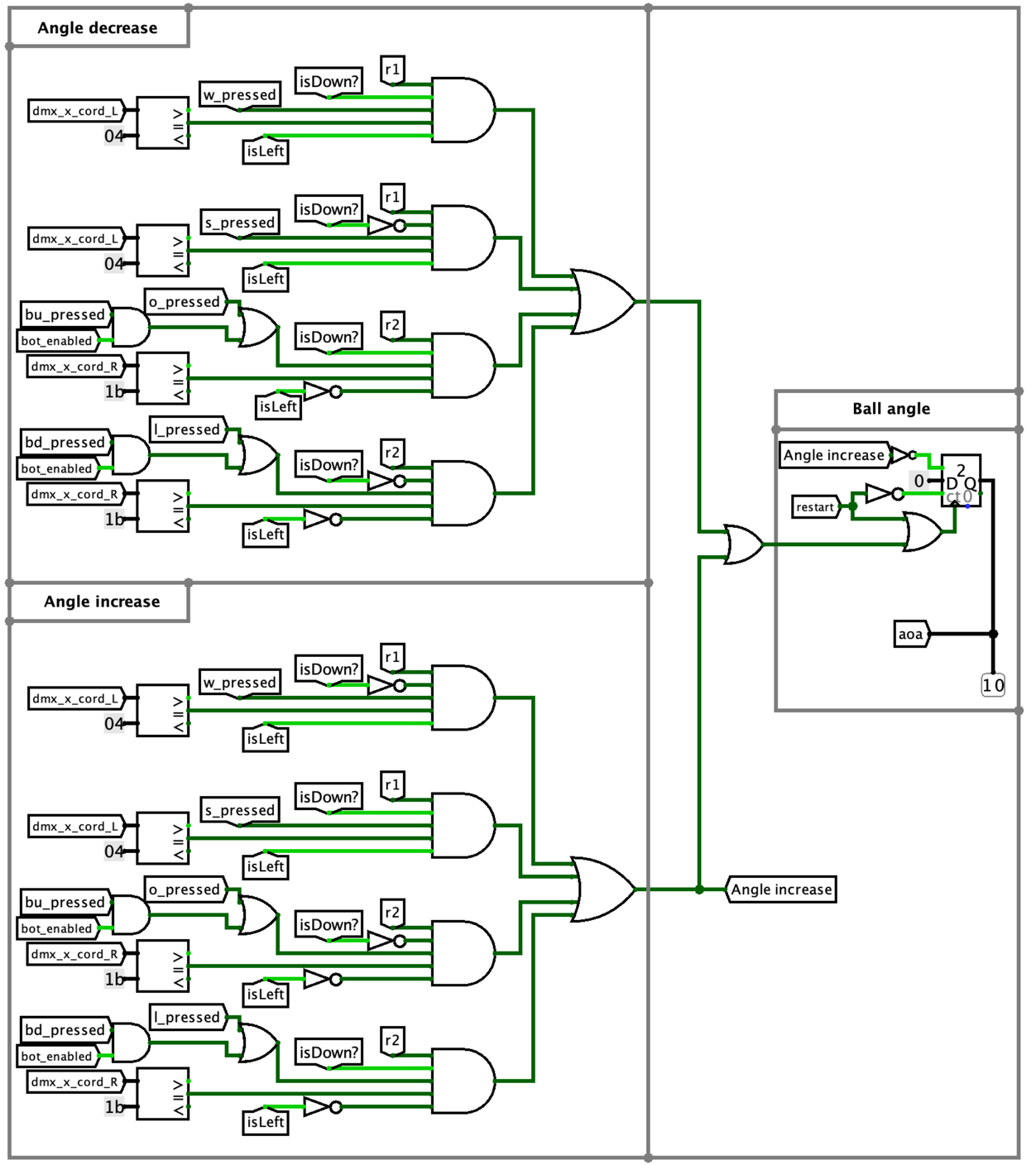
- Single player mode, enabled by BotBorris
- Bigger screen
- More colors
- Improved game mechanics

Thank you

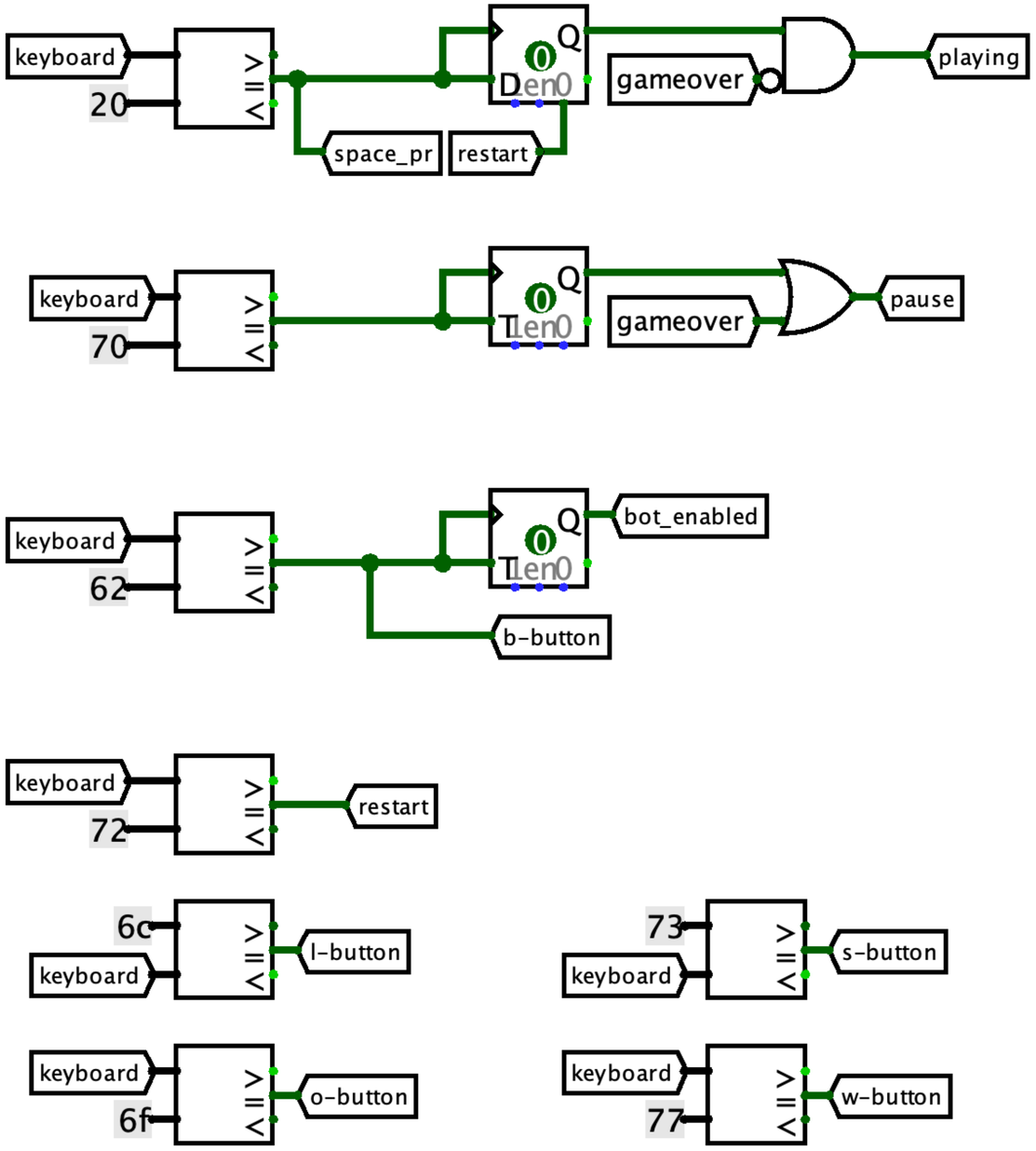
DEMO


00:15
PING
PONG
BOT
003
001

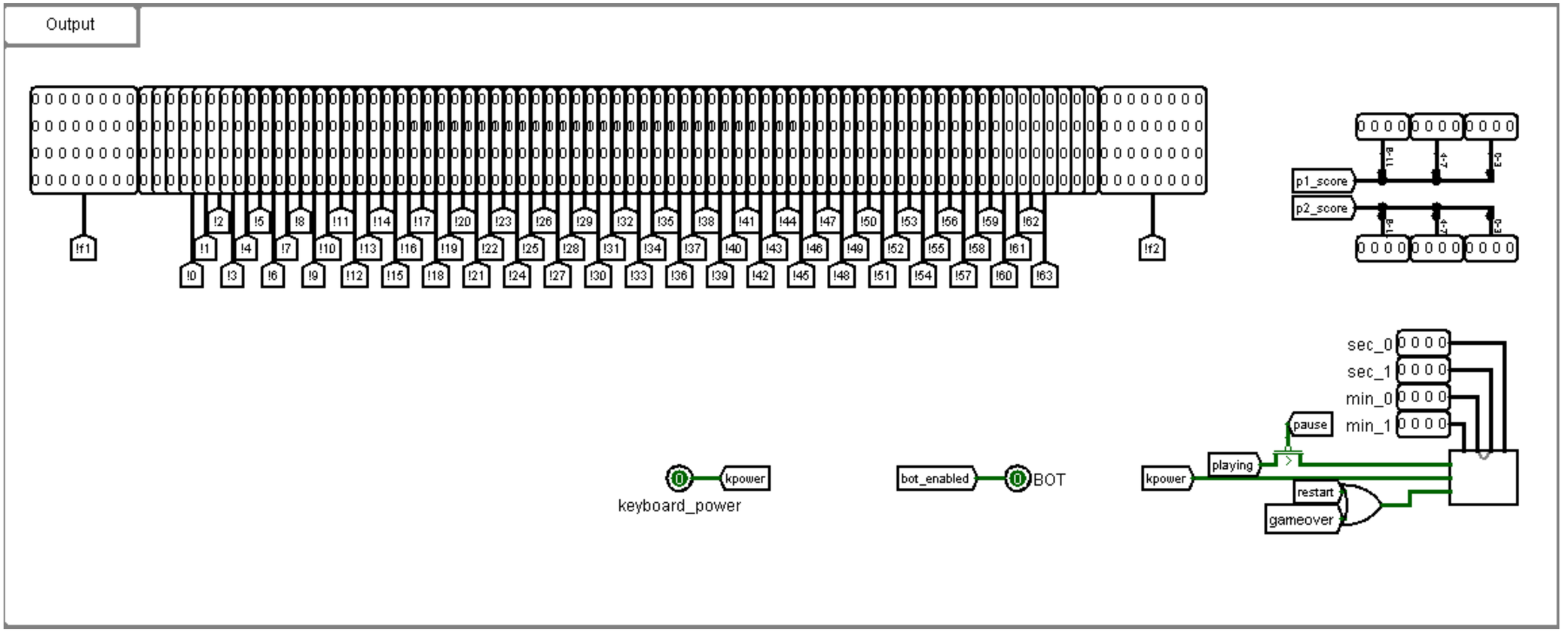
Changing the angle of attack of the ball



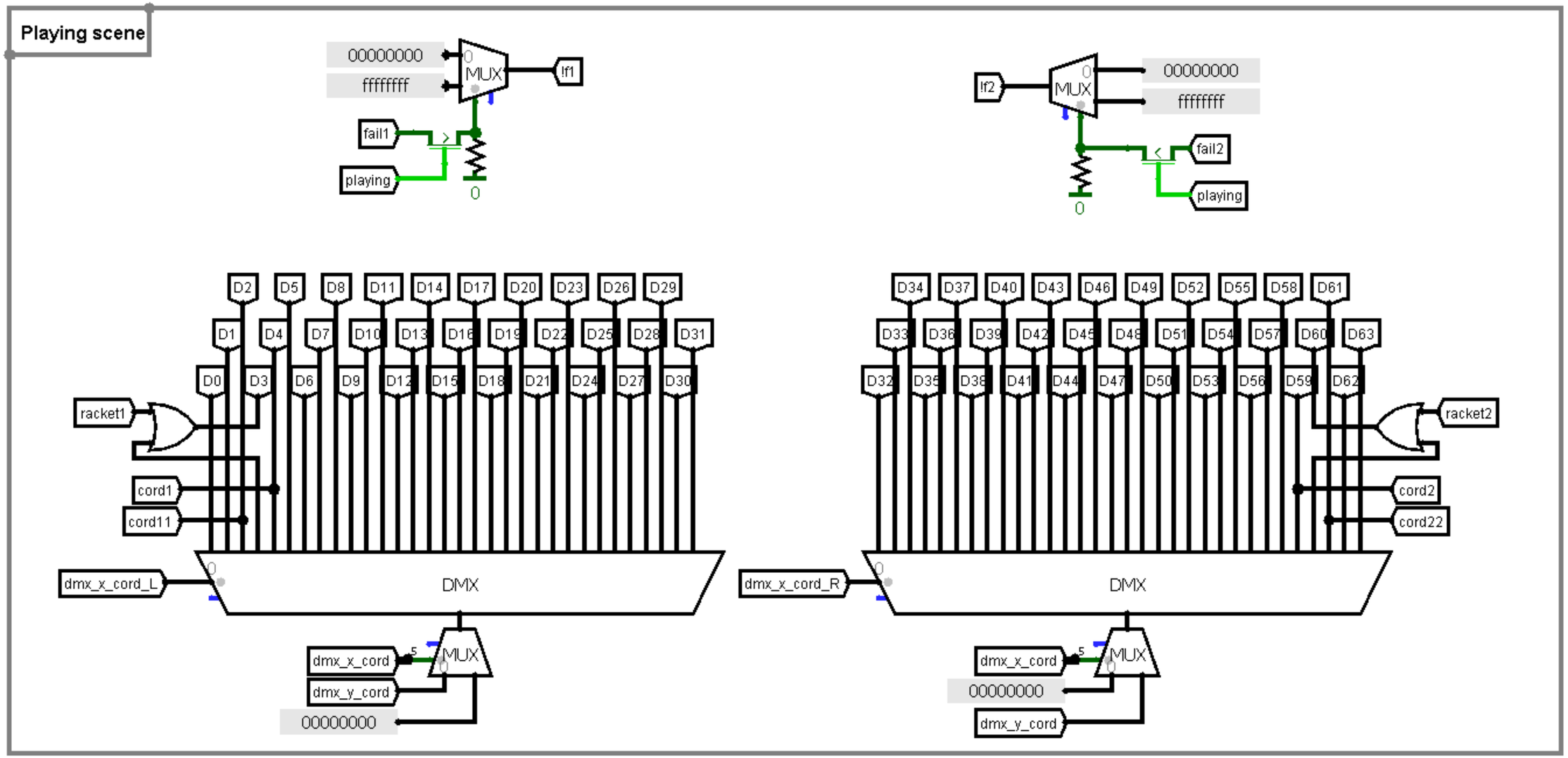
Keyboard Input



Contacts for data transfer between two circuits



Playing screen



Keyboard Input

Remembering the last button press