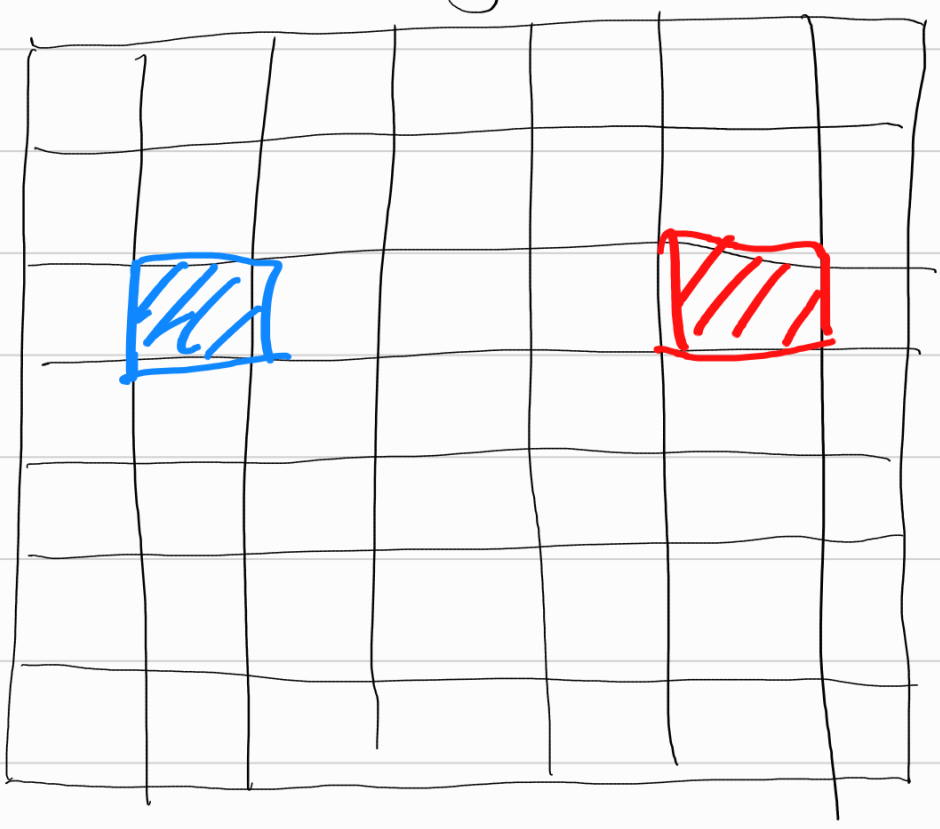
Ball Bouncing Simulation



m x n board

Angle

Power

Wind

Gravity

Time

Start | Reset

Start

Target

Set angle
Power
horizatal gravity
gravity 9.8 m/s²
time

> will replace with Velocity X, Y

- "Wind" or acceleration X
- acceleration Y, will scale down to 1 block/s²

Add walls by clicking on the board and on edges

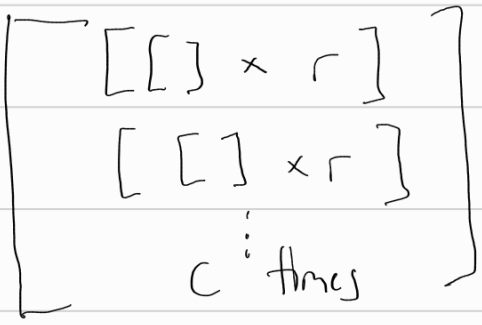
Setting Up the board

col = c , row = r , board = [ ]

push c to board → [[ ], [ ], ... c times]

push r to each c

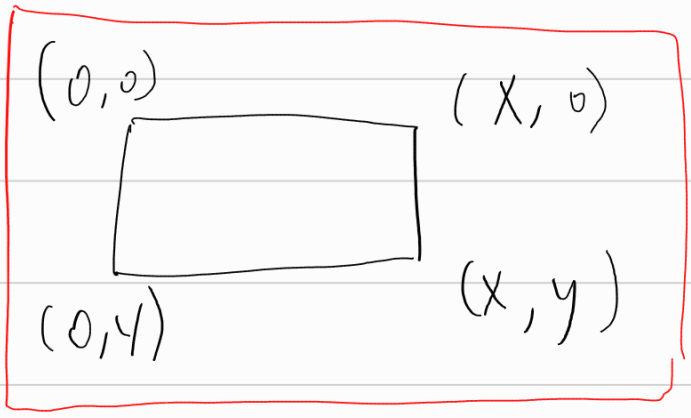
[[ ] × r] [[ ] × r] ... c times]



becomes (row, col)

instead of (col, row)

(pushing array swaps the convention, but col = x , row = y still)



+X (col)

+Y (row)

start w/ $V_y = 0$?

above wall = true

to make sure if $V_y = 0$, ball will not move upward based on velocity / collision equations

Collision Cases Velocity - if these walls exist

→ before

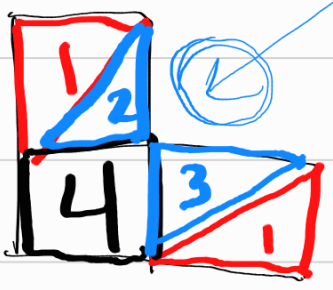
→ after

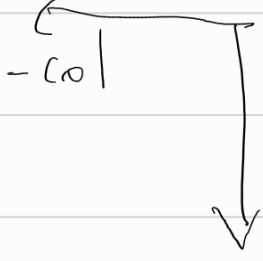
1 - 1st check

2 3 - 2nd, 3rd check
order does not matter

4 - 4th check

Case 1: (bottom left) (-x, +y)

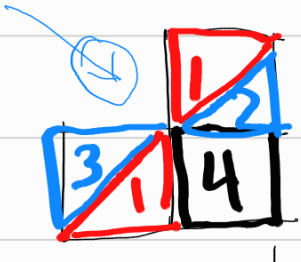


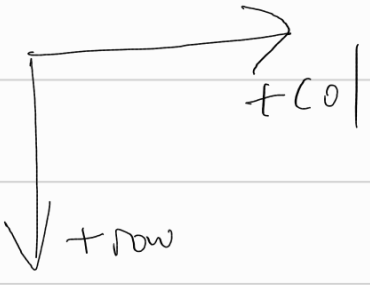


if 1: reverse both V_x, V_y
2: reverse V_x
3: reverse V_y
4: reverse V_x, V_y

* Due to odd interactions, 1st check will end the run

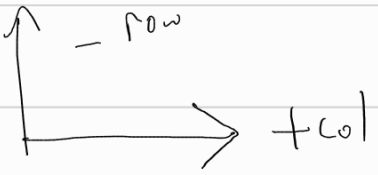
Case 2: (bottom right) (+x, +y)





if 1: reverse both V_x, V_y
2: reverse V_x
3: reverse V_y
4: reverse both

Case 3: (top right) (+X, -y)



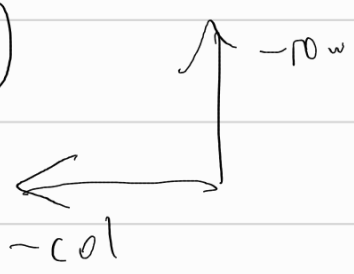
if 1: reverse both V_x, V_y

2: reverse V_y

3: reverse V_x

4: reverse V_x, V_y

Case 4: (top left) (-X, -y)



if 1: reverse both V_x, V_y

2: reverse V_x

3: reverse V_y

4: reverse V_x, V_y

Case 5: (right) (+X, 0)

reverse V_x

Case 6: (left) (-X, 0) ← - col

reverse V_x

Case 7: (down) (0, +Y) ↓ row

reverse V_y

Case 8: (up) (0, -Y) ↑ -row

Reverse V_y

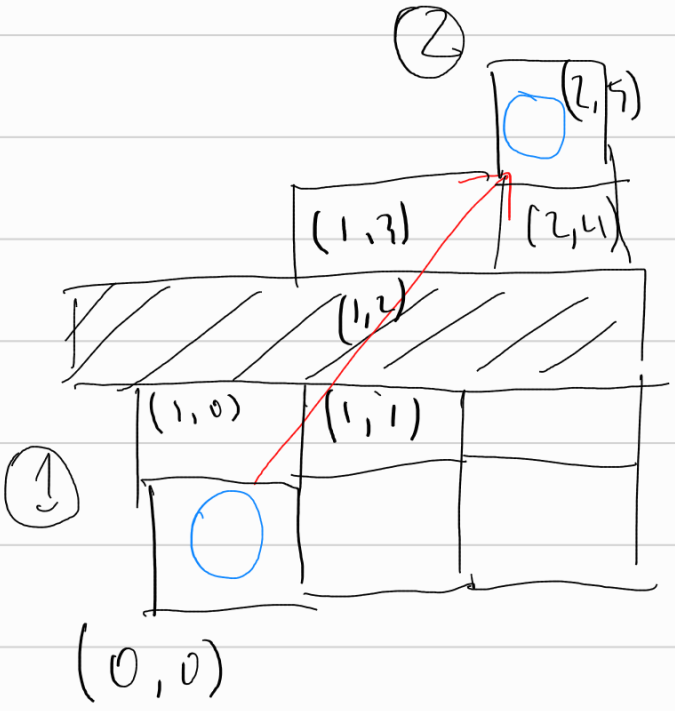
Case 9: (+col, 0) $g != 0$

Case 10: (−col, 0) $g != 0$

$g != 0$ means wall under the ball

Find Closest Non Wall

Ball jumps over wall (current algo. can only handle ball moving 1 cell)



1- Take max, absolute value of $\Delta X, \Delta Y$

2- Record sign (+/-) of delta to relocate in correct direction ball is heading

iterate to go from farthest to closest

temp X = reference old position
temp Y

```
for ( let i = 1 ; i < max(ΔX, ΔY) , i++)
```

- for higher delta, increment by 1
- for lower delta, increment by $\frac{\Delta X \text{ or } \Delta Y}{\Delta max + i}$
- if both are equal, increment both by 1

Assume Start (16,33) - i.e: (if (17,39) and (17,40) are walls and (17,38) is not, (17,40) relocates to (17,38))

End (17,40)

S - Start (0,0)

E - End (2,5)

W - All possible wall checks

(find Closest NonWall)

if all checkpoints are walls, relocates to initial starting, will stop if non wall is found

$\Delta Y = 7$ $\Delta Y > \Delta X$

$\Delta X = 1$

$i=1$: $y = 40-1 = 39$, $x = x_1 + \frac{\Delta x}{1} = 17$

$i=2$: $y = 39-1 = 38$, $x = 16 + \frac{1}{2} \approx 17$

$i=3$: $y = 38-1 = 37$, $x = 16 + \frac{1}{3} \approx 16$

$i=4$: $y = 37-1 = 36$, $x = 16 + \frac{1}{4} \approx 16$

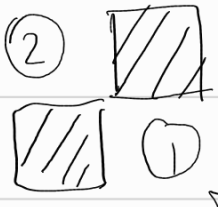
$i=5$: $y = 36-1 = 35$, $x = 16 + \frac{1}{5} \approx 16$

$i=6$: $y = 35-1 = 34$, $x = 16 + \frac{1}{6} \approx 16$

Special Case:

— wall

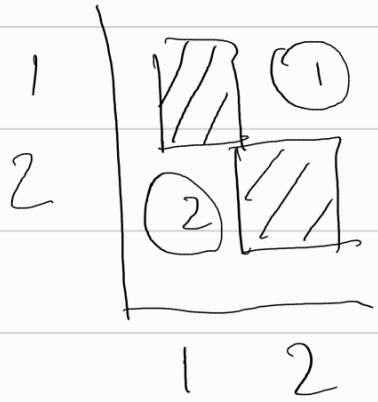
must check for walls in 1 row and 1 col blocks otherwise, can jump over



(-col, -row)

add to findClosestNonWall using Position(x,y)f − Position(x,y)i to determine which of 4 cases this is

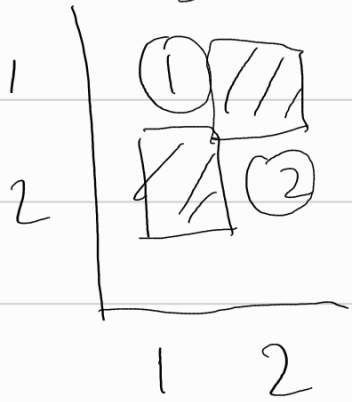
Case 1 bot left



$\Delta P_y = 2-1 = +$

$\Delta P_x = 1-2 = -$

Case 2 bot right

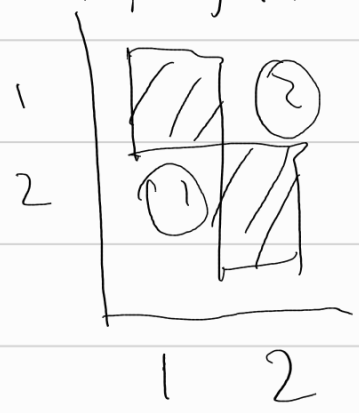


$\Delta P_y = 2-1 = +$

$\Delta P_x = 2-1 = +$

Case 3

top right

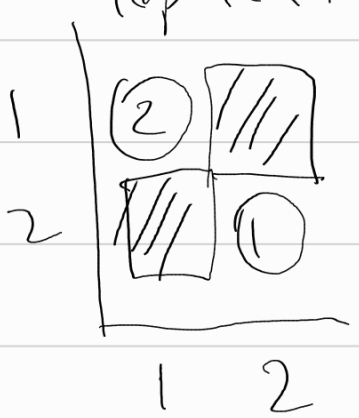


$\Delta P_y = 1 - 2 = (-)$

$\Delta P_x = 2 - 1 = (+)$

Case 4

top left



$\Delta P_y = 1 - 2 = (-)$

$\Delta P_x = 1 - 2 = (-)$

Since code iterates from farthest to inital position, need to add checks from (2) to both walls for all 4 cases