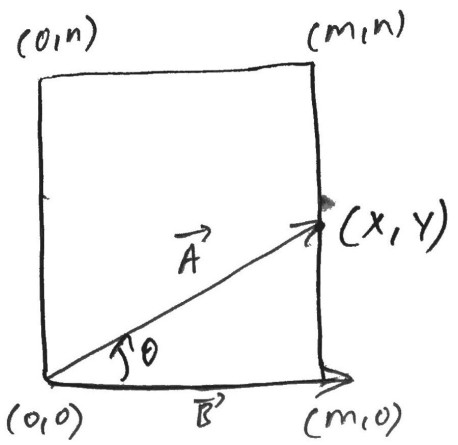
Algorigthm (Mathematical)



To find the point (x, Y) where it collides, two vectors are drawn, $\vec{A}$ & $\vec{B}$. $\vec{A}$ connects the point where it is currently & the future point. $\vec{B}$ connects the curre point and the extreme point on the wall of the table. $\vec{B}$ is called the base vector.

$$\vec{A} = [(x-0), (y-0)]$$

$$\vec{B} = [m-0, 0-0]$$

First, Dot product is computed.

$$\vec{A} \cdot \vec{B} = (x-0)(m-0) + (y-0)(0-0) = xm$$

$$\vec{A} \cdot \vec{B} = \|\vec{A}\| \, \|\vec{B}\| \cos\theta = [\|\vec{A}\| \cos\theta] \, \|\vec{B}\| = \|\vec{B}\|^2$$

$$\Rightarrow xm = (m-0)^2 \Rightarrow x = m //$$

Next, cross product is computed.

$$\vec{B} \times \vec{A} = -(\vec{A} \times \vec{B}) = \begin{vmatrix} i & j & k \\ m & 0 & 0 \\ x & y & 0 \end{vmatrix} = (my)\hat{k} \longrightarrow \|\vec{B} \times \vec{A}\| = my$$

But, $\|\vec{B} \times \vec{A}\| = \|\vec{B}\| \, \|\vec{A}\| \sin\theta$

$$= \frac{\vec{A} \cdot \vec{B}}{\cos\theta} \cdot \sin\theta$$

$$= \vec{A} \cdot \vec{B} \tan\theta \quad [\text{But}, \because \theta = 45]$$

$$= \vec{A} \cdot \vec{B} = \|B\|^2$$

$\Rightarrow my = \|B\|^2 = m^2$

$\Rightarrow y = m$ //

This way, future point (x, y) can be calculated at any current location (x', y')

Note: The direction of cross-product, i.e., $\vec{B} \times \vec{A}$ (or) $\vec{A} \times \vec{B}$ will be based on the direction of motion of the ball.

$\vec{B} \times \vec{A}$ ⟶ Anti-clock wise

$\vec{A} \times \vec{B}$ ⟶ clock wise

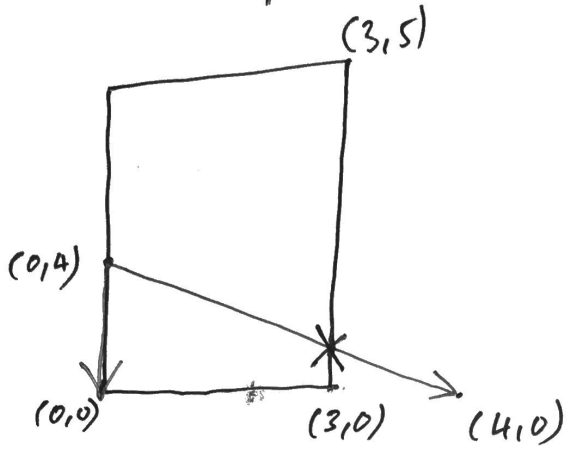
This is to maintain a +ve result (or) output from the cross-product

---

Exceptional Cases:

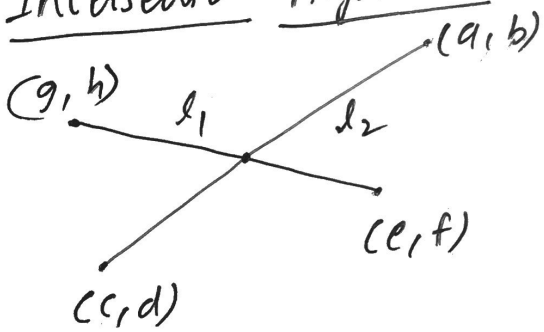
→ When the dimensions of the table are large, the output of cross-product & dot-product will be outsi the table. What this means is that the ball has colided with a different wall.

For example, when we take a 3x5 table, the 3rd collision is at (0,4). When trying to find the next point the output is outside the table.



In this scenario, the ball hits the right wall instead of the bottom wall. Therefore, the next point is the intersection of line segments made by [(0,4),(4,0)] & [(3,0),(3,5)]. The direction of the ball is also reversed.

Intersection Algorithm



Before finding the intersection point, cross product is computed to find if the lines ARE PARALLEL OR NOT.

If $l_1 \parallel l_2$, $l_1 \times l_2 = 0$

If the cross product gives a value other than 0, affine lines theorem is used to find the intersection.

$$l_1 : \begin{pmatrix} g \\ h \end{pmatrix} + s\begin{pmatrix} g-e \\ h-f \end{pmatrix}$$

$$l_2 : \begin{pmatrix} a \\ b \end{pmatrix} + t\begin{pmatrix} c-a \\ d-b \end{pmatrix}$$

$$l_1 = l_2$$

$$\begin{pmatrix} g \\ h \end{pmatrix} + s\begin{pmatrix} g-e \\ h-f \end{pmatrix} = t\begin{pmatrix} c-a \\ d-b \end{pmatrix} + \begin{pmatrix} a \\ b \end{pmatrix}$$

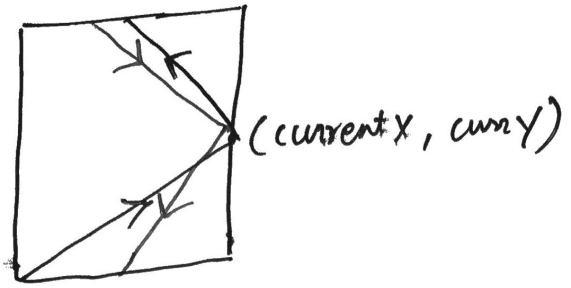
$$\Rightarrow \begin{pmatrix} g-a \\ h-b \end{pmatrix} = \begin{pmatrix} t(c-a) - s(g-e) \\ t(d-b) - s(h-f) \end{pmatrix}$$

⇒ These two equations are solved to find s, t and later plugged into either l_1 or l_2 to find the intersection point.

To determine if the intersection point actually lies on the line segment, we check if BOTH s, t are ≤ 1.

Deciding the base vector

current point at right wall

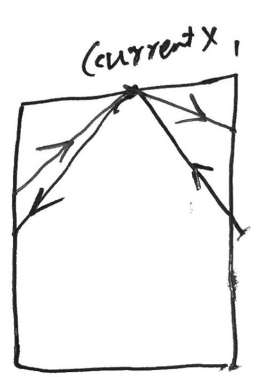


current Y - Previous Y

: >0 → Base : [Point → (m,n)]

: <0 → Base : [Point → (m,0)]

current point at Top wall

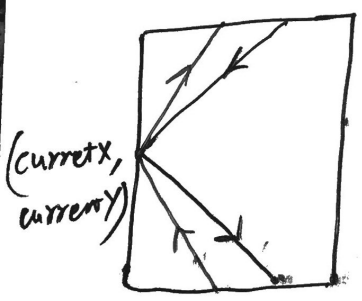


current X - Previous X

: >0 → Base : [Point → (m,n)]

: <0 → Base : [Point → (0,n)]

current point at left wall



current Y - Previous Y

: <0 → [Point → (0,0)]

: >0 → [Point → (0,n)]

current point at bottom wall

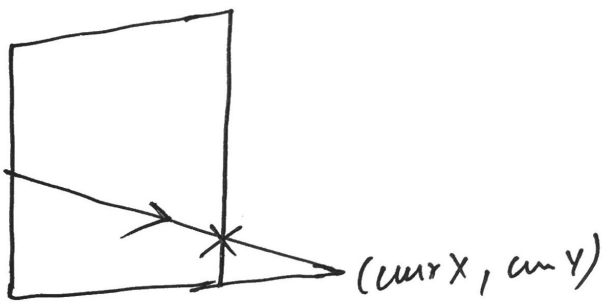
current x - previous x

: < 0 → [Point → (0,0)]

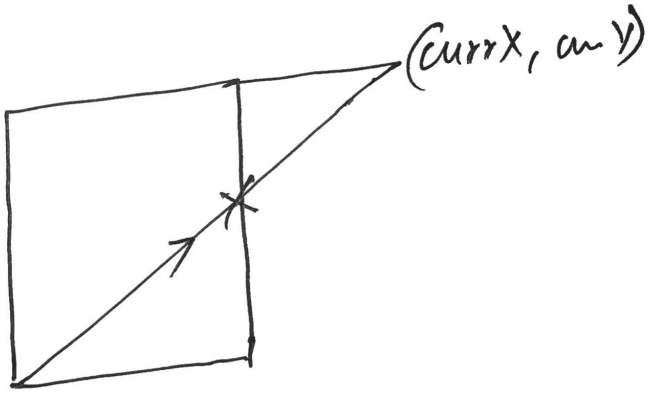
: > 0 → [Point → (m,0)]

All Exceptional Cases (when the algorith gives a point out of the table)

$X_{output} > m$:

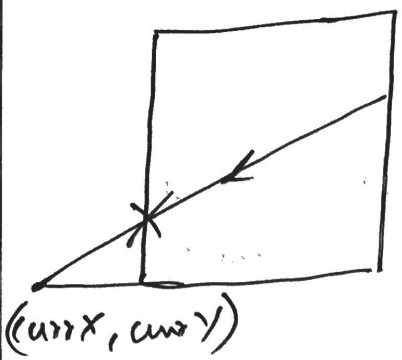


curr Y - Previa Y < 0

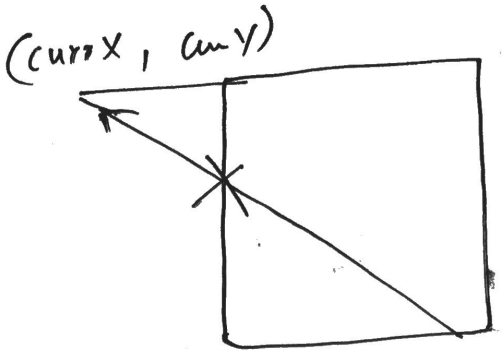


curr Y - Prev Y > 0

$X_{output} < 0$:

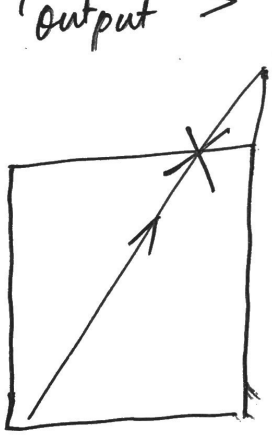


currY - PrevY < 0

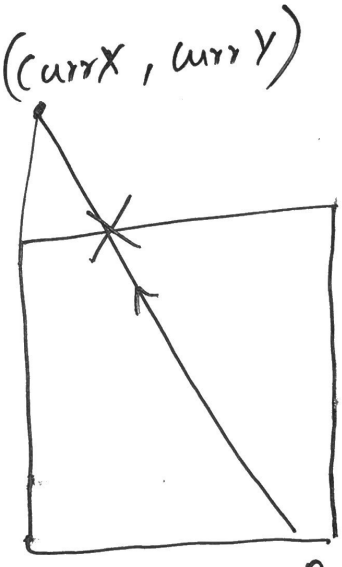


currY - PreY > 0

$Y_{output} > n$:

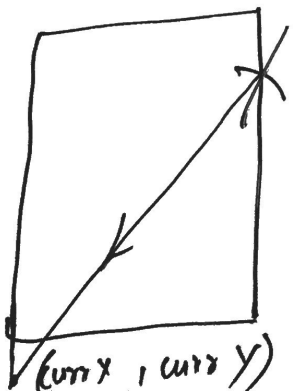


currX - PrevX > 0

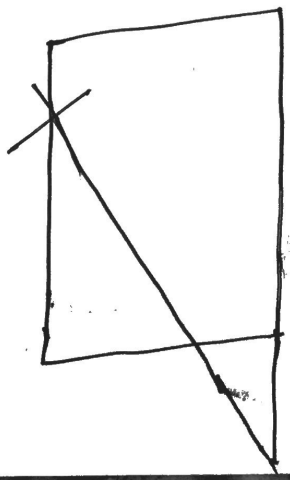


currX - PrevX < 0

$Y_{output} < n$:



currX - PrevX < 0

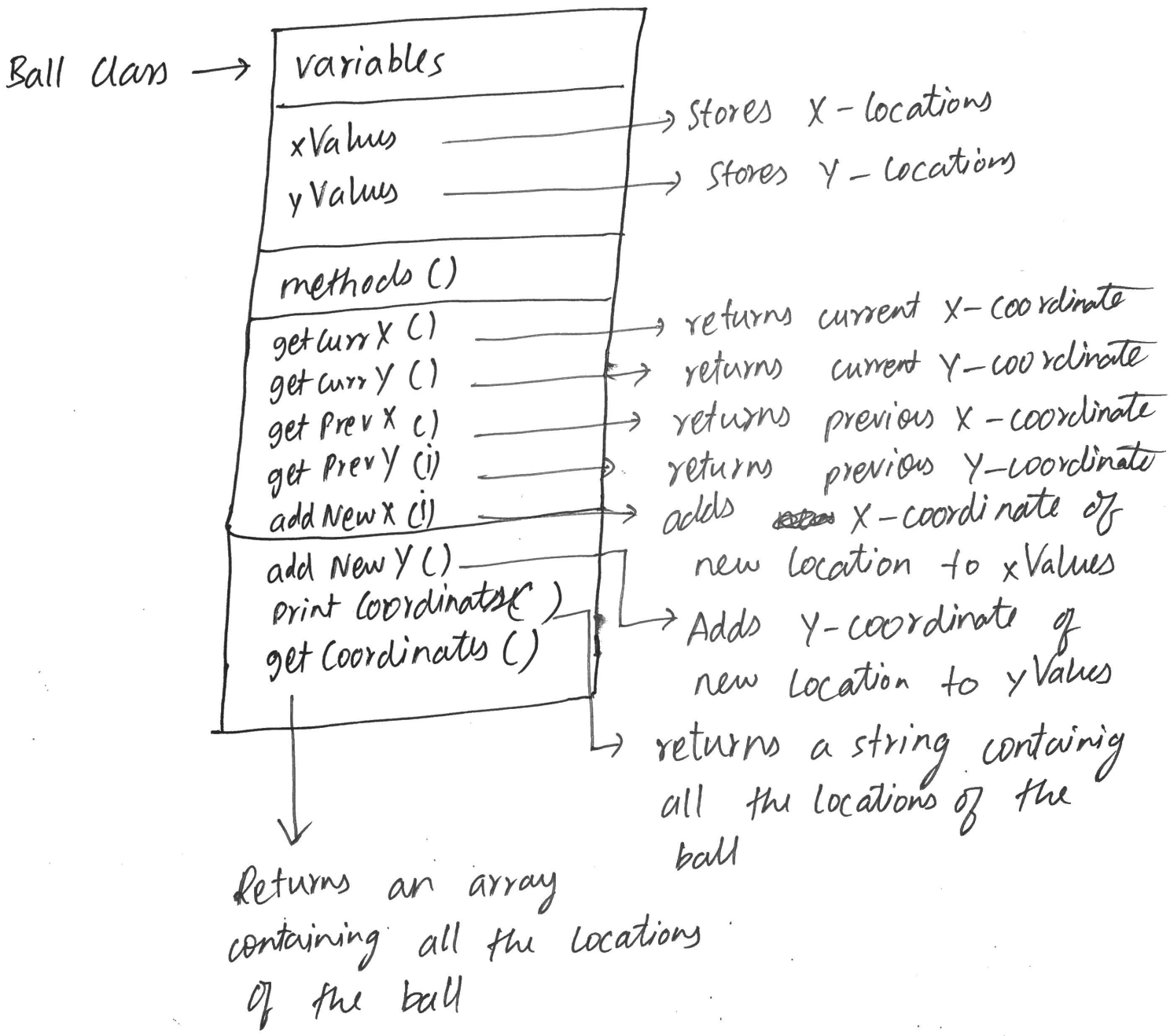


currX - PrevX >0

Algorithm Description (High level):

The entire program has been organized in 2 modules.

→ Ball Module ⟶ Contains Ball class
→ Game Module



Game Module:

→ The method that runs is the game method.

→ Initially, it calls the "start()" method which moves the ball to its first location "(m, m)".

→ Next, a while loop starts which continuously calls the getNextLocation() function to get the new location's coordinates & adds them to the ball by calling ball.addX(i) & ball.addY(i)

→ The condition for the while loop is inHole() == False. inHole() checks if the ball has reached the pocket. by isClose() function.

→ The getNextLocation() calls the getBaseVector() function to find the base vector. The getBaseVector() calls collinear() fuction to find the wall at which it is currently.

→ After the while loop is terminated, the results are printed & plot and animation is made with plotPath() function.

Tolerances → Tolerances used are 10^{-8}. The following is chosen to maintain high accuracy.