2D Collision

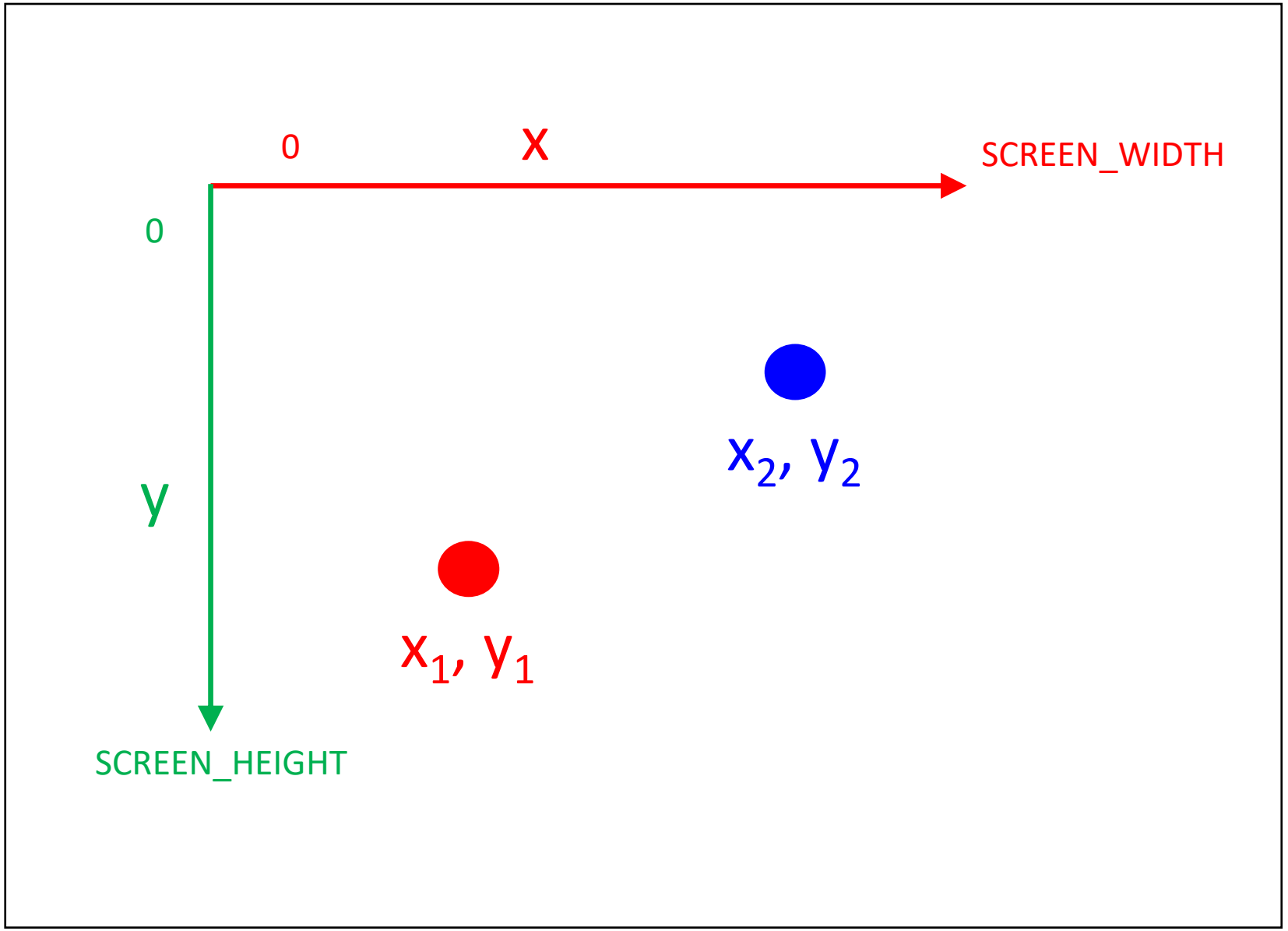
Knox Game Design

February 2022

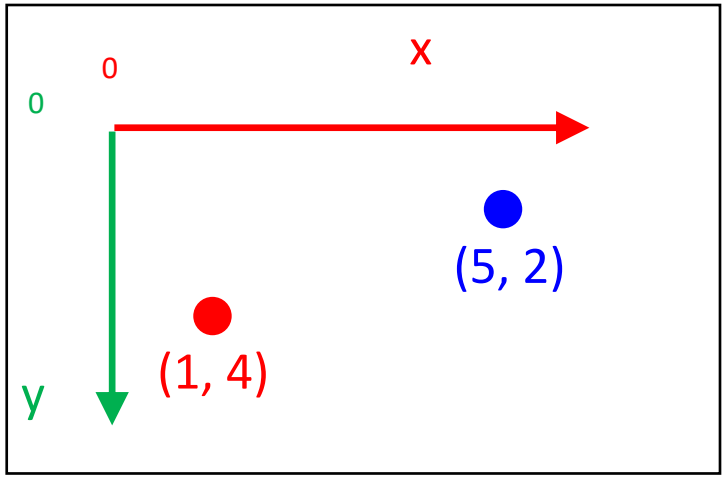
Levi D. Smith

Collision between two points

- $x_1 = x_2$, $y_1 = y_2$
- Useful for grid style games
  - rows = y
  - columns = x
- Tolerance value (especially for non-integers)
- Screen coordinates
  - x increases left to right
  - y increases top to bottom

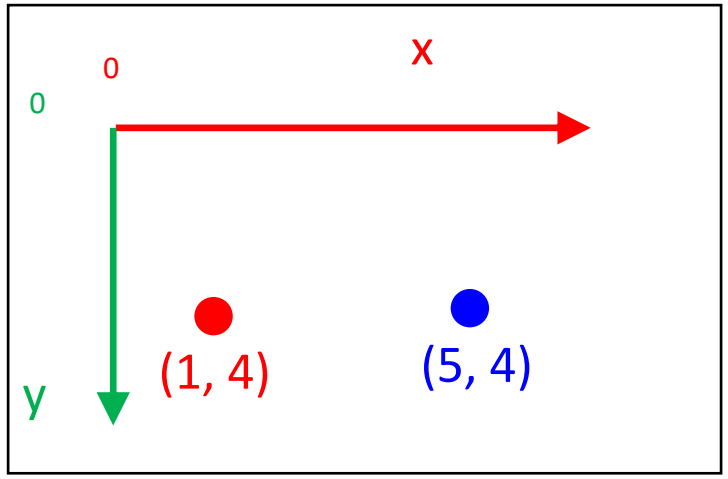



```
if (p1.x == p2.x &&
    p1.y == p2.y) {
    //collision
}
```



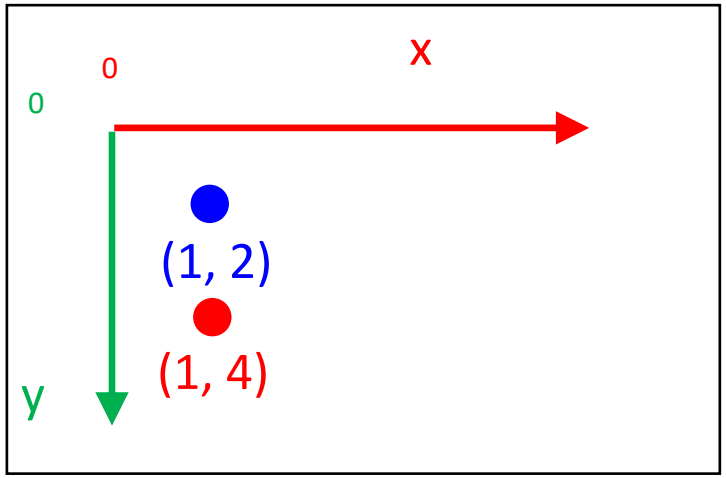
$5 \neq 1$ ✗

$2 \neq 4$ ✗



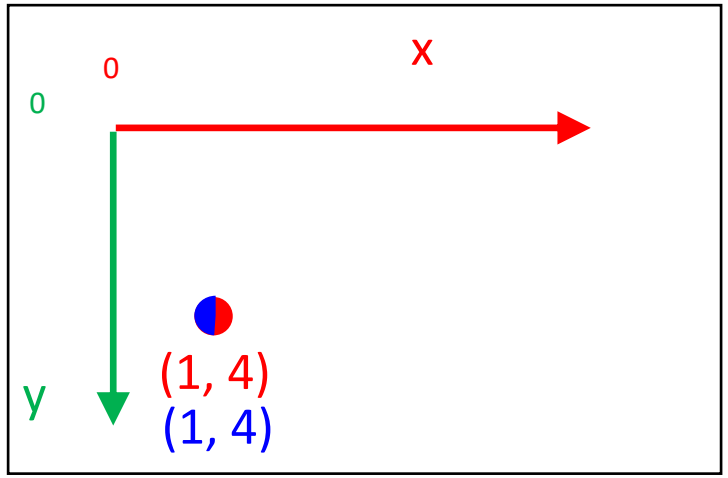
$5 \neq 1$ ✗

$4 = 4$ ✓



$1 = 1$ ✓

$2 \neq 4$ ✗

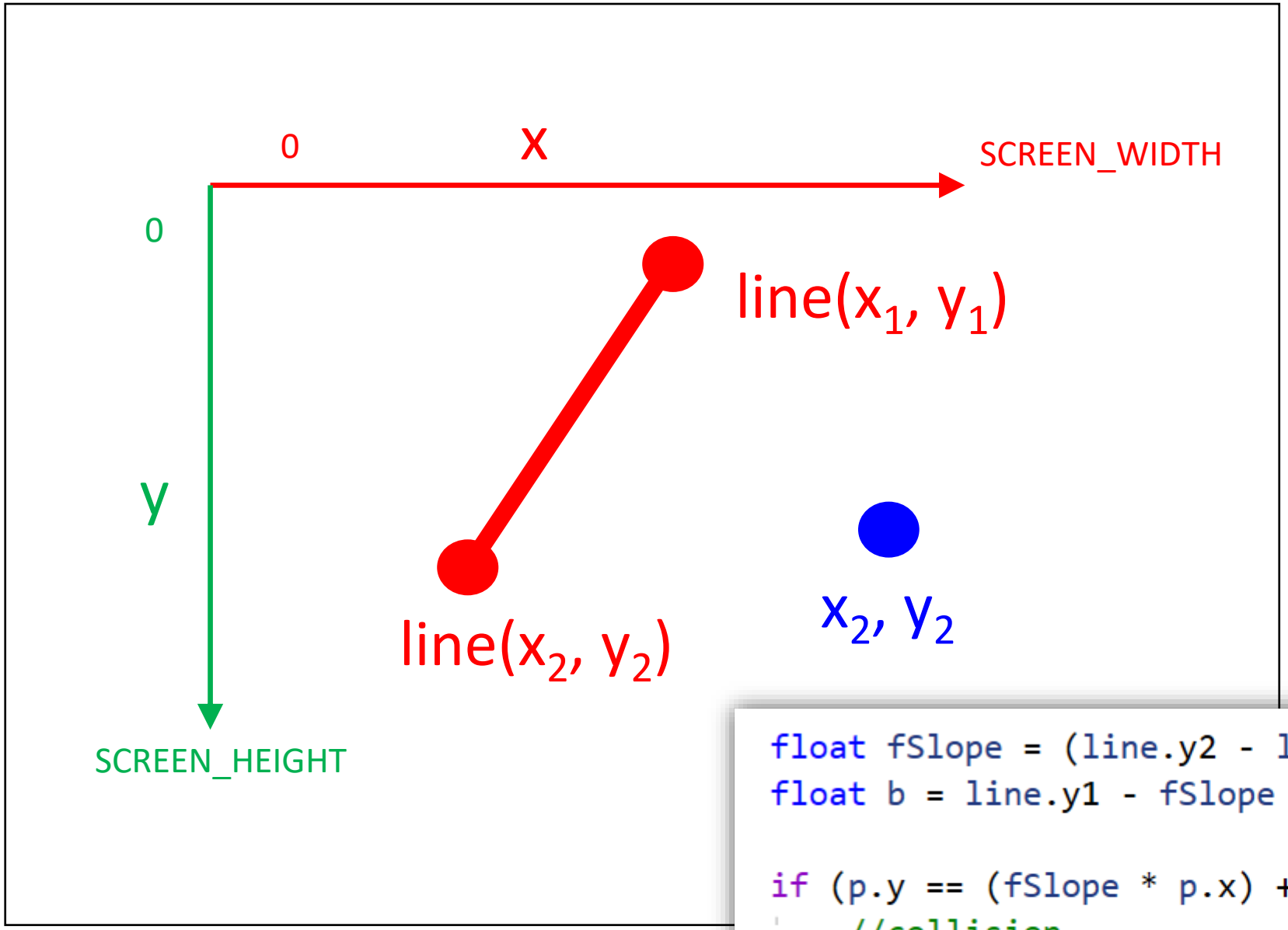


$1 = 1$ ✓

$4 = 4$ ✓

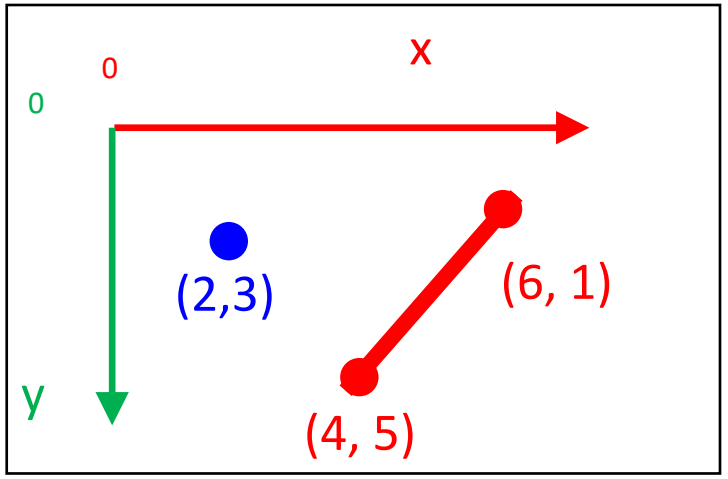
Collision between point and line

- line is defined by two points (x_1, y_1) - (x_2, y_2)
- $y = mx + b$
- m: slope, rise over run $(y_2 - y_1) / (x_2 - x_1)$
  - Watch out for integer rounding!
- b: Apply line point and slope values and solve
  - $b = y_1 - (m * x_1)$
  - Can use either line end point, should get the same result
- Plug in point to test for collision into equation to test for validity




```
float fSlope = (line.y2 - line.y1) / (line.x2 - line.x1);
float b = line.y1 - fSlope * line.x1;

if (p.y == (fSlope * p.x) + b) {
    //collision
}
```



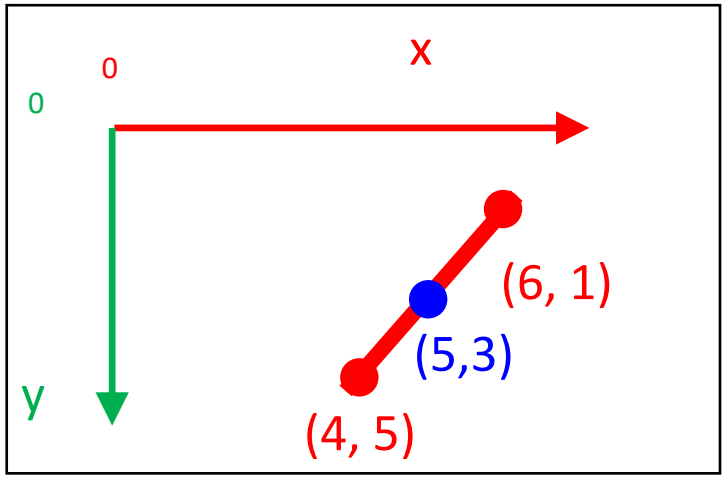
$y = mx + b$

$m = (1 - 5) / (6 - 4)$
$= -4 / 2$
$= -2$

$b = y - mx$
$= 1 - (-2 * 6)$
$= 1 - (-12)$
$= 13$

$y = -2x + 13$

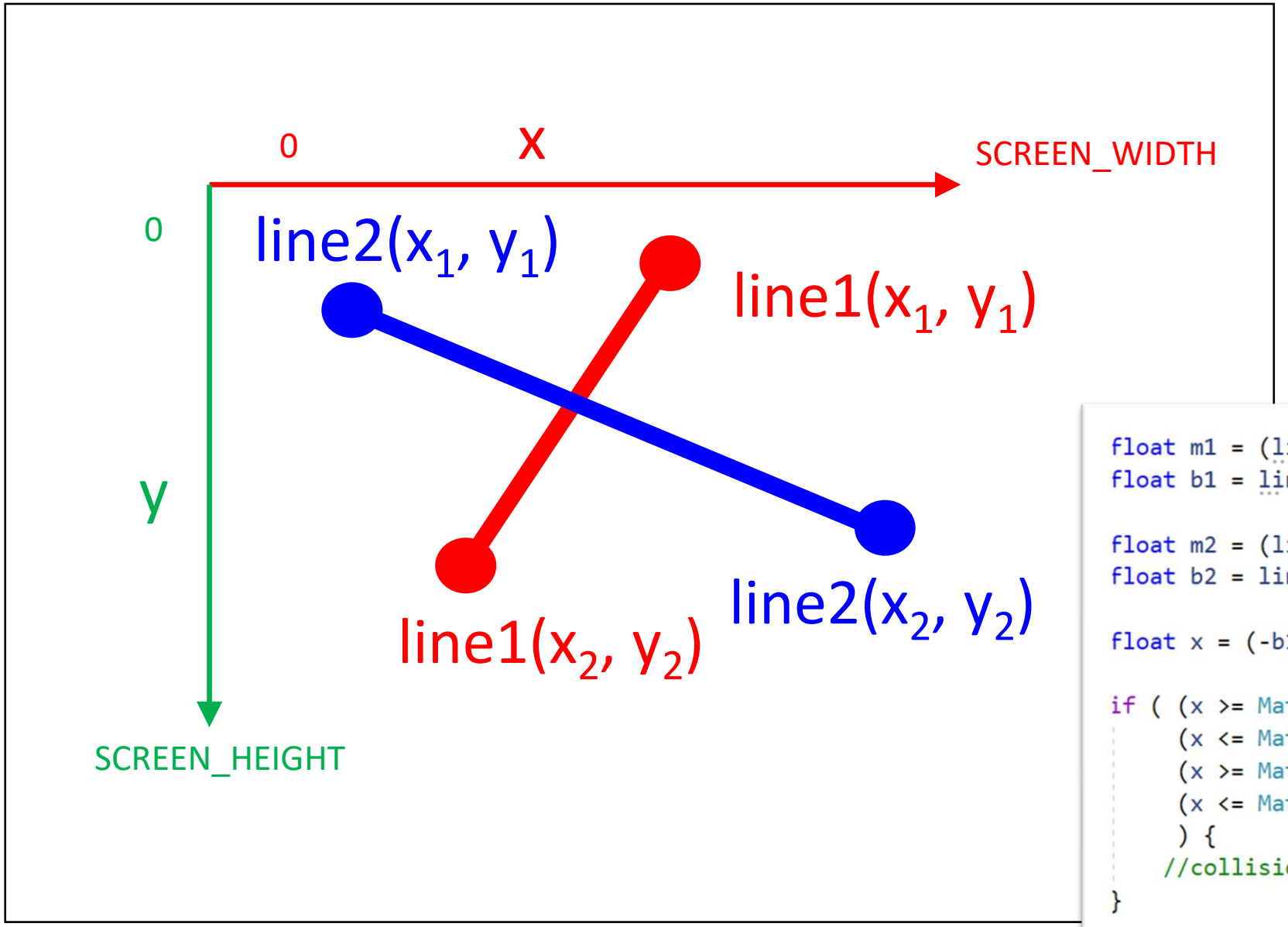
$3 = (-2 * 2) + 13$
$3 = (-4) + 13$
$3 \neq 9$ ✗



$3 = (-2 * 5) + 13$
$3 = (-10) + 13$
$3 = 3$ ✓

Collision between line and line

- Two lines: $y = m_1x+b_1$, $y = m_2x+b_2$
- Determine where the lines cross:
  - $m_1x+b_1 = m_2x+b_2$
  - Solve for x
  - $x = (-b_1 + b_2)/(m_1 - m_2)$
  - Verify that x falls between min and max x values for both segments
    - $x >= line1_{min(x)}$
    - $x <= line1_{max(x)}$
    - $x >= line2_{min(x)}$
    - $x <= line2_{max(x)}$
  - Alternatively, use determinants

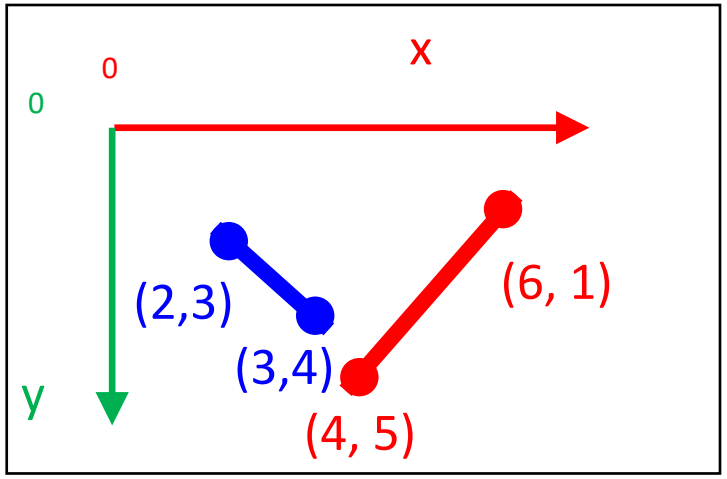



```
float m1 = (line1.y2 - line1.y1) / (line1.x2 - line1.x1);
float b1 = line1.y1 - (m1 * line1.x1);

float m2 = (line2.y2 - line2.y1) / (line2.x2 - line2.x1);
float b2 = line2.y1 - (m2 * line2.x1);

float x = (-b1 + b2) / (m1 - m2);

if ( (x >= Mathf.Min(line1.x1, line1.x2)) &&
     (x <= Mathf.Max(line1.x1, line1.x2)) &&
     (x >= Mathf.Min(line2.x1, line2.x2)) &&
     (x <= Mathf.Max(line2.x1, line2.x2))
     ) {
    //collision
}
```

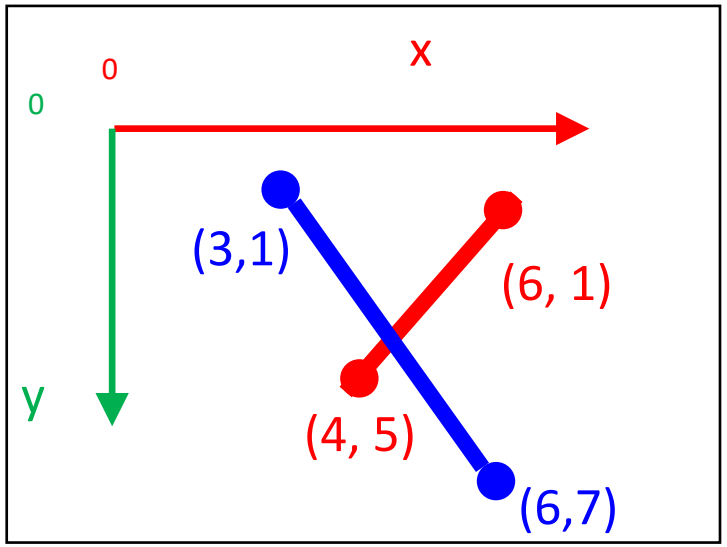


$m_1 = (1 - 5) / (6 - 4)$
$= -2$
$b_1 = 1 - (-2 * 6)$
$= 13$

$m_2 = (3 - 4)/(2-3)$
$= 1$
$b_2 = 4 - (1 * 3)$
$= 1$

$x = (-13 + 1) / (-2 - 1)$
$= -12 / -3$
$= 4$

$4 >= 4$ ✔
$4 <= 6$ ✔
$4 >= 2$ ✔
$4 > 3$ ✘



$m_1 = (1 - 5) / (6 - 4)$
$= -2$
$b_1 = 1 - (-2 * 6)$
$= 13$

$m_2 = (1 - 7)/(3 - 6)$
$= 2$
$b_2 = 1 - (2 * 3)$
$= -5$

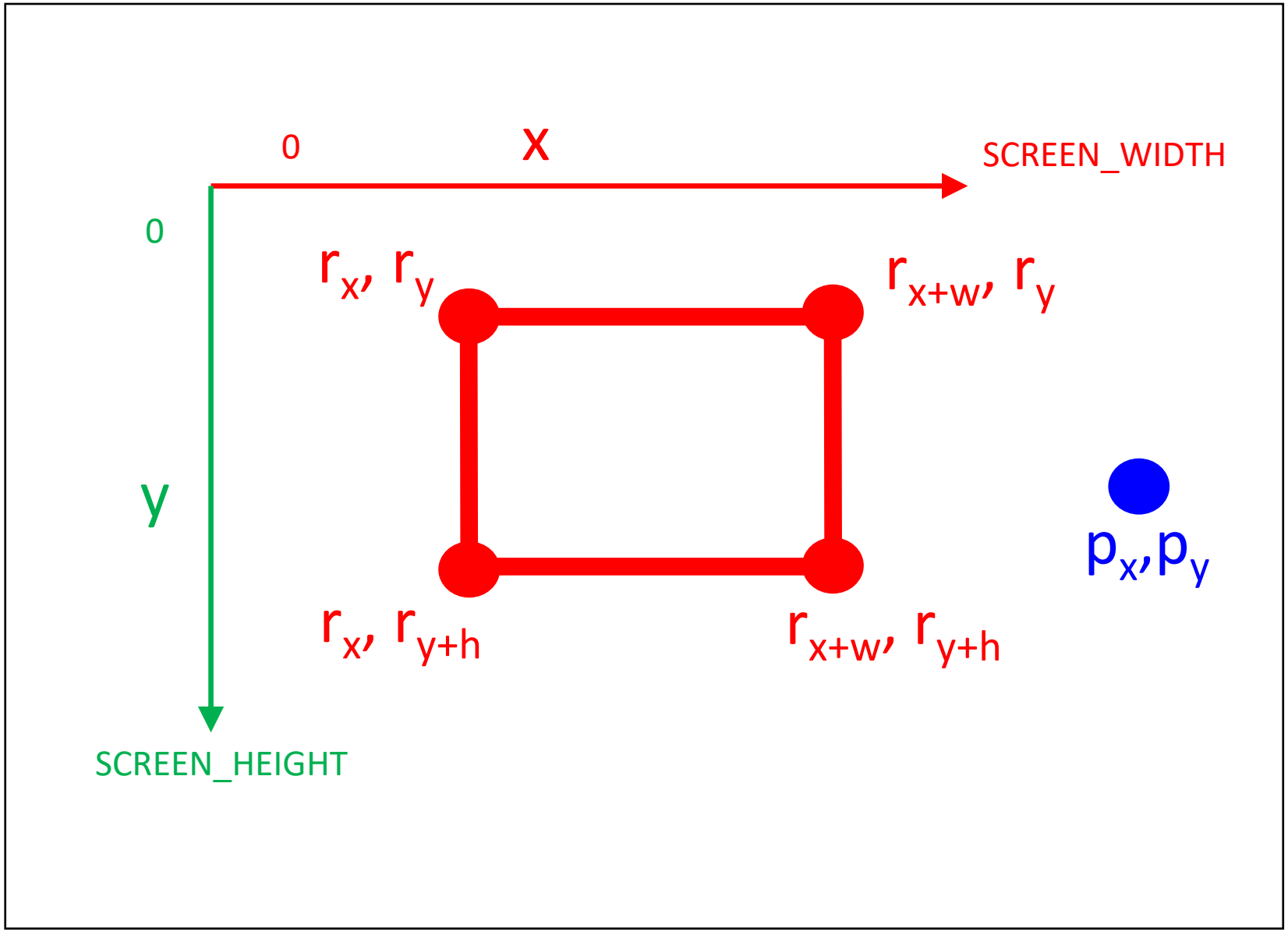
$x = (-13 + -5) / (-2 - 2)$
$= -18 / -4$
$= 4.5$

$4.5 >= 4$ ✔
$4.5 <= 6$ ✔
$4.5 >= 3$ ✔
$4.5 <= 6$ ✔

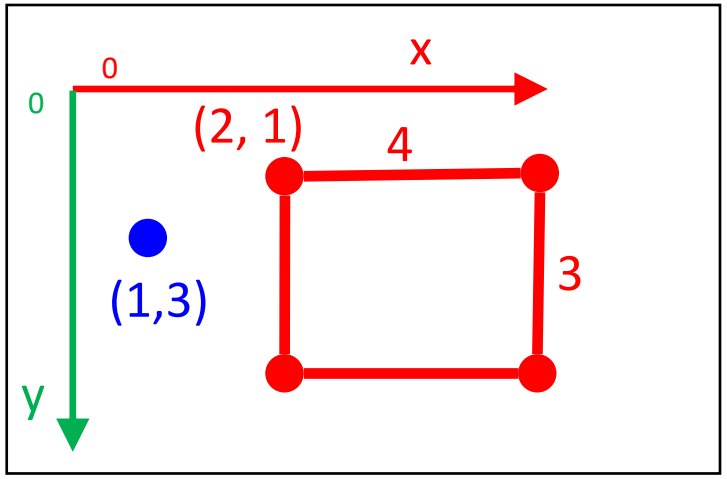
Collision between point and rectangle

- Point: p_x, p_y
- Rectangle: r_x, r_y, r_w, r_h
  - $p_x >= r_x$
  - $p_x <= r_x + r_w$
  - $p_y >= r_y$
  - $p_y <= r_y + r_h$
- All rules must be valid for collision

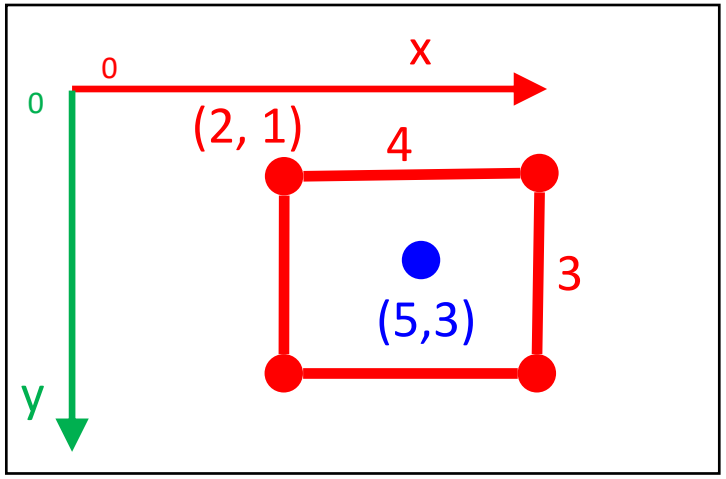
KNOX GAME DESIGN



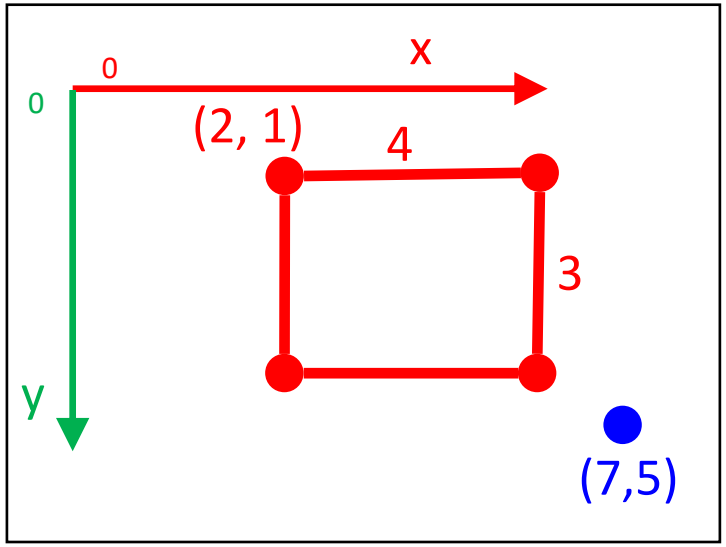

```
if (p1.x >= r1.x &&
    p1.x <= r1.x + r1.w &&
    p1.y >= r1.y &&
    p1.y <= r1.y + r1.h) {
    //collision
}
```



$1 < 2$ ✗
$1 < 2 + 4$ ✓
$3 > 1$ ✓
$3 < 1 + 3$ ✓



$5 > 2$ ✓
$5 < 2 + 4$ ✓
$3 > 1$ ✓
$3 < 1 + 3$ ✓



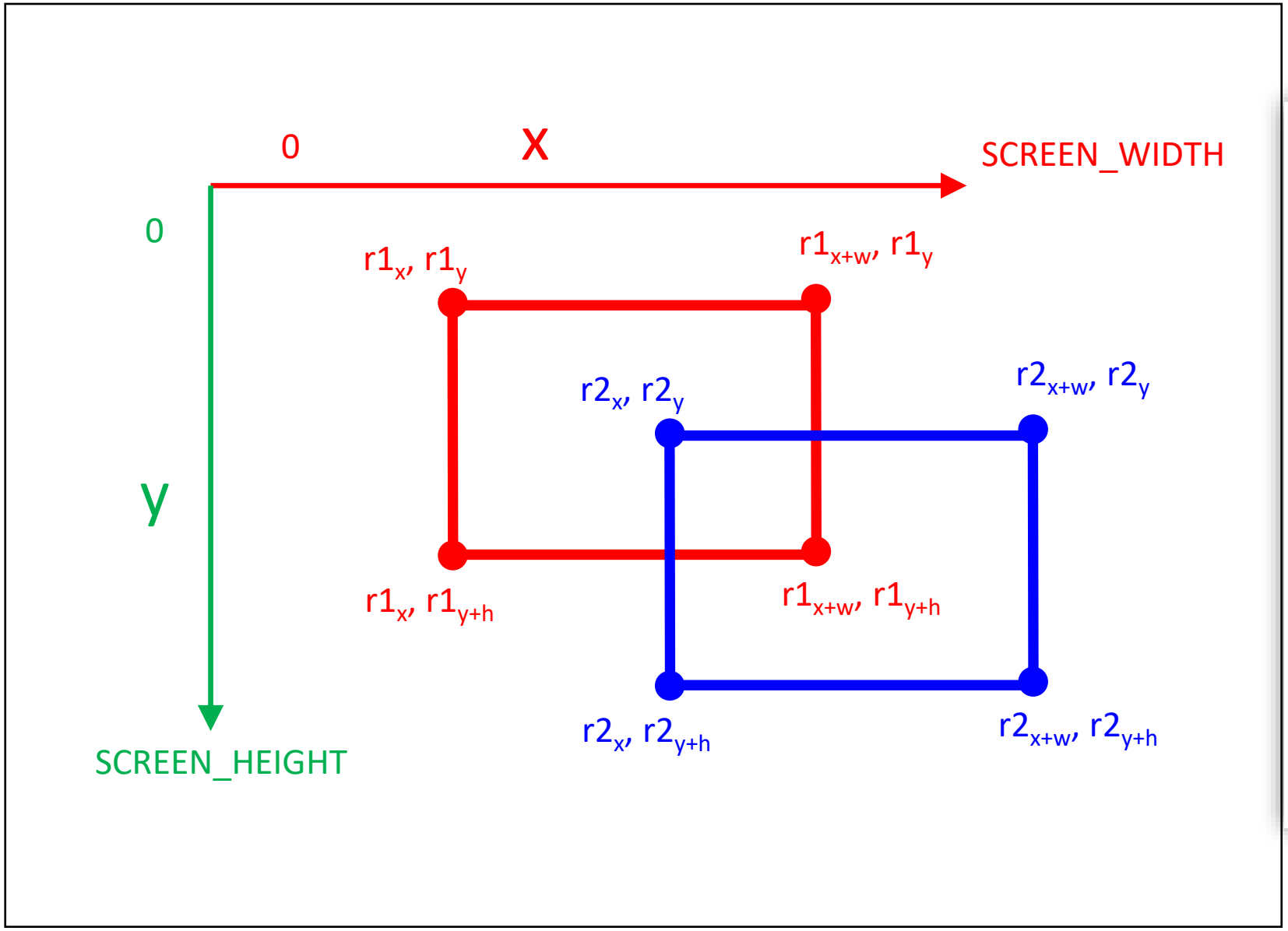
$7 > 2$ ✓
$7 > 2 + 4$ ✗
$5 > 1$ ✓
$5 > 1 + 3$ ✗

Note - ✓ and ✗ signify whether it broke the rule; does not signify if it is a valid statement

Collision between two rectangles

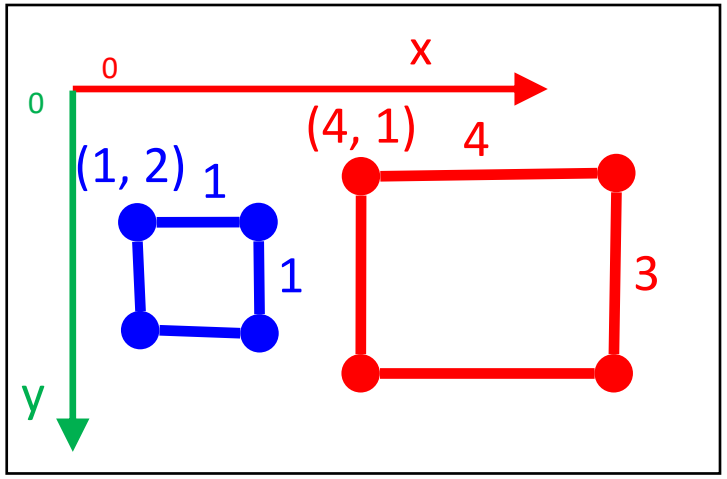
- Easier to test when rectangles don't collide, then NOT the result
  - r2.x + r2.w < r1.x (to the left)
  - r2.x > r1.x + r1.w (to the right)
  - r2.y + r2.h < r1.y (above)
  - r2.y > r1.y + r1.h (below)
- DeMorgan's Theorem
  - (NOT a) AND (NOT b) = NOT (a OR b)
- Note - Square is a special instance of rectangle where $r_w = r_h$

KNOX GAME DESIGN



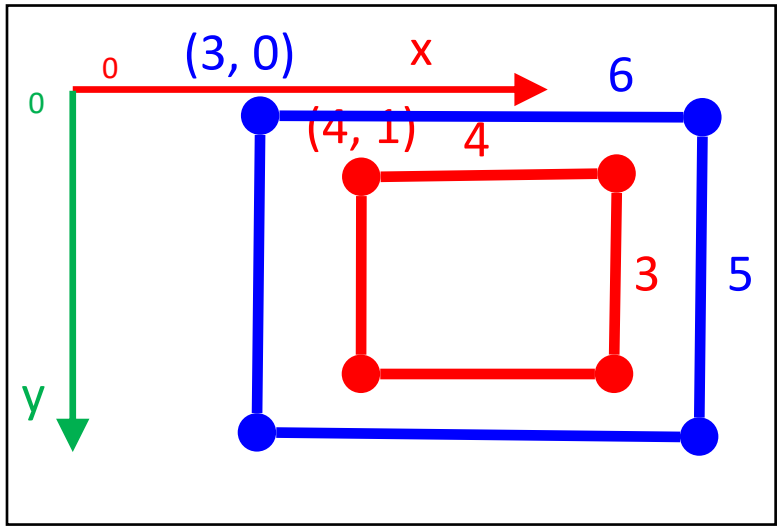

```
if (!(r2.x + r2.w < r1.x) &&
    !(r2.x > r1.x + r1.w) &&
    !(r2.y + r2.h < r1.y) &&
    !(r2.y > r1.y + r1.h)) {
    //collision
}


//Using DeMorgan's Theorem
if (!(r2.x + r2.w < r1.x ||
    r2.x > r1.x + r1.w ||
    r2.y + r2.h < r1.y ||
    r2.y > r1.y + r1.h)) {
    //collision
}
```



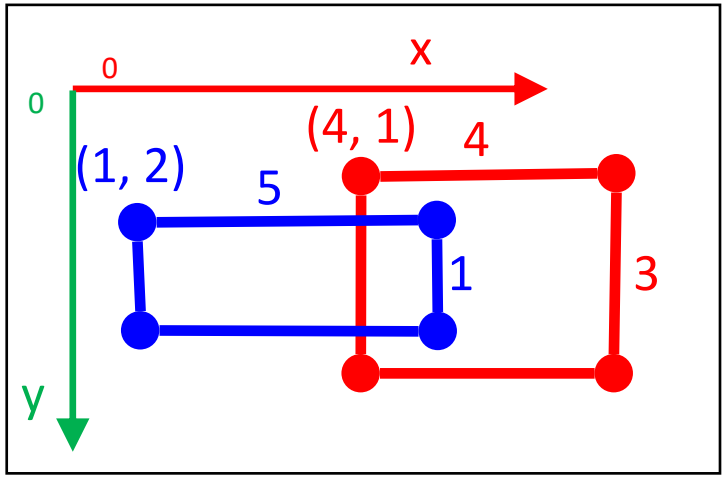
left: $1 + 1 < 4$ ✓
right: $1 < 4 + 4$ ✗
above: $2 + 1 > 1$ ✗
below: $2 < 1 + 3$ ✗

not all ✗



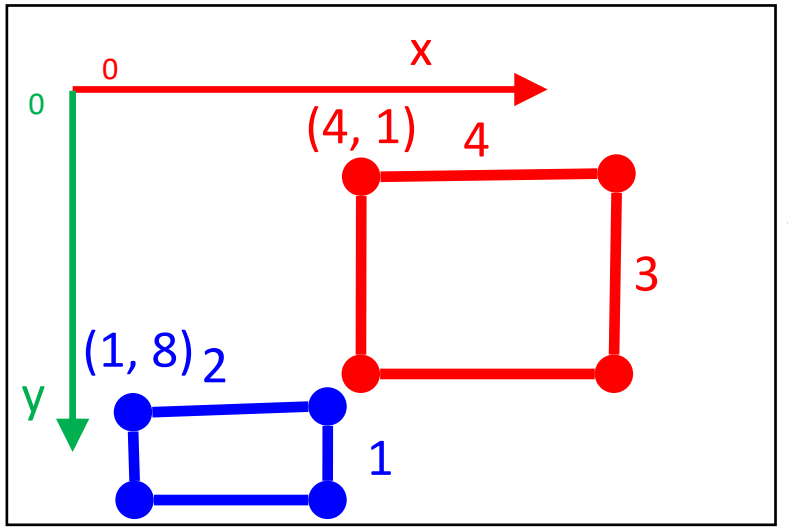
left: $3 + 6 > 4$ ✗
right: $3 < 4 + 4$ ✗
above: $0 + 5 > 1$ ✗
below: $0 < 1 + 3$ ✗

not all ✓



left: $1 + 5 > 4$ ✗
right: $1 < 4 + 4$ ✗
above: $2 + 1 > 1$ ✗
below: $2 < 1 + 3$ ✗

not all ✓



left: $1 + 2 < 4$ ✓
right: $1 < 4 + 4$ ✗
above: $8 + 1 > 1$ ✗
below: $8 > 1 + 3$ ✓

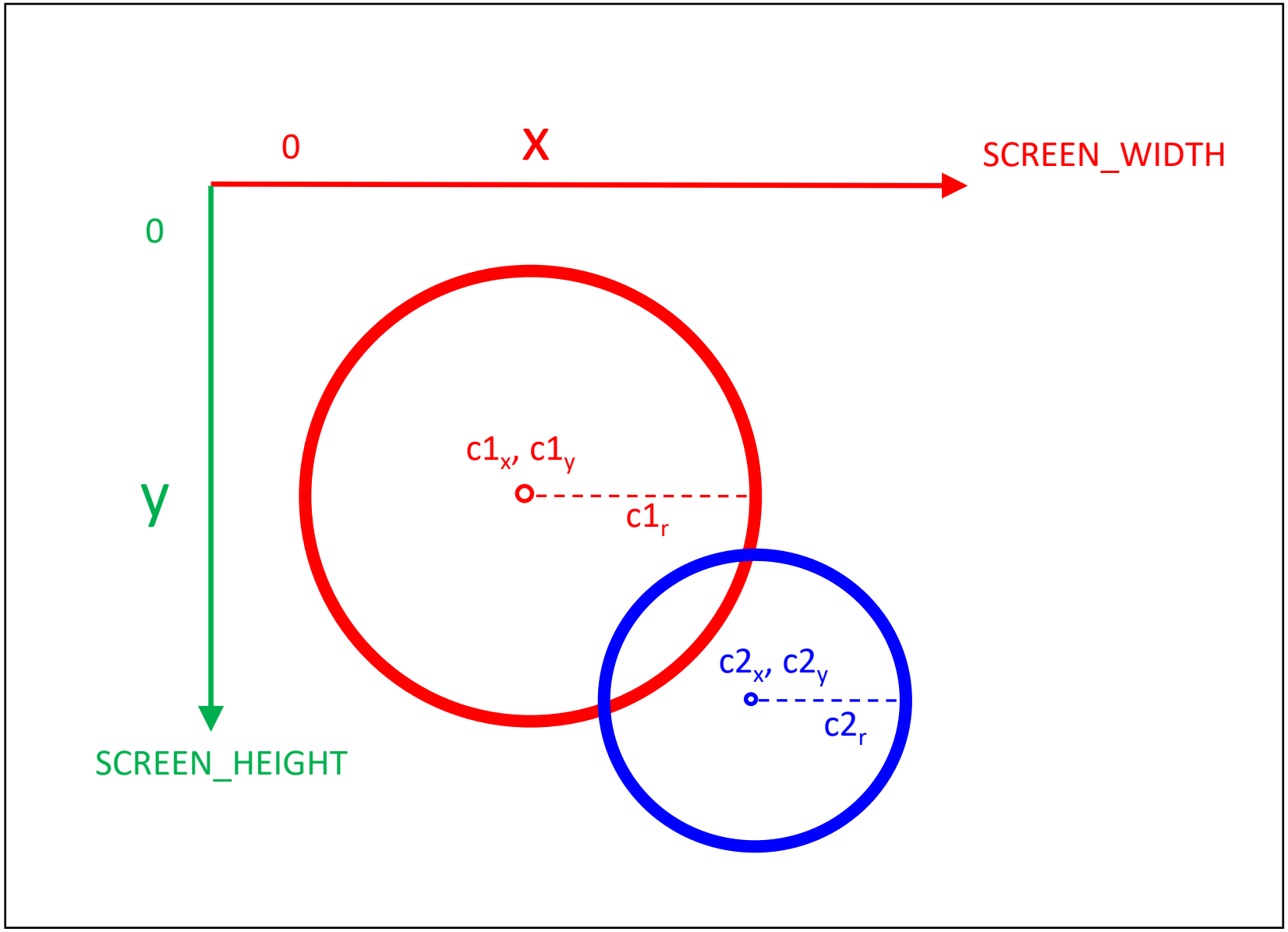
not all ✗

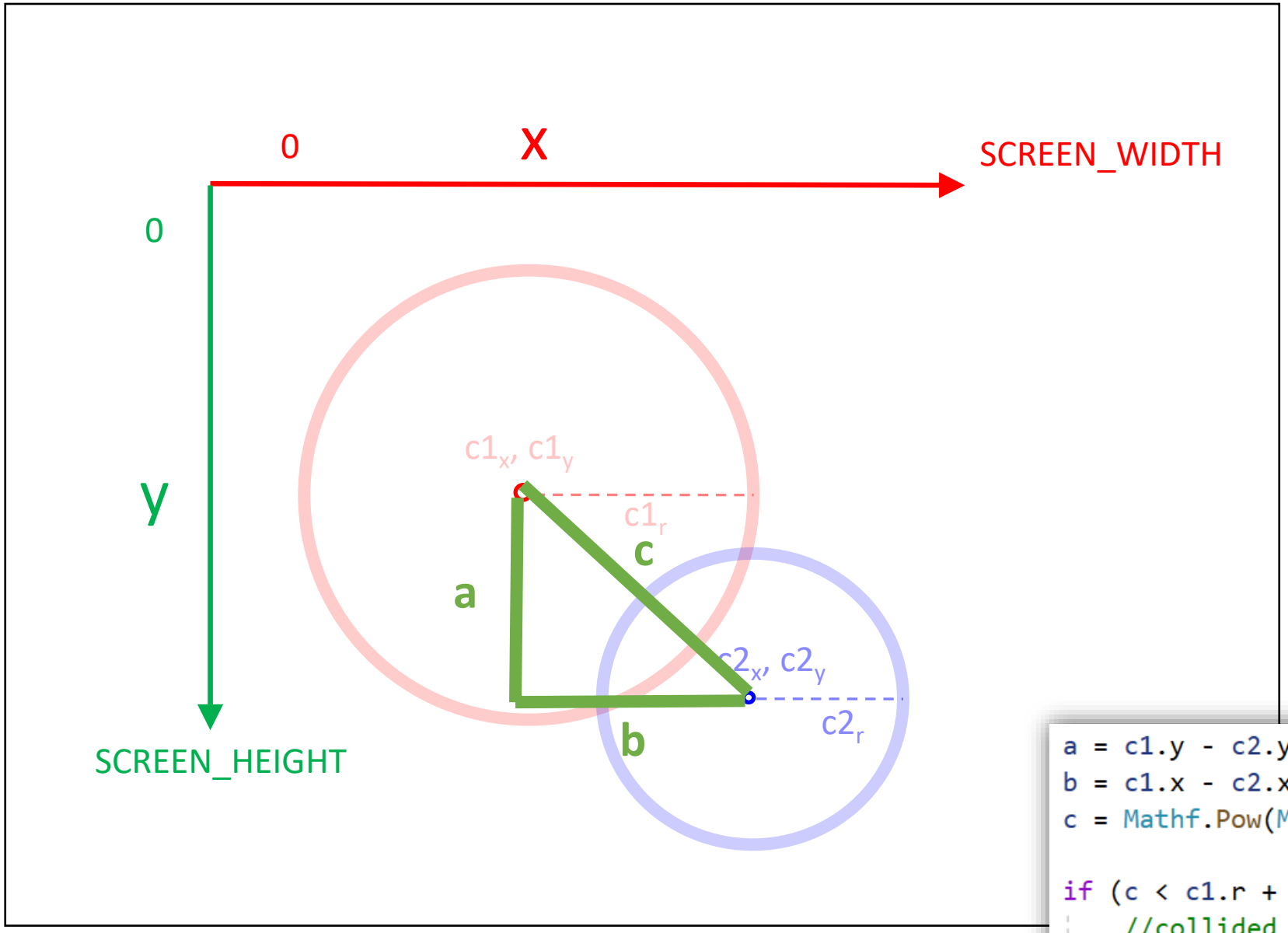
Any ✓ signifies that it satisfied the rule, meaning that it does not collide. Must break all rules (all ✗) to confirm collision.

Collision between two circles

- Pythagorean Theorem
  - $a^2 + b^2 = c^2$
  - Using a triangle to detect collision between circles doesn't seem intuitive
- Find distance between the centers of the two circles
- If the distance is less than the sum of the two radius values, then the two circles collide
- Note - Collision between point an circle, use circle for point with radius equal to zero
  - Theoretically, could test collision between two points using Pythagorean Theorem and check for distance of zero
- Does not work for ovals, sorry!

KNOX GAME DESIGN


0
x
SCREEN_WIDTH
0
y
SCREEN_HEIGHT
c1x, c1y
c1r
c2x, c2y
c2r

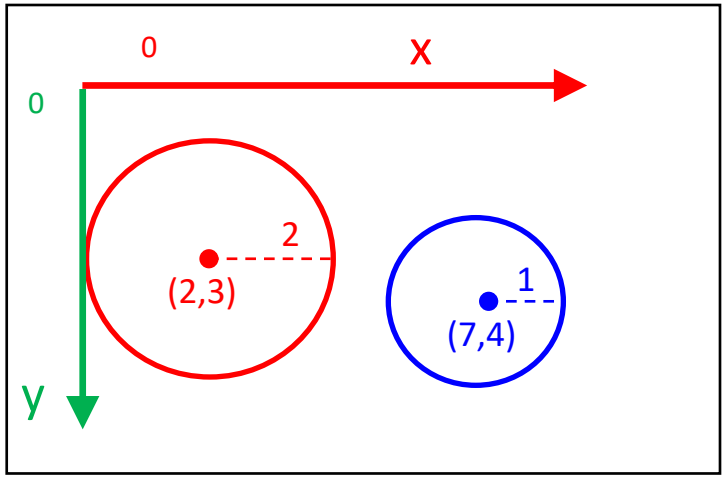


$a = c1_y - c2_y$
$b = c1_x - c2_x$
$c = (a^2 + b^2)^{1/2}$

if c < (c1r + c2r)
then collided

```
a = c1.y - c2.y;
b = c1.x - c2.x;
c = Mathf.Pow(Mathf.Pow(a, 2) + Mathf.Pow(b, 2), 0.5f);

if (c < c1.r + c2.r) {
    //collided
}
```



$a = 3 - 4$

$= -1$

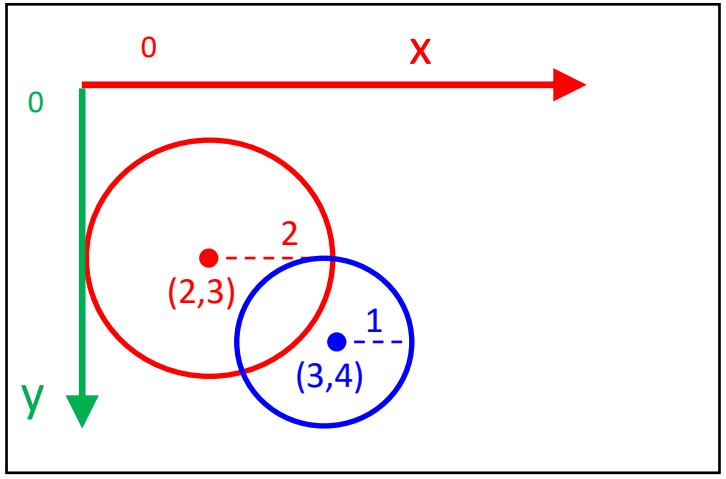
$b = 2 - 7$

$= -5$

$c = (-1^2 + -5^2)^{1/2}$

$= 5.01$

$5.01 > 2 + 1$ ✗



$a = 3 - 4$

$= -1$

$b = 2 - 3$

$= -1$

$c = (-1^2 + -1^2)^{1/2}$

$= 1.41$

$1.41 < 2 + 1$ ✓

Built in collision functions

- GameMaker
  - place_meeting

https://manual.yoyogames.com/#t=GameMaker_Language%2FGML_Reference%2FMovement_And_Collisions%2FCollisions%2FCollisions.htm

- Unity
  - OnCollisionEnter2D

https://docs.unity3d.com/ScriptReference/Collision2D.html

- Godot
  - CollisionShapes, collide

https://docs.godotengine.org/en/latest/tutorials/physics/collision_shapes_2d.html