

We're almost ready to  
make a game!

# What we've learned so far...

Initializing SDL, OpenGL, creating a window.

Drawing a triangle.

Transformations: translation, rotation, scale.

Handling timing.

Loading images and rendering textures.

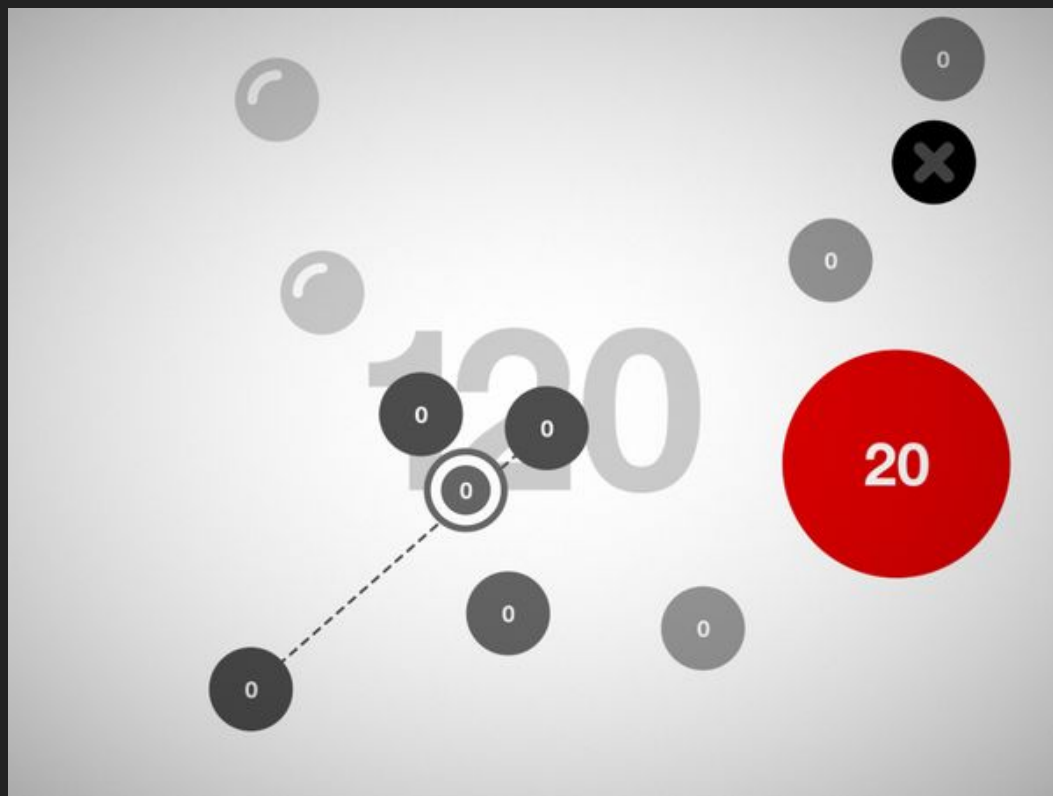
Keyboard, controller and mouse input.

Movement based on unit vector, speed and elapsed time.

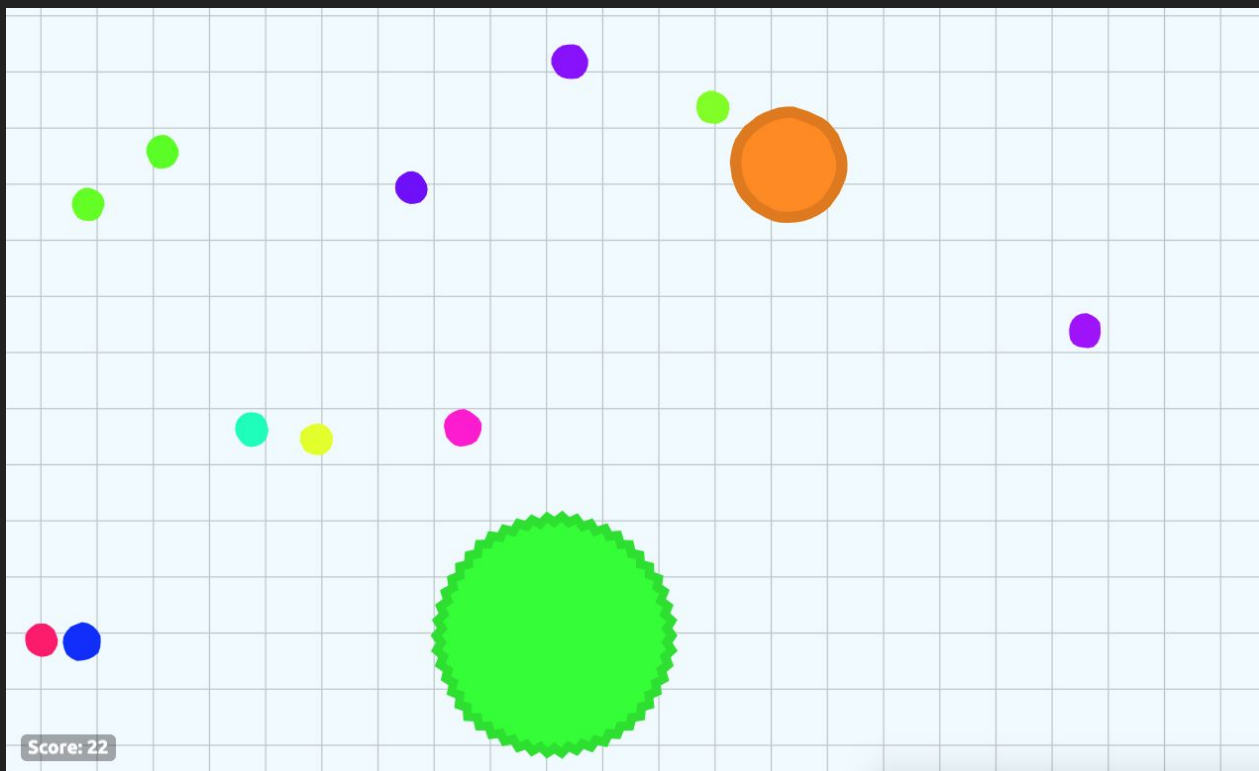
## We need one more thing to make our first game...

# Collision Detection!

# Circle - Circle Collision Detection



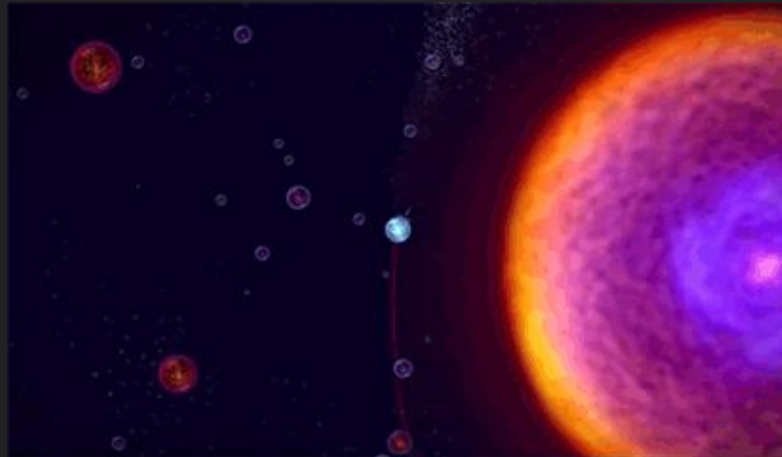
Hundreds (iOS/Android)



<https://agar.io>



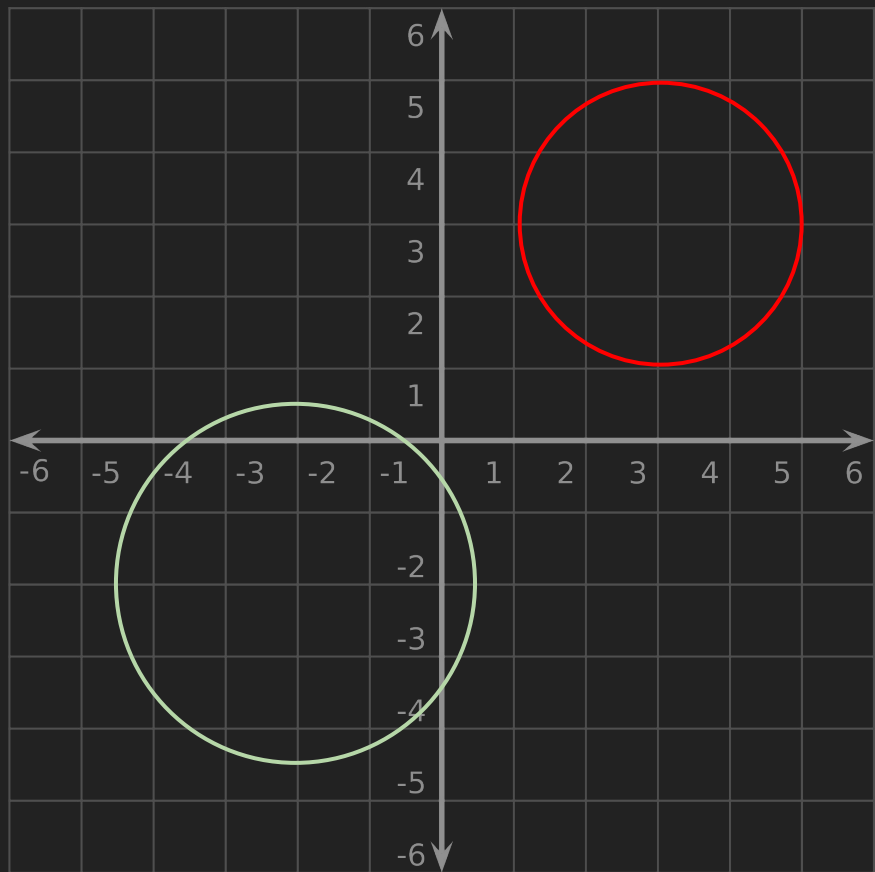
Asteroids - 1979 Arcade

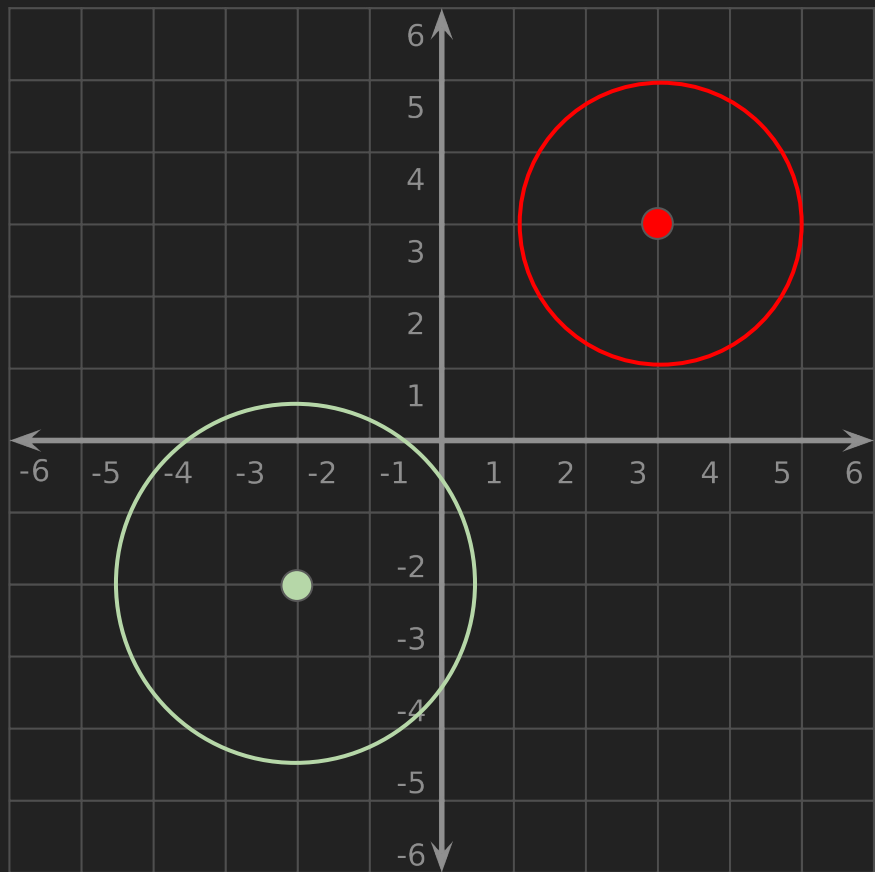


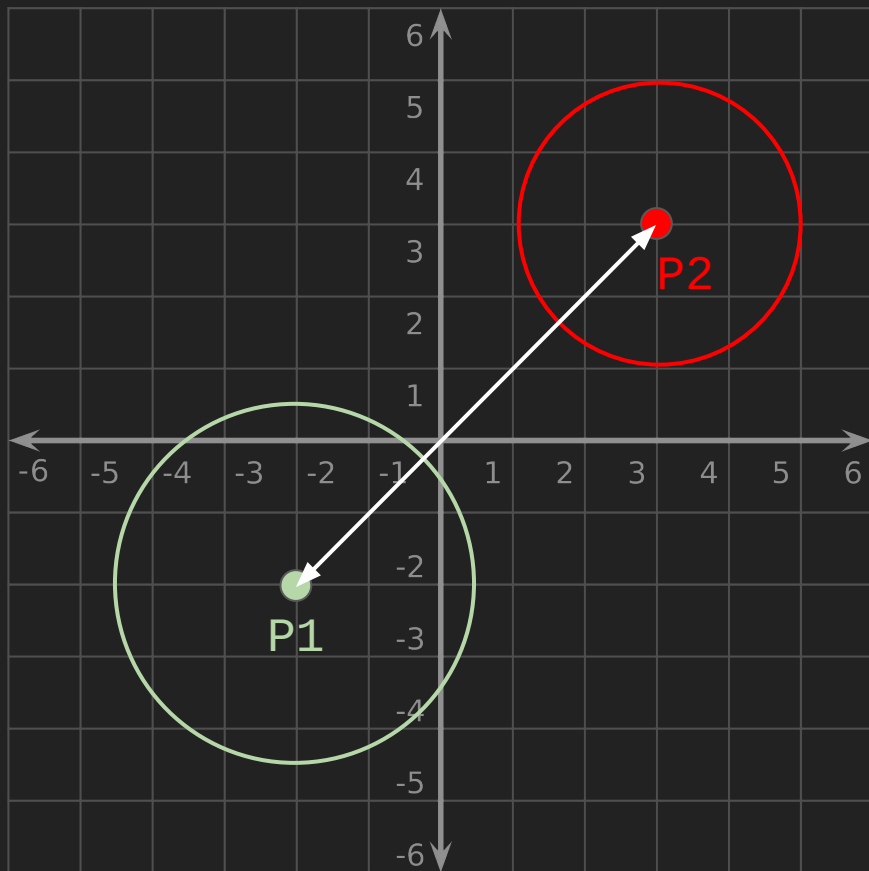
Osmos  
by Andy Nealen  
NYU Professor!











Radius: 3

P1: -2, -2

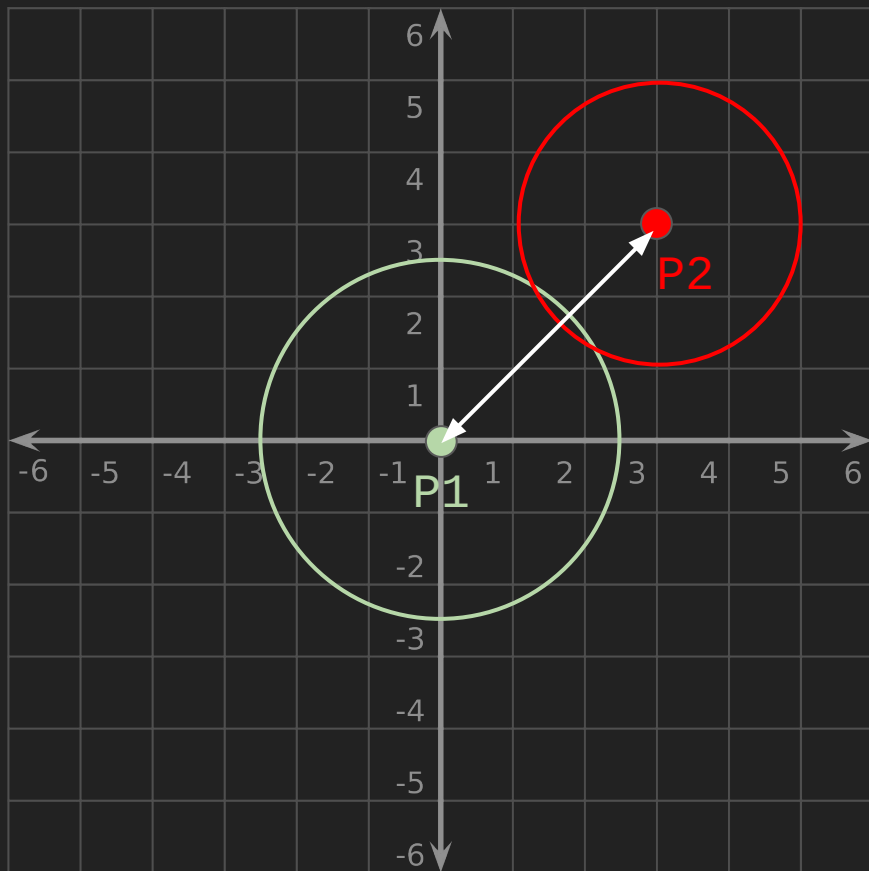
Radius: 2

P2: 3, 3

Radius + Radius = 5

Distance: 7.07  
(Pythagorean Theorem)

distance > (radius + radius)



Radius: 3

$P_1: 0, 0$

Radius: 2

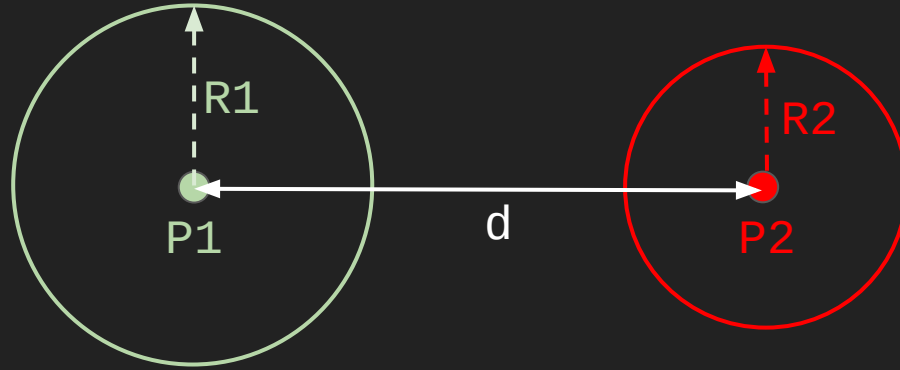
$P_2: 3, 3$

Radius + Radius = 5

Distance: 4.24  
(Pythagorean Theorem)

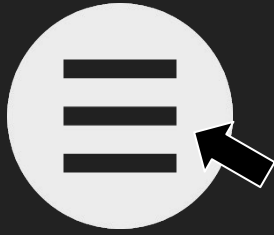
distance < (radius + radius)

# Circle - Circle Collision Detection



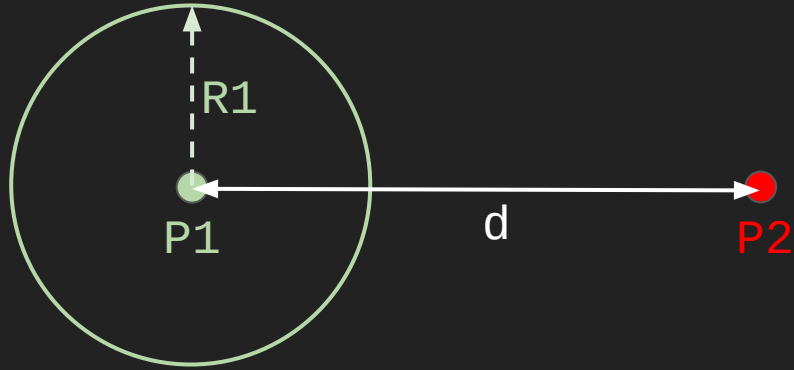
If the distance between the center of the circles is less than the sum of the radii, they are colliding.

# Point - Circle Collision Detection



Click on UI, move the player, select a target.

# Point - Circle Collision Detection



If the distance between the point and the circle center is less than the radius, they are colliding.



# Box - Box Collision Detection



The Legend of Zelda (NES)

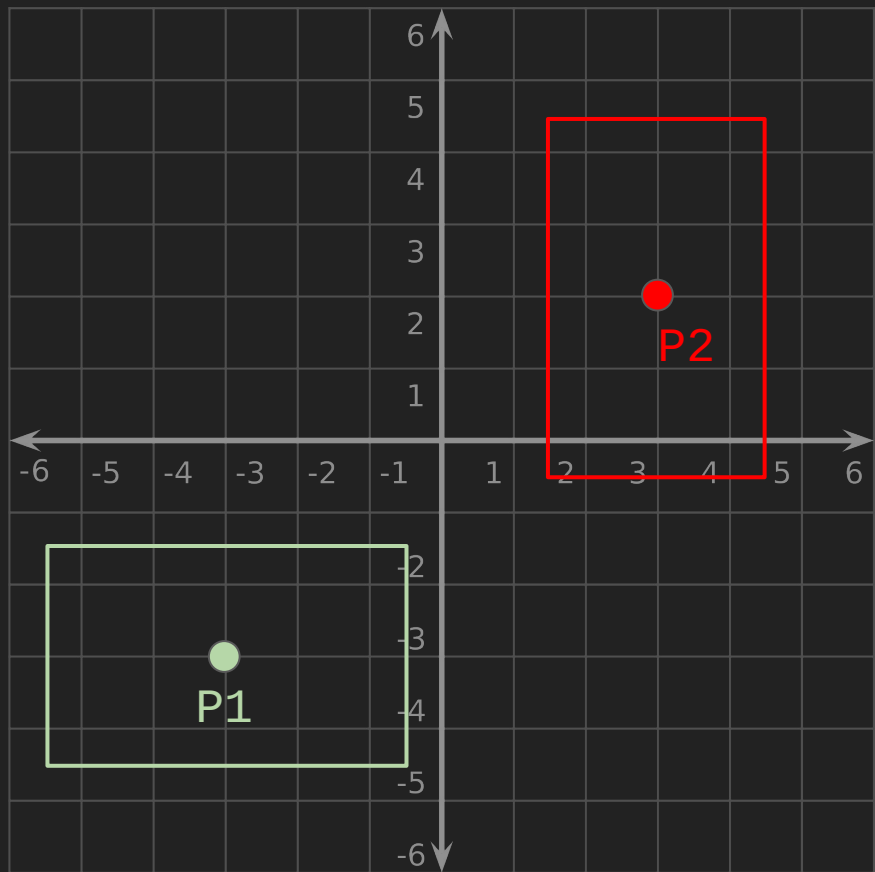


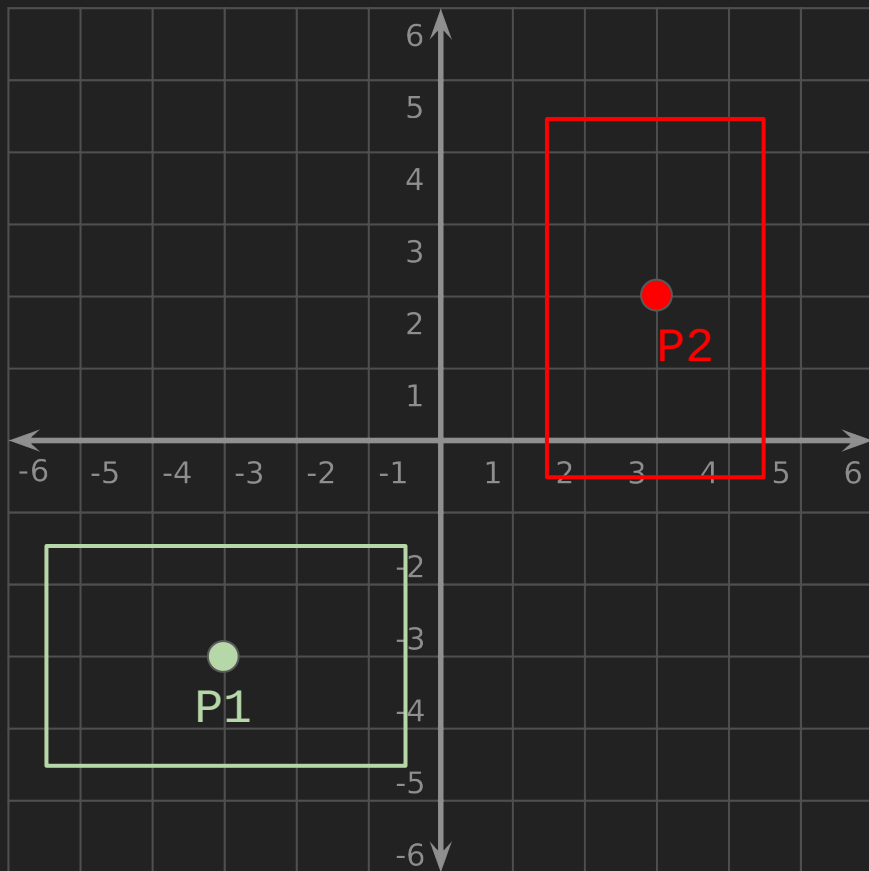
A Link to the Past (SNES)



Street Fighter (Arcade)





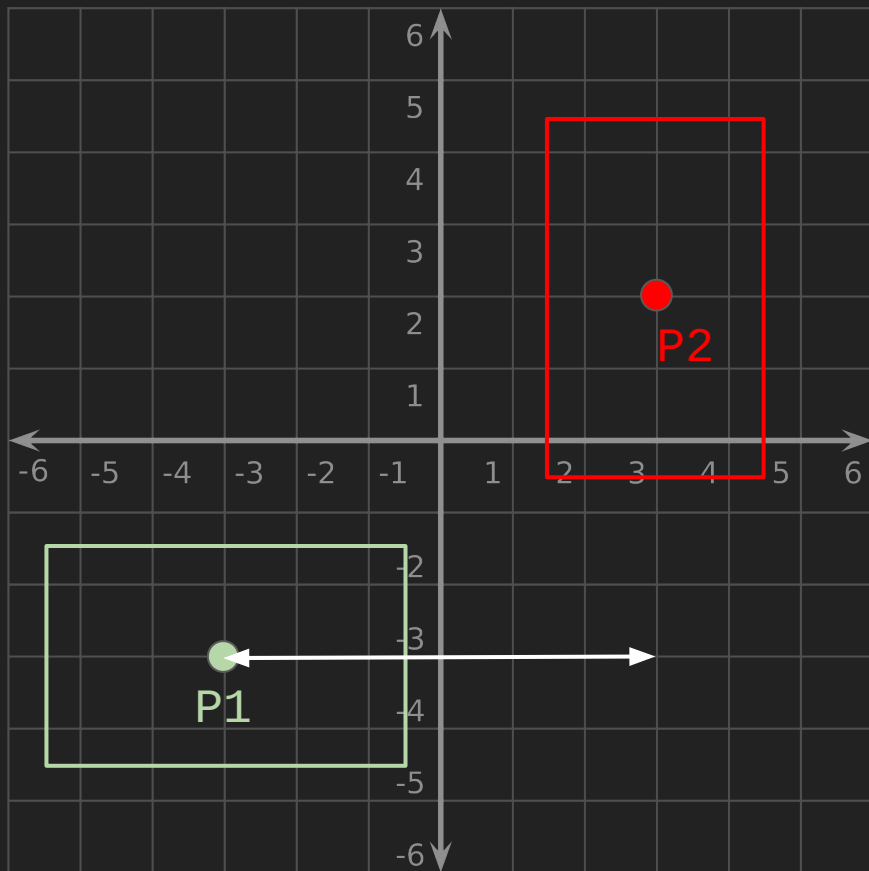


Width: 5      Height: 3

P1: -3, -3

Width: 3      Height: 5

P2: 3, 2



Width: 5

Height: 3

P1: -3, -3

Width: 3

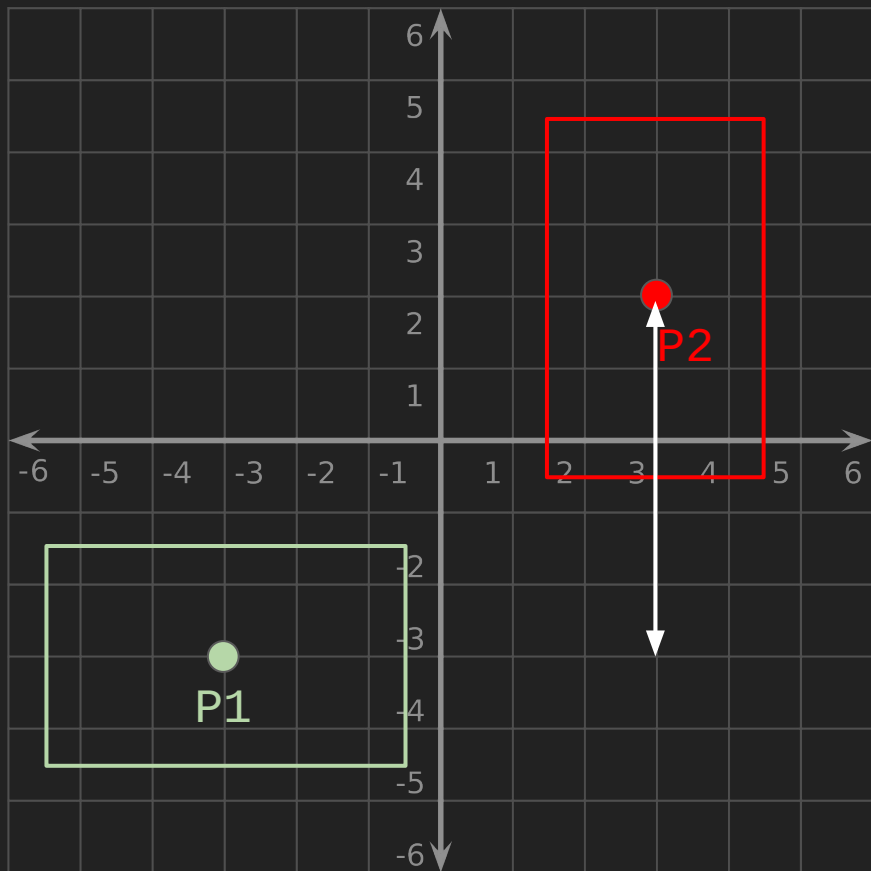
Height: 5

P2: 3, 2

X Diff: 6

$\text{fabs}(x2 - x1)$





Width: 5

P1: -3, -3

Height: 3

Width: 3

P2: 3, 2

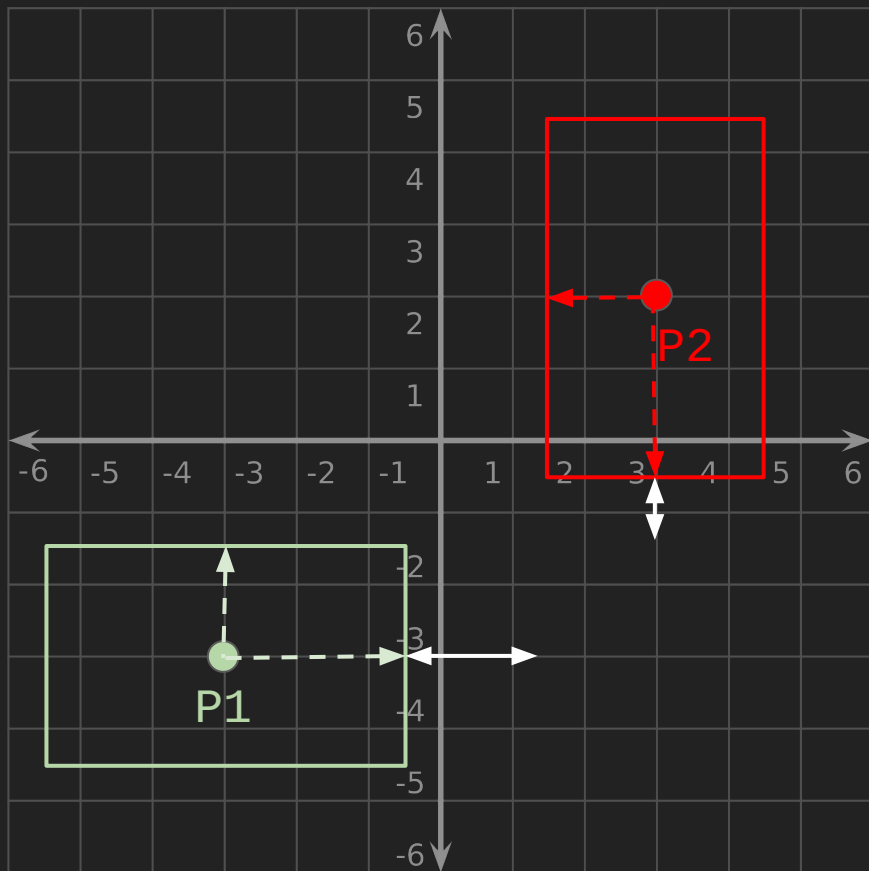
Height: 5

X Diff: 6

Y Diff: 5

$\text{fabs}(x2 - x1)$

$\text{fabs}(y2 - y1)$



Width: 5

Height: 3

P1: -3, -3

Width: 3

Height: 5

P2: 3, 2

X Diff: 6

$\text{fabs}(x2 - x1)$

Y Diff: 5

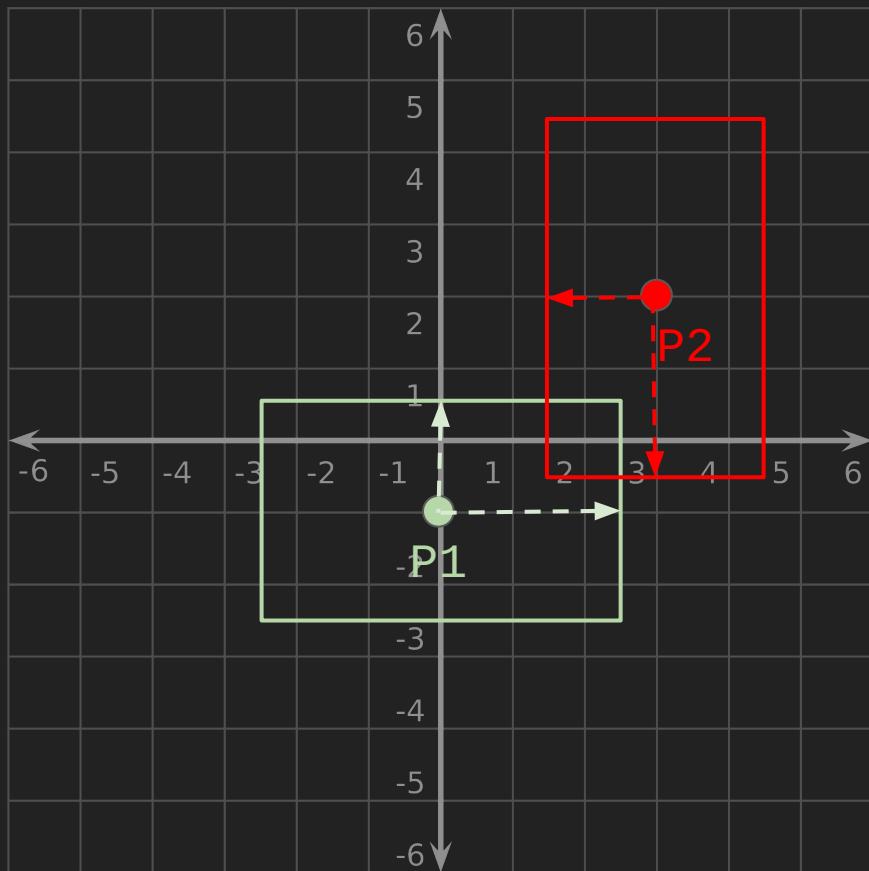
$\text{fabs}(y2 - y1)$

$X \text{ Distance} = XDiff - (W1 + W2) / 2$

$Y \text{ Distance} = YDiff - (H1 + H2) / 2$

$X \text{ Distance} = 2, Y \text{ Distance} = 1$

(both need to be  $< 0$  to be colliding)



Width: 5  
P1: 0, -1

Height: 3

Width: 3  
P2: 3, 2

Height: 5

X Diff: 3  
Y Diff: 3

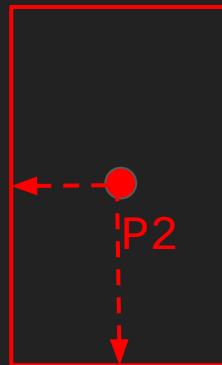
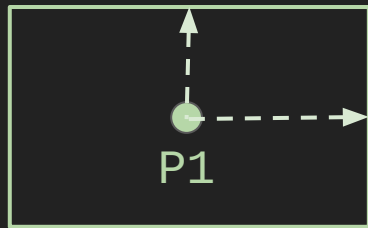
$\text{fabs}(x2 - x1)$   
 $\text{fabs}(y2 - y1)$

X Distance =  $XDiff - (W1 + W2) / 2$

Y Distance =  $YDiff - (H1 + H2) / 2$

X Distance = -1, Y Distance = -1  
(both are  $< 0$  = colliding)

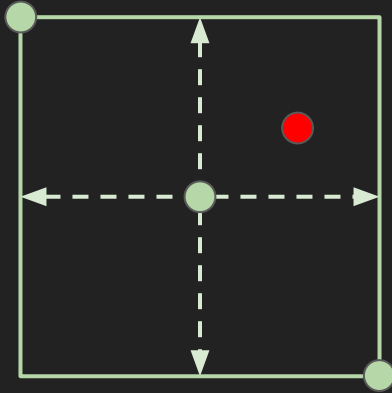
# Box - Box Collision Detection



```
float xdist = fabs(x2 - x1) - ((w1 + w2) / 2.0f);  
float ydist = fabs(y2 - y1) - ((h1 + h2) / 2.0f);  
  
if (xdist < 0 && ydist < 0) // Colliding!
```

# Point - Box Collision Detection

# Point - Box Collision Detection



Get the top left and bottom right corners.  
Check if the X,Y of the point is inside.

Let's get organized!



```
modelMatrix = glm::mat4(1.0f);
modelMatrix = glm::translate(modelMatrix, player_position);
program.SetModelMatrix(modelMatrix);

float vertices[] = { -0.5, -0.5, 0.5, -0.5, 0.5, 0.5, -0.5, -0.5, 0.5, 0.5, -0.5, 0.5 };
float texCoords[] = { 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0 };

glBindTexture(GL_TEXTURE_2D, playerTextureID);

glVertexAttribPointer(program.positionAttribute, 2, GL_FLOAT, false, 0, vertices);
glEnableVertexAttribArray(program.positionAttribute);

glVertexAttribPointer(program.texCoordAttribute, 2, GL_FLOAT, false, 0, texCoords);
glEnableVertexAttribArray(program.texCoordAttribute);

glDrawArrays(GL_TRIANGLES, 0, 6);

glDisableVertexAttribArray(program.positionAttribute);
glDisableVertexAttribArray(program.texCoordAttribute);
```

“I’ve got a  
bad feeling  
about this.”



# Entities!



# Make a Class/Header

```
class Entity {  
public:  
  
    glm::vec3 position;  
    glm::vec3 movement;  
    float speed;  
  
    GLuint textureID;  
  
    Entity();  
  
    void Update(float deltaTime);  
    void Render(ShaderProgram *program);  
};
```

# Make a Class/Header

```
void Entity::Update(float deltaTime)
{
    position += movement * speed * deltaTime;
}

void Entity::Render(ShaderProgram *program) {
    glm::mat4 modelMatrix = glm::mat4(1.0f);
    modelMatrix = glm::translate(modelMatrix, position);
    program->SetModelMatrix(modelMatrix);

    float vertices[] = { -0.5, -0.5, 0.5, -0.5, 0.5, 0.5, -0.5, -0.5, 0.5, 0.5, -0.5, 0.5 };
    float texCoords[] = { 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0 };

    glBindTexture(GL_TEXTURE_2D, textureID);

    glVertexAttribPointer(program->positionAttribute, 2, GL_FLOAT, false, 0, vertices);
    glEnableVertexAttribArray(program->positionAttribute);

    glVertexAttribPointer(program->texCoordAttribute, 2, GL_FLOAT, false, 0, texCoords);
    glEnableVertexAttribArray(program->texCoordAttribute);

    glDrawArrays(GL_TRIANGLES, 0, 6);

    glDisableVertexAttribArray(program->positionAttribute);
    glDisableVertexAttribArray(program->texCoordAttribute);
}
```

# The Game Loop

## ProcessInput()

- player.movement = ...

## Update()

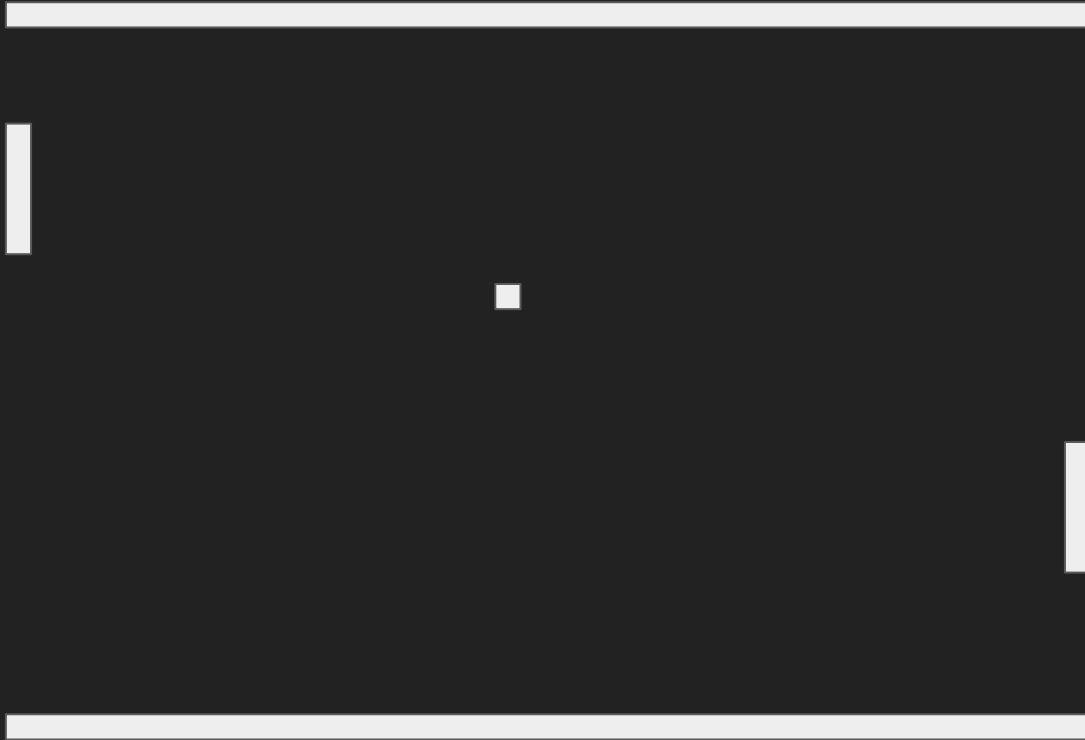
- Calculate deltaTime
- for each entity:
  - entity.Update(deltaTime);

## Render()

- Clear the screen
- for each entity:
  - entity.Render(program);

You're ready to make  
your first game!

# Pong!



# Project 2: Pong

Make Pong!

- Needs a paddle on each side that can move individually.

- Needs a object that bounces off of the paddles and top/bottom wall.

- Does not need to keep score but must detect when someone wins.

- Can use images or untextured polygons.

- Can use keyboard, mouse or joystick input.

- You can have both players using the keyboard if you want.

- Commit your code to your GitHub repository.

- Post the link in the Assignments area.

For example, your link might look like:

<https://github.com/tonystark/CS3113/P2/>

If there is time left in class:

Start working on Project 2!