Abertay University

Collision Detection

CMP105 Games Programming

This week

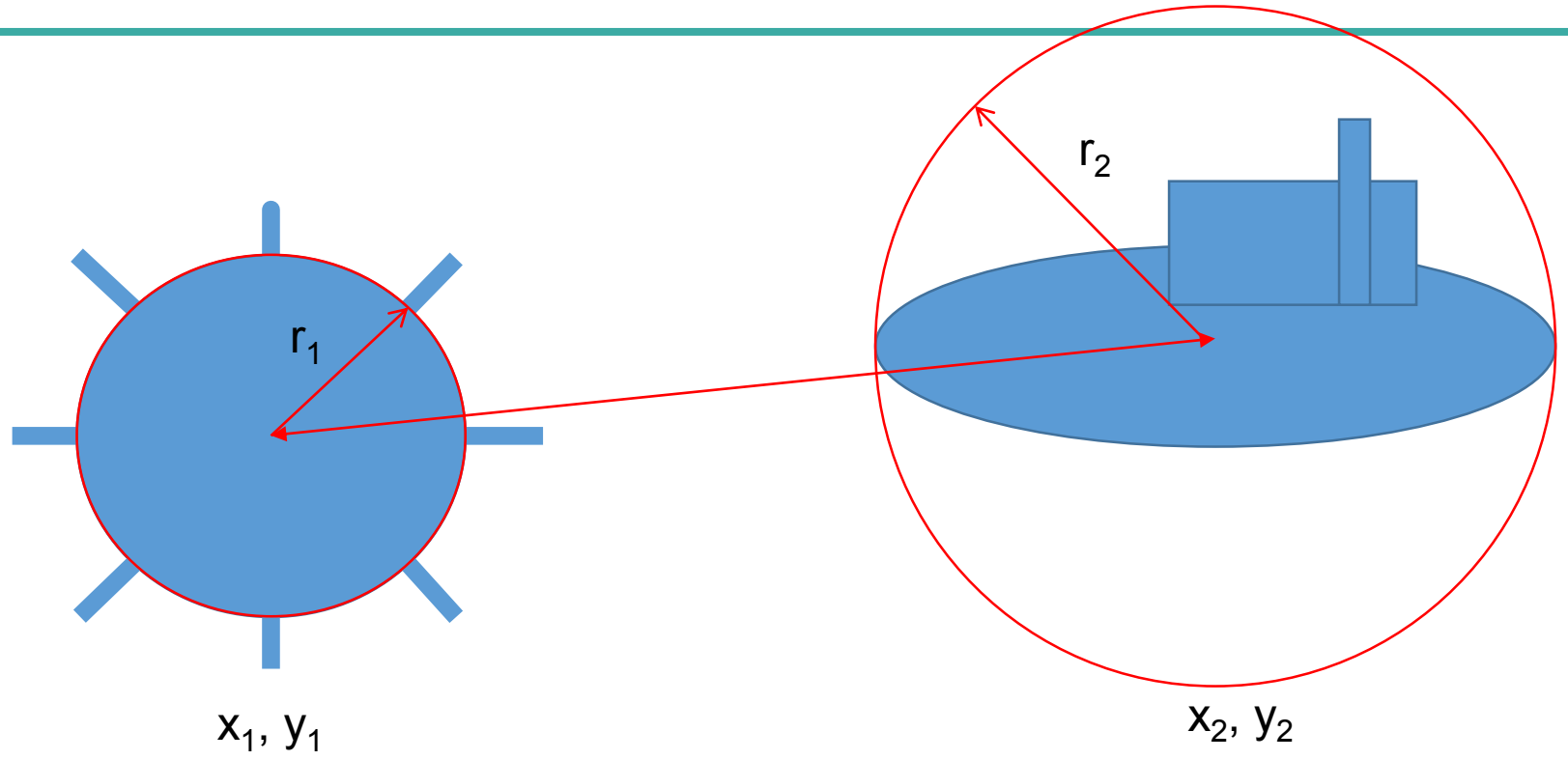
- Collision detection
  - Bounding circle
  - Axis Aligned Bounding Box
  - Object Orientated Bounding Box
  - Optimisations
- Collision resolution
- Examples
  - Sphere bounding
  - AABB

Collision terminology

- Collision detection:
  - Determine if two objects occupy the same space within a game world (2D/3D).
  - Determine if an object has interacted with the game environment (Walls, floor, etc).

- Collision response
  - Specification/calculation of what happens to the objects and/or the game environment after a collision has been detected.

Bounding circle



- A collision has occurred if:

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} < (r_2 + r_1)$$

Bounding circle

- Optimising the distance calculation
  - Don’t computer the square root
    - Too resource intensive
  - Instead:

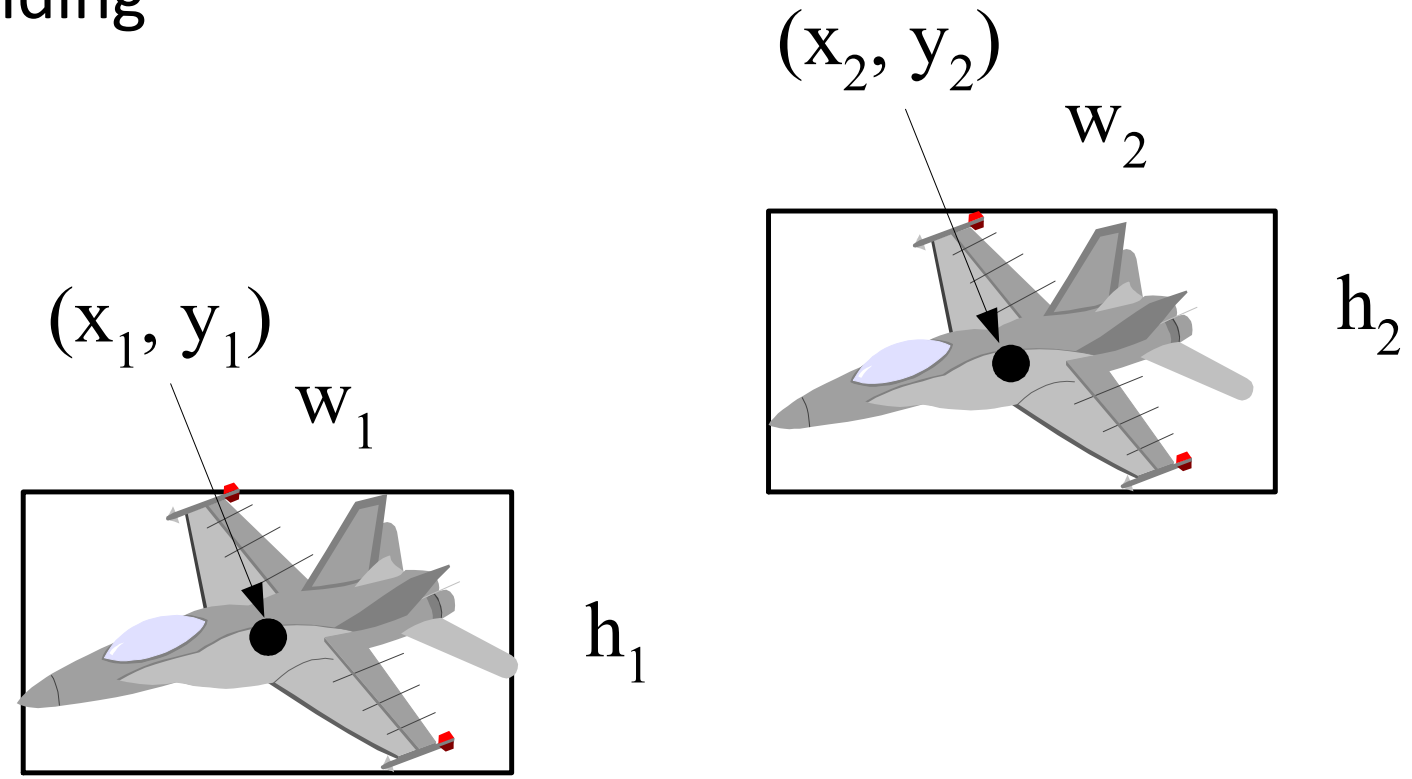
$$(x_2 - x_1)^2 + (y_2 - y_1)^2 < (r_2 + r_1)^2$$

Bounding circle

- What to use for centre?
  - Origin of shape (ours is in the top left) / Centre of shape
  - Centroid (average of all points)
  - Centre of bounding box

Axis Aligned Bounding Boxes (AABB)

- How we determine if the boxes overlap
- Easier to check if **NOT** colliding



Axis Aligned Bounding Boxes (AABB)

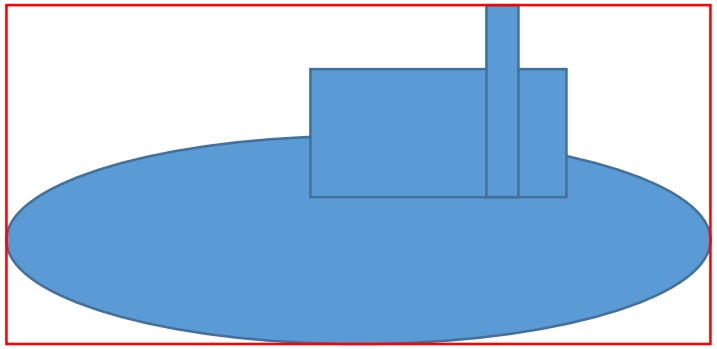
- if Sprite1.right is less than Sprite2.left
  - Return false
- If Sprite1.left is greater than Sprite2.right
  - Return false
- If Sprite1.bottom is less than Sprite2.top
  - Return false
- If Sprite1.top is greater than Sprite2.bottom
  - Return false
- Return true

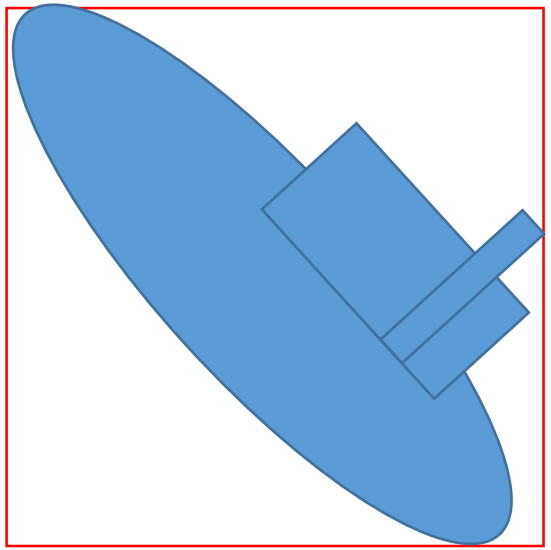
Axis Aligned Bounding Boxes (AABB)

```
// check AABB
bool Collision::checkBoundingBox(GameObject* s1, GameObject* s2)
{
    if (s1->getCollisionBox().left + s1->getCollisionBox().width < s2->getCollisionBox().left)
        return false;
    if (s1->getCollisionBox().left > s2->getCollisionBox().left + s2->getCollisionBox().width)
        return false;
    if (s1->getCollisionBox().top + s1->getCollisionBox().height < s2->getCollisionBox().top)
        return false;
    if (s1->getCollisionBox().top > s2->getCollisionBox().top + s2->getCollisionBox().height)
        return false;

return true;
}
```

Axis Aligned Bounding Boxes (AABB)

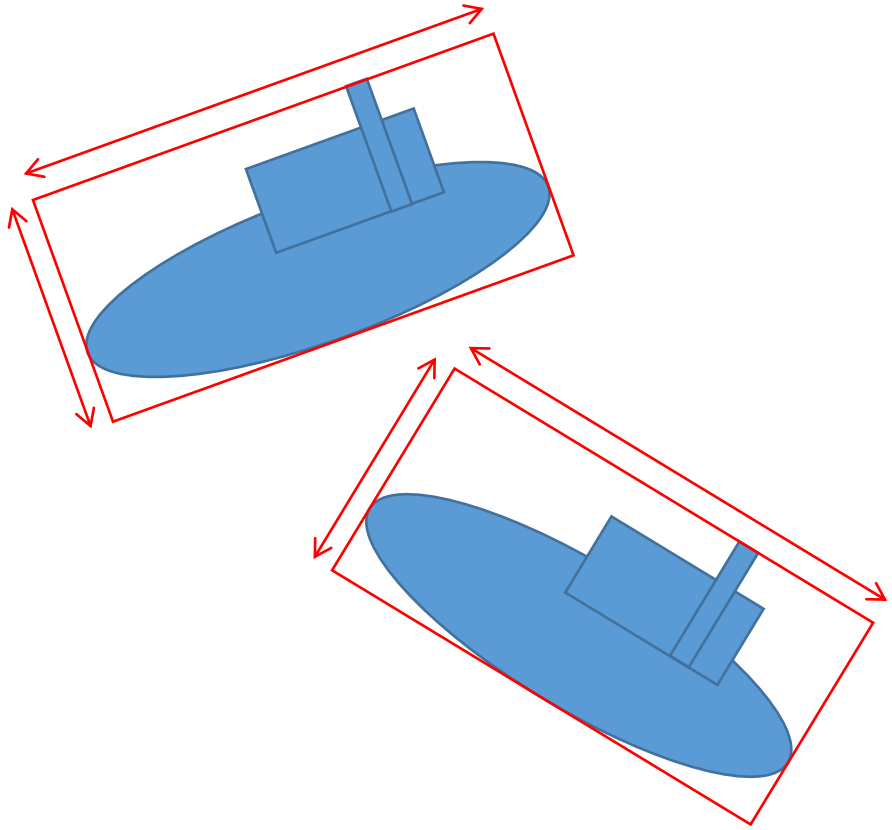
- AABB box edges are aligned with world axes
  - Recalculate when the object changes orientation
  - AABB will change depending on orientation of the bounding shape
  - This is computationally inexpensive but can be inaccurate





Object Orientated Bounding Boxes (OOBB)

- OOBB
  - Box edges aligned with local object coordinate system
  - Much tighter, but collision calcs costly
- Solved accurately with “axes of separation theorem”
- Find an axis which separate the object.
- An axes exists perpendicular to each edge of the shape
- There are four separating axes for this situation



http://www.essentialmath.com/CollisionDetection.pps
http://www.metanetsoftware.com/technique/tutorialA.html
http://www.gamasutra.com/view/feature/131790/simple_intersection_tests_for_games.php

You get a collision, you get a collision, …

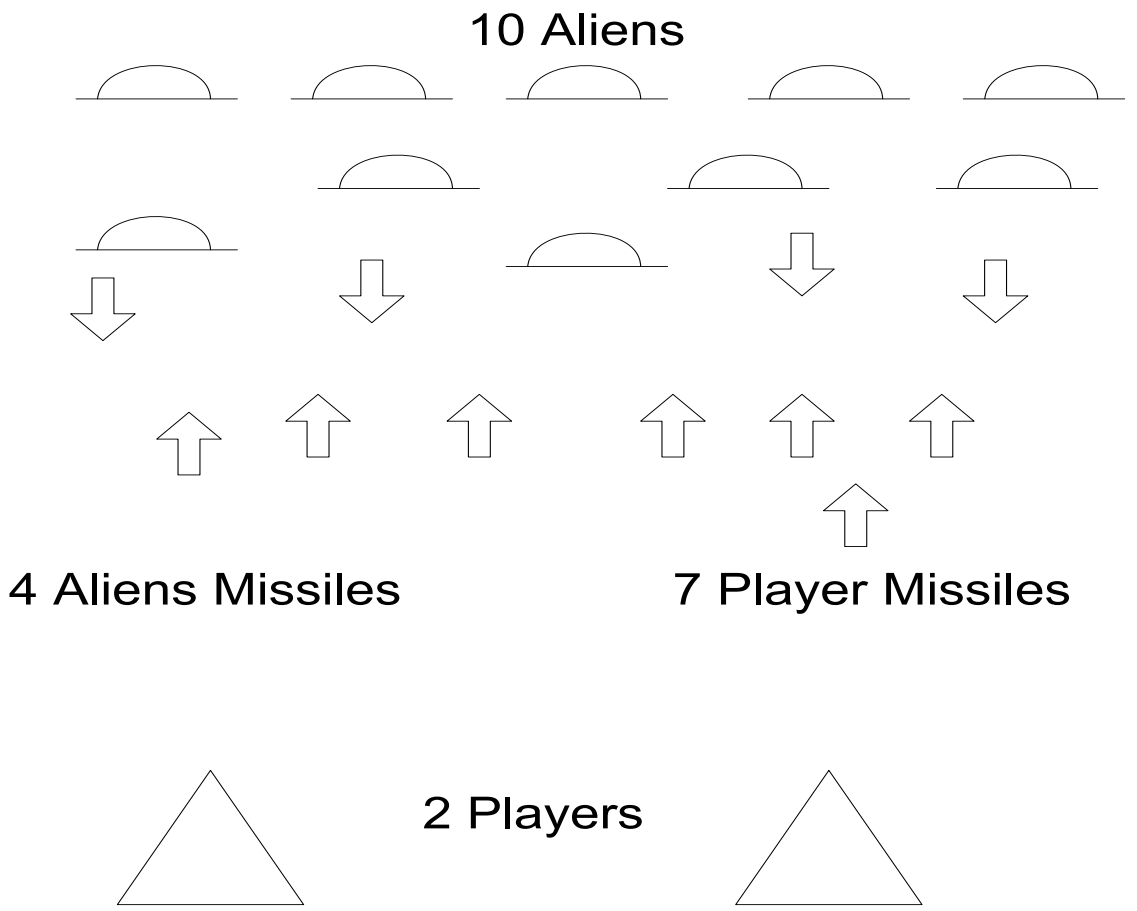
Objects	Collision Tests
2	1
3	3
4	6
5	10
6	15
7	21
8	28
9	36
10	45
15	105
20	190

- Rapidly increasing number of collision tests.
- The numbers are derived from the formula $(n^2 - n)/2$
- This algorithm is said to be of order $O(n^2)$
- It could be worse: $O(n^3)$, $O(2^n)$
- $O(n)$ is good
- $O(1)$ is pure bliss

- How can we reduce the collision test requirements?

Using game rules

- How many potential collision detections per frame?
- $(23^2 – 23)/2 = 253$



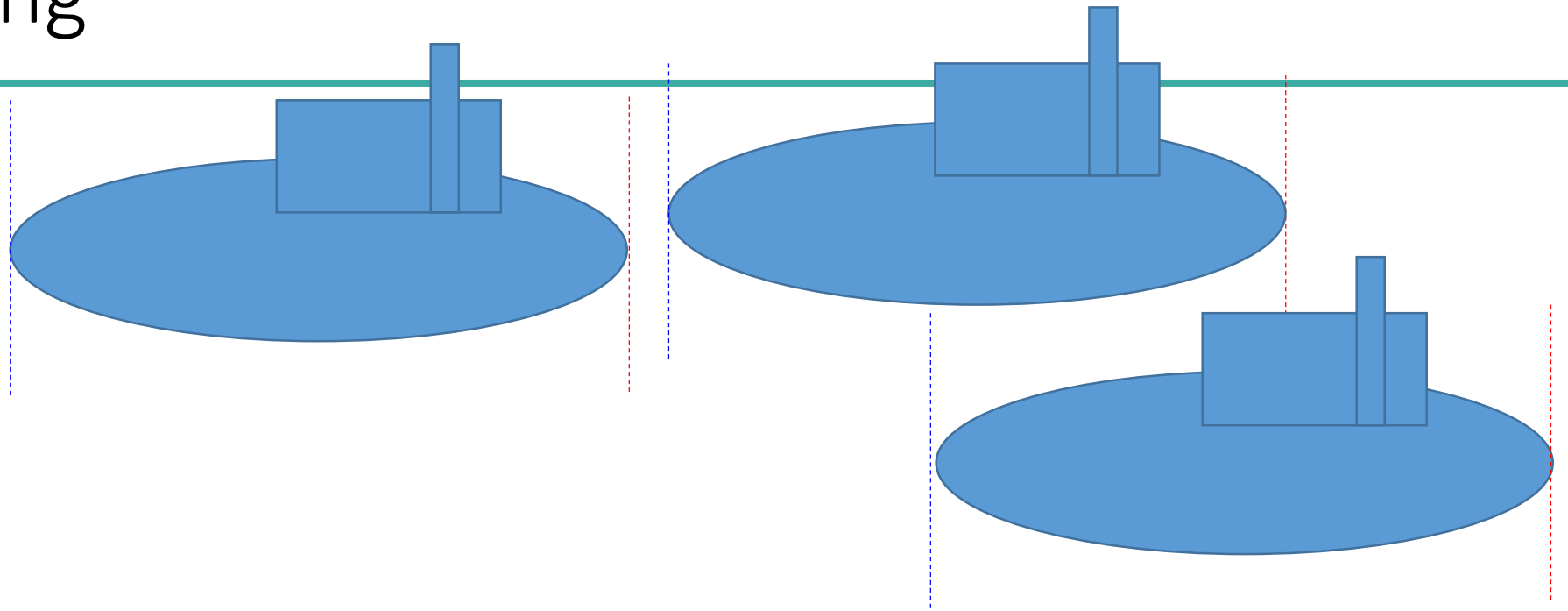
Using game rules

- No similar missile - missile collisions
- No alien – alien collisions
- No alien missile – alien collisions
- No player missile – player collisions

- Alien missiles colliding with players (8)
- Player missiles colliding with aliens (70)
- Aliens colliding with players (20)

- **98 tests per frame instead of 253**

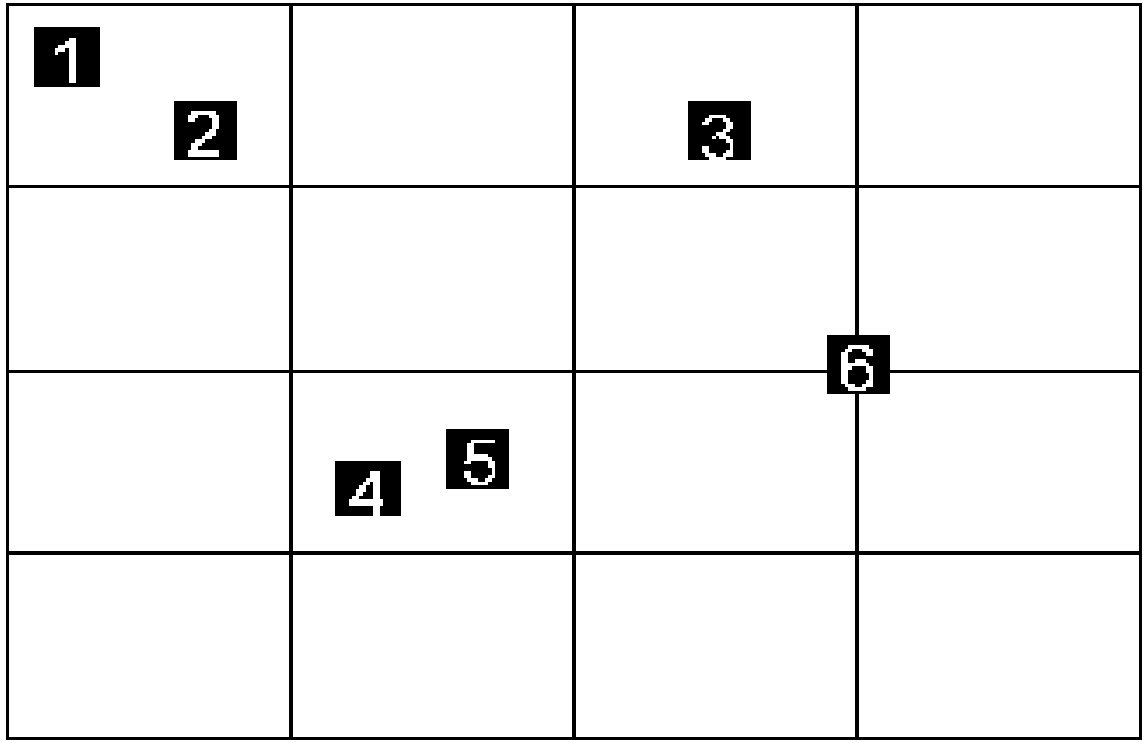
Axis sorting



- Sort objects according to their position
- Only necessary to compare objects close to each other in the table
- Overheads associated with maintaining the table

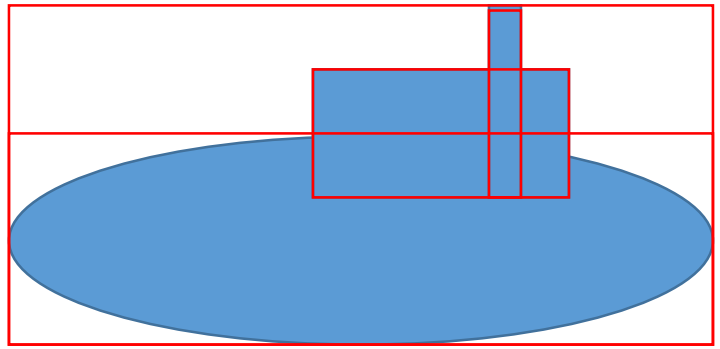
Spatial partitioning

- Only test objects in the same partition
- Overheads associated with maintaining data



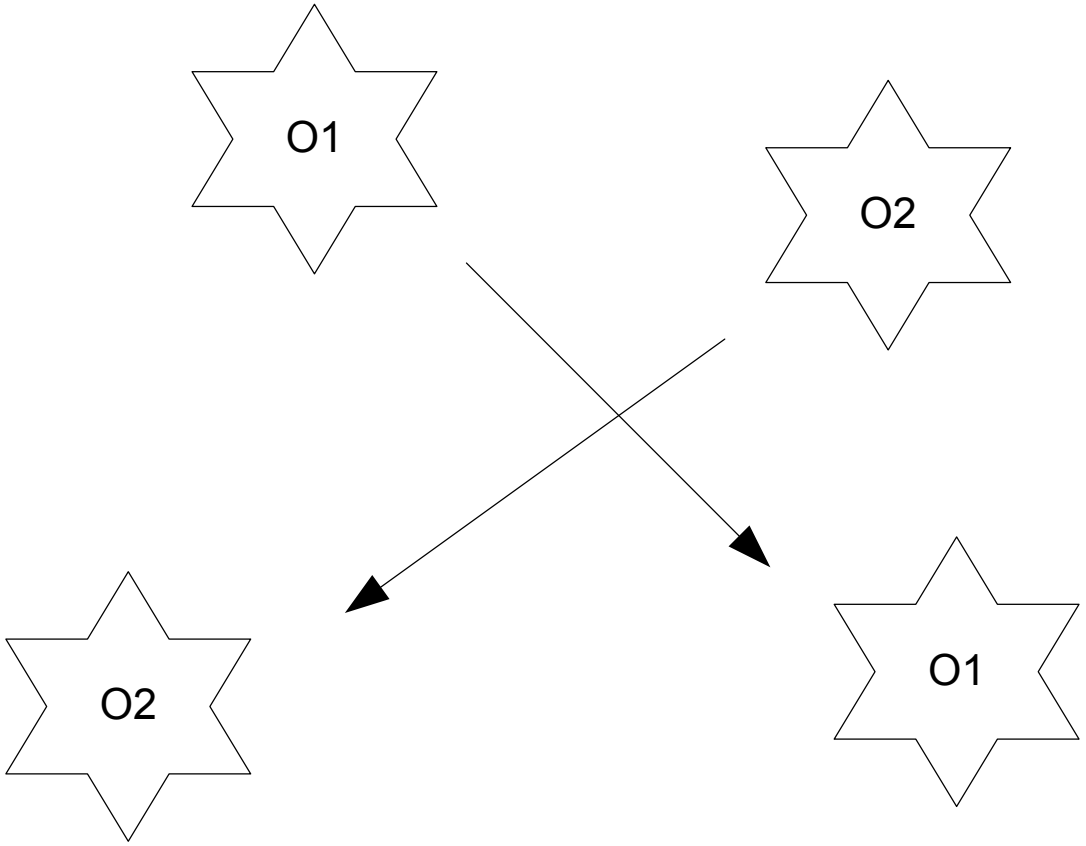
Hierarchical Collision Detection

- Simple test to reject most possibilities
- Increased level of detection depending upon game situation



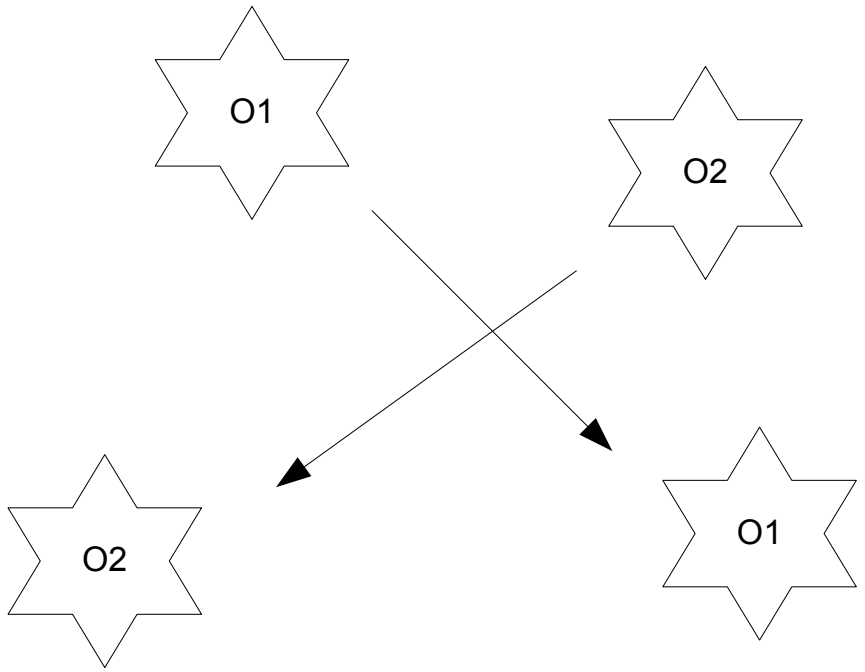
Velocity problems

- Is there a collision or not?



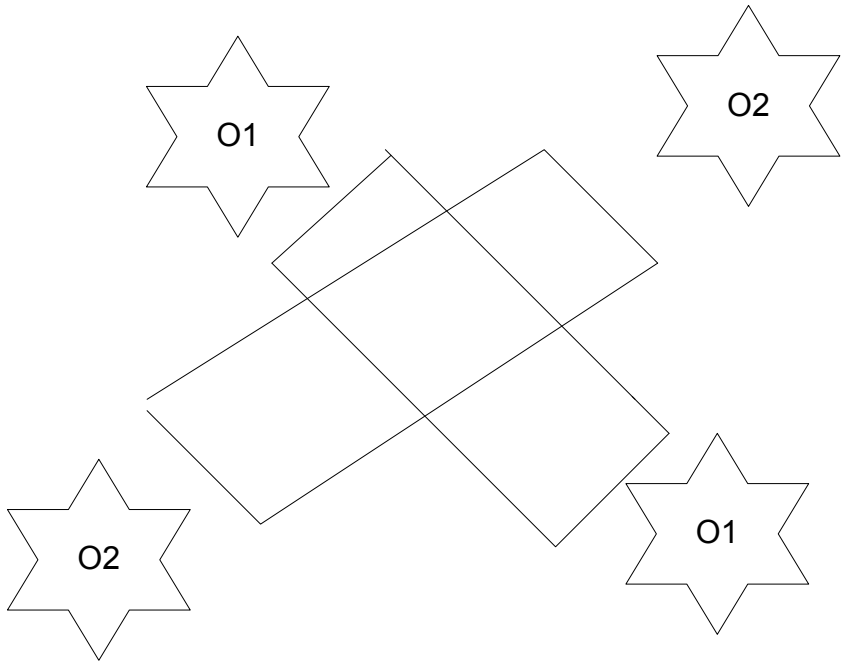
Line segment intersection

- Do the trace line intersect?
  - Yes, but not necessarily a collision!



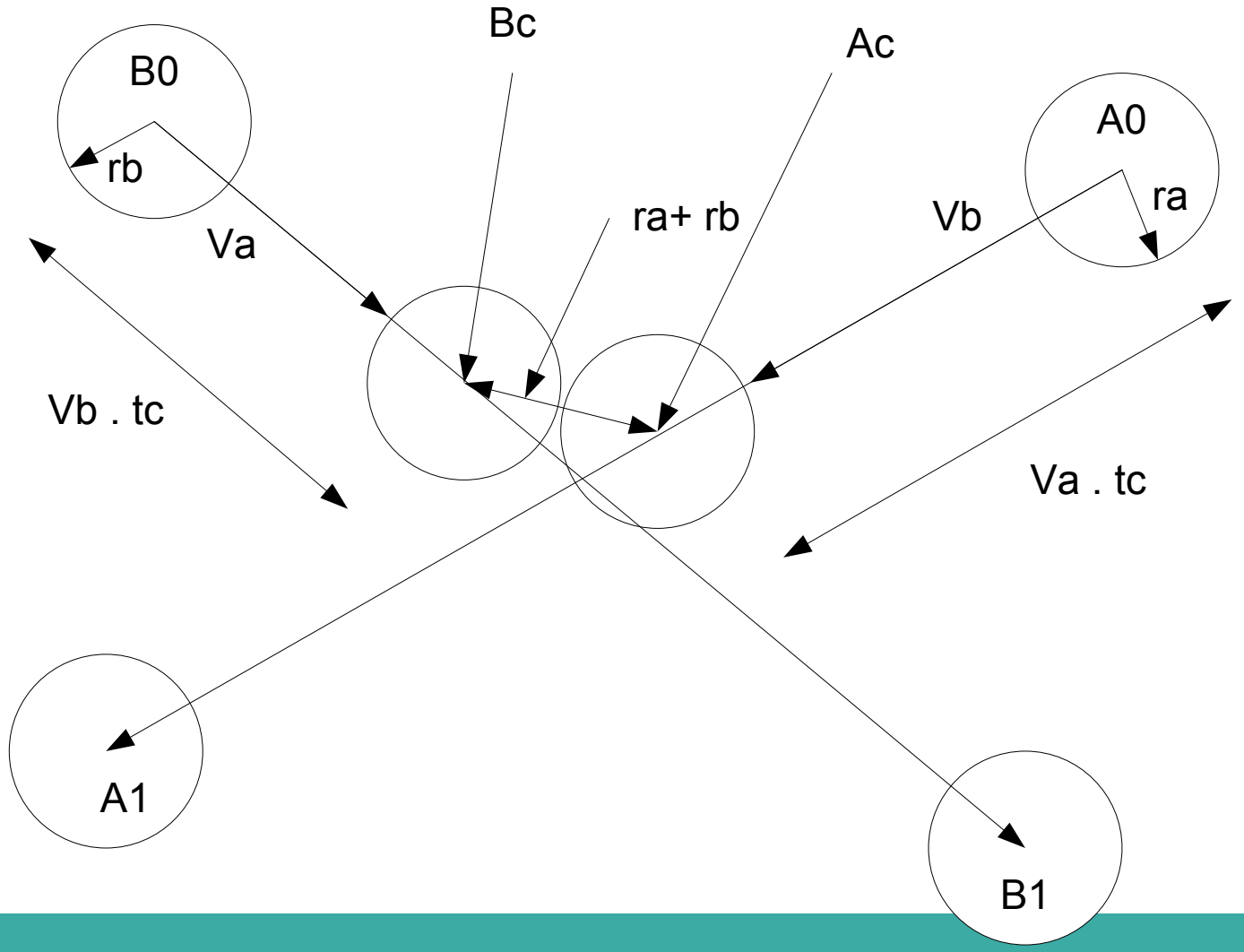
Swept volume tests

- Do swept volumes intersect?
  - Non axis aligned bounding boxes
  - Yes, still not necessarily a collision!



Analytic solution

- Time to collision
- Position of objects at that time



Collision resolution / response

- Depends upon the game play
  - Increment a score
  - Create an explosion
  - Change the object velocity
  - Prevent object moving though a wall
  - Kill the player/object
  - etc. etc.

Physics engines

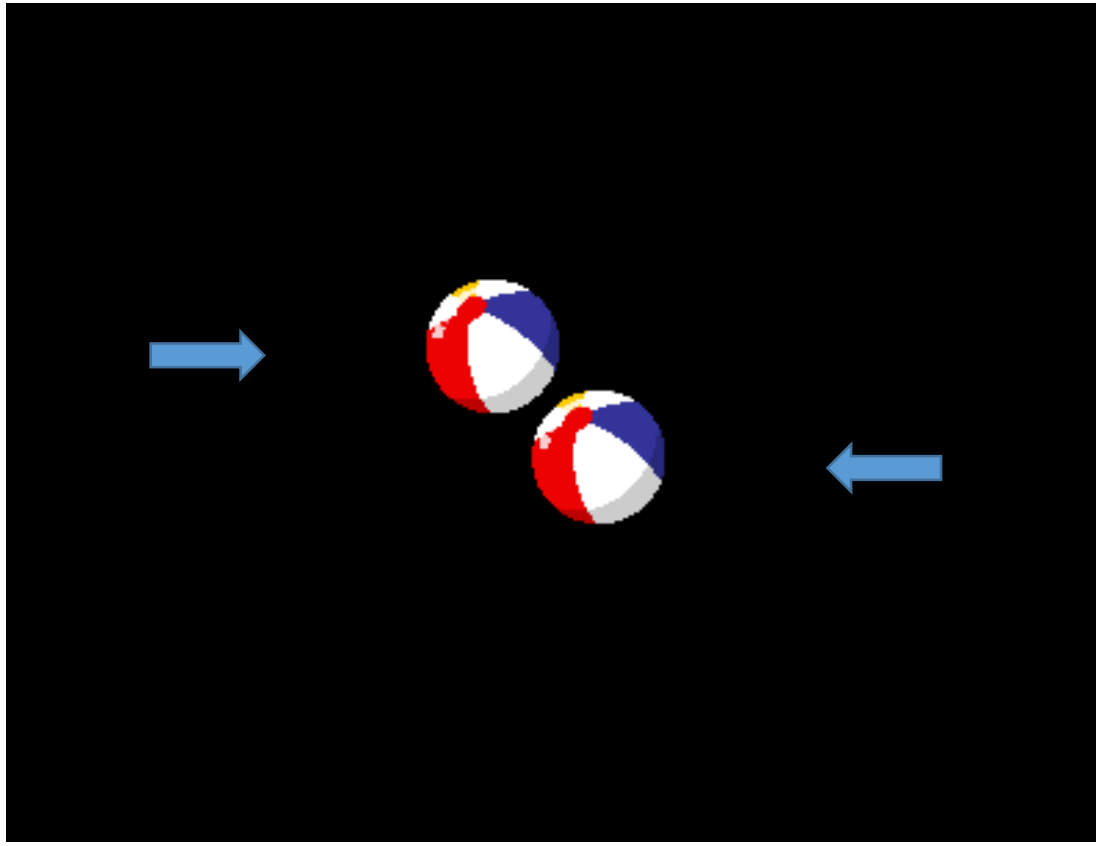
- Physics engines contain collision detection systems
  - Box2D, PhysX, ODE, Havok, Bullet, etc..
- For complex collision detection, best to use an engine rather than implement your own
  - We will be focusing on the major detection algorithms
    - Bounding circle/sphere
    - AABB

Examples

- I will provide a static class that has collision detection functions
- Bounding circle
  - Function that compares two Game objects
    - Using the bounding circle calculation
    - Returns true if sprites are colliding
- AABB (two versions)
  - Function that compares two Game Objects
    - Using the AABB calculation
    - Returns true if sprites are colliding
  - Sprites need to configure a bounding box
- Second version compares a Game Object to a Vector2i position

Sphere bounding example

- Two objects with ball texture
- Moving towards each other
- Every frame
  - Update the ball objects
  - Check for collision
    - If collision, resolve collision
  - Render objects



Game update()

```
ball1.update(dt);
ball2.update(dt);
if (Collision::checkBoundingCircle(&ball1, &ball2))
{
   ball1.collisionResponse(NULL);
   ball2.collisionResponse(NULL);
}
```

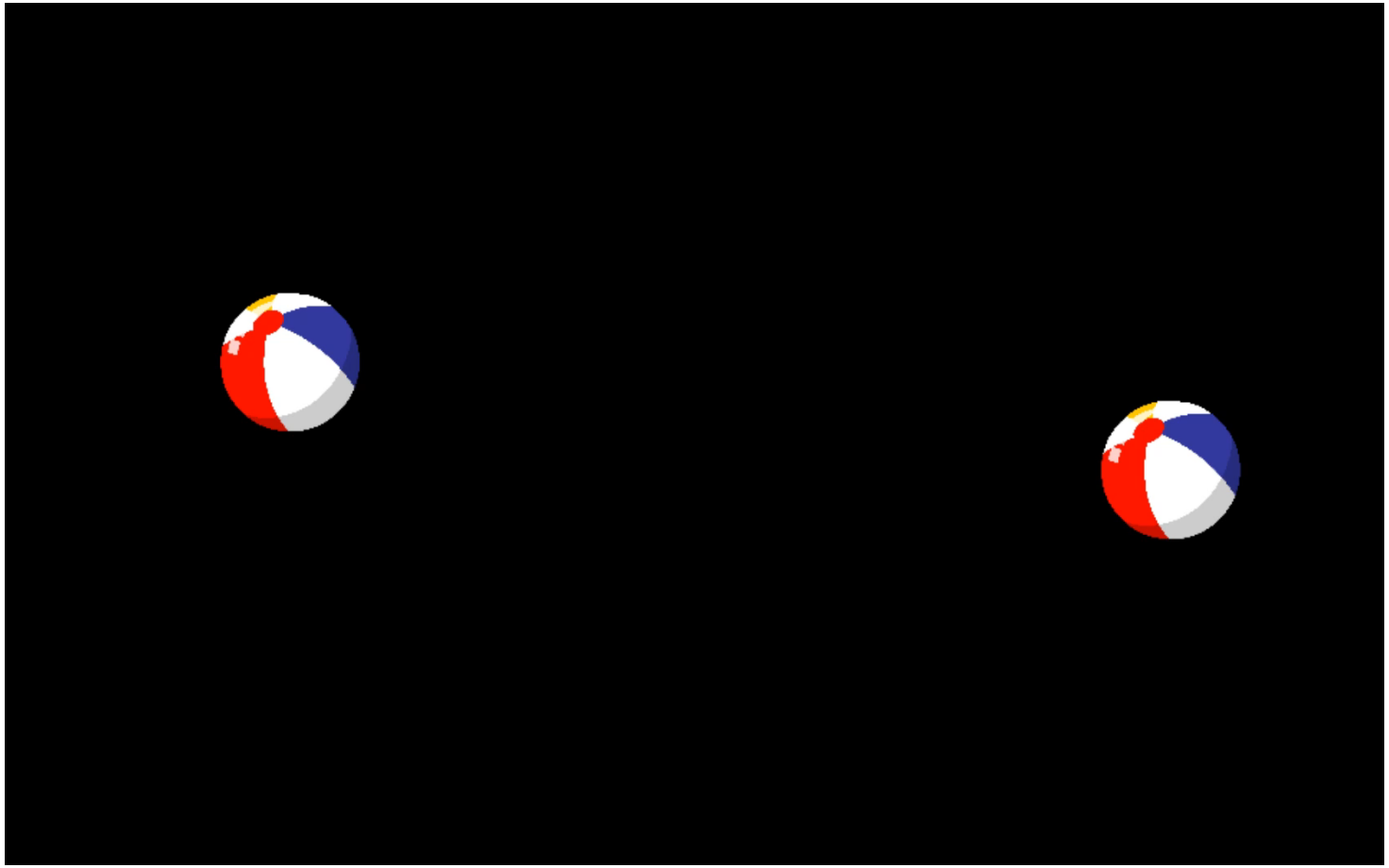
Ball update

```
void Ball::update(float dt)
{
    move(velocity*dt);

    if (getPosition().x < 0)
    {
        setPosition(0, getPosition().y);
        velocity.x = -velocity.x;
    }
    if (getPosition().x > 750)
    {
        setPosition(750, getPosition().y);
        velocity.x = -velocity.x;
    }
}
void Ball::collisionResponse(GameObject* collider)
{
        velocity.x = -velocity.x;
}
```

Bounding circle/sphere detection

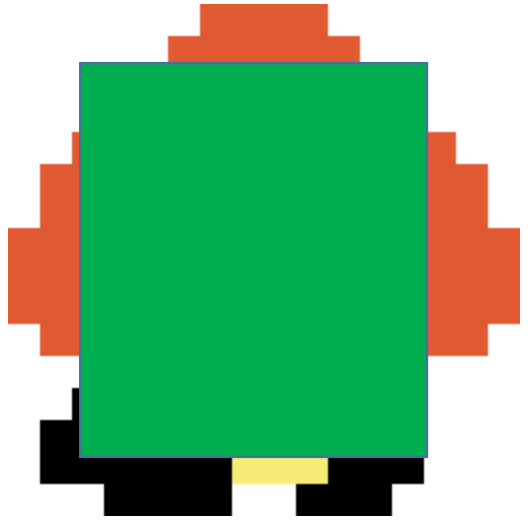
```
// check Sphere bounding collision
bool Collision::checkBoundingCircle(GameObject* s1, GameObject* s2)
{
    // Get radius and centre of sprites.
    float radius1 = s1->getSize().x / 2;
    float radius2 = s2->getSize().x / 2;
    float xpos1 = s1->getPosition().x + radius1;
    float xpos2 = s2->getPosition().x + radius2;
    float ypos1 = s1->getPosition().y + radius1;
    float ypos2 = s2->getPosition().y + radius2;

    if(pow(xpos2 - xpos1, 2) + pow(ypos2 - ypos1, 2) < pow(radius1 + radius2, 2))
    {
         return true;
    }
return false;
}
```

Example



AABB example

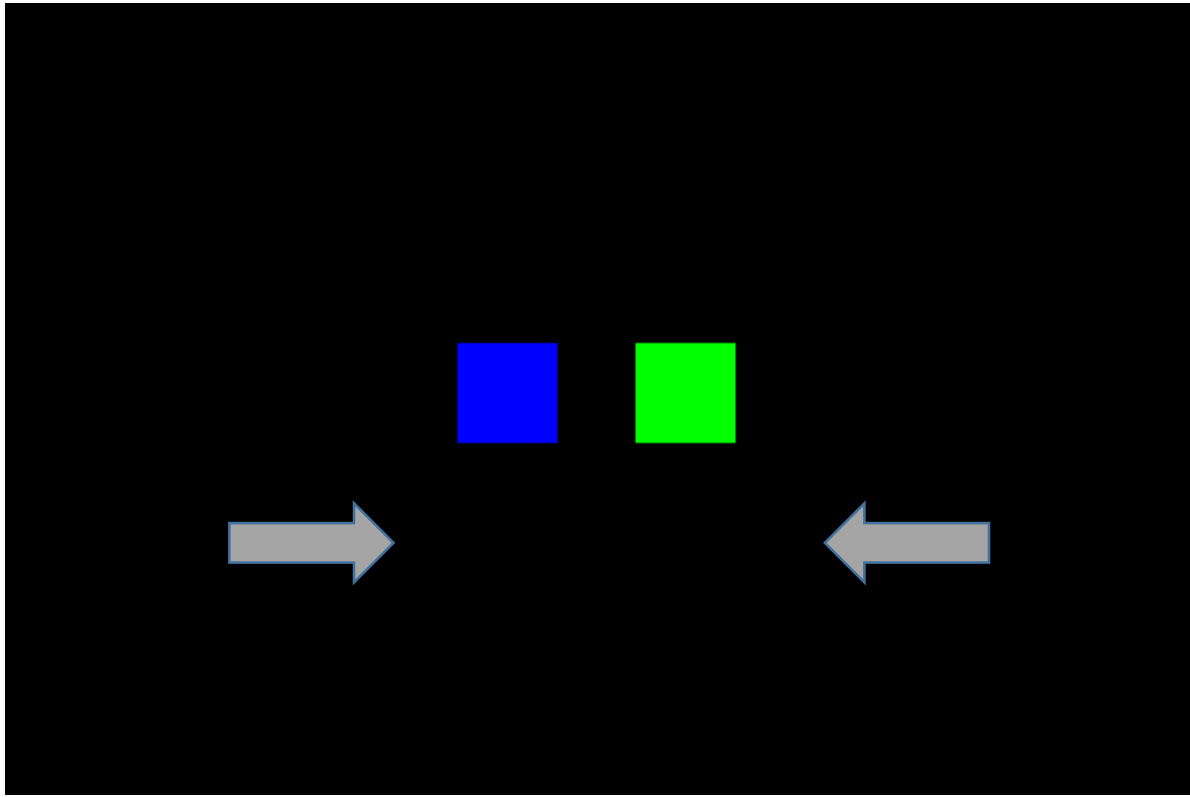
- GameObject class already contains a variable for representing the bounding box
  - sf::`FloatRect` collisionBox;
- Allows different sized bounding box instead of just sprite size
- Needs to be configured
  - Is set in relation to the sprite origin
  - Could change based on sprite changes/rotation/animation



Mario bounding box example

AABB Example

- Two quads moving towards each other
  - Configure collision box
- Every frame
  - Update objects
    - Move
  - Check for collision
    - Resolve collision
  - Render



Configure shapes

```
// bouncing squares
square1.setSize(sf::Vector2f(50, 50));
square1.setCollisionBox(sf::FloatRect(0, 0, 50, 50));
square1.setPosition(0, 200);
square1.setVelocity(50, 0);
square1.setFillColor(sf::Color::Blue);

square2.setPosition(750, 200);
square2.setSize(sf::Vector2f(50, 50));
square2.setCollisionBox(sf::FloatRect(0, 0, 50, 50));
square2.setVelocity(-50, 0);
square2.setFillColor(sf::Color::Green);
```

Game.update()

```
square1.update(dt);
square2.update(dt);

if (Collision::checkBoundingBox(&square1, &square2))
{
	square1.collisionResponse(NULL);
	square2.collisionResponse(NULL);
}
```

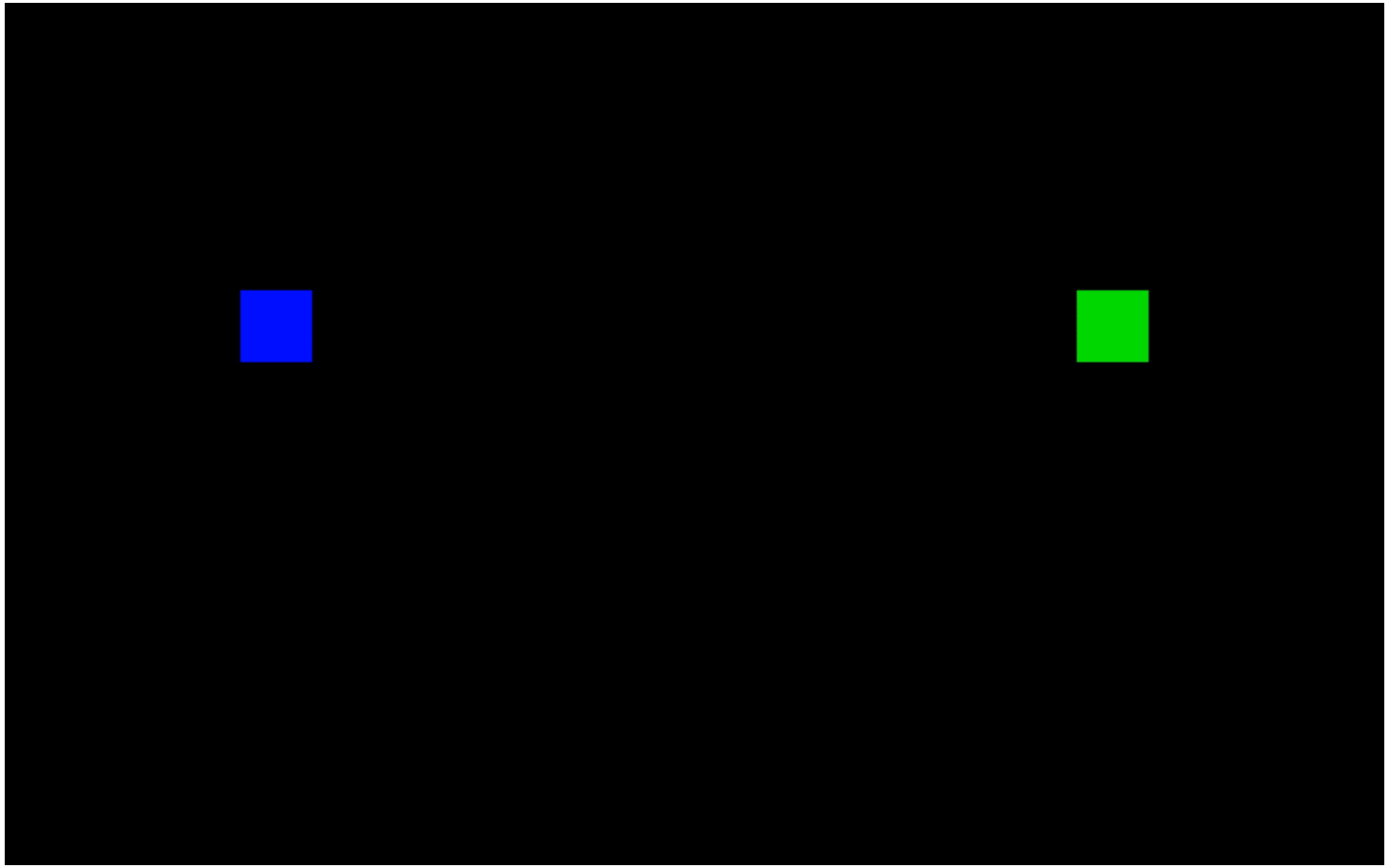
Square update

```
void Square::update(float dt)
{
	move(velocity*dt);
	if (getPosition().x < 0)
	{
	setPosition(0, getPosition().y);
	velocity.x = -velocity.x;
	}
	if (getPosition().x > 750)
	{
	setPosition(750, getPosition().y);
	velocity.x = -velocity.x;
	}
}
void Square::collisionResponse(GameObject* collider)
{
	velocity.x = -velocity.x;
}
```

AABB Collision detection

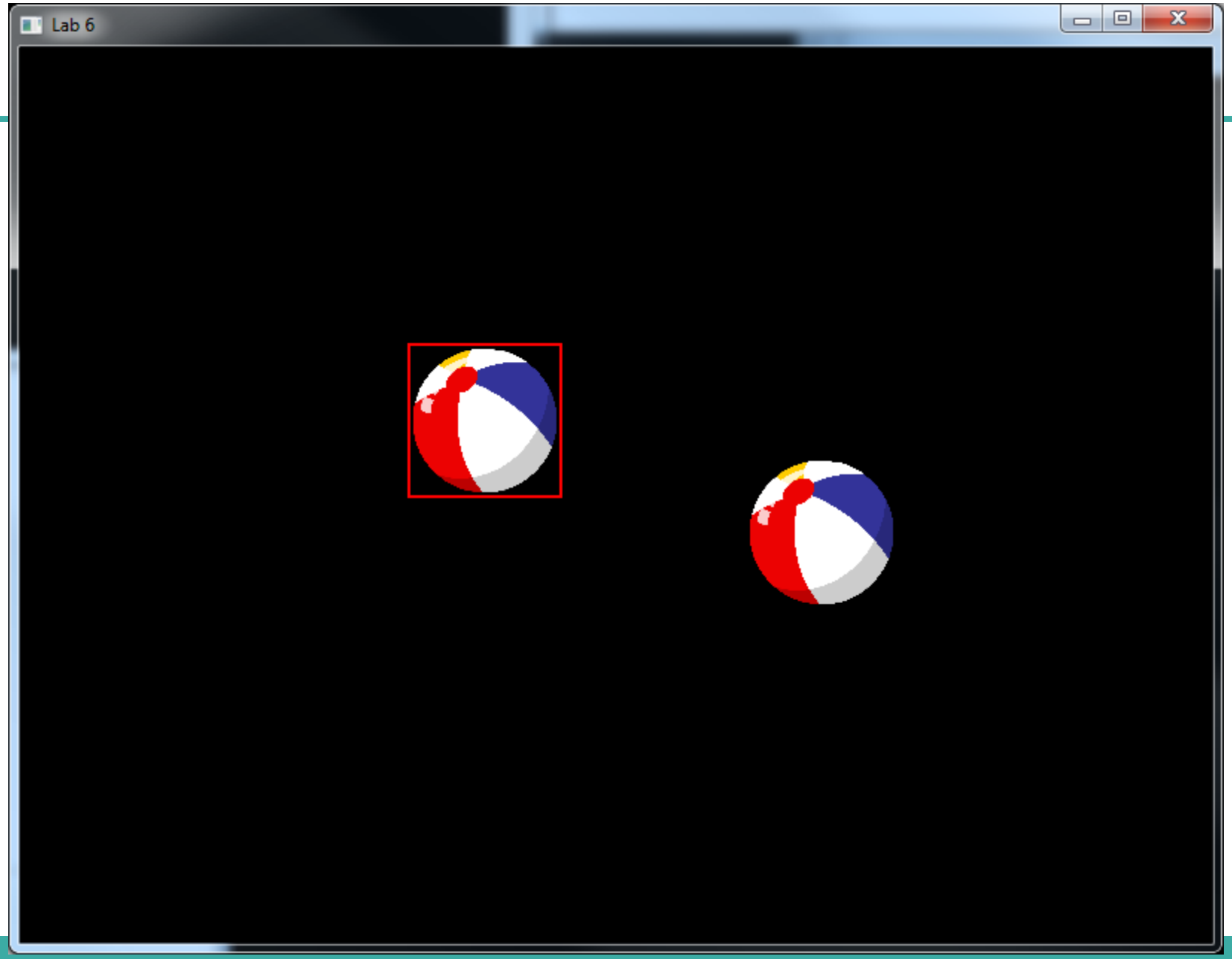
```
// check AABB
bool Collision::checkBoundingBox(GameObject* s1, GameObject* s2)
{
    if (s1->getCollisionBox().left + s1->getCollisionBox().width < s2->getCollisionBox().left)
        return false;
    if (s1->getCollisionBox().left > s2->getCollisionBox().left + s2->getCollisionBox().width)
        return false;
    if (s1->getCollisionBox().top + s1->getCollisionBox().height < s2->getCollisionBox().top)
        return false;
    if (s1->getCollisionBox().top > s2->getCollisionBox().top + s2->getCollisionBox().height)
        return false;

    return true;
}
```

AABB example



Debugging

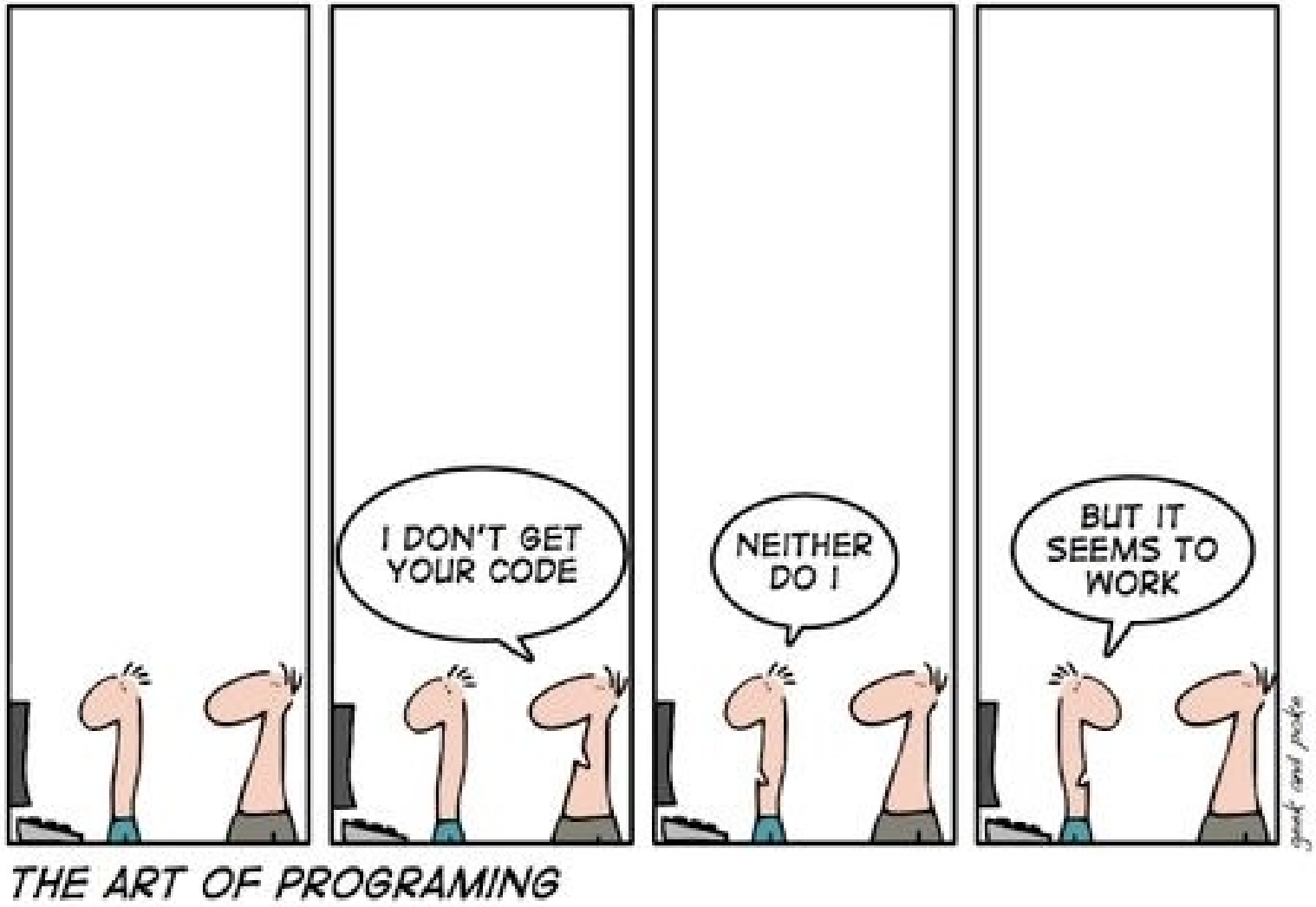
- To check if your collision are correct you could render the collision box
- The collision box is a rectangle, using that data build and render a rectangle
  - Update it’s position every frame


Lab 6

In the labs

- Example project updates the framework adding the collision class
- Making objects that collide
- Making Pong!
- Thinking about coursework
  - If you haven't already, you should be starting your coursework

- Remember – a computer game is an illusion
  - Correct balance between realism and accuracy


I DON'T GET YOUR CODE
NEITHER DO I
BUT IT SEEMS TO WORK
THE ART OF PROGRAMING