CMP105 Games Programming

Collision Detection

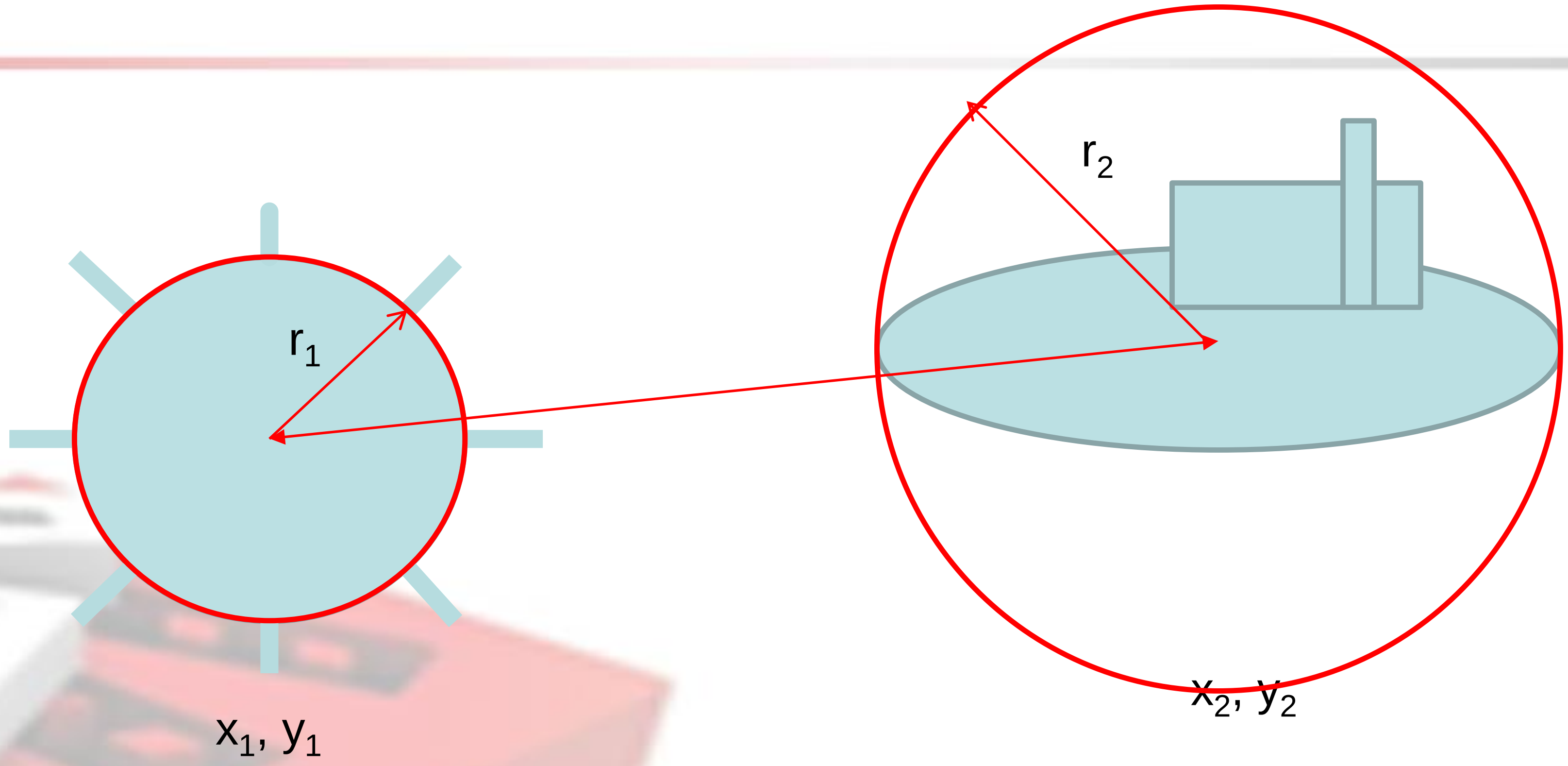
This week

- 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:

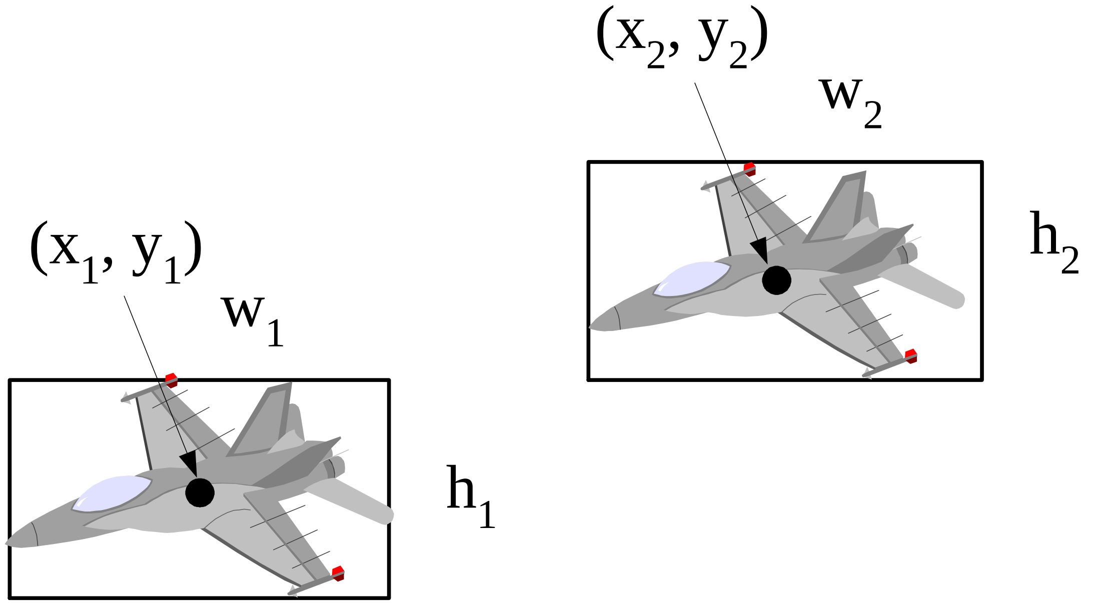
$$(x_2 - x_1)^2 + (y_2 - y_1)^2 < (r_2 + r_1)^2$$

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)

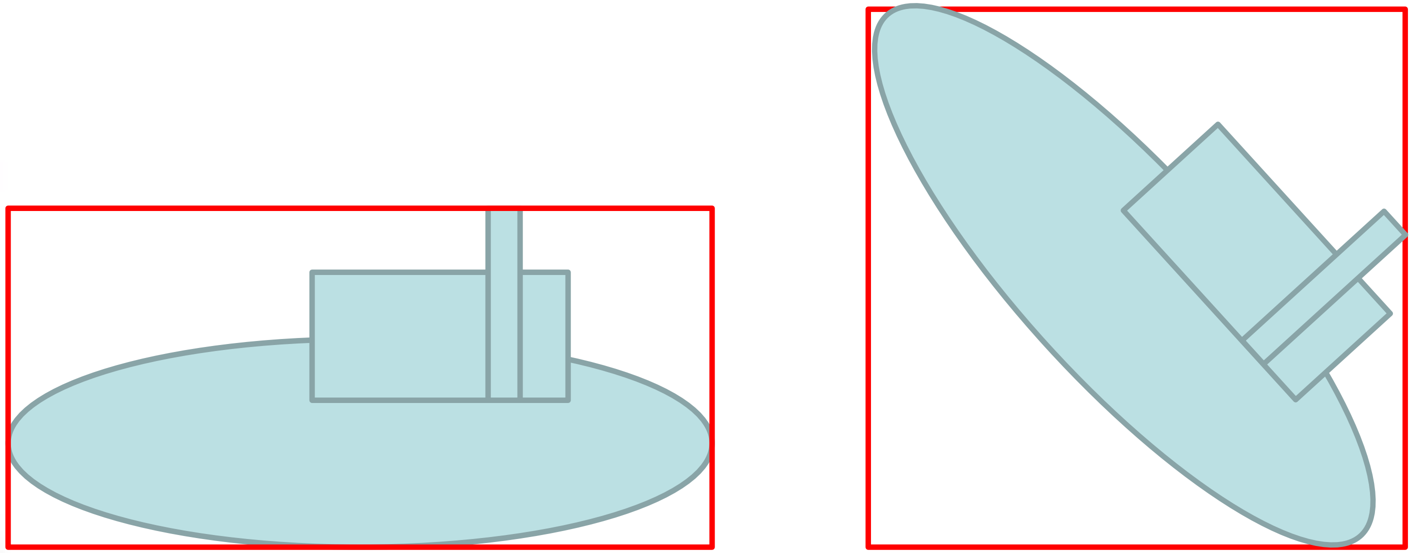
- if Sprite1.right is less than Sprite2.left
  - Return false
- If Sprite1.left is greater than Sprite2.right
  - Return false
- If Sprite1.bottom is less than Sprite2.top
  - Return false
- If Sprite1.top is greater than Sprite2.bottom
  - Return false
- Return true

Axis Aligned Bounding Boxes (AABB)

```
bool Game::checkCollision(Sprite* s1, Sprite* s2)
{
if (s1->getAABB().left + s1->getAABB().width < s2->getAABB().left)
        return false;
if (s1->getAABB().left > s2->getAABB().left + s2->getAABB().width)
        return false;
if (s1->getAABB().top + s1->getAABB().height < s2->getAABB().top)
        return false;
if (s1->getAABB().top > s2->getAABB().top + s2->getAABB().height)
        return false;

return true;
}
```

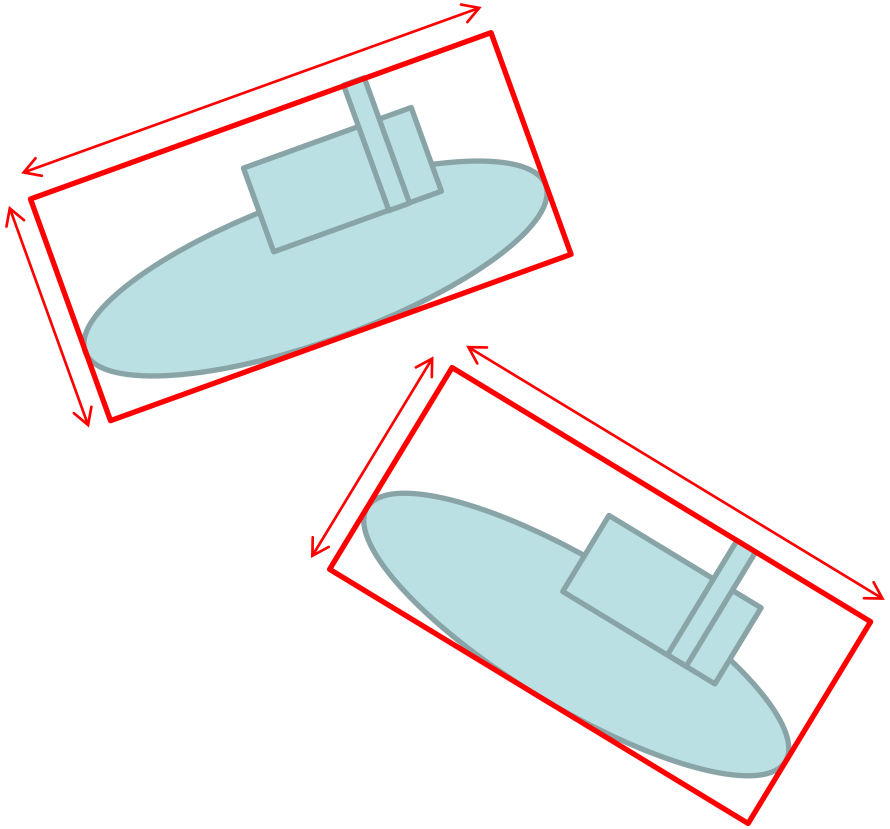
Axis Aligned Bounding Boxes (AABB)

- AABB box edges are aligned with world aces
  - 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, …

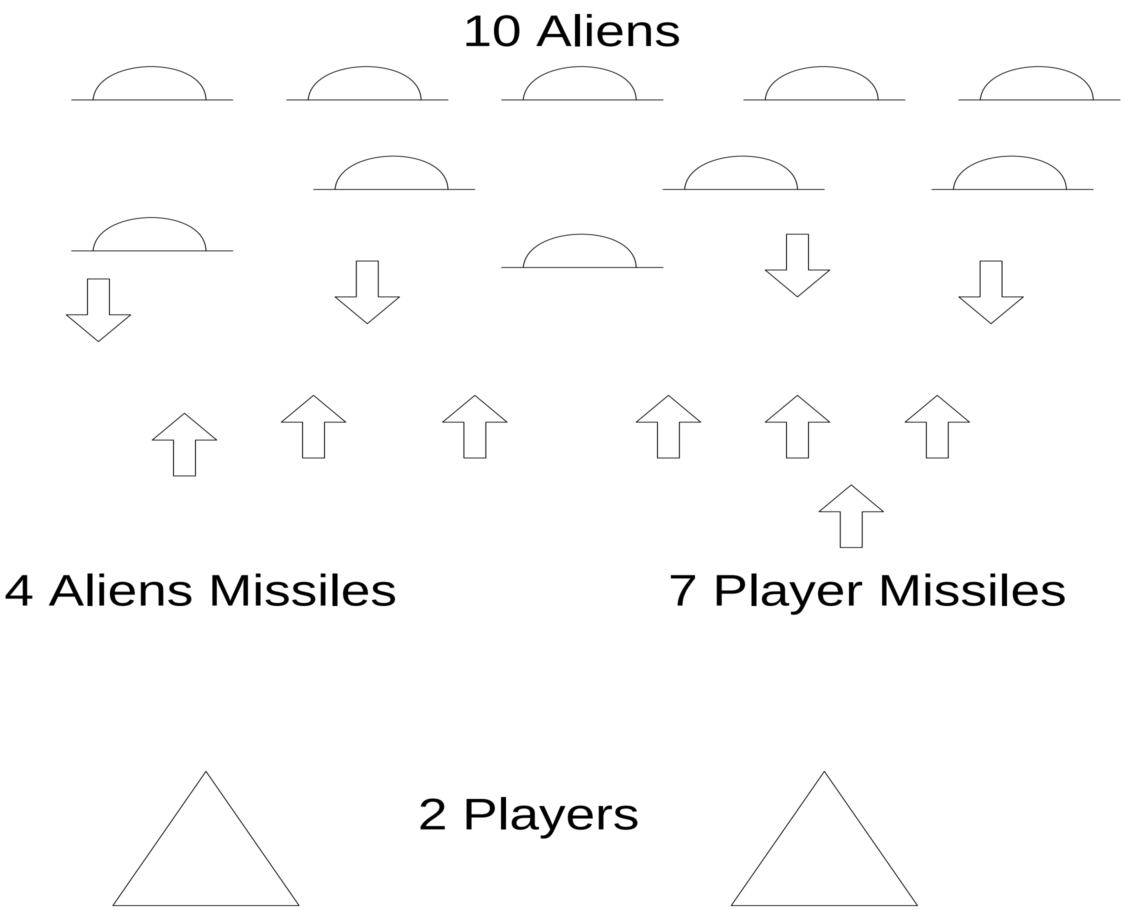
Objects	Collision Tests
2	1
3	3
4	6
5	10
6	15
7	21
8	28
9	36
10	45
15	105
20	190

- 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?
- $(23^2 – 23)/2 = 253$



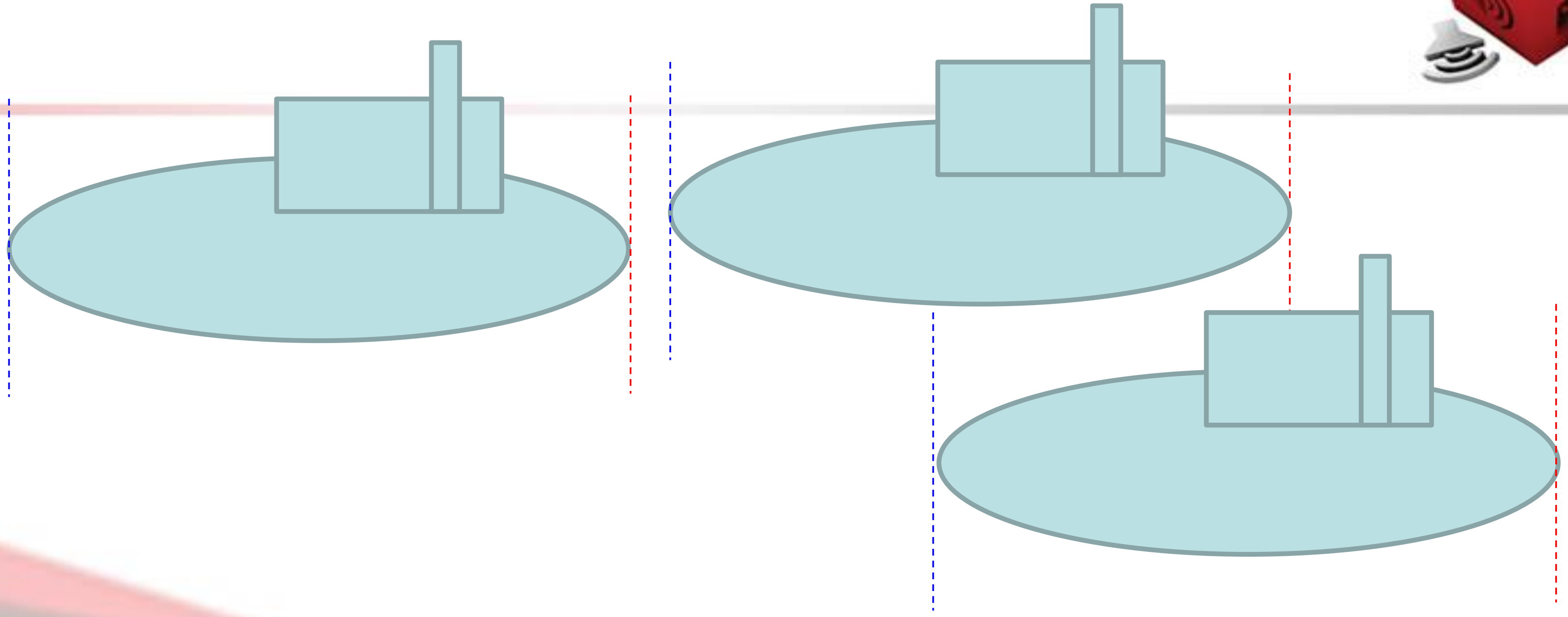
Using game rules

- 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**

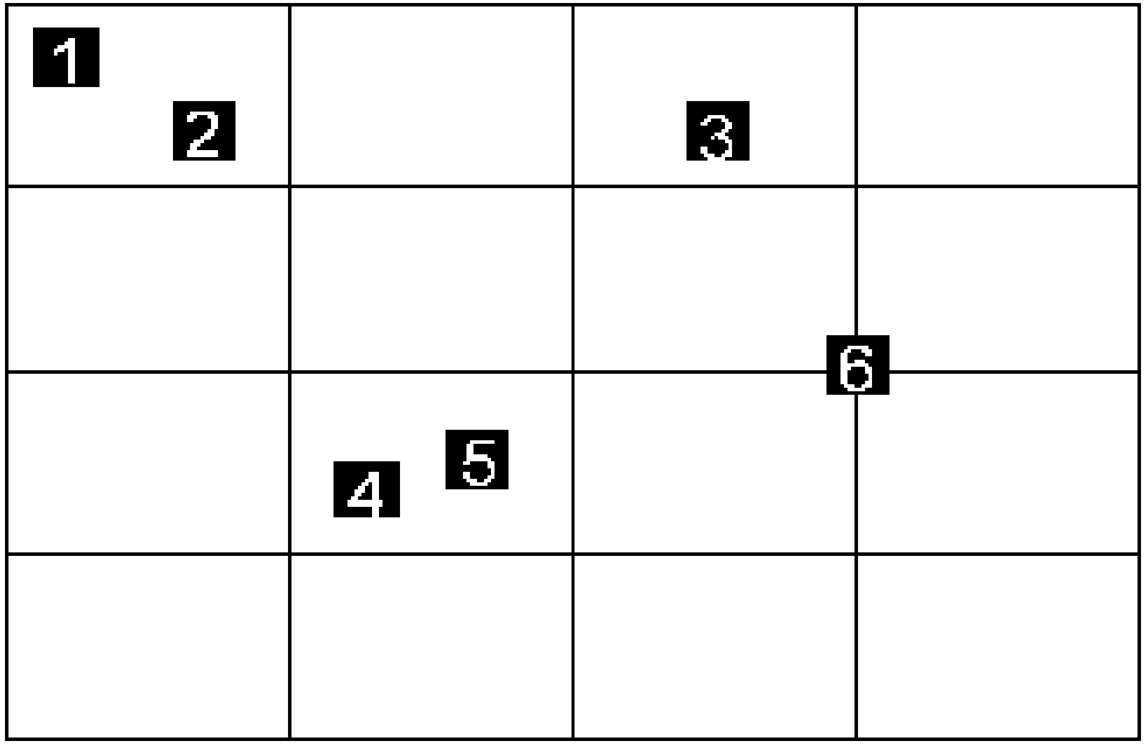
Axis sorting



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

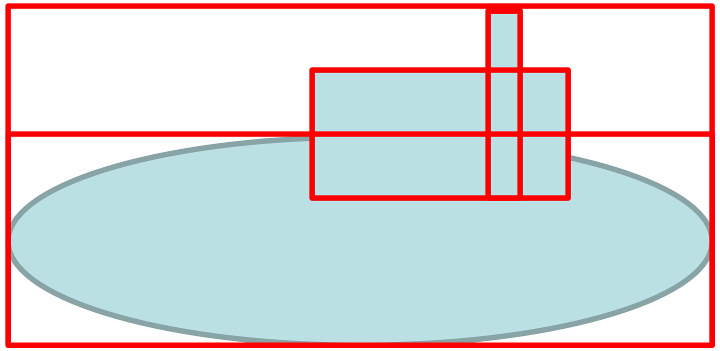
Spatial partitioning

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



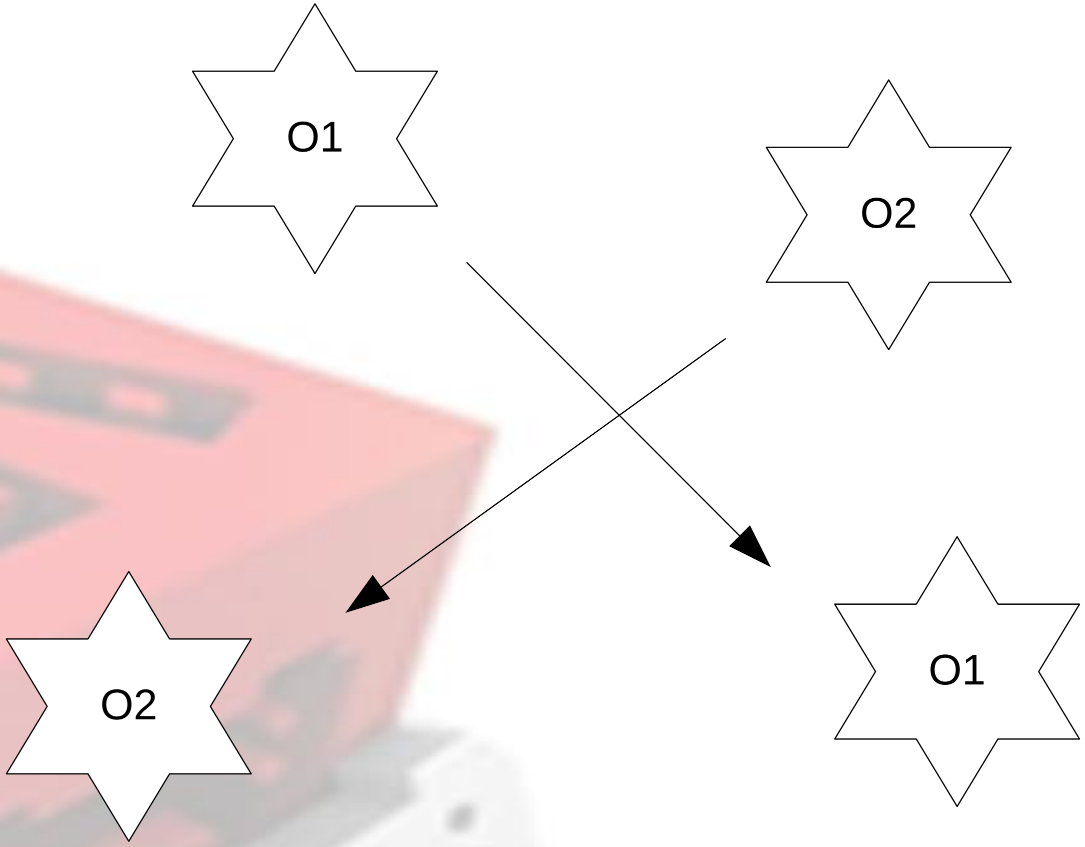
Hierarchical Collision Detection

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



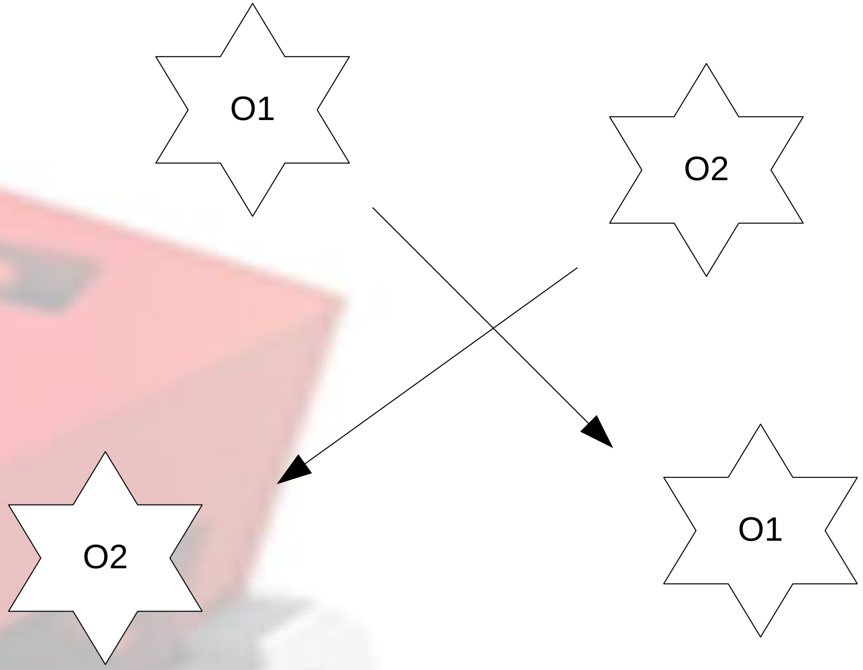
Velocity problems

- Is there a collision or not?



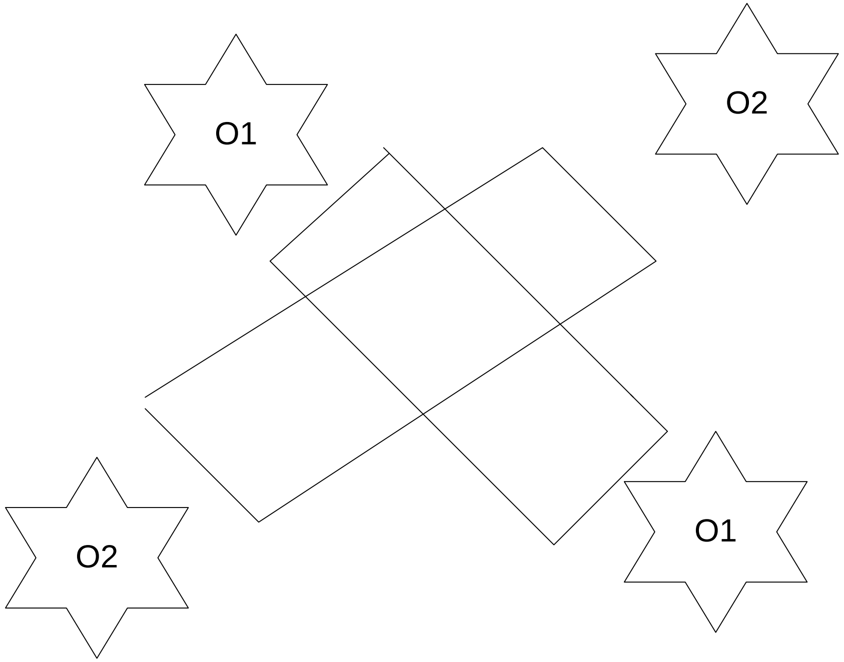
Line segment intersection

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



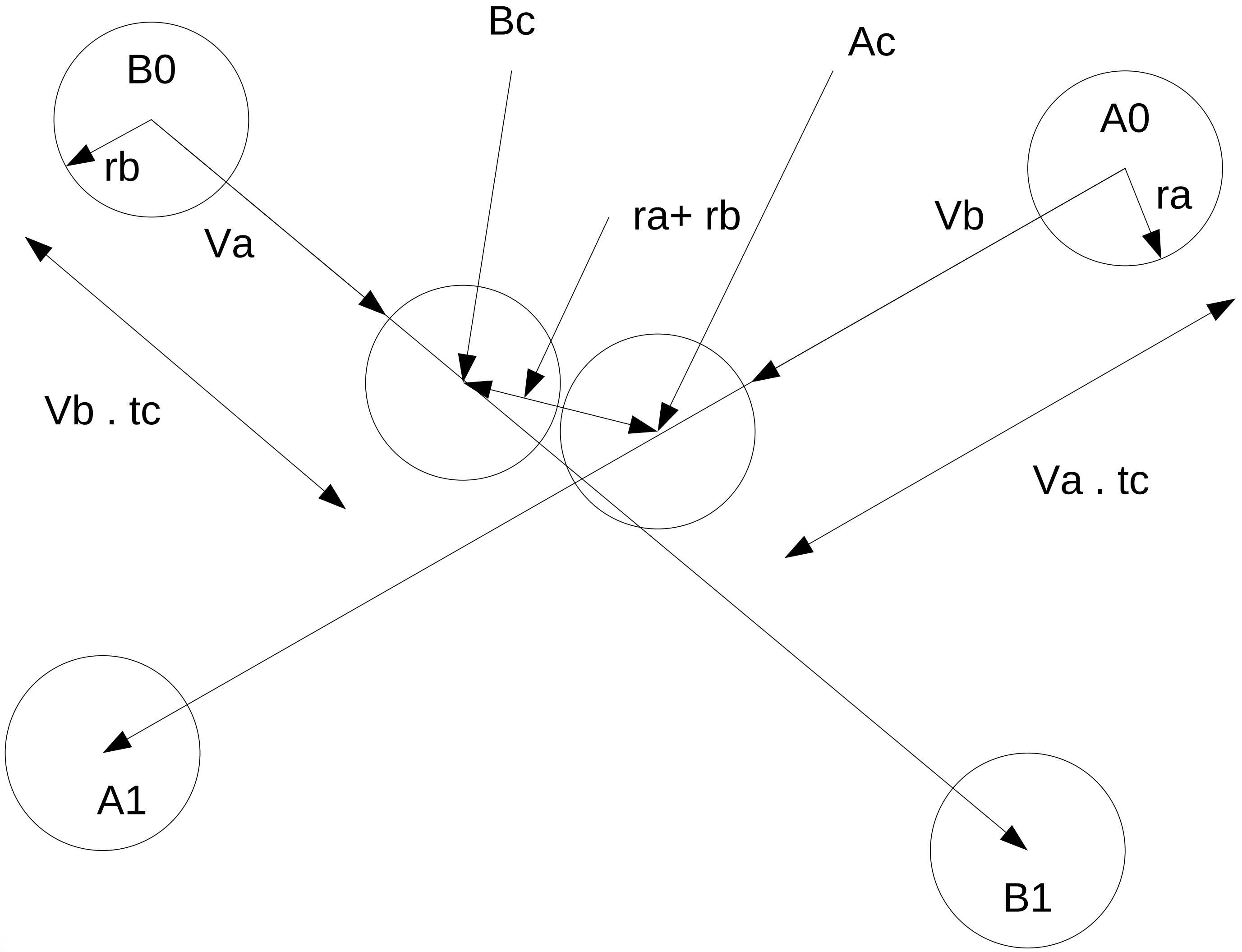
Swept volume tests

- 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

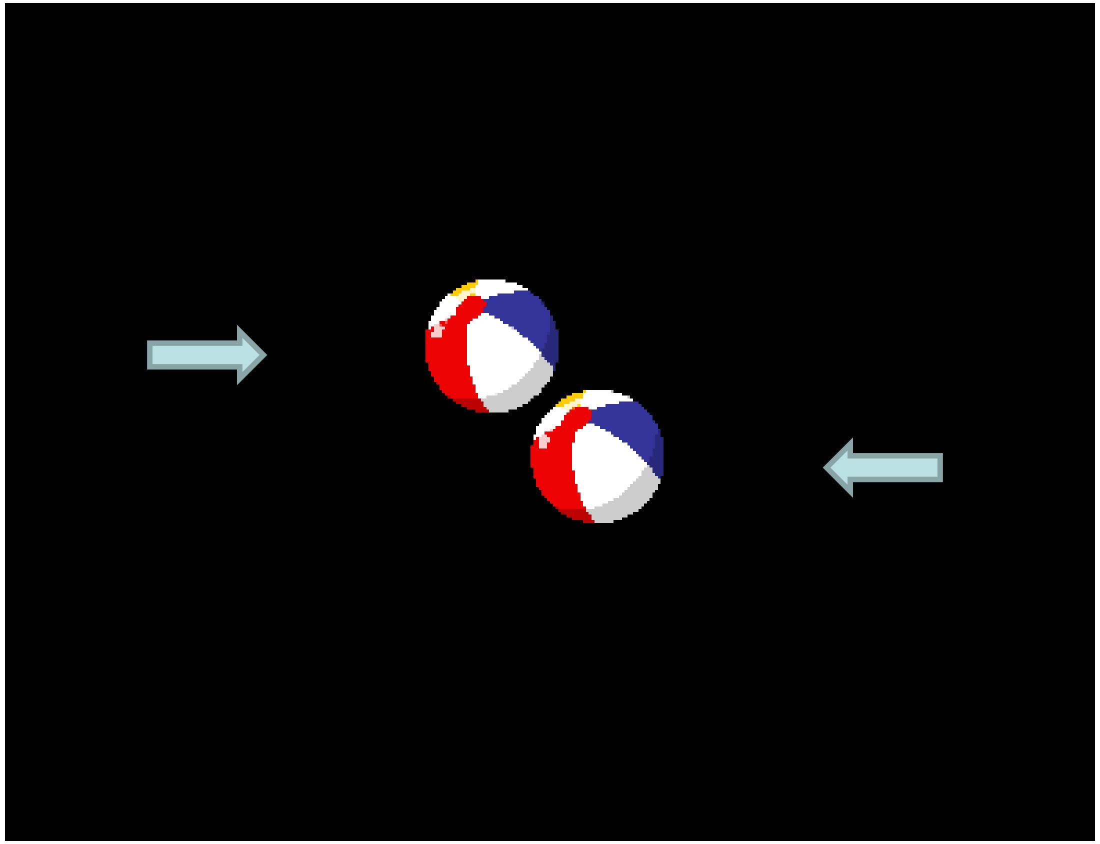
- Physics engine contain collision detection systems
  - Box2D, PhysX, ODR, Havok, Bullet, etc..
- For complex collision detection, best to use an engine rather than implement your own
  - We will be focusing on the major detection algorithms
    - Bounding circle/sphere
    - AABB

Examples

- Bounding sphere
  - – Implement function in game class that compares two Sprites
    - Using the bounding circle/shere calculation
    - Returns true if sprites are colliding
- AABB
  - – Implement another function that compares two Sprites
    - Using the AABB calculation
    - Returns true if sprites are colliding
  - – Sprites need to maintain a bounding box

Sphere bounding example

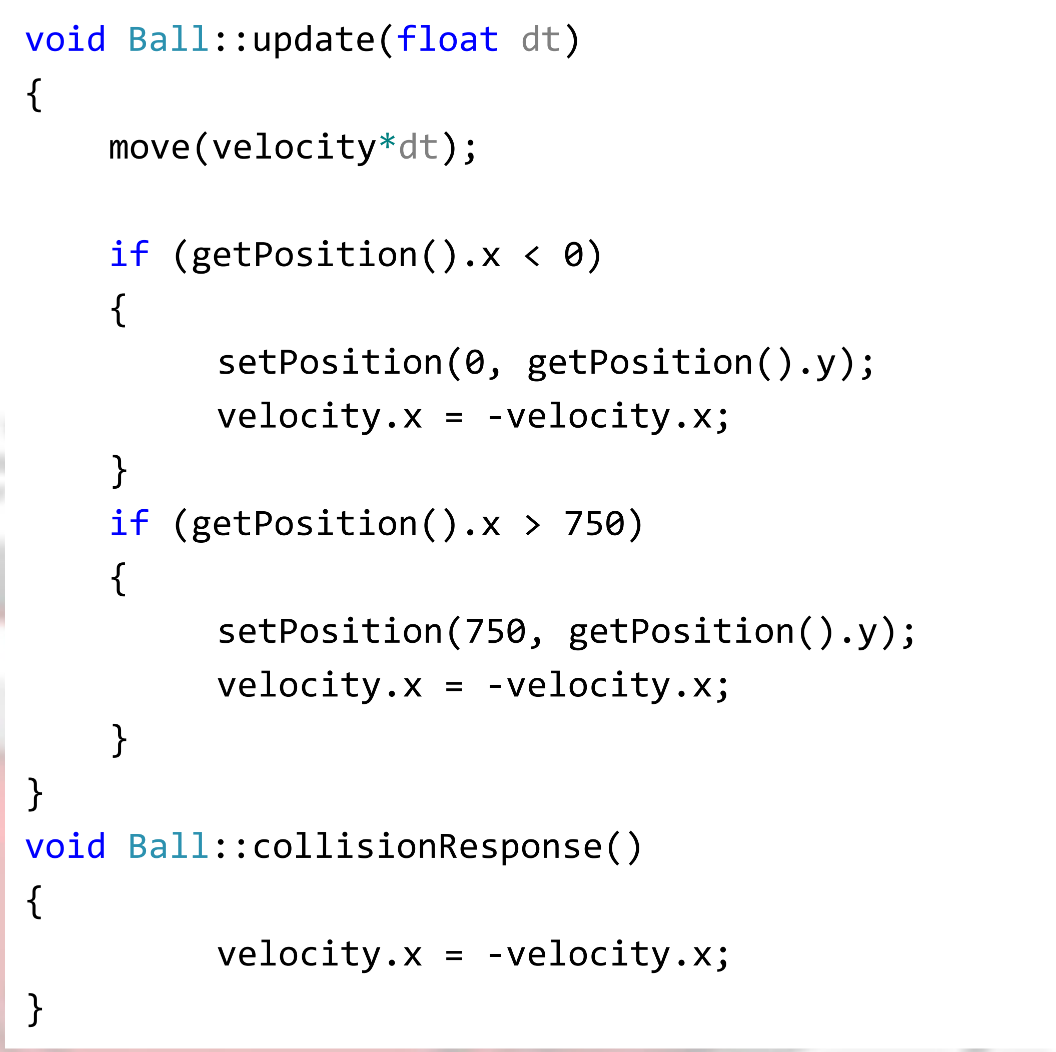
- Two Sprite objects with ball texture
- Moving towards each other
- Every frame
  - Update the ball objects
  - Check for collision
    - If collision, resolve collision
  - Render objects



Game update()

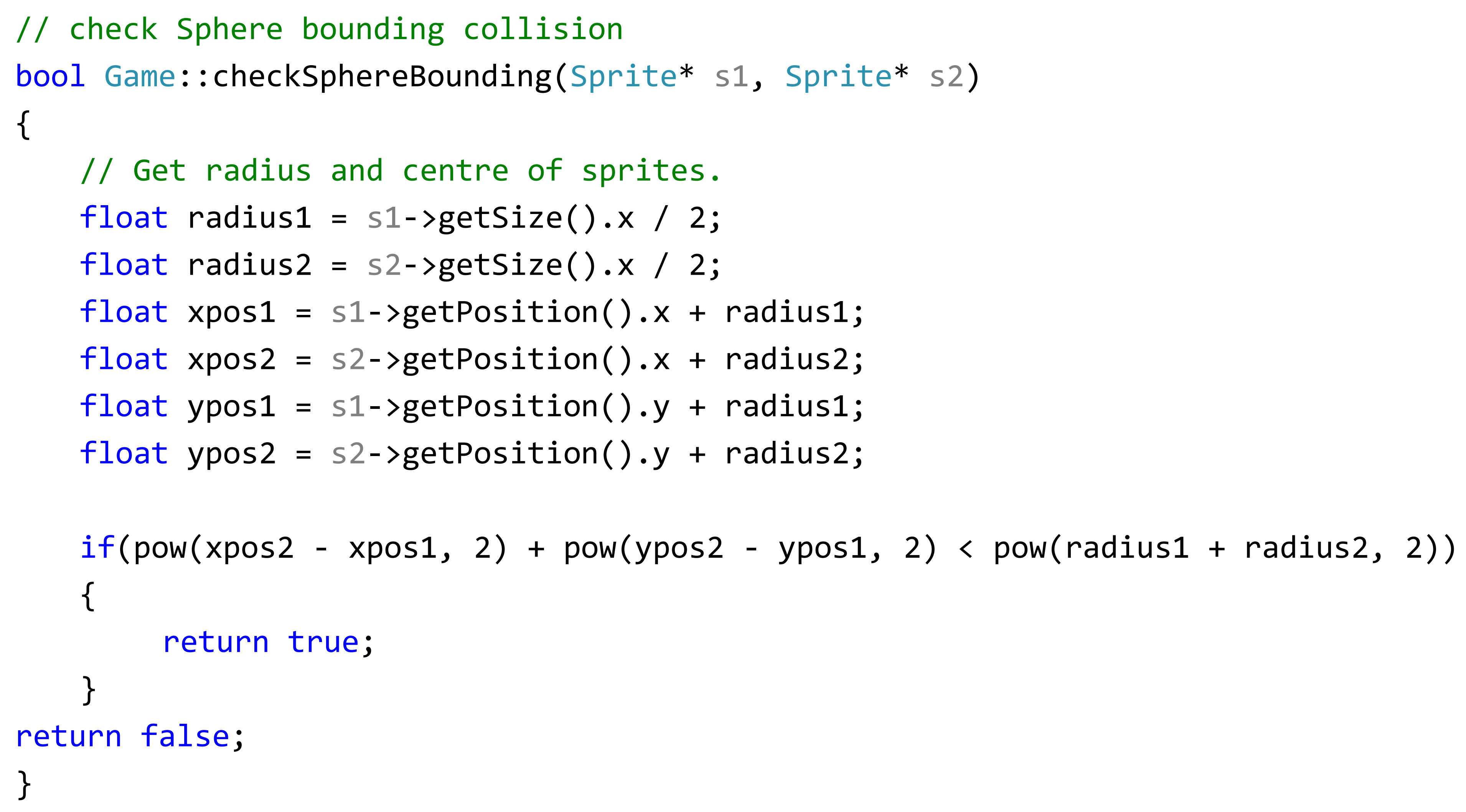
```
ball1.update(dt);
ball2.update(dt);
if (checkSphereBounding(&ball1, &ball2))
{
   ball1.collisionResponse();
   ball2.collisionResponse();
}
```

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()
{
        velocity.x = -velocity.x;
}
```

Bounding circle/sphere detection



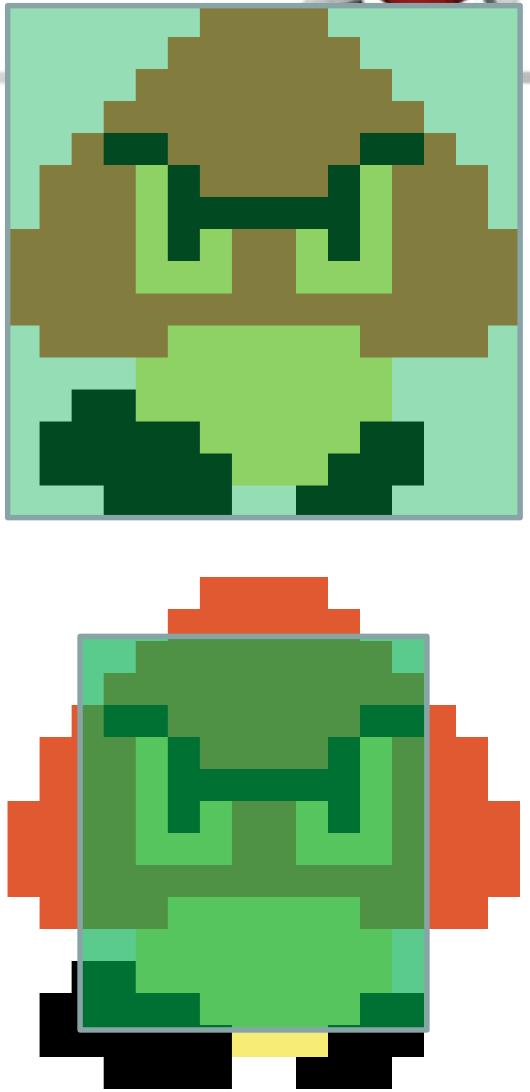
```
// check Sphere bounding collision
bool Game::checkSphereBounding(Sprite* s1, Sprite* s2)
{
    // Get radius and centre of sprites.
    float radius1 = s1->getSize().x / 2;
    float radius2 = s2->getSize().x / 2;
    float xpos1 = s1->getPosition().x + radius1;
    float xpos2 = s2->getPosition().x + 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;
}
```

Live demo

- Bounding sphere example

AABB example

- Further updates to the Sprite class
  - Added a variable to represent our bounding box
    - sf::FloatRect AABB;
- Allows different sized bounding box instead of just sprite size
- Needs to be updated every frame because
  - Sprite movement
  - Change in animation
  - etc

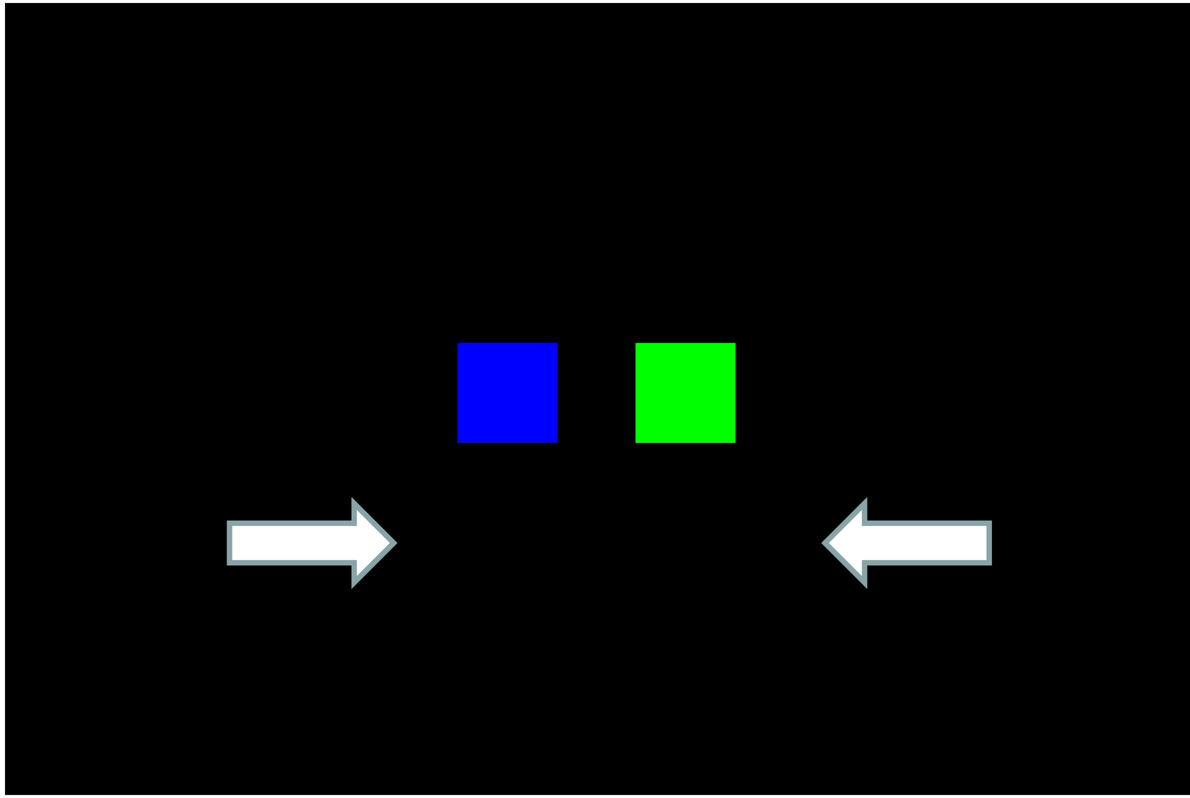


Updating AABB

```
void Sprite::updateAABB()
{
        // Axis Aligned Bounding Box, based on sprite size and position.
        // Shape could be smaller/larger and offset if required.
        // Can be overwritten by child classes
        AABB.left = getPosition().x;
        AABB.top = getPosition().y;
        AABB.width = getSize().x;
        AABB.height = getSize().y;

}
```

AABB Example

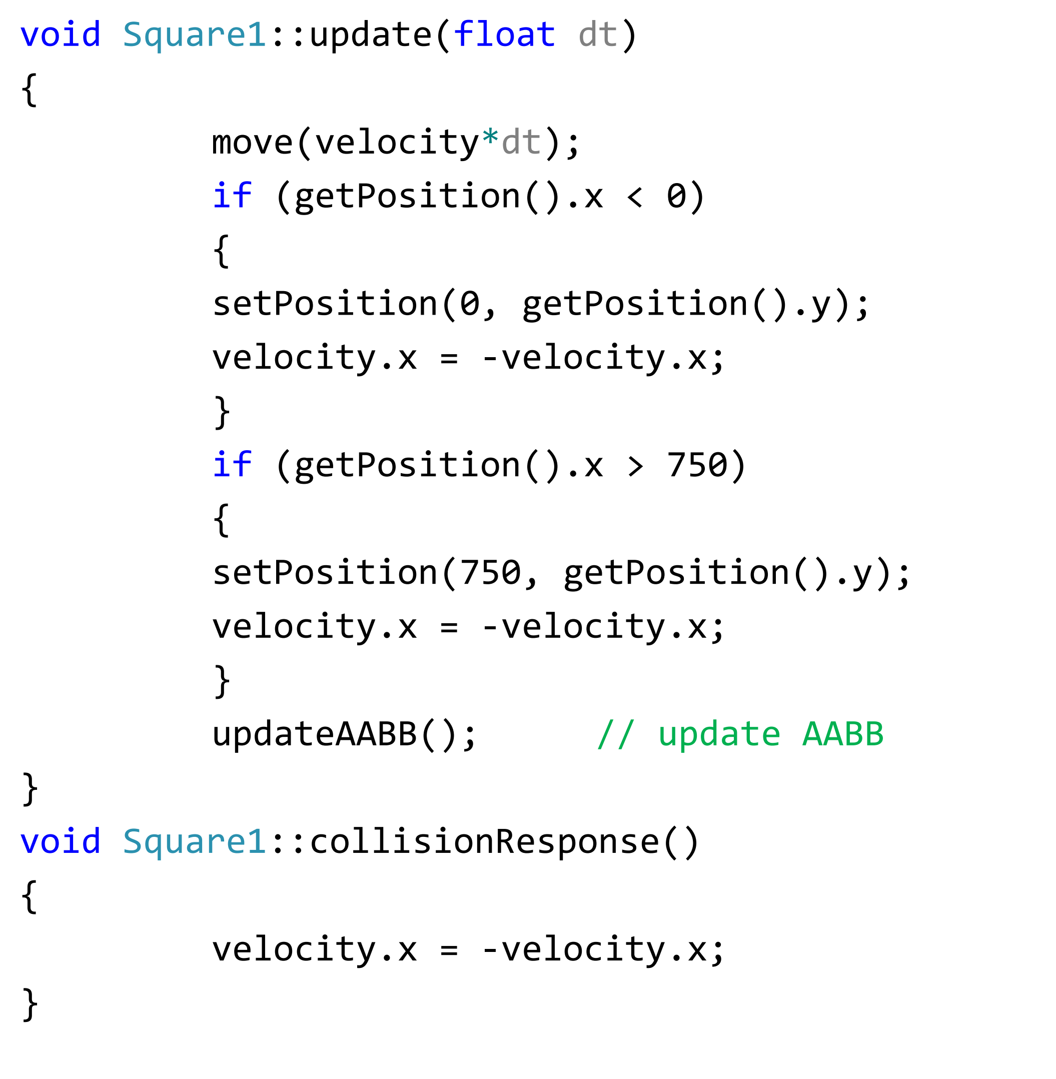
- Two quads moving towards each other
- Every frame
  - Update objects
    - Move
    - Update AABB
  - Check for collision
    - Resolve collision
  - Render



Game.update()

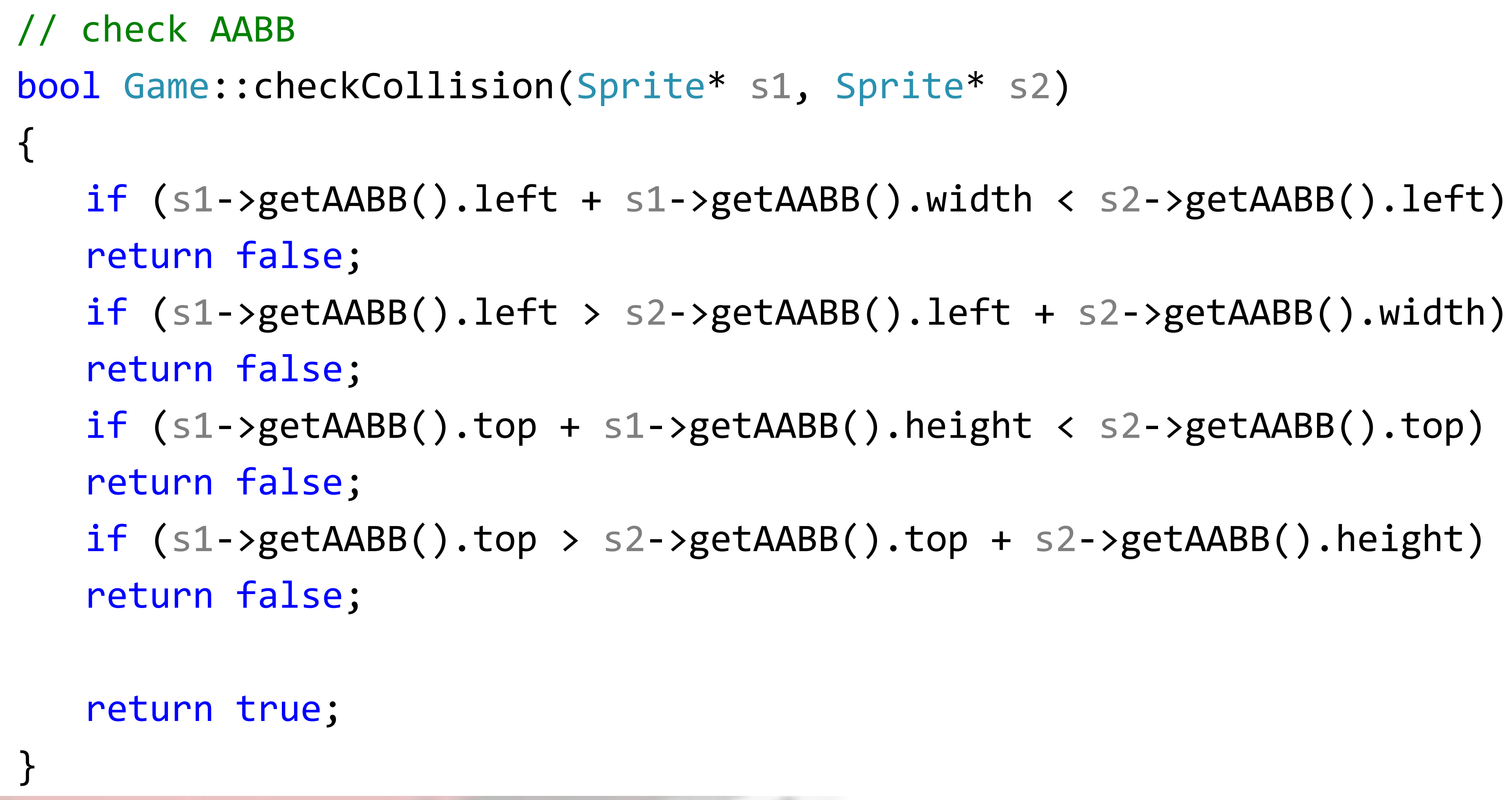
```
square1.update(dt);
square2.update(dt);

if (checkCollision(&square1, &square2))
{
      square1.collisionResponse();
      square2.collisionResponse();
}
```

Square update



```
void Square1::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;
        }
        updateAABB();     // update AABB
}
void Square1::collisionResponse()
{
        velocity.x = -velocity.x;
}
```

AABB Collision detection

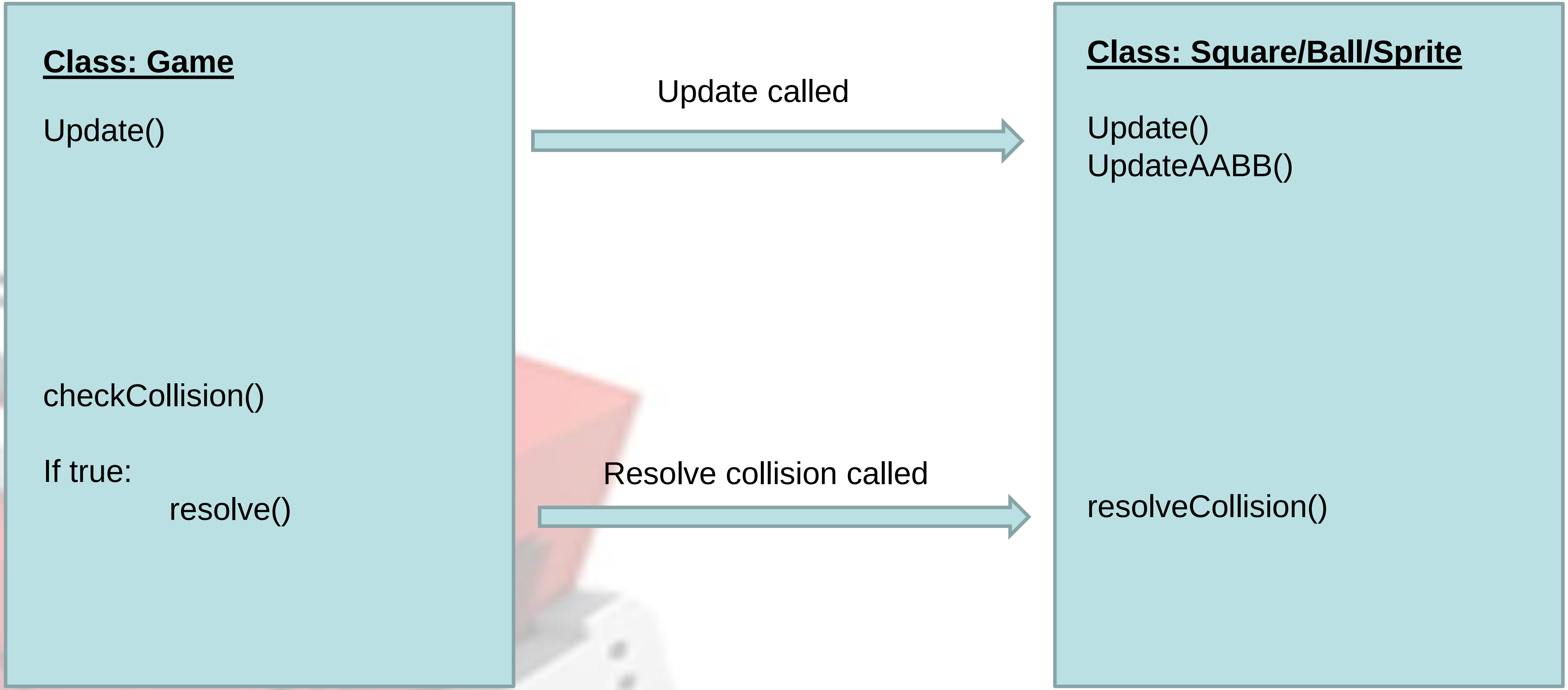


```
// check AABB
bool Game::checkCollision(Sprite* s1, Sprite* s2)
{
    if (s1->getAABB().left + s1->getAABB().width < s2->getAABB().left)
    return false;
    if (s1->getAABB().left > s2->getAABB().left + s2->getAABB().width)
    return false;
    if (s1->getAABB().top + s1->getAABB().height < s2->getAABB().top)
    return false;
    if (s1->getAABB().top > s2->getAABB().top + s2->getAABB().height)
    return false;

    return true;
}
```

Live demo

- AABB example

Dreaded diagram



Class: Game

Update()

checkCollision()

If true:

    resolve()

Update called

Resolve collision called

Class: Square/Ball/Sprite

Update()
UpdateAABB()

resolveCollision()

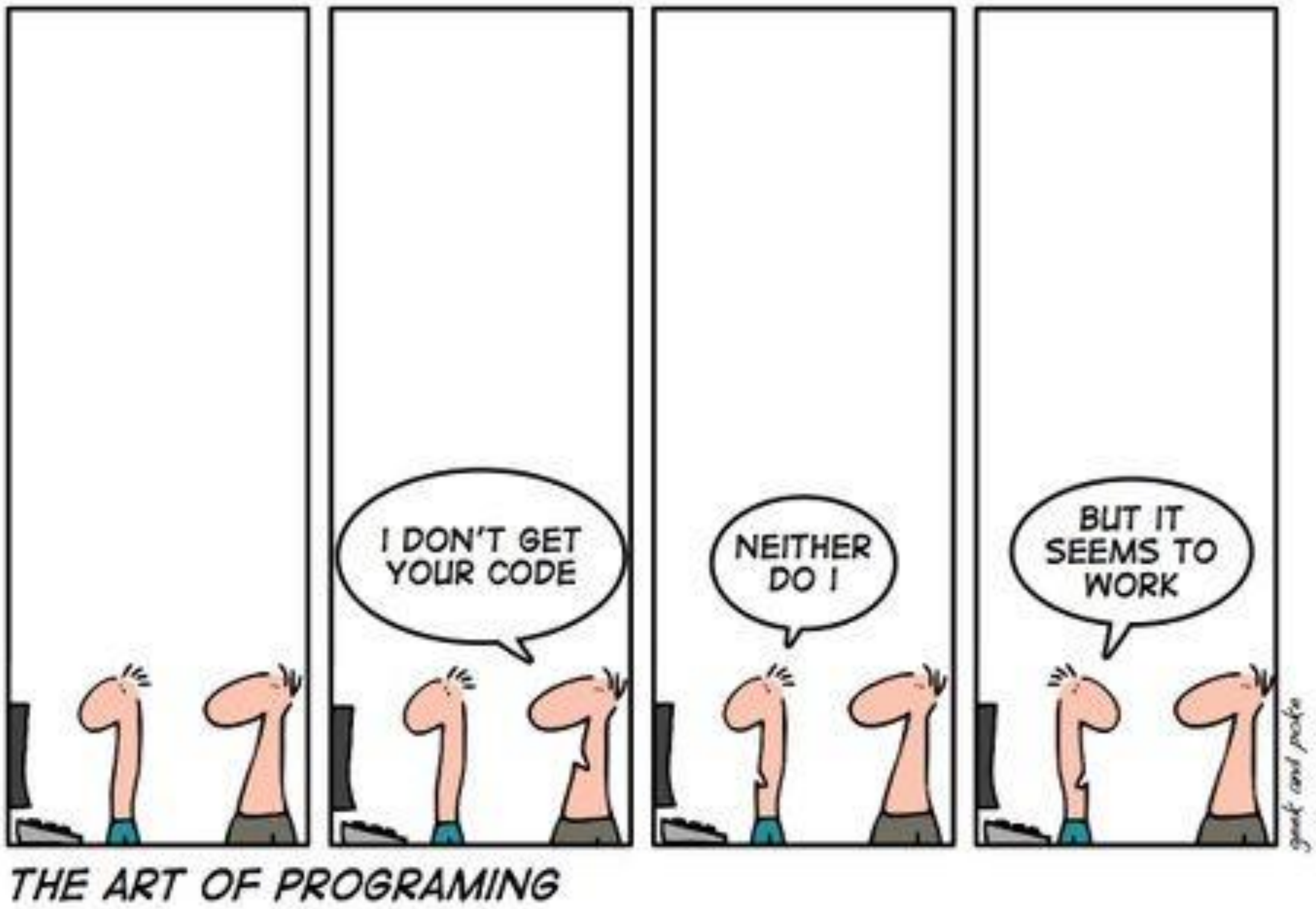
In the labs

- Updating Sprite and Game class to handle collision detection and resolution.
- Making objects that collide.
- Making Pong!
- Thinking about coursework.

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

Previous coursework examples

- New module so no previous examples
  - Yours could be an example for next year
- There have been similar coursework as part of other modules
  - Not as complex
  - Only for half a module
  - Couple video examples



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