Pseudocódigo para hacer la detección de colisiones entre bicho-torre y bicho-bicho y las estructuras de datos para hacerlas.

Estructuras de datos de los bounding boxes

```
class ObjectBoundingBoxes { //Según el enunciado, las torres han de tener una esfera englobante y los bichos han de tener una esfera, y dentro varios AABB.
    static int global_index = 0
    SphereBB sphere
    AABB[] internos
    int index
    Object parent //La torre o bicho al que está asociado

    ObjectBoundingBoxes(Object o) {
            index = global_index
            global_index++
            parent = o
    {

    void actualizarPos(int x,int y,int z) {
            sphere.actualizarPos(x,y,z)
            internos.actualizarPos(x,y,z)
    }
}

class AABB {
    int x_max
    int x_min
    int y_max
    int y_min
    int z_max
    int z_min
    int tam_x
    int tam_y
    int tam_z

    void actualizarPos(int x,int y,int z) {
            x_min = x;
            y_min = y;
            z_min = z;
            x_max = x_min + tam_x;
            y_max = y_min + tam_y;
            z_max = z_min + tam_z;
    }
```

```
}

class SphereBB{
    float radio
    int x
    int y
    int z

    void actualizarPos(int x0,int y0,int z0) {
            x = x0;
            y = y0;
            z = z0;
    }
}
```

Funciones de detección de colision

```
bool colision_sphere(SphereBB a, SphereBB b){
    return sqrt((a.x-b.x)^2 + (a.y-b.y)^2 + (a.z-b.z)^2) < (a.radio + b.radio)
}

bool colision_aabb(AABB a, AABB b){
    if(a.x_max < b.x_min || b.x_max < a.x_min
            || a.y_max < b.y_min || b.y_max < a.y_min
            || a.z_max < b.z_min || b.z_max < a.z_min
    ) {
            return false
    }

    return true
}

bool colision_aabb_sphere(AABB a, SphereBB b){
    if(b.x+b.radio < a.x_min || b.x-b.radio > a.x_max
            b.y+b.radio < a.y_min || b.y-b.radio > a.y_max
            b.z+b.radio < a.z_min || b.z-b.radio > a.z_max
    ) {
            return false
    }

    return true
}

bool colision(ObjectBoundingBoxes a, ObjectBoundingBoxes b) {
```

```
    if(!colision_sphere(a.sphere, b.sphere))
    {
        return false
    }
    else if(a.internos.size == 0 && b.internos.size == 0) { //Si colisionan las esferas y no tienen AABB, según el enunciado nunca pasaría ya que las torres no se mueven pero está por completitud
        return true
    }
    else if(a.internos.size == 0) {
        for(AABB y in b.internos) {
            if(colision_aabb_esfera(y, a.esfera))
            {
                return true
            }
        }
    }
    else if(b.internos.size == 0) {
        for(AABB x in a.internos) {
            if(colision_aabb_sphere(x, b.esfera)) {
                return true
            }
        }
    }
    else { //Ambos tienen AABB además de la esfera
        for(AABB x in a.internos) {
            for(AABB y in b.internos) {
                if(colision_aabb(x,y)) {
                    return true
                }
            }
        }
    }
    return false
}
```

Clases torre/bicho

```
class Torre
    ObjectBoundingBoxes hitbox
    ...
    Torre() {
        ...
        hitbox.sphere = new SphereBB(x,y,z)
```

```
            ...
    }
    ...
}


class Bicho {
    int id
    ObjectBoundingBoxes hitbox;
    ...
    Bicho() {
            ...
            hitbox.sphere = new SphereBB(x,y,z)
            for(.....) {
                    hitbox.internos.add(new AABB(/*Calcular valores para AABB de patas o
cuerpo*/))
            }
            ...
    }
    ...
}
```

Funciones

```
void update() { //Función “global” de actualizar el juego cada frame
    …
    for(Bicho b in mundo) {
        b.actualizarPosicionBoundingBoxes() //Es necesario actualizar la posición de la esfera
y de los AABB si se ha trasladado, y se ha de recalcular los AABB si se ha rotado.
    }

    int i = -1
    for(ObjectBoundingBoxes o in mundo) {
        for(ObjectBoundingBoxes o2 in mundo)
            if(o.index <= o2.index) {
                    continue
            }
            if(colision(o, o2)) {
                    choque(o.parent, o2.parent)
            }
        }
    }
    ...
}
```

```
void choque(Object o, Object o2) {
   if(o.isBicho() && o2.isBicho()) { //Según el GDD, la interacción entre bichos es igual independientemente de si pertenecen a la misma calle o no, el de id menor (es decir, el que se creó primero) tiene prioridad y el de id mayor para
            if(o.id < o2.id) {
                     o2.wait()
            }
            else{
                     o.wait()
            }
   }
   else if(o.isBicho() && o2.isTorre()) {
            o.atacar(o2)
   }
   else if(o.isTorre() && o2.isBicho()) {
            o2.atacar(o)
   }
}
```