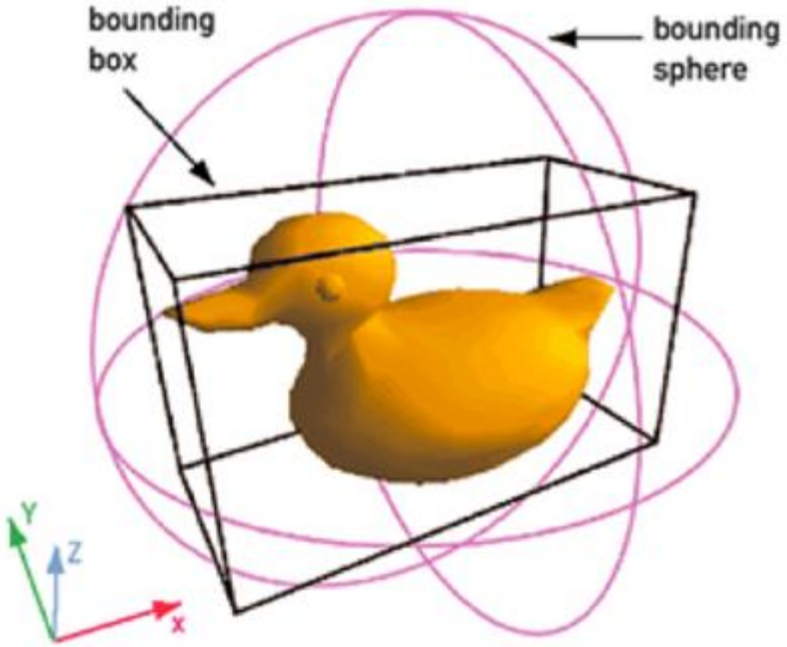
P.PORTO

POLITÉCNICO
DO PORTO
ESMAD

SISTEMAS GRÁFICOS
TSIW

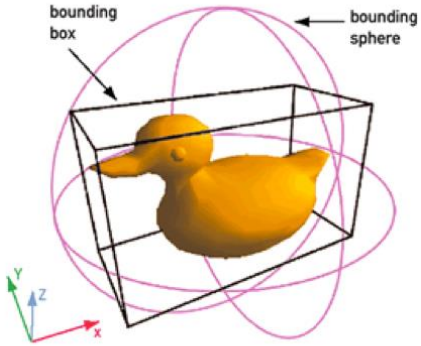
# Three.js - Collisions

- Three.js does not provide a system for collision detection or colliders
- Two options:
  - implement collision detection with some math and coarse bounding volumes like THREE.Sphere or THREE.Box3
  - integrate a physics engine
- For most apps, a real physics engine is an overkill
  - Check the code for this example, where in the render function it is ensured the balls are kept inside the room and collide against each other using simple math logic



# Three.js - Collisions

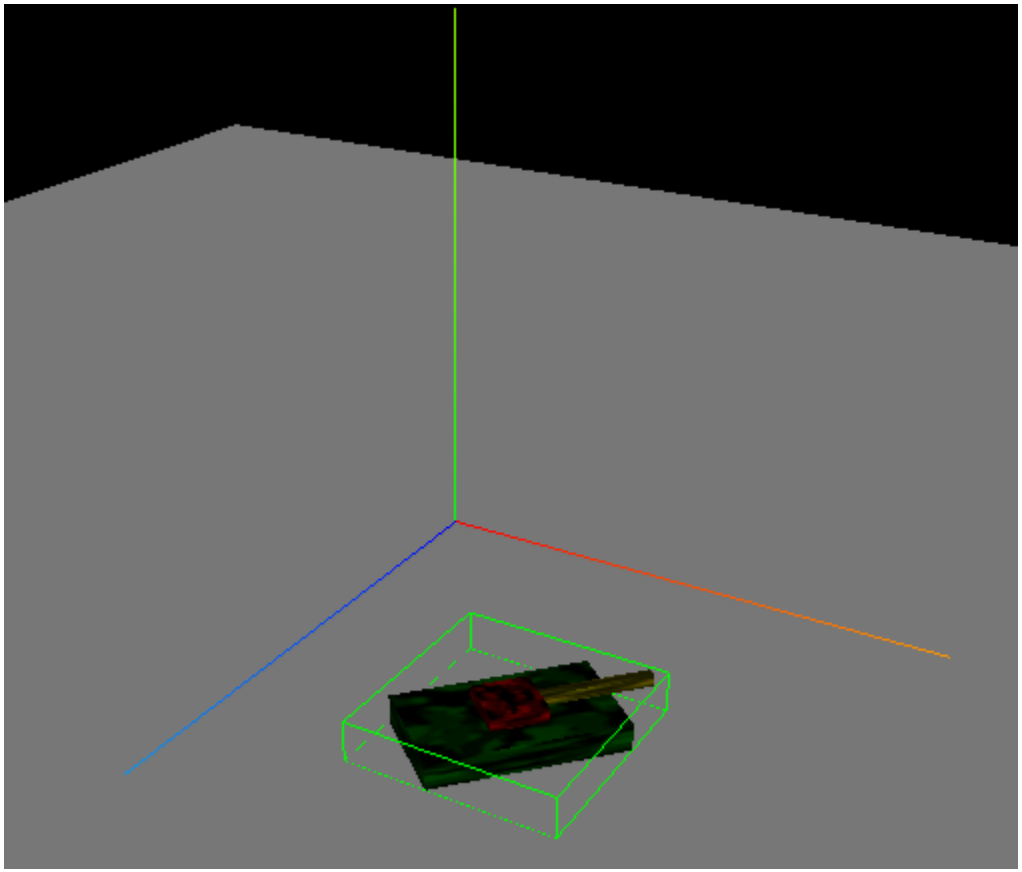
- Fastest way to detect collisions between objects - ***Bounding Volumes***
  - Most common volumes: boxes and spheres




- Other techniques: using the **raycasting** technique
  - https://stemkoski.github.io/Three.js/Collision-Detection.html
  - Sets a ray using the moving object position and a direction (in the example, determines a ray passing by each of the moving cube vertices)
  - Checks if they intersect any mesh in the array of target meshes (for increased collision accuracy, one can add more vertices to the cube)
  - HOWEVER: when the origin of the ray is within the target mesh, collisions do not occur

# Three.js - Collisions

- **Three.js bounding volumes**: the box definition surrounding an object is of type AABB - ***Axis-Aligned*** *Bounding Box*



# Three.js - Collisions

- Three.js:

  1. **Build the geometry *Bounding Volume*:**

  *mesh.geometry*.computeBoundingBox() → computes a THREE.Box3, the object's Minimum Axis-Aligned Bounding Box
  *mesh.geometry*.computeBoundingSphere() → computes a THREE.Sphere, the object's Minimum Bounding Sphere

  **BE AWARE**: those methods take as reference the geometry (not the mesh and its transformations), so they **ignore** any transformation that is applied to the mesh!

  So, you must use:

```
let BBox = new THREE.Box3().setFromObject(mesh);
```

  (the Bounding Sphere doesn't have the same method)

```
let BSphere = new THREE.Sphere().setFromPoints(mesh.vertices);
Bsphere.applyMatrix4(mesh.matrixWorld);
```

# Three.js - Collisions

- Three.js:
  2. Intersection methods - return **true** or **false** whether or not the given volume intersects another given geometry:
     *boundingVolume*.**containsPoint**(*point*)
     *boundingVolume*.**intersectsBox**(*box*)
     *boundingVolume*.**intersectsSphere**(*sphere*)
     *boundingVolume*.**intersectsPlane**(*plane*)
  3. If an object is in motion, it is necessary to determine the updated bounding volume, detect collisions (intersections) and only change the position if there is no intersection

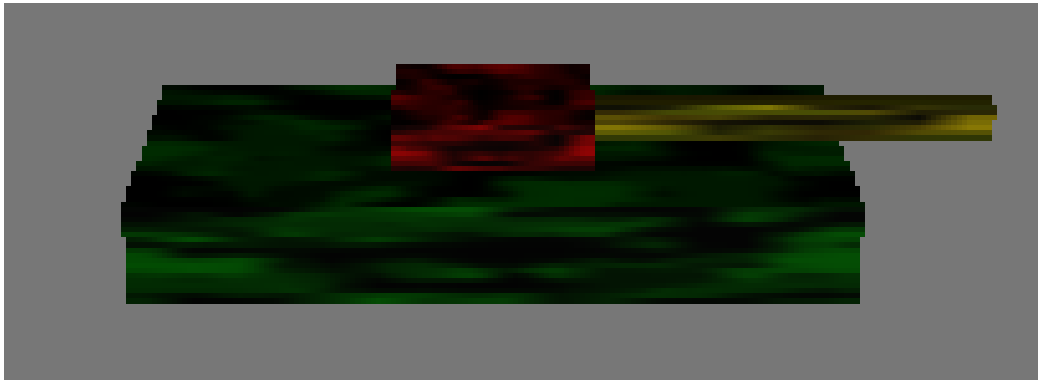
# Three.js - Collisions

- Three.js:

```
// object definition
…
// helper object to show the world-axis-aligned bounding box around an object
mesh.geometry.computeBoundingBox();
bbHelper = new THREE.BoxHelper(mesh, 0x00FFFF);
scene.add(bbHelper); // adds AABB to the scene
…
// animation function
…
bbHelper.update(mesh); // updates helper object
…
let BBox = new THREE.Box3().setFromObject(mesh);
let BBox2 = new THREE.Box3().setFromObject(othermesh);
let collision = BBox.intersectsBox(BBox2); // checks collision between mesh and othermesh
…
```

# Exercise - Collisions

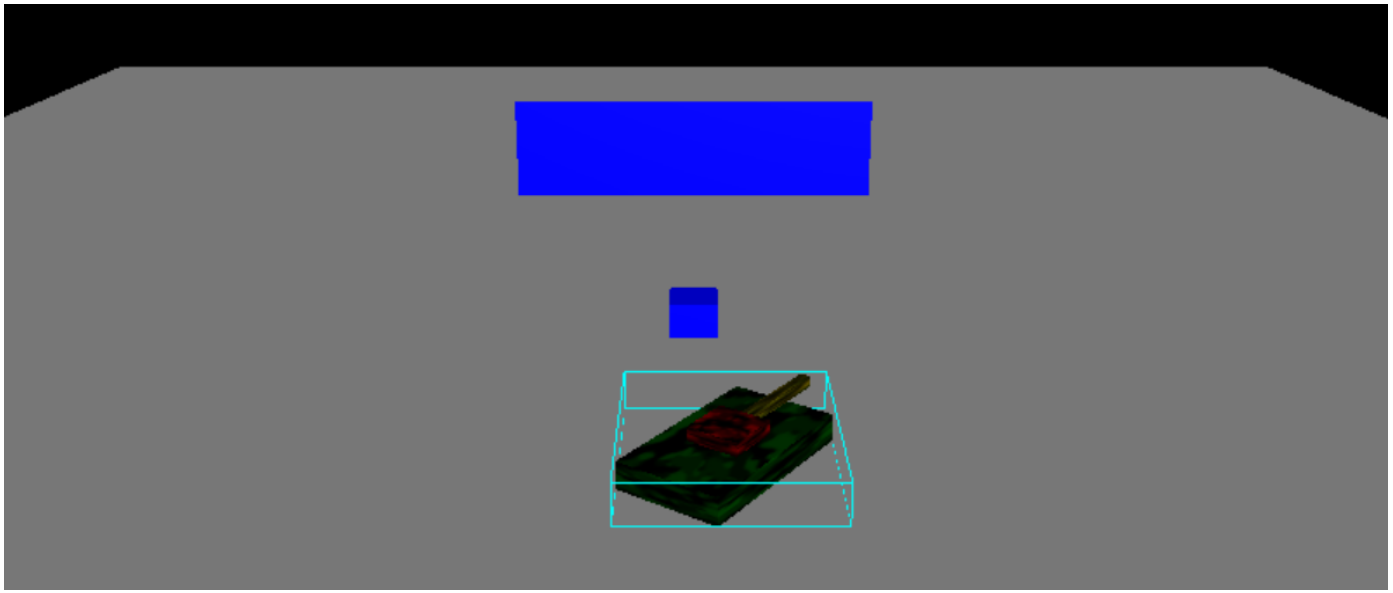
- Remember the TANK exercise?



Keys W and S: tank moving forward or backward, respectively (constant velocity of 0.1 units per frame)
Keys D and A: tank moving right or left (increase or decrease base rotation by 0.01 radians per frame)
Keys Z and X: rotate tower / Keys V and B: rotate cannon
Key C: shift between static camera and “third person view” (offset to the base center: Y=5, Z=-15)

# Exercise - Collisions

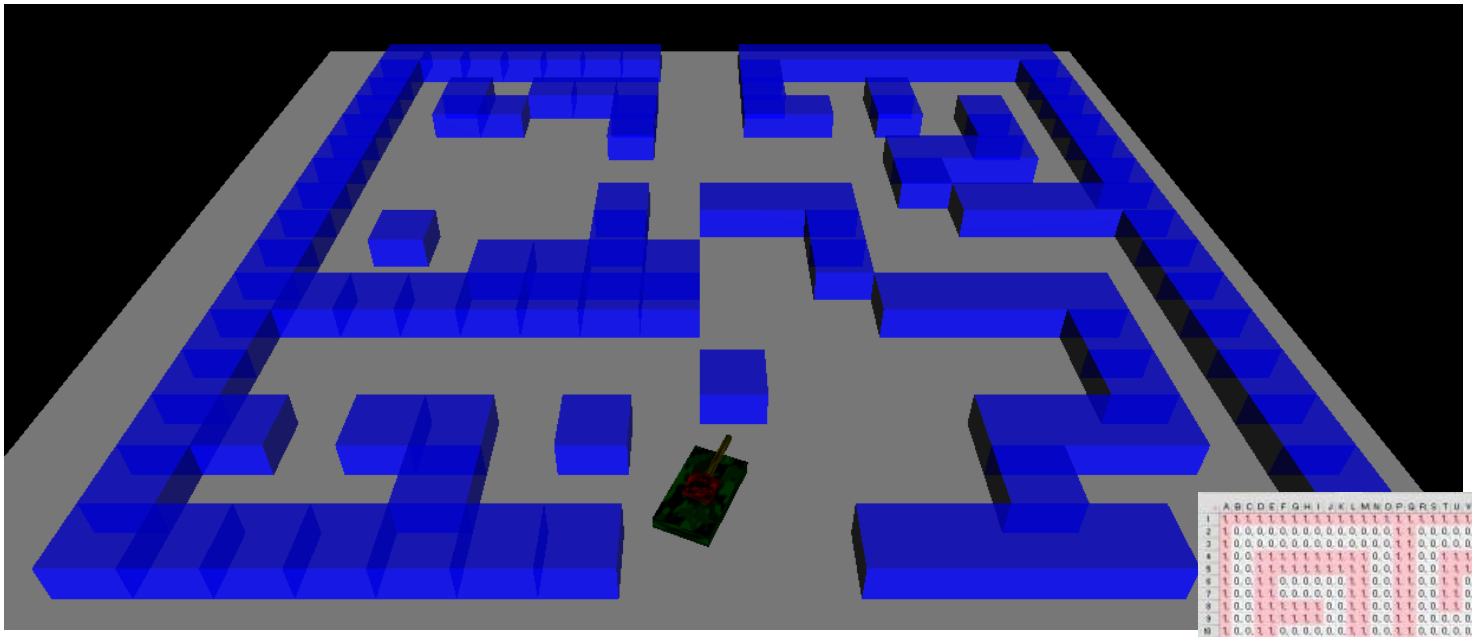
- Add some obstacles (blue rectangles) to the scene
- Compute the tank bounding volume and visualize it using a BoxHelper

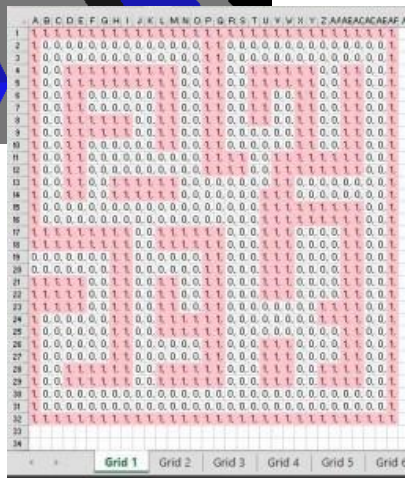


- Move the tank only if there is no collision between it and any of the obstacles

# Exercise - Collisions

- Create a maze and navigate the tank through it

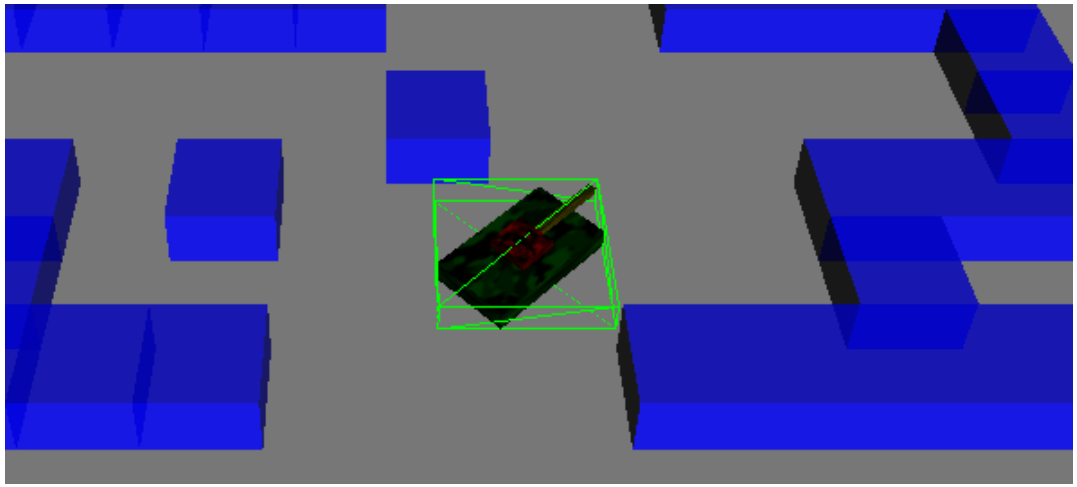




  - HINT: build the labyrinth using a matrix
    MAZE = [“*** ***”, “* * *”,…] where the “*” represent the maze cubes (use Excel)

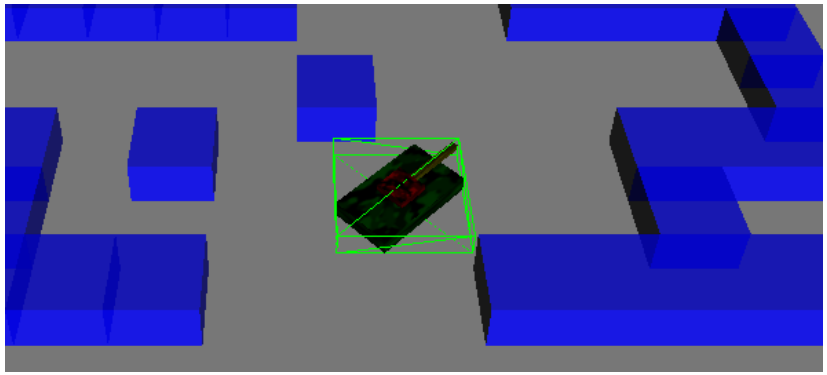
# Exercise - Collisions

- You may find thate the *Bounding Box* may not be the ideal solution for all cases!



  - HINT: determine the collision using the tank **vertices** and use function `containsPoint(`*`point`*`)` to determine if any vertex point collides with one of the labyrinth cubes

# Exercise - Collisions

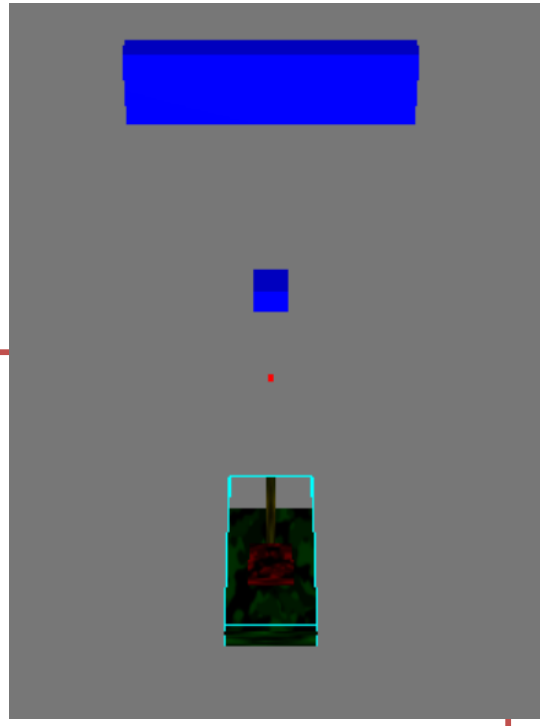


- To compute the correct world position of hierarchichal objects, you must traverse all its children and apply the mesh's global transform

```
object.traverse( function(child) {
        if (child instanceof THREE.Mesh) {
                child.updateMatrixWorld(); //update the global transform of the object
                let len = child.geometry.vertices.length;
                for (let i = 0; i < len; i++) {
                        let v = child.geometry.vertices[i].clone();
                        //gets the mesh vertex in world coordinates
                        v.applyMatrix4(child.matrixWorld);
                }
        }
});
```

# Exercise - Collisions

- Let's shot some bullets! Key "H": create a bullet in front of the tank's cannon and make it move forward



```
//create bullet (sphere geometry)
…
//bullet position
sphere.position = meshCannon.position.clone();
sphere.position.applyMatrix4(meshCannon.matrixWorld);
//bullet direction (get from rotation of world matrix
and normal vector of face #8 of cannon geometry)
let normalMatrix = new THREE.Matrix4().extractRotation(meshCannon.matrixWorld);
let normal = meshCannon.geometry.faces[8].normal;
bulletDirection = normal.clone(). applyMatrix4(normalMatrix);
…
//update bullet position (make it move): variable inc value increases each frame
let n = bulletDirection.clone();
sphere.position.addVectors(sphere.position, n.multiplyScalar(inc));
inc++;
```