Group 5 Presentation 2

Alex, Matt, Terry and Dylan
Week Two

Which Brings Us to Raycasters...

.set (origin : Vector3, direction : Vector3) : null

origin — The origin vector where the ray casts from.
direction — The normalized direction vector that gives direction to the ray.

Updates the ray with a new origin and direction.

.setFromCamera (coords : Vector2, camera : Camera) : null

coords — 2D coordinates of the mouse, in normalized device coordinates (NDC)---X and Y components should be between -1 and 1.
camera — camera from which the ray should originate

Updates the ray with a new origin and direction.

```
[ { distance, point, face, faceIndex, object }, ... ]
```

distance – distance between the origin of the ray and the intersection
point – point of intersection, in world coordinates
face – intersected face
faceIndex – index of the intersected face
object – the intersected object
uv - U,V coordinates at point of intersection
uv2 - Second set of U,V coordinates at point of intersection
instanceId – The index number of the instance where the ray intersects

.intersectObjects (objects : Array, recursive : Boolean, optionalTarget : Array) : Array

objects — The objects to check for intersection with the ray.
recursive — If true, it also checks all descendants of the objects. Otherwise it only checks intersection with the objects. Default is false.
optionalTarget — (optional) target to set the result. Otherwise a new Array is instantiated. If set, you must clear this array prior to each call (i.e., array.length = 0;).

The Array of Intersections Gives Us a Distance!

So it’s just a simple matter of not letting the player move if there is an object too close to them (so they can’t move through it).

```
for(i = 0; i < collisionsB.length; i++){
    if(collisionsB[i].distance < 3.3){
        moveBackward = false;
    }
}
```

The array approach may not be strictly necessary for collision, but...

Monster Freezing

We DO need to loop through every object for monster freezing: (Or use intersectObject method)

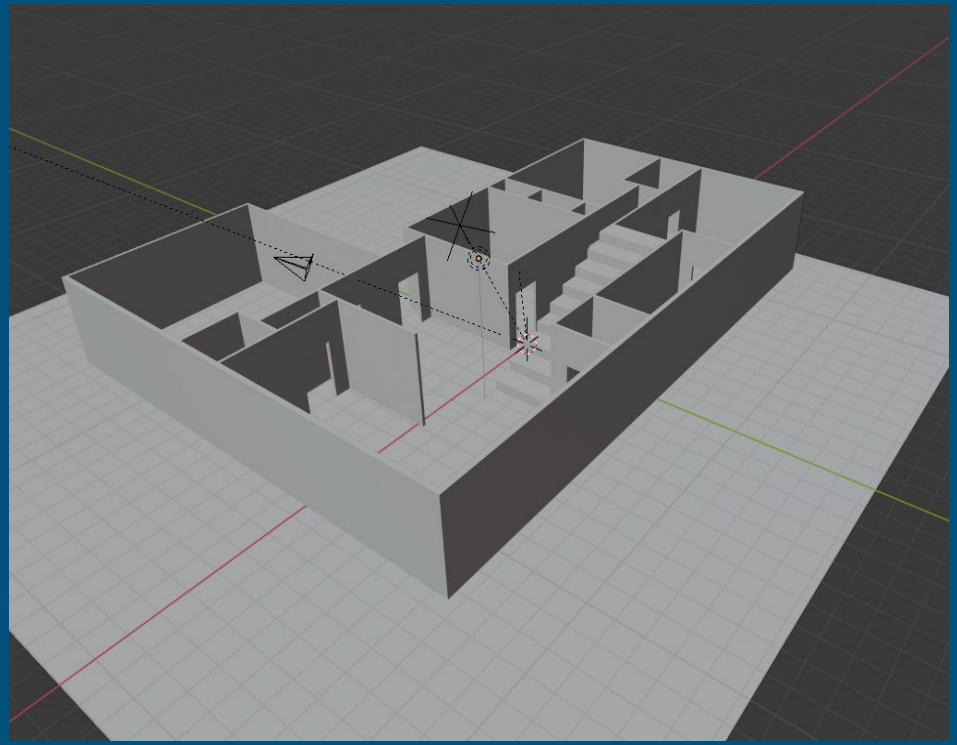
```
for(i = 0; i < collisions.length; i++){
    if(collisions[i].distance < 3.3){
        moveForward = false;
    }
    if(collisions[i].object == cone){
        seeCone = true;
    }
}
```

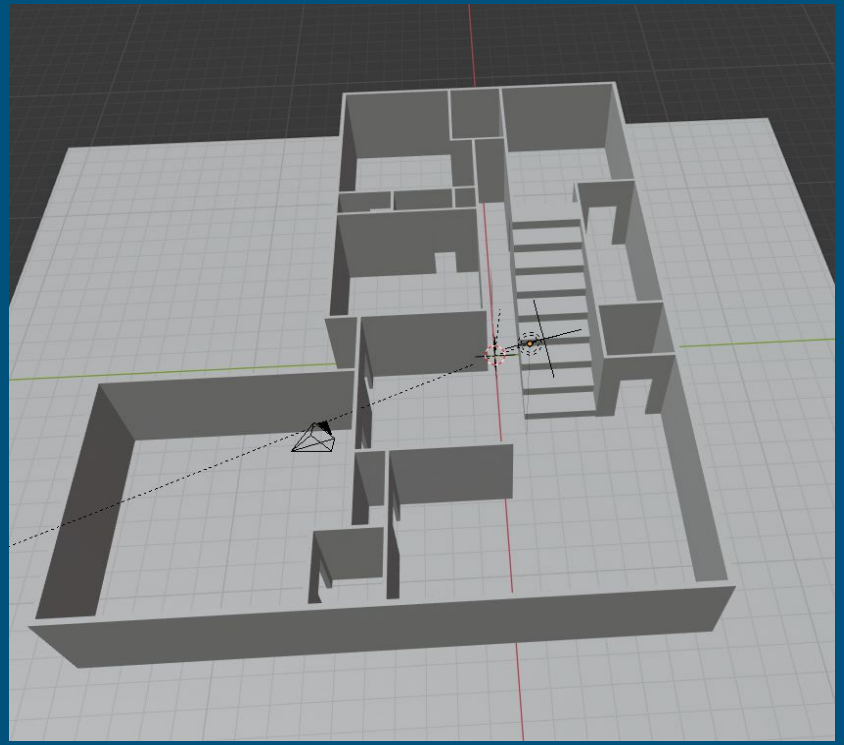
I’m a 1st Level Raycaster

- We need several raycasters to make this work, and if you’re a bit rusty on linear algebra, it can take some trial and error to get them set properly.
- console.log(vector) was useful, but we wanted it on a timer…
- Note that setInterval takes the *name* of a function as a parameter, not a function call!

```
function logDirection(){
    console.log(direction);
    console.log(directionL);
}
setInterval(logDirection, 1000);
```

Modeling





Global Audio

- .ogg
- AudioListener - listens for audio effects in scene

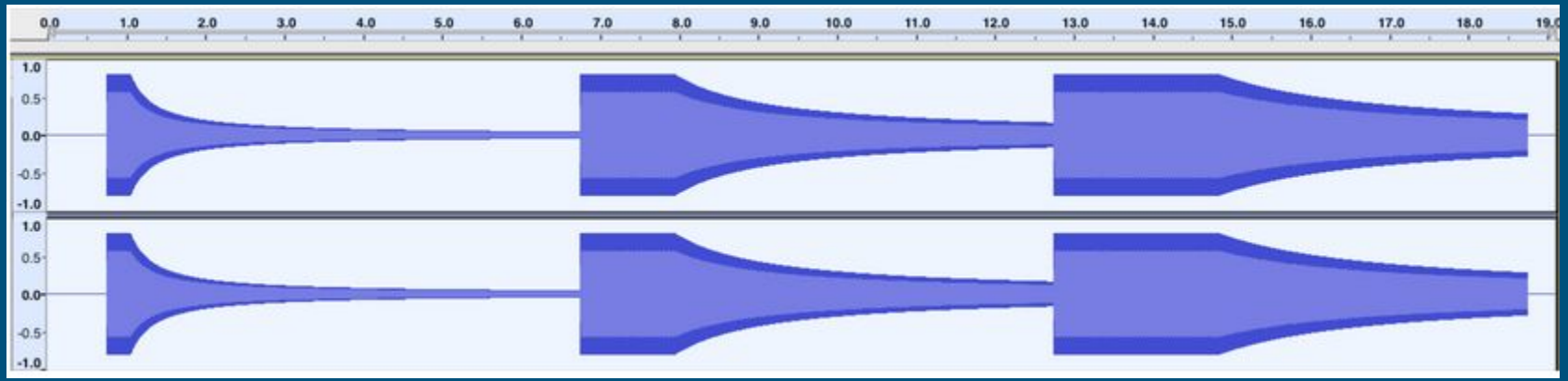
```
//WORKING ON SOUND HERE
var listener = new THREE.AudioListener();
camera.add(listener);
var audioLoader = new THREE.AudioLoader();

//AMBIENT SOUND
var ambientFloor = new THREE.Audio(listener);
audioLoader.load('js/Sounds/floor_boards.ogg', function (buffer) {
    ambientFloor.setBuffer(buffer);
    ambientFloor.setLoop(true);
    ambientFloor.setVolume(.5);
    ambientFloor.play();
});
```

Positional Audio

- RefDistance - represents distance for reducing volume based on distance of audio source

```
//CHASE SOUND
var doom1 = new THREE.PositionalAudio(listener);
audioLoader.load('js/Sounds/rip_and_tear.ogg', function (buffer) {
    doom1.setBuffer(buffer);
    doom1.setRefDistance(4);
    doom1.play();
});
cone.add(doom1);
```



Audio Orientation

- Directional Sound - transforms omnidirectional sound into directional sound

```
.setDirectionalCone( 180, 230, 0.1 );
```

- PositionalAudioHelper

  https://threejs.org/docs/#examples/en/helpers/PositionalAudioHelper

What Next

- Getting the object to follow the player.
- Working out kinks of positional audio function.
- Continue creation of the house model.
- Animation, Model Rigging.