3 Game

1. Valorous Rabbit

HTML

```
<div id="world" />
<div id="gameoverInstructions">
  Game Over
</div>
<div id="dist">
    <div class="label">distance</div>
    <div id="distValue">000</div>
</div>

<div id="instructions">Click to jump<span class="lightInstructions"> — Grab the carrots /
avoid the hedgehogs</span></div>


<div id="credits">
  <p><a href="https://codepen.io/Yakudoo/" target="blank">other codepens</a> | <a
href="https://www.epic.net" target="blank">epic.net</a></p>
</div>
```

CSS

```
@import url('https://fonts.googleapis.com/css?family=Voltaire');


#world{
  position: absolute;
  width:100%;
  height: 100%;
  background-color: #dbe6e6;
  overflow: hidden;
}


#gameoverInstructions{
  position:absolute;
  font-family:'Voltaire', sans-serif;
  font-weight:bold;
  text-transform: uppercase;
  font-size:120px;
  text-align:center;
  color:#ffc5a2;
  opacity:0;
  left:50%;
  top:50%;
  width:100%;
  transform : translate(-50%,-100%);
  user-select: none;
  transition: all 500ms ease-in-out;

  &.show{
    opacity:1;
    transform : translate(-50%,-50%);
    transition: all 500ms ease-in-out;
  };
}


#dist{
  position:absolute;
  left:50%;
  top:50px;
  transform:translate(-50%,0%);
  user-select: none;
}

.label{
  position:relative;
  font-family:'Voltaire', sans-serif;
  text-transform:uppercase;
  color:#ffa873;//100707;
  font-size:12px;
  letter-spacing:2px;
  text-align:center;
  margin-bottom:5px;
}


#distValue{
  position:relative;
  text-transform:uppercase;
  color:#dc5f45;//dc5f45;
  font-size:40px;
  font-family:'Voltaire';
  text-align:center;
}

#credits{
  position:absolute;
  width:100%;
  margin: auto;
  bottom:0;
  margin-bottom:20px;
  font-family:'Voltaire', sans-serif;
  color:#544027;
  font-size:12px;
  letter-spacing:0.5px;
```

```css
  text-transform: uppercase;
  text-align : center;
  user-select: none;
}
#credits a {
  color:#544027;

}

#credits a:hover {
  color:#dc5f45;
}

#instructions{
  position:absolute;
  width:100%;
  bottom:0;
  margin: auto;
  margin-bottom:50px;
  font-family:'Voltaire', sans-serif;
  color:#dc5f45;
  font-size:16px;
  letter-spacing:1px;
  text-transform: uppercase;
  text-align : center;
  user-select: none;
}
.lightInstructions {
  color:#5f9042;
}
```

JS

```javascript
//THREEJS RELATED VARIABLES

var scene,
  camera, fieldOfView, aspectRatio, nearPlane, farPlane,
  gobalLight, shadowLight, backLight,
  renderer,
  container,
  controls,
  clock;
var delta = 0;
var floorRadius = 200;
var speed = 6;
var distance = 0;
var level = 1;
var levelInterval;
var levelUpdateFreq = 3000;
var initSpeed = 5;
var maxSpeed = 48;
var monsterPos = .65;
var monsterPosTarget = .65;
var floorRotation = 0;
var collisionObstacle = 10;
var collisionBonus = 20;
var gameStatus = "play";
var cameraPosGame = 160;
var cameraPosGameOver = 260;
var monsterAcceleration = 0.004;
var malusClearColor = 0xb44b39;
var malusClearAlpha = 0;
var audio = new Audio('https://s3-us-west-2.amazonaws.com/s.cdpn.io/264161/Antonio-
Vivaldi-Summer_01.mp3');

var fieldGameOver, fieldDistance;

//SCREEN & MOUSE VARIABLES

var HEIGHT, WIDTH, windowHalfX, windowHalfY,
  mousePos = {
    x: 0,
    y: 0
  };

//3D OBJECTS VARIABLES

var hero;


// Materials
var blackMat = new THREE.MeshPhongMaterial({
    color: 0x100707,
    shading:THREE.FlatShading,
  });

var brownMat = new THREE.MeshPhongMaterial({
    color: 0xb44b39,
    shininess:0,
    shading:THREE.FlatShading,
  });

var greenMat = new THREE.MeshPhongMaterial({
    color: 0x7abf8e,
    shininess:0,
    shading:THREE.FlatShading,
  });

  var pinkMat = new THREE.MeshPhongMaterial({
    color: 0xdc5f45,//0xb43b29,//0xff5b49,
    shininess:0,
    shading:THREE.FlatShading,
  });

  var lightBrownMat = new THREE.MeshPhongMaterial({
    color: 0xe07a57,
    shading:THREE.FlatShading,
  });

  var whiteMat = new THREE.MeshPhongMaterial({
    color: 0xa49789,
    shading:THREE.FlatShading,
  });
  var skinMat = new THREE.MeshPhongMaterial({
    color: 0xff9ea5,
    shading:THREE.FlatShading
  });


// OTHER VARIABLES

var PI = Math.PI;

//INIT THREE JS, SCREEN AND MOUSE EVENTS

function initScreenAnd3D() {

  HEIGHT = window.innerHeight;
  WIDTH = window.innerWidth;
  windowHalfX = WIDTH / 2;
  windowHalfY = HEIGHT / 2;

  scene = new THREE.Scene();

  scene.fog = new THREE.Fog(0xd6eae6, 160,350);

  aspectRatio = WIDTH / HEIGHT;
  fieldOfView = 50;
  nearPlane = 1;
  farPlane = 2000;
  camera = new THREE.PerspectiveCamera(
    fieldOfView,
    aspectRatio,
    nearPlane,
    farPlane
  );
  camera.position.x = 0;
  camera.position.z = cameraPosGame;
  camera.position.y = 30;
  camera.lookAt(new THREE.Vector3(0, 30, 0));

  renderer = new THREE.WebGLRenderer({
    alpha: true,
    antialias: true
  });
  renderer.setPixelRatio(window.devicePixelRatio);
  renderer.setClearColor( malusClearColor, malusClearAlpha);

  renderer.setSize(WIDTH, HEIGHT);
  renderer.shadowMap.enabled = true;

  container = document.getElementById('world');
  container.appendChild(renderer.domElement);

  window.addEventListener('resize', handleWindowResize, false);
  document.addEventListener('mousedown', handleMouseDown, false);
  document.addEventListener("touchend", handleMouseDown, false);

  /*
  controls = new THREE.OrbitControls(camera, renderer.domElement);
  //controls.minPolarAngle = -Math.PI / 2;
  //controls.maxPolarAngle = Math.PI / 2;
  //controls.noZoom = true;
  controls.noPan = true;
  //*/

  clock = new THREE.Clock();

}

function handleWindowResize() {
  HEIGHT = window.innerHeight;
  WIDTH = window.innerWidth;
  windowHalfX = WIDTH / 2;
  windowHalfY = HEIGHT / 2;
  renderer.setSize(WIDTH, HEIGHT);
  camera.aspect = WIDTH / HEIGHT;
  camera.updateProjectionMatrix();
}


function handleMouseDown(event){
  if (gameStatus == "play") hero.jump();
  else if (gameStatus == "readyToReplay"){
    replay();
  }
}

function createLights() {
  globalLight = new THREE.AmbientLight(0xffffff, .9);

  shadowLight = new THREE.DirectionalLight(0xffffff, 1);
  shadowLight.position.set(-30, 40, 20);
  shadowLight.castShadow = true;
  shadowLight.shadow.camera.left = -400;
  shadowLight.shadow.camera.right = 400;
  shadowLight.shadow.camera.top = 400;
  shadowLight.shadow.camera.bottom = -400;
  shadowLight.shadow.camera.near = 1;
  shadowLight.shadow.camera.far = 2000;
  shadowLight.shadow.mapSize.width = shadowLight.shadow.mapSize.height = 2048;

  scene.add(globalLight);
  scene.add(shadowLight);

}

function createFloor() {

  floorShadow = new THREE.Mesh(new THREE.SphereGeometry(floorRadius, 50, 50), new
THREE.MeshPhongMaterial({
    color: 0x7abf8e,
    specular:0x000000,
    shininess:1,
    transparent:true,
    opacity:.5
  }));
  //floorShadow.rotation.x = -Math.PI / 2;
  floorShadow.receiveShadow = true;

  floorGrass = new THREE.Mesh(new THREE.SphereGeometry(floorRadius-.5, 50, 50), new
THREE.MeshBasicMaterial({
    color: 0x7abf8e
  }));
  //floor.rotation.x = -Math.PI / 2;
  floorGrass.receiveShadow = false;

  floor = new THREE.Group();
  floor.position.y = -floorRadius;

  floor.add(floorShadow);
  floor.add(floorGrass);
  scene.add(floor);

}

Hero = function() {
  this.status = "running";
  this.runningCycle = 0;
  this.mesh = new THREE.Group();
  this.body = new THREE.Group();
  this.mesh.add(this.body);

  var torsoGeom = new THREE.CubeGeometry(7, 7, 10, 1);
```

```
this.torso = new THREE.Mesh(torsoGeom, brownMat);
this.torso.position.z = 0;
this.torso.position.y = 7;
this.torso.castShadow = true;
this.body.add(this.torso);

var pantsGeom = new THREE.CubeGeometry(9, 9, 5, 1);
this.pants = new THREE.Mesh(pantsGeom, whiteMat);
this.pants.position.z = -3;
this.pants.position.y = 0;
this.pants.castShadow = true;
this.torso.add(this.pants);

var tailGeom = new THREE.CubeGeometry(3, 3, 3, 1);
tailGeom.applyMatrix(new THREE.Matrix4().makeTranslation(0,0,-2));
this.tail = new THREE.Mesh(tailGeom, lightBrownMat);
this.tail.position.z = -4;
this.tail.position.y = 5;
this.tail.castShadow = true;
this.torso.add(this.tail);

this.torso.rotation.x = -Math.PI/8;

var headGeom = new THREE.CubeGeometry(10, 10, 13, 1);

headGeom.applyMatrix(new THREE.Matrix4().makeTranslation(0,0,7.5));
this.head = new THREE.Mesh(headGeom, brownMat);
this.head.position.z = 2;
this.head.position.y = 11;
this.head.castShadow = true;
this.body.add(this.head);

var cheekGeom = new THREE.CubeGeometry(1, 4, 4, 1);
this.cheekR = new THREE.Mesh(cheekGeom, pinkMat);
this.cheekR.position.x = -5;
this.cheekR.position.z = 7;
this.cheekR.position.y = -2.5;
this.cheekR.castShadow = true;
this.head.add(this.cheekR);

this.cheekL = this.cheekR.clone();
this.cheekL.position.x = - this.cheekR.position.x;
this.head.add(this.cheekL);


var noseGeom = new THREE.CubeGeometry(6, 6, 3, 1);
this.nose = new THREE.Mesh(noseGeom, lightBrownMat);
this.nose.position.z = 13.5;
this.nose.position.y = 2.6;
this.nose.castShadow = true;
this.head.add(this.nose);

var mouthGeom = new THREE.CubeGeometry(4, 2, 4, 1);
mouthGeom.applyMatrix(new THREE.Matrix4().makeTranslation(0,0,3));
mouthGeom.applyMatrix(new THREE.Matrix4().makeRotationX(Math.PI/12));
this.mouth = new THREE.Mesh(mouthGeom, brownMat);
this.mouth.position.z = 8;
this.mouth.position.y = -4;
this.mouth.castShadow = true;
this.head.add(this.mouth);


var pawFGeom = new THREE.CubeGeometry(3,3,3, 1);
this.pawFR = new THREE.Mesh(pawFGeom, lightBrownMat);
this.pawFR.position.x = -2;
this.pawFR.position.z = 6;
this.pawFR.position.y = 1.5;
this.pawFR.castShadow = true;
this.body.add(this.pawFR);

this.pawFL = this.pawFR.clone();
this.pawFL.position.x = - this.pawFR.position.x;
this.pawFL.castShadow = true;
this.body.add(this.pawFL);

var pawBGeom = new THREE.CubeGeometry(3,3,6, 1);
this.pawBL = new THREE.Mesh(pawBGeom, lightBrownMat);
this.pawBL.position.y = 1.5;
this.pawBL.position.z = 0;
this.pawBL.position.x = 5;
this.pawBL.castShadow = true;
this.body.add(this.pawBL);

this.pawBR = this.pawBL.clone();
this.pawBR.position.x = - this.pawBL.position.x;
this.pawBR.castShadow = true;
this.body.add(this.pawBR);

var earGeom = new THREE.CubeGeometry(7, 18, 2, 1);
earGeom.vertices[6].x+=2;
earGeom.vertices[6].z+=.5;

earGeom.vertices[7].x+=2;
earGeom.vertices[7].z-=.5;

earGeom.vertices[2].x-=2;
earGeom.vertices[2].z-=.5;

earGeom.vertices[3].x-=2;
earGeom.vertices[3].z+=.5;
earGeom.applyMatrix(new THREE.Matrix4().makeTranslation(0,9,0));
```

```
  this.earL = new THREE.Mesh(earGeom, brownMat);
  this.earL.position.x = 2;
  this.earL.position.z = 2.5;
  this.earL.position.y = 5;
  this.earL.rotation.z = -Math.PI/12;
  this.earL.castShadow = true;
  this.head.add(this.earL);

  this.earR = this.earL.clone();
  this.earR.position.x = -this.earL.position.x;
  this.earR.rotation.z = -this.earL.rotation.z;
  this.earR.castShadow = true;
  this.head.add(this.earR);

  var eyeGeom = new THREE.CubeGeometry(2,4,4);

  this.eyeL = new THREE.Mesh(eyeGeom, whiteMat);
  this.eyeL.position.x = 5;
  this.eyeL.position.z = 5.5;
  this.eyeL.position.y = 2.9;
  this.eyeL.castShadow = true;
  this.head.add(this.eyeL);

  var irisGeom = new THREE.CubeGeometry(.6,2,2);

  this.iris = new THREE.Mesh(irisGeom, blackMat);
  this.iris.position.x = 1.2;
  this.iris.position.y = 1;
  this.iris.position.z = 1;
  this.eyeL.add(this.iris);

  this.eyeR = this.eyeL.clone();
  this.eyeR.children[0].position.x = -this.iris.position.x;


  this.eyeR.position.x = -this.eyeL.position.x;
  this.head.add(this.eyeR);

  this.body.traverse(function(object) {
    if (object instanceof THREE.Mesh) {
      object.castShadow = true;
      object.receiveShadow = true;
    }
  });
}

BonusParticles = function(){
  this.mesh = new THREE.Group();
  var bigParticleGeom = new THREE.CubeGeometry(10,10,10,1);
  var smallParticleGeom = new THREE.CubeGeometry(5,5,5,1);
  this.parts = [];
  for (var i=0; i<10; i++){
    var partPink = new THREE.Mesh(bigParticleGeom, pinkMat);
    var partGreen = new THREE.Mesh(smallParticleGeom, greenMat);
    partGreen.scale.set(.5,.5,.5);
    this.parts.push(partPink);
    this.parts.push(partGreen);
    this.mesh.add(partPink);
    this.mesh.add(partGreen);
  }
}

BonusParticles.prototype.explose = function(){
  var _this = this;
  var explosionSpeed = .5;
  for(var i=0; i<this.parts.length; i++){
    var tx = -50 + Math.random()*100;
    var ty = -50 + Math.random()*100;
    var tz = -50 + Math.random()*100;
    var p = this.parts[i];
    p.position.set(0,0,0);
    p.scale.set(1,1,1);
    p.visible = true;
    var s = explosionSpeed + Math.random()*.5;
    TweenMax.to(p.position, s,{x:tx, y:ty, z:tz, ease:Power4.easeOut});
    TweenMax.to(p.scale, s,{x:.01, y:.01, z:.01, ease:Power4.easeOut,
onComplete:removeParticle, onCompleteParams:[p]});
  }
}

function removeParticle(p){
  p.visible = false;
}

Hero.prototype.run = function(){
  this.status = "running";

  var s = Math.min(speed,maxSpeed);

  this.runningCycle += delta * s * .7;
  this.runningCycle = this.runningCycle % (Math.PI*2);
  var t = this.runningCycle;

  var amp = 4;
  var disp = .2;

  // BODY

  this.body.position.y = 6+ Math.sin(t - Math.PI/2)*amp;
  this.body.rotation.x = .2 + Math.sin(t - Math.PI/2)*amp*.1;
  this.torso.rotation.x =  Math.sin(t - Math.PI/2)*amp*.1;
  this.torso.position.y =  7 + Math.sin(t - Math.PI/2)*amp*.5;

  // MOUTH
  this.mouth.rotation.x = Math.PI/16 + Math.cos(t)*amp*.05;

  // HEAD
  this.head.position.z = 2 + Math.sin(t - Math.PI/2)*amp*.5;
  this.head.position.y = 8 + Math.cos(t - Math.PI/2)*amp*.7;
  this.head.rotation.x = -.2 + Math.sin(t + Math.PI)*amp*.1;

  // EARS
  this.earL.rotation.x = Math.cos(-Math.PI/2 + t)*(amp*.2);
  this.earR.rotation.x = Math.cos(-Math.PI/2 + .2 + t)*(amp*.3);

  // EYES
  this.eyeR.scale.y = this.eyeL.scale.y = .7 +  Math.abs(Math.cos(-Math.PI/4 + t*.5))*.6;

  // TAIL
  this.tail.rotation.x = Math.cos(Math.PI/2 + t)*amp*.3;

  // FRONT RIGHT PAW
  this.pawFR.position.y = 1.5 + Math.sin(t)*amp;
  this.pawFR.rotation.x = Math.cos(t ) * Math.PI/4;


  this.pawFR.position.z = 6 - Math.cos(t)*amp*2;

  // FRONT LEFT PAW

  this.pawFL.position.y = 1.5 + Math.sin(disp + t)*amp;
  this.pawFL.rotation.x = Math.cos( t ) * Math.PI/4;


  this.pawFL.position.z = 6 - Math.cos(disp+t)*amp*2;

  // BACK RIGHT PAW
  this.pawBR.position.y = 1.5 + Math.sin(Math.PI + t)*amp;
  this.pawBR.rotation.x = Math.cos(t + Math.PI*1.5) * Math.PI/3;


  this.pawBR.position.z = - Math.cos(Math.PI + t)*amp;

  // BACK LEFT PAW
  this.pawBL.position.y = 1.5 + Math.sin(Math.PI + t)*amp;
  this.pawBL.rotation.x = Math.cos(t + Math.PI *1.5) * Math.PI/3;


  this.pawBL.position.z = - Math.cos(Math.PI + t)*amp;
```

```
}

Hero.prototype.jump = function(){
  if (this.status == "jumping") return;
  this.status = "jumping";
  var _this = this;
  var totalSpeed = 10 / speed;
  var jumpHeight = 45;

  TweenMax.to(this.earL.rotation, totalSpeed, {x:"+=.3", ease:Back.easeOut});
  TweenMax.to(this.earR.rotation, totalSpeed, {x:"-=.3", ease:Back.easeOut});

  TweenMax.to(this.pawFL.rotation, totalSpeed, {x:"+=.7", ease:Back.easeOut});
  TweenMax.to(this.pawFR.rotation, totalSpeed, {x:"-=.7", ease:Back.easeOut});
  TweenMax.to(this.pawBL.rotation, totalSpeed, {x:"+=.7", ease:Back.easeOut});
  TweenMax.to(this.pawBR.rotation, totalSpeed, {x:"-=.7", ease:Back.easeOut});

  TweenMax.to(this.tail.rotation, totalSpeed, {x:"+=1", ease:Back.easeOut});

  TweenMax.to(this.mouth.rotation, totalSpeed, {x:.5, ease:Back.easeOut});

  TweenMax.to(this.mesh.position, totalSpeed/2, {y:jumpHeight, ease:Power2.easeOut});
  TweenMax.to(this.mesh.position, totalSpeed/2, {y:0, ease:Power4.easeIn, delay:totalSpeed/2,
onComplete: function(){
    //t = 0;
    _this.status="running";
  }});

}



Monster = function(){

  this.runningCycle = 0;

  this.mesh = new THREE.Group();
  this.body = new THREE.Group();

  var torsoGeom = new THREE.CubeGeometry(15,15,20, 1);
  this.torso = new THREE.Mesh(torsoGeom, blackMat);

  var headGeom = new THREE.CubeGeometry(20,20,40, 1);
  headGeom.applyMatrix(new THREE.Matrix4().makeTranslation(0,0,20));
  this.head = new THREE.Mesh(headGeom, blackMat);
  this.head.position.z = 12;
  this.head.position.y = 2;

  var mouthGeom = new THREE.CubeGeometry(10,4,20, 1);
  mouthGeom.applyMatrix(new THREE.Matrix4().makeTranslation(0,-2,10));
  this.mouth = new THREE.Mesh(mouthGeom, blackMat);
  this.mouth.position.y = -8;
this.mouth.rotation.x = .4;
this.mouth.position.z = 4;

this.heroHolder = new THREE.Group();
this.heroHolder.position.z = 20;
this.mouth.add(this.heroHolder);

var toothGeom = new THREE.CubeGeometry(2,2,1,1);

toothGeom.vertices[1].x-=1;
toothGeom.vertices[4].x+=1;
toothGeom.vertices[5].x+=1;
toothGeom.vertices[0].x-=1;

for(var i=0; i<3; i++){
  var toothf = new THREE.Mesh(toothGeom, whiteMat);
  toothf.position.x = -2.8 + i*2.5;
  toothf.position.y = 1;
  toothf.position.z = 19;

  var toothl = new THREE.Mesh(toothGeom, whiteMat);
  toothl.rotation.y = Math.PI/2;
  toothl.position.z = 12 + i*2.5;
  toothl.position.y = 1;
  toothl.position.x = 4;

  var toothr = toothl.clone();
  toothl.position.x = -4;

  this.mouth.add(toothf);
  this.mouth.add(toothl);
  this.mouth.add(toothr);
}

var tongueGeometry = new THREE.CubeGeometry(6,1,14);
tongueGeometry.applyMatrix(new THREE.Matrix4().makeTranslation(0,0,7));

this.tongue = new THREE.Mesh(tongueGeometry, pinkMat);
this.tongue.position.z = 2;
this.tongue.rotation.x = -.2;
this.mouth.add(this.tongue);

var noseGeom = new THREE.CubeGeometry(4,4,4, 1);
this.nose = new THREE.Mesh(noseGeom, pinkMat);
this.nose.position.z = 39.5;
this.nose.position.y = 9;
this.head.add(this.nose);

this.head.add(this.mouth);

var eyeGeom = new THREE.CubeGeometry(2,3,3);
```

```
this.eyeL = new THREE.Mesh(eyeGeom, whiteMat);
this.eyeL.position.x = 10;
this.eyeL.position.z = 5;
this.eyeL.position.y = 5;
this.eyeL.castShadow = true;
this.head.add(this.eyeL);

var irisGeom = new THREE.CubeGeometry(.6,1,1);

this.iris = new THREE.Mesh(irisGeom, blackMat);
this.iris.position.x = 1.2;
this.iris.position.y = -1;
this.iris.position.z = 1;
this.eyeL.add(this.iris);

this.eyeR = this.eyeL.clone();
this.eyeR.children[0].position.x = -this.iris.position.x;
this.eyeR.position.x = -this.eyeL.position.x;
this.head.add(this.eyeR);


var earGeom = new THREE.CubeGeometry(8, 6, 2, 1);
earGeom.vertices[1].x-=4;
earGeom.vertices[4].x+=4;
earGeom.vertices[5].x+=4;
earGeom.vertices[5].z-=2;
earGeom.vertices[0].x-=4;
earGeom.vertices[0].z-=2;


earGeom.applyMatrix(new THREE.Matrix4().makeTranslation(0,3,0));

this.earL = new THREE.Mesh(earGeom, blackMat);
this.earL.position.x = 6;
this.earL.position.z = 1;
this.earL.position.y = 10;
this.earL.castShadow = true;
this.head.add(this.earL);

this.earR = this.earL.clone();
this.earR.position.x = -this.earL.position.x;
this.earR.rotation.z = -this.earL.rotation.z;
this.head.add(this.earR);

var eyeGeom = new THREE.CubeGeometry(2,4,4);

var tailGeom = new THREE.CylinderGeometry(5,2, 20, 4, 1);
tailGeom.applyMatrix(new THREE.Matrix4().makeTranslation(0,10,0));
tailGeom.applyMatrix(new THREE.Matrix4().makeRotationX(-Math.PI/2));
tailGeom.applyMatrix(new THREE.Matrix4().makeRotationZ(Math.PI/4));
```

```
  this.tail = new THREE.Mesh(tailGeom, blackMat);
  this.tail.position.z = -10;
  this.tail.position.y = 4;
  this.torso.add(this.tail);


  var pawGeom = new THREE.CylinderGeometry(1.5,0,10);
  pawGeom.applyMatrix(new THREE.Matrix4().makeTranslation(0,-5,0));
  this.pawFL = new THREE.Mesh(pawGeom, blackMat);
  this.pawFL.position.y = -7.5;
  this.pawFL.position.z = 8.5;
  this.pawFL.position.x = 5.5;
  this.torso.add(this.pawFL);

  this.pawFR = this.pawFL.clone();
  this.pawFR.position.x = - this.pawFL.position.x;
  this.torso.add(this.pawFR);

  this.pawBR = this.pawFR.clone();
  this.pawBR.position.z = - this.pawFL.position.z;
  this.torso.add(this.pawBR);

  this.pawBL = this.pawBR.clone();
  this.pawBL.position.x = this.pawFL.position.x;
  this.torso.add(this.pawBL);

  this.mesh.add(this.body);
  this.torso.add(this.head);
  this.body.add(this.torso);

  this.torso.castShadow = true;
  this.head.castShadow = true;
  this.pawFL.castShadow = true;
  this.pawFR.castShadow = true;
  this.pawBL.castShadow = true;
  this.pawBR.castShadow = true;

  this.body.rotation.y = Math.PI/2;
}

Monster.prototype.run = function(){
  var s = Math.min(speed,maxSpeed);
  this.runningCycle += delta * s * .7;
  this.runningCycle = this.runningCycle % (Math.PI*2);
  var t = this.runningCycle;

  this.pawFR.rotation.x = Math.sin(t)*Math.PI/4;
  this.pawFR.position.y = -5.5 - Math.sin(t);
  this.pawFR.position.z = 7.5 + Math.cos(t);
  this.pawFL.rotation.x = Math.sin(t+.4)*Math.PI/4;
  this.pawFL.position.y = -5.5 - Math.sin(t+.4);
  this.pawFL.position.z = 7.5 + Math.cos(t+.4);

  this.pawBL.rotation.x = Math.sin(t+2)*Math.PI/4;
  this.pawBL.position.y = -5.5 - Math.sin(t+3.8);
  this.pawBL.position.z = -7.5 + Math.cos(t+3.8);

  this.pawBR.rotation.x = Math.sin(t+2.4)*Math.PI/4;
  this.pawBR.position.y = -5.5 - Math.sin(t+3.4);
  this.pawBR.position.z = -7.5 + Math.cos(t+3.4);

  this.torso.rotation.x = Math.sin(t)*Math.PI/8;
  this.torso.position.y = 3-Math.sin(t+Math.PI/2)*3;

  //this.head.position.y = 5-Math.sin(t+Math.PI/2)*2;
  this.head.rotation.x = -.1+Math.sin(-t-1)*.4;
  this.mouth.rotation.x = .2 + Math.sin(t+Math.PI+.3)*.4;

  this.tail.rotation.x = .2 + Math.sin(t-Math.PI/2);

  this.eyeR.scale.y = .5 + Math.sin(t+Math.PI)*.5;
}

Hero.prototype.nod = function(){
  var _this = this;
  var sp = .5 + Math.random();

  // HEAD
  var tHeadRotY = -Math.PI/6 + Math.random()* Math.PI/3;
  TweenMax.to(this.head.rotation, sp, {y:tHeadRotY, ease:Power4.easeInOut,
onComplete:function(){_this.nod()}});

  // EARS
  var tEarLRotX =  Math.PI/4 + Math.random()* Math.PI/6;
  var tEarRRotX =  Math.PI/4 + Math.random()* Math.PI/6;

  TweenMax.to(this.earL.rotation, sp, {x:tEarLRotX, ease:Power4.easeInOut});
  TweenMax.to(this.earR.rotation, sp, {x:tEarRRotX, ease:Power4.easeInOut});


  // PAWS BACK LEFT

  var tPawBLRot = Math.random()*Math.PI/2;
  var tPawBLY = -4 + Math.random()*8;

  TweenMax.to(this.pawBL.rotation, sp/2, {x:tPawBLRot, ease:Power1.easeInOut, yoyo:true,
repeat:2});
  TweenMax.to(this.pawBL.position, sp/2, {y:tPawBLY, ease:Power1.easeInOut, yoyo:true,
repeat:2});
```

```
  // PAWS BACK RIGHT

  var tPawBRRot = Math.random()*Math.PI/2;
  var tPawBRY = -4 + Math.random()*8;
  TweenMax.to(this.pawBR.rotation, sp/2, {x:tPawBRRot, ease:Power1.easeInOut, yoyo:true,
repeat:2});
  TweenMax.to(this.pawBR.position, sp/2, {y:tPawBRY, ease:Power1.easeInOut, yoyo:true,
repeat:2});

  // PAWS FRONT LEFT

  var tPawFLRot = Math.random()*Math.PI/2;
  var tPawFLY = -4 + Math.random()*8;

  TweenMax.to(this.pawFL.rotation, sp/2, {x:tPawFLRot, ease:Power1.easeInOut, yoyo:true,
repeat:2});

  TweenMax.to(this.pawFL.position, sp/2, {y:tPawFLY, ease:Power1.easeInOut, yoyo:true,
repeat:2});

  // PAWS FRONT RIGHT

  var tPawFRRot = Math.random()*Math.PI/2;
  var tPawFRY = -4 + Math.random()*8;

  TweenMax.to(this.pawFR.rotation, sp/2, {x:tPawFRRot, ease:Power1.easeInOut, yoyo:true,
repeat:2});

  TweenMax.to(this.pawFR.position, sp/2, {y:tPawFRY, ease:Power1.easeInOut, yoyo:true,
repeat:2});

  // MOUTH
  var tMouthRot = Math.random()*Math.PI/8;
  TweenMax.to(this.mouth.rotation, sp, {x:tMouthRot, ease:Power1.easeInOut});
  // IRIS
  var tIrisY = -1 + Math.random()*2;
  var tIrisZ = -1 + Math.random()*2;
  var iris1 = this.iris;
  var iris2 = this.eyeR.children[0];
  TweenMax.to([iris1.position, iris2.position], sp, {y:tIrisY, z:tIrisZ, ease:Power1.easeInOut});

  //EYES
  if (Math.random()>.2) TweenMax.to([this.eyeR.scale, this.eyeL.scale], sp/8, {y:0,
ease:Power1.easeInOut, yoyo:true, repeat:1});

}

Hero.prototype.hang = function(){
  var _this = this;
  var sp = 1;
  var ease = Power4.easeOut;

  TweenMax.killTweensOf(this.eyeL.scale);
  TweenMax.killTweensOf(this.eyeR.scale);

  this.body.rotation.x = 0;
  this.torso.rotation.x = 0;
  this.body.position.y = 0;
  this.torso.position.y = 7;

  TweenMax.to(this.mesh.rotation, sp, {y:0, ease:ease});
  TweenMax.to(this.mesh.position, sp, {y:-7, z:6, ease:ease});
  TweenMax.to(this.head.rotation, sp, {x:Math.PI/6, ease:ease,
onComplete:function(){_this.nod();}});

  TweenMax.to(this.earL.rotation, sp, {x:Math.PI/3, ease:ease});
  TweenMax.to(this.earR.rotation, sp, {x:Math.PI/3, ease:ease});

  TweenMax.to(this.pawFL.position, sp, {y:-1, z:3, ease:ease});
  TweenMax.to(this.pawFR.position, sp, {y:-1, z:3, ease:ease});
  TweenMax.to(this.pawBL.position, sp, {y:-2, z:-3, ease:ease});
  TweenMax.to(this.pawBR.position, sp, {y:-2, z:-3, ease:ease});

  TweenMax.to(this.eyeL.scale, sp, {y:1, ease:ease});
  TweenMax.to(this.eyeR.scale, sp, {y:1, ease:ease});
}

Monster.prototype.nod = function(){
  var _this = this;
  var sp = 1 + Math.random()*2;

  // HEAD
  var tHeadRotY = -Math.PI/3 + Math.random()*.5;
  var tHeadRotX = Math.PI/3 - .2 +  Math.random()*.4;
  TweenMax.to(this.head.rotation, sp, {x:tHeadRotX, y:tHeadRotY, ease:Power4.easeInOut,
onComplete:function(){_this.nod()}});

  // TAIL

  var tTailRotY = -Math.PI/4;
  TweenMax.to(this.tail.rotation, sp/8, {y:tTailRotY, ease:Power1.easeInOut, yoyo:true,
repeat:8});

  // EYES

  TweenMax.to([this.eyeR.scale, this.eyeL.scale], sp/20, {y:0, ease:Power1.easeInOut,
yoyo:true, repeat:1});
}

Monster.prototype.sit = function(){
  var sp = 1.2;
  var ease = Power4.easeOut;
  var _this = this;
  TweenMax.to(this.torso.rotation, sp, {x:-1.3, ease:ease});
  TweenMax.to(this.torso.position, sp, {y:-5, ease:ease, onComplete:function(){
    _this.nod();
    gameStatus = "readyToReplay";
  }});

  TweenMax.to(this.head.rotation, sp, {x:Math.PI/3, y :-Math.PI/3, ease:ease});
  TweenMax.to(this.tail.rotation, sp, {x:2, y:Math.PI/4, ease:ease});
  TweenMax.to(this.pawBL.rotation, sp, {x:-.1, ease:ease});
  TweenMax.to(this.pawBR.rotation, sp, {x:-.1, ease:ease});
  TweenMax.to(this.pawFL.rotation, sp, {x:1, ease:ease});
  TweenMax.to(this.pawFR.rotation, sp, {x:1, ease:ease});
  TweenMax.to(this.mouth.rotation, sp, {x:.3, ease:ease});
  TweenMax.to(this.eyeL.scale, sp, {y:1, ease:ease});
  TweenMax.to(this.eyeR.scale, sp, {y:1, ease:ease});

  //TweenMax.to(this.body.rotation, sp, {y:Math.PI/4});

}


Carrot = function() {
  this.angle = 0;
  this.mesh = new THREE.Group();

  var bodyGeom = new THREE.CylinderGeometry(5,3, 10, 4,1);
  bodyGeom.vertices[8].y+=2;
  bodyGeom.vertices[9].y-=3;

  this.body = new THREE.Mesh(bodyGeom, pinkMat);

  var leafGeom = new THREE.CubeGeometry(5,10,1,1);
  leafGeom.applyMatrix(new THREE.Matrix4().makeTranslation(0,5,0));
  leafGeom.vertices[2].x-=1;
  leafGeom.vertices[3].x-=1;
  leafGeom.vertices[6].x+=1;
  leafGeom.vertices[7].x+=1;

  this.leaf1 = new THREE.Mesh(leafGeom,greenMat);
  this.leaf1.position.y = 7;
  this.leaf1.rotation.z = .3;
  this.leaf1.rotation.x = .2;

  this.leaf2 = this.leaf1.clone();
  this.leaf2.scale.set(1,1.3,1);
  this.leaf2.position.y = 7;
  this.leaf2.rotation.z = -.3;
  this.leaf2.rotation.x = -.2;
  this.mesh.add(this.body);
  this.mesh.add(this.leaf1);
  this.mesh.add(this.leaf2);

  this.body.traverse(function(object) {
    if (object instanceof THREE.Mesh) {
      object.castShadow = true;
      object.receiveShadow = true;
    }
  });
}

Hedgehog = function() {
  this.angle = 0;
  this.status="ready";
  this.mesh = new THREE.Group();
  var bodyGeom = new THREE.CubeGeometry(6,6,6,1);
  this.body = new THREE.Mesh(bodyGeom, blackMat);

  var headGeom = new THREE.CubeGeometry(5,5,7,1);
  this.head= new THREE.Mesh(headGeom, lightBrownMat);
  this.head.position.z = 6;
  this.head.position.y = -.5;

  var noseGeom = new THREE.CubeGeometry(1.5,1.5,1.5,1);
  this.nose = new THREE.Mesh(noseGeom, blackMat);
  this.nose.position.z = 4;
  this.nose.position.y = 2;

  var eyeGeom = new THREE.CubeGeometry(1,3,3);

  this.eyeL = new THREE.Mesh(eyeGeom, whiteMat);
  this.eyeL.position.x = 2.2;
  this.eyeL.position.z = -.5;
  this.eyeL.position.y = .8;
  this.eyeL.castShadow = true;
  this.head.add(this.eyeL);

  var irisGeom = new THREE.CubeGeometry(.5,1,1);

  this.iris = new THREE.Mesh(irisGeom, blackMat);
  this.iris.position.x = .5;
  this.iris.position.y = .8;
  this.iris.position.z = .8;
  this.eyeL.add(this.iris);

  this.eyeR = this.eyeL.clone();
  this.eyeR.children[0].position.x = -this.iris.position.x;
  this.eyeR.position.x = -this.eyeL.position.x;

  var spikeGeom = new THREE.CubeGeometry(.5,2,.5,1);
```

```
spikeGeom.applyMatrix(new THREE.Matrix4().makeTranslation(0,1,0));

for (var i=0; i<9; i++){
  var row = (i%3);
  var col = Math.floor(i/3);
  var sb = new THREE.Mesh(spikeGeom, blackMat);
  sb.rotation.x =-Math.PI/2 + (Math.PI/12*row) -.5 +  Math.random();
  sb.position.z = -3;
  sb.position.y = -2 + row*2;
  sb.position.x = -2 + col*2;
  this.body.add(sb);
  var st = new THREE.Mesh(spikeGeom, blackMat);
  st.position.y = 3;
  st.position.x = -2 + row*2;
  st.position.z = -2 + col*2;
  st.rotation.z = Math.PI/6 - (Math.PI/6*row) -.5 +  Math.random();
  this.body.add(st);

  var sr = new THREE.Mesh(spikeGeom, blackMat);
  sr.position.x = 3;
  sr.position.y = -2 + row*2;
  sr.position.z = -2 + col*2;
  sr.rotation.z = -Math.PI/2 + (Math.PI/12*row) -.5 +  Math.random();
  this.body.add(sr);

  var sl = new THREE.Mesh(spikeGeom, blackMat);
  sl.position.x = -3;
  sl.position.y = -2 + row*2;
  sl.position.z = -2 + col*2;
  sl.rotation.z = Math.PI/2  - (Math.PI/12*row) -.5 +  Math.random();;
  this.body.add(sl);
}

this.head.add(this.eyeR);
var earGeom = new THREE.CubeGeometry(2, 2, .5, 1);
this.earL = new THREE.Mesh(earGeom, lightBrownMat);
this.earL.position.x = 2.5;
this.earL.position.z = -2.5;
this.earL.position.y = 2.5;
this.earL.rotation.z = -Math.PI/12;
this.earL.castShadow = true;
this.head.add(this.earL);

this.earR = this.earL.clone();
this.earR.position.x = -this.earL.position.x;
this.earR.rotation.z = -this.earL.rotation.z;
this.earR.castShadow = true;
this.head.add(this.earR);

var mouthGeom = new THREE.CubeGeometry( 1, 1,.5, 1);
this.mouth = new THREE.Mesh(mouthGeom, blackMat);
```

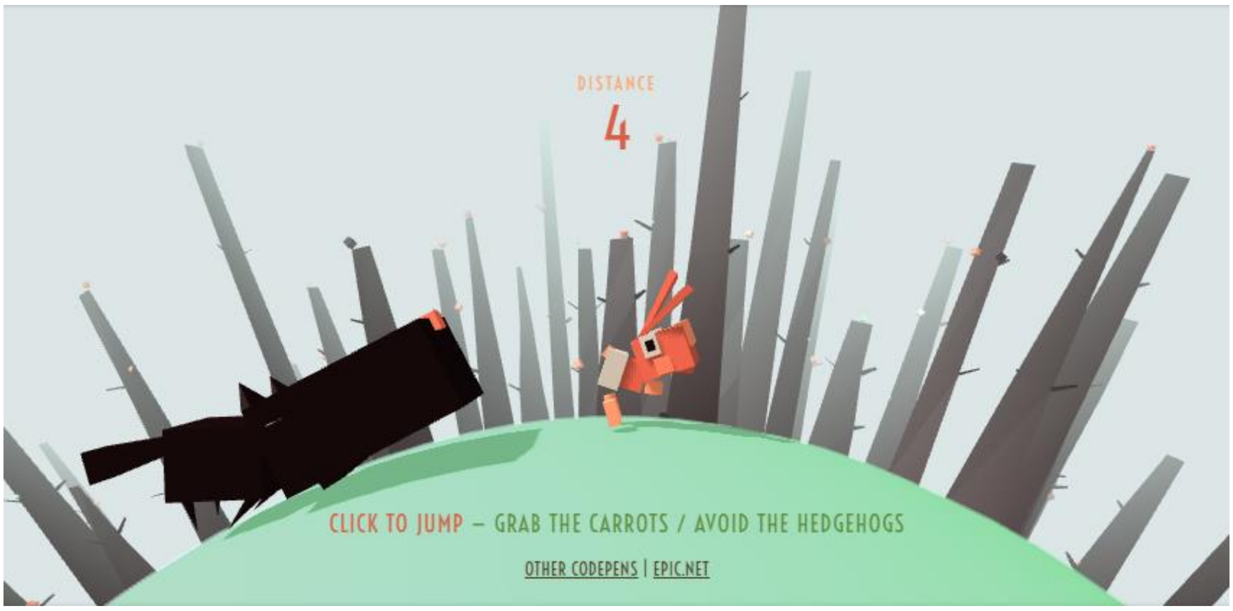
```
  this.mouth.position.z = 3.5;
  this.mouth.position.y = -1.5;
  this.head.add(this.mouth);


  this.mesh.add(this.body);
  this.body.add(this.head);
  this.head.add(this.nose);

  this.mesh.traverse(function(object) {
    if (object instanceof THREE.Mesh) {
      object.castShadow = true;
      object.receiveShadow = true;
    }
  });
}

Hedgehog.prototype.nod = function(){
  var _this = this;
  var speed = .1 + Math.random()*.5;
  var angle = -Math.PI/4 + Math.random()*Math.PI/2;
  TweenMax.to(this.head.rotation, speed, {y:angle, onComplete:function(){
    _this.nod();
  }});
}


function createHero() {
  hero = new Hero();
  hero.mesh.rotation.y = Math.PI/2;
  scene.add(hero.mesh);
  hero.nod();
}

function createMonster() {

  monster = new Monster();
  monster.mesh.position.z = 20;
  //monster.mesh.scale.set(1.2,1.2,1.2);
  scene.add(monster.mesh);
  updateMonsterPosition();

}

function updateMonsterPosition(){
  monster.run();
  monsterPosTarget -= delta*monsterAcceleration;
  monsterPos += (monsterPosTarget-monsterPos) *delta;
  if (monsterPos < .56){
    gameOver();
  }
  var angle = Math.PI*monsterPos;
  monster.mesh.position.y = - floorRadius + Math.sin(angle)*(floorRadius + 12);
  monster.mesh.position.x = Math.cos(angle)*(floorRadius+15);
  monster.mesh.rotation.z = -Math.PI/2 + angle;
}

function gameOver(){
  fieldGameOver.className = "show";
  gameStatus = "gameOver";
  monster.sit();
  hero.hang();
  monster.heroHolder.add(hero.mesh);
  TweenMax.to(this, 1, {speed:0});
  TweenMax.to(camera.position, 3, {z:cameraPosGameOver, y: 60, x:-30});
  carrot.mesh.visible = false;
  obstacle.mesh.visible = false;
  clearInterval(levelInterval);
}

function replay(){

  gameStatus = "preparingToReplay"

  fieldGameOver.className = "";

  TweenMax.killTweensOf(monster.pawFL.position);
  TweenMax.killTweensOf(monster.pawFR.position);
  TweenMax.killTweensOf(monster.pawBL.position);
  TweenMax.killTweensOf(monster.pawBR.position);

  TweenMax.killTweensOf(monster.pawFL.rotation);
  TweenMax.killTweensOf(monster.pawFR.rotation);
  TweenMax.killTweensOf(monster.pawBL.rotation);
  TweenMax.killTweensOf(monster.pawBR.rotation);

  TweenMax.killTweensOf(monster.tail.rotation);
  TweenMax.killTweensOf(monster.head.rotation);
  TweenMax.killTweensOf(monster.eyeL.scale);
  TweenMax.killTweensOf(monster.eyeR.scale);

  //TweenMax.killTweensOf(hero.head.rotation);

  monster.tail.rotation.y = 0;

  TweenMax.to(camera.position, 3, {z:cameraPosGame, x:0, y:30, ease:Power4.easeInOut});
  TweenMax.to(monster.torso.rotation,2, {x:0, ease:Power4.easeInOut});
  TweenMax.to(monster.torso.position,2, {y:0, ease:Power4.easeInOut});
  TweenMax.to(monster.pawFL.rotation,2, {x:0, ease:Power4.easeInOut});
  TweenMax.to(monster.pawFR.rotation,2, {x:0, ease:Power4.easeInOut});
  TweenMax.to(monster.mouth.rotation,2, {x:.5, ease:Power4.easeInOut});
  TweenMax.to(monster.head.rotation,2, {y:0, x:-.3, ease:Power4.easeInOut});

  TweenMax.to(hero.mesh.position, 2, { x:20, ease:Power4.easeInOut});
  TweenMax.to(hero.head.rotation, 2, { x:0, y:0, ease:Power4.easeInOut});
  TweenMax.to(monster.mouth.rotation, 2, {x:.2, ease:Power4.easeInOut});
  TweenMax.to(monster.mouth.rotation, 1, {x:.4, ease:Power4.easeIn, delay: 1,
onComplete:function(){

    resetGame();
  }});

}

Fir = function() {
  var height = 200;
  var truncGeom = new THREE.CylinderGeometry(2,2,height, 6,1);
  truncGeom.applyMatrix(new THREE.Matrix4().makeTranslation(0,height/2,0));
  this.mesh = new THREE.Mesh(truncGeom, greenMat);
  this.mesh.castShadow = true;
}

var firs = new THREE.Group();

function createFirs(){

  var nTrees = 100;
   for(var i=0; i< nTrees; i++){
    var phi = i*(Math.PI*2)/nTrees;
    var theta = Math.PI/2;
    //theta += .25 + Math.random()*.3;
    theta += (Math.random()>.05)? .25 + Math.random()*.3 : - .35 -  Math.random()*.1;

    var fir = new Tree();
    fir.mesh.position.x = Math.sin(theta)*Math.cos(phi)*floorRadius;
    fir.mesh.position.y = Math.sin(theta)*Math.sin(phi)*(floorRadius-10);
    fir.mesh.position.z = Math.cos(theta)*floorRadius;

    var vec = fir.mesh.position.clone();
    var axis = new THREE.Vector3(0,1,0);
    fir.mesh.quaternion.setFromUnitVectors(axis, vec.clone().normalize());
    floor.add(fir.mesh);
  }
}

function createCarrot(){
  carrot = new Carrot();
  scene.add(carrot.mesh);
}
```

```
function updateCarrotPosition(){
  carrot.mesh.rotation.y += delta * 6;
  carrot.mesh.rotation.z = Math.PI/2 - (floorRotation+carrot.angle);
  carrot.mesh.position.y = -floorRadius + Math.sin(floorRotation+carrot.angle) *
(floorRadius+50);
  carrot.mesh.position.x = Math.cos(floorRotation+carrot.angle) * (floorRadius+50);

}

function updateObstaclePosition(){
  if (obstacle.status=="flying")return;

  // TODO fix this,
  if (floorRotation+obstacle.angle > 2.5 ){
    obstacle.angle = -floorRotation + Math.random()*.3;
    obstacle.body.rotation.y = Math.random() * Math.PI*2;
  }

  obstacle.mesh.rotation.z = floorRotation + obstacle.angle - Math.PI/2;
  obstacle.mesh.position.y = -floorRadius + Math.sin(floorRotation+obstacle.angle) *
(floorRadius+3);
  obstacle.mesh.position.x = Math.cos(floorRotation+obstacle.angle) * (floorRadius+3);

}

function updateFloorRotation(){
  floorRotation += delta*.03 * speed;
  floorRotation = floorRotation%(Math.PI*2);
  floor.rotation.z = floorRotation;
}

function createObstacle(){
  obstacle = new Hedgehog();
  obstacle.body.rotation.y = -Math.PI/2;
  obstacle.mesh.scale.set(1.1,1.1,1.1);
  obstacle.mesh.position.y = floorRadius+4;
  obstacle.nod();
  scene.add(obstacle.mesh);
}

function createBonusParticles(){
  bonusParticles = new BonusParticles();
  bonusParticles.mesh.visible = false;
  scene.add(bonusParticles.mesh);

}



function checkCollision(){
  var db = hero.mesh.position.clone().sub(carrot.mesh.position.clone());
  var dm = hero.mesh.position.clone().sub(obstacle.mesh.position.clone());

  if(db.length() < collisionBonus){
    getBonus();
  }

  if(dm.length() < collisionObstacle && obstacle.status != "flying"){
    getMalus();
  }
}

function getBonus(){
  bonusParticles.mesh.position.copy(carrot.mesh.position);
  bonusParticles.mesh.visible = true;
  bonusParticles.explose();
  carrot.angle += Math.PI/2;
  //speed*=.95;
  monsterPosTarget += .025;

}

function getMalus(){
  obstacle.status="flying";
  var tx = (Math.random()>.5)? -20-Math.random()*10 : 20+Math.random()*5;
  TweenMax.to(obstacle.mesh.position, 4, {x:tx, y:Math.random()*50, z:350,
ease:Power4.easeOut});
  TweenMax.to(obstacle.mesh.rotation, 4, {x:Math.PI*3, z:Math.PI*3, y:Math.PI*6,
ease:Power4.easeOut, onComplete:function(){
    obstacle.status = "ready";
    obstacle.body.rotation.y = Math.random() * Math.PI*2;
    obstacle.angle = -floorRotation - Math.random()*.4;

    obstacle.angle = obstacle.angle%(Math.PI*2);
    obstacle.mesh.rotation.x = 0;
    obstacle.mesh.rotation.y = 0;
    obstacle.mesh.rotation.z = 0;
    obstacle.mesh.position.z = 0;

  }});
  //
  monsterPosTarget -= .04;
  TweenMax.from(this, .5, {malusClearAlpha:.5, onUpdate:function(){
    renderer.setClearColor(malusClearColor, malusClearAlpha );
  }})
}

function updateDistance(){
  distance += delta*speed;
  var d = distance/2;
  fieldDistance.innerHTML = Math.floor(d);
}
```

```
function updateLevel(){
  if (speed >= maxSpeed) return;
  level++;
  speed += 2;
}

function loop(){
  delta = clock.getDelta();
  updateFloorRotation();

  if (gameStatus == "play"){

    if (hero.status == "running"){
      hero.run();
    }
    updateDistance();
    updateMonsterPosition();
    updateCarrotPosition();
    updateObstaclePosition();
    checkCollision();
  }

  render();
  requestAnimationFrame(loop);
}

function render(){
  renderer.render(scene, camera);
}

window.addEventListener('load', init, false);

function init(event){
  initScreenAnd3D();
  createLights();
  createFloor()
  createHero();
  createMonster();
  createFirs();
  createCarrot();
  createBonusParticles();
  createObstacle();
  initUI();
  resetGame();
  loop();

  //setInterval(hero.blink.bind(hero), 3000);
}

function resetGame(){
  scene.add(hero.mesh);
  hero.mesh.rotation.y = Math.PI/2;
  hero.mesh.position.y = 0;
  hero.mesh.position.z = 0;
  hero.mesh.position.x = 0;

  monsterPos = .56;
  monsterPosTarget = .65;
  speed = initSpeed;
  level = 0;
  distance = 0;
  carrot.mesh.visible = true;
  obstacle.mesh.visible = true;
  gameStatus = "play";
  hero.status = "running";
  hero.nod();
  audio.play();
  updateLevel();
  levelInterval = setInterval(updateLevel, levelUpdateFreq);
}

function initUI(){
  fieldDistance = document.getElementById("distValue");
  fieldGameOver = document.getElementById("gameoverInstructions");

}



////////////////////////////////////////////////
//                              MODELS
////////////////////////////////////////////////

// TREE

Tree = function(){
            this.mesh = new THREE.Object3D();
            this.trunc = new Trunc();
            this.mesh.add(this.trunc.mesh);
}


Trunc = function(){
  var truncHeight = 50 + Math.random()*150;
  var topRadius = 1+Math.random()*5;
  var bottomRadius = 5+Math.random()*5;
  var mats = [blackMat, brownMat, pinkMat, whiteMat, greenMat, lightBrownMat, pinkMat];
  var matTrunc = blackMat;//mats[Math.floor(Math.random()*mats.length)];
  var nhSegments = 3;//Math.ceil(2 + Math.random()*6);
  var nvSegments = 3;//Math.ceil(2 + Math.random()*6);
```

```
 var geom = new THREE.CylinderGeometry(topRadius,bottomRadius,truncHeight,
nhSegments, nvSegments);
 geom.applyMatrix(new THREE.Matrix4().makeTranslation(0,truncHeight/2,0));

 this.mesh = new THREE.Mesh(geom, matTrunc);

 for (var i=0; i<geom.vertices.length; i++){
   var noise = Math.random() ;
   var v = geom.vertices[i];
   v.x += -noise + Math.random()*noise*2;
   v.y += -noise + Math.random()*noise*2;
   v.z += -noise + Math.random()*noise*2;

   geom.computeVertexNormals();

   // FRUITS

   if (Math.random()>.7){
     var size = Math.random()*3;
     var fruitGeometry = new THREE.CubeGeometry(size,size,size,1);
     var matFruit = mats[Math.floor(Math.random()*mats.length)];
     var fruit = new THREE.Mesh(fruitGeometry, matFruit);
     fruit.position.x = v.x;
     fruit.position.y = v.y+3;
     fruit.position.z = v.z;
     fruit.rotation.x = Math.random()*Math.PI;
     fruit.rotation.y = Math.random()*Math.PI;

     this.mesh.add(fruit);
   }

   // BRANCHES

   if (Math.random()>.5 && v.y > 10 && v.y < truncHeight - 10){
     var h = 3 + Math.random()*5;
     var thickness = .2 + Math.random();

     var branchGeometry = new THREE.CylinderGeometry(thickness/2, thickness, h, 3, 1);
     branchGeometry.applyMatrix(new THREE.Matrix4().makeTranslation(0,h/2,0));
     var branch = new THREE.Mesh(branchGeometry, matTrunc);
     branch.position.x = v.x;
     branch.position.y = v.y;
     branch.position.z = v.z;

     var vec = new THREE.Vector3(v.x, 2, v.z);
     var axis = new THREE.Vector3(0,1,0);
     branch.quaternion.setFromUnitVectors(axis, vec.clone().normalize());


     this.mesh.add(branch);
   }
```

```
  }


  this.mesh.castShadow = true;
}
```


DISTANCE
4
CLICK TO JUMP – GRAB THE CARROTS / AVOID THE HEDGEHOGS
OTHER CODEPENS | EPIC.NET

2. AI Chess

HTML

```
<section id="view">
  <aside>
    <div>
      <h3 title="Autoplay">Auto</h3>
      <label for="white-random"><input type="checkbox" id="white-random" /><svg
width="170" height="170" viewBox="0 0 170 170" fill="none"
xmlns="http://www.w3.org/2000/svg" class="white"><use href="#king" /></svg></label>
      <label for="black-random"><input type="checkbox" id="black-random" checked /><svg
width="170" height="170" viewBox="0 0 170 170" fill="none"
xmlns="http://www.w3.org/2000/svg" class="black"><use href="#king" /></svg></label>
    </div>
    <div>
      <h3 title="Actions per Second">APS</h3>
      <label for="speed-slow"><input type="radio" name="speed" id="speed-slow"
/><span>1</span></label>
      <label for="speed-medium"><input type="radio" name="speed" id="speed-medium"
checked /><span>2</span></label>
      <label for="speed-fast"><input type="radio" name="speed" id="speed-fast"
/><span>4</span></label>
      <label for="speed-asap"><input type="radio" name="speed" id="speed-asap"
/><span>20</span></label>
    </div>
    <div>
      <h3 title="Point of View">POV</h3>
      <label for="white-perspective"><input type="radio" name="perspective" id="white-
perspective" checked /><svg width="170" height="170" viewBox="0 0 170 170" fill="none"
xmlns="http://www.w3.org/2000/svg" class="white"><use href="#king" /></svg></label>
      <label for="black-perspective"><input type="radio" name="perspective" id="black-
perspective" /><svg width="170" height="170" viewBox="0 0 170 170" fill="none"
xmlns="http://www.w3.org/2000/svg" class="black"><use href="#king" /></svg></label>
    </div>
  </aside>
  <div id="board"></div>
</section>

<svg aria-hidden style="display: none;" width="170" height="170" viewBox="0 0 170 170"
fill="none" xmlns="http://www.w3.org/2000/svg">
  <g id="pawn">
    <path fill="var(--stroke)" d="M63.9596 75.8417L71.9157 80.049C73.2376 77.5492
73.3078 74.5734 72.1052 72.0141C70.9025 69.4547 68.5669 67.6094 65.7987
67.0316L63.9596 75.8417ZM53.8303 94.9961L61.7863 99.2034L61.7864 99.2034L53.8303
94.9961ZM53.2512 96.2739L44.7646 93.2777L44.7646 93.2777L53.2512
96.2739ZM54.1798 112V121C57.4071 121 60.3874 119.272 61.9907 116.471C63.5939
113.67 63.5747 110.225 61.9403 107.442L54.1798 112ZM114.82 112L107.06
107.442C105.425 110.225 105.406 113.67 107.009 116.471C108.613 119.272 111.593 121
114.82 121V112ZM115.749 96.2739L124.235 93.2776L124.235 93.2776L115.749
96.2739ZM115.17 94.9961L107.214 99.2034L107.214 99.2034L115.17 94.9961ZM105.111
75.9755L104.388 67.0046C101.37 67.2479 98.6777 68.9917 97.2213 71.6453C95.7649
74.2989 95.74 77.5069 97.1551 80.1828L105.111 75.9755ZM101.706 61L94.1937
56.0436C92.3701 58.8076 92.2116 62.3498 93.7811 65.2657C95.3506 68.1816 98.3945 70
101.706 70V61ZM68.294 61V70C71.6055 70 74.6494 68.1816 76.2189 65.2657C77.7884
62.3498 77.6299 58.8076 75.8063 56.0436L68.294 61ZM56.0036 71.6343L45.8743
90.7887L61.7864 99.2034L71.9157 80.049L56.0036 71.6343ZM45.8743 90.7887C45.5683
91.3674 45.1338 92.2321 44.7646 93.2777L61.7378 99.2702C61.7062 99.3598 61.685
99.4043 61.691 99.3911C61.6938 99.3851 61.7021 99.3674 61.718 99.3356C61.7341 99.3035
61.756 99.2608 61.7863 99.2034L45.8743 90.7887ZM44.7646 93.2777C43.8488 95.8717
40.5174 106.509 46.4192 116.558L61.9403 107.442C61.2075 106.194 60.9315 104.686
61.0138 103C61.0971 101.296 61.5308 99.8565 61.7378 99.2702L44.7646
93.2777ZM54.1798 103H54V121H54.1798V103ZM54 103C41.2975 103 31 113.297 31
126H49C49 123.239 51.2386 121 54 121V103ZM31 126C31 138.703 41.2974 149 54
149V131C51.2386 131 49 128.761 49 126H31ZM54 149H116V131H54V149ZM116
149C128.703 149 139 138.703 139 126H121C121 128.761 118.761 131 116 131V149ZM139
126C139 113.297 128.703 103 116 103V121C118.761 121 121 123.239 121 126H139ZM116
103H114.82V121H116V103ZM122.581 116.558C128.483 106.509 125.151 95.8716 124.235
93.2776L107.262 99.2702C107.469 99.8565 107.903 101.296 107.986 103C108.069 104.686
107.793 106.194 107.06 107.442L122.581 116.558ZM124.235 93.2776C123.866 92.2321
123.432 91.3674 123.126 90.7887L107.214 99.2034C107.244 99.2608 107.266 99.3035
107.282 99.3356C107.298 99.3674 107.306 99.3851 107.309 99.3912C107.315 99.4043
107.294 99.3598 107.262 99.2702L124.235 93.2776ZM123.126 90.7888L113.067
71.7681L97.1551 80.1828L107.214 99.2034L123.126 90.7888ZM103 68.5C103 67.7083
103.607 67.0675 104.388 67.0046L105.835 84.9463C114.328 84.2614 121 77.1644 121
68.5H103ZM104.5 70C103.672 70 103 69.3284 103 68.5H121C121 59.3873 113.613 52
104.5 52V70ZM101.706 70H104.5V52H101.706V70ZM109.218 65.9564C112.241 61.375
```

```
114 55.8793 114 50H96C96 52.248 95.3352 54.3134 94.1937 56.0436L109.218
65.9564ZM114 50C114 33.9837 101.016 21 85 21V39C91.0751 39 96 43.9249 96
50H114ZM85 21C68.9837 21 56 33.9837 56 50H74C74 43.9249 78.9249 39 85 39V21ZM56
50C56 55.8793 57.7591 61.375 60.7818 65.9564L75.8063 56.0436C74.6648 54.3134 74
52.248 74 50H56ZM65.5 70H68.294V52H65.5V70ZM67 68.5C67 69.3284 66.3284 70 65.5
70V52C56.3873 52 49 59.3873 49 68.5H67ZM65.7987 67.0316C66.4826 67.1743 67 67.7665
67 68.5H49C49 76.4618 54.6302 83.0882 62.1205 84.6518L65.7987 67.0316Z" />
   <path fill-rule="evenodd" clip-rule="evenodd" d="M63.9596 75.8417L53.8303
94.9961C53.6109 95.4111 53.4075 95.8312 53.2512 96.2739C52.4848 98.4448 50.4371
105.627 54.1798 112H54C46.268 112 40 118.268 40 126C40 133.732 46.268 140 54
140H116C123.732 140 130 133.732 130 126C130 118.268 123.732 112 116
112H114.82C118.563 105.627 116.515 98.4447 115.749 96.2739C115.592 95.8312 115.389
95.4111 115.17 94.9961L105.111 75.9755C108.967 75.6645 112 72.4364 112 68.5C112
64.3579 108.642 61 104.5 61H101.706C103.788 57.8442 105 54.0636 105 50C105 38.9543
96.0457 30 85 30C73.9543 30 65 38.9543 65 50C65 54.0636 66.2119 57.8442 68.294
61H65.5C61.3579 61 58 64.3579 58 68.5C58 72.1142 60.5564 75.1312 63.9596 75.8417Z" />
  </g>

  <g id="rook">
   <path fill="var(--stroke)" d="M96 40H105C105 35.0294 100.971 31 96 31V40ZM75
40V31C70.0294 31 66 35.0294 66 40H75ZM75 53L75 62C77.3869 62 79.6761 61.0518
81.364 59.364C83.0518 57.6761 84 55.387 84 53H75ZM58 53H49C49 57.9706 53.0294 62
58 62L58 53ZM58 40H67C67 35.0294 62.9706 31 58 31V40ZM39 40V31C34.0294 31 30
35.0294 30 40H39ZM39 69H30C30 69.379 30.0239 69.7577 30.0717 70.1337L39
69ZM46.8667 130.951L51.4589 138.691C54.5432 136.861 56.2467 133.375 55.795
129.817L46.8667 130.951ZM123.133 130.951L114.205 129.817C113.753 133.375 115.457
136.861 118.541 138.691L123.133 130.951ZM131 69L139.928 70.1338C139.976 69.7577
140 69.3791 140 69L131 69ZM131 40H140C140 35.0294 135.971 31 131 31V40ZM113
40V31C108.029 31 104 35.0294 104 40H113ZM113 53L113 62C115.387 62 117.676 61.0518
119.364 59.364C121.052 57.6761 122 55.387 122 53H113ZM96 53H87C87 57.9706 91.0294
62 96 62L96 53ZM96 31H75V49H96V31ZM66 40V53H84V40H66ZM75 44L58 44L58
62L75 62L75 44ZM67 53V40H49V53H67ZM58 31H39V49H58V31ZM30
40V69H48V40H30ZM30.0717 70.1337L37.9384 132.085L55.795 129.817L47.9283
67.8663L30.0717 70.1337ZM42.2745 123.211C35.5441 127.204 30.9998 134.566 30.9998
143H48.9998C48.9998 141.18 49.9671 139.576 51.4589 138.691L42.2745
123.211ZM30.9998 143C30.9998 155.703 41.2973 166 53.9998 166V148C51.2384 148
48.9998 145.761 48.9998 143H30.9998ZM53.9998 166H116V148H53.9998V166ZM116
166C128.702 166 139 155.703 139 143H121C121 145.761 118.761 148 116 148V166ZM139
143C139 134.567 134.456 127.204 127.725 123.211L118.541 138.691C120.033 139.576 121
141.18 121 143H139ZM132.061 132.085L139.928 70.1338L122.072 67.8662L114.205
129.817L132.061 132.085ZM140 69V40H122V69H140ZM131 31H113V49H131V31ZM104
40V53H122V40H104ZM113 44L96 44L96 62L113 62L113 44ZM105 53V40H87V53H105Z"
/>
   <path fill-rule="evenodd" clip-rule="evenodd" d="M96 40H75V53L58
53V40H39V69L46.8667 130.951C42.7556 133.39 39.9998 137.873 39.9998 143C39.9998
150.732 46.2678 157 53.9998 157H116C123.732 157 130 150.732 130 143C130 137.873
127.244 133.39 123.133 130.951L131 69V40H113V53L96 53V40Z" />
  </g>


  <g id="knight">
  <path fill="var(--stroke)" d="M49.1052 38.3193L55.4587 44.6937L55.4587
44.6937L49.1052 38.3193ZM94.2401 29.9368L87.874 36.2986L87.8741 36.2987L94.2401
29.9368ZM47.3161 124.557L55.6423 121.14L55.6423 121.14L47.3161 124.557ZM51.766
130.503L55.8329 138.532C58.5612 137.15 60.401 134.477 60.7176 131.435C61.0343
128.393 59.7845 125.399 57.3994 123.484L51.766 130.503ZM127.628 131.204L122.091
124.109C119.79 125.905 118.502 128.705 118.637 131.621C118.773 134.537 120.314
137.207 122.772 138.782L127.628 131.204ZM132.387 123.702L141.165 125.687L141.165
125.687L132.387 123.702ZM127.584 110.339L133.091 103.22L133.091 103.22L127.584
110.339ZM110.576 97.1821L105.069 104.301L105.069 104.301L110.576 97.1821ZM126.76
52.2539L121.858 59.802L121.858 59.802L126.76 52.2539ZM99.5001 34.552L104.402
27.0039L104.402 27.0039L99.5001 34.552ZM94.4956 30.1927L100.866 23.8348L100.866
23.8347L94.4956 30.1927ZM55.4587 44.6937C66.4763 33.7121 74.1214 31.8771 78.4041
32.0063C82.8219 32.1396 86.0037 34.427 87.874 36.2986L100.606 23.575C96.9524 19.9187
89.6658 14.3378 78.9468 14.0145C68.0925 13.6871 55.9323 18.8073 42.7516
31.9449L55.4587 44.6937ZM55.6423 121.14C51.8581 111.919 46.772 97.1895 45.3694
82.2455C43.9517 67.1402 46.4349 53.688 55.4587 44.6937L42.7516 31.9449C28.1598
46.4892 25.8204 66.5842 27.4482 83.9274C29.091 101.432 34.9182 118.052 38.99
127.974L55.6423 121.14ZM57.3994 123.484C56.6293 122.866 56.029 122.082 55.6423
121.14L38.99 127.974C40.5712 131.827 43.0657 135.061 46.1326 137.522L57.3994
123.484ZM53.0838 143C53.0838 141.063 54.183 139.368 55.8329 138.532L47.699
122.475C40.2331 126.256 35.0838 134.019 35.0838 143H53.0838ZM58.0838 148C55.3224
148 53.0838 145.761 53.0838 143H35.0838C35.0838 155.703 45.3812 166 58.0838
166V148ZM120.084 148H58.0838V166H120.084V148ZM125.084 143C125.084 145.761
122.845 148 120.084 148V166C132.786 166 143.084 155.703 143.084
143H125.084ZM122.772 138.782C124.181 139.685 125.084 141.24 125.084
143H143.084C143.084 134.852 138.84 127.7 132.484 123.627L122.772 138.782ZM123.608
121.717C123.387 122.695 122.843 123.522 122.091 124.109L133.165 138.3C137.095
135.233 140 130.837 141.165 125.687L123.608 121.717ZM122.077 117.458C123.372
118.459 123.969 120.121 123.608 121.717L141.165 125.687C143.069 117.265 139.92
108.503 133.091 103.22L122.077 117.458ZM105.069 104.301L122.077 117.458L133.091
103.22L116.083 90.0634L105.069 104.301ZM98.3876 90.6851C98.3876 96.0131 100.855
101.041 105.069 104.301L116.083 90.0634C116.275 90.2123 116.388 90.4418 116.388
90.6851H98.3876ZM115.602 73.471C106.095 73.471 98.3876 81.178 98.3876
90.6851H116.388C116.388 91.1191 116.036 91.471 115.602 91.471V73.471ZM120.815
73.471H115.602V91.471H120.815V73.471ZM125.243 69.0432C125.243 71.4886 123.261
73.471 120.815 73.471V91.471C133.202 91.471 143.243 81.4297 143.243
69.0432H125.243ZM125.243 66.0369V69.0432H143.243V66.0369H125.243ZM121.858
59.802C123.969 61.1731 125.243 63.5195 125.243 66.0369H143.243C143.243 57.4241
138.885 49.3965 131.661 44.7058L121.858 59.802ZM94.5985 42.1001L121.858
59.802L131.661 44.7058L104.402 27.0039L94.5985 42.1001ZM88.1255 36.5506C89.5726
38.0004 91.8096 40.2891 94.5985 42.1001L104.402 27.0039C103.476 26.4028 102.551
25.5233 100.866 23.8348L88.1255 36.5506ZM87.8741 36.2987C87.9569 36.3816 88.04
36.4649 88.1256 36.5506L100.866 23.8347C100.781 23.7498 100.694 23.6625 100.606
23.5749L87.8741 36.2987Z" />
  <path fill-rule="evenodd" clip-rule="evenodd" d="M94.2401 29.9368C88.716 24.409
73.3034 14.2001 49.1052 38.3193C25.4895 61.8579 39.4602 105.414 47.3161
124.557C48.3001 126.955 49.8475 128.963 51.766 130.503C47.208 132.812 44.0838 137.541
44.0838 143C44.0838 150.732 50.3518 157 58.0838 157H120.084C127.816 157 134.084
150.732 134.084 143C134.084 138.046 131.511 133.693 127.628 131.204C129.969 129.378
131.694 126.766 132.387 123.702C133.519 118.693 131.646 113.481 127.584
110.339L110.576 97.1821C108.565 95.6265 107.388 93.2275 107.388 90.6851C107.388
```

```
86.1486 111.065 82.471 115.602 82.471H120.815C128.231 82.471 134.243 76.4592 134.243
69.0432V66.0369C134.243 60.4718 131.427 55.2848 126.76 52.2539L99.5001
34.552C97.6428 33.3459 96.0618 31.7619 94.4956 30.1927C94.4104 30.1073 94.3252 30.022
94.2401 29.9368Z" />
  </g>

  <g id="bishop">
    <path fill="var(--stroke)" d="M97.7142 44.0856L89.4897 40.4306L87.1055
45.7955L91.0547 50.1396L97.7142 44.0856ZM107.204 54.5248L113.843 60.6014L119.386
54.5454L113.864 48.4708L107.204 54.5248ZM90.0638 73.2519L96.7027 79.3284L96.7027
79.3284L90.0638 73.2519ZM90.5325 83.8481L84.4559 90.4871L84.4559 90.4871L90.5325
83.8481ZM101.129 83.3795L107.768 89.456L107.768 89.456L101.129 83.3795ZM117.337
65.6709L123.997 59.6168L117.36 52.3162L110.698 59.5943L117.337 65.6709ZM122.204
71.0243L115.544 77.0783L115.544 77.0783L122.204 71.0243ZM126.158 88.1953L134.795
90.7267L134.795 90.7267L126.158 88.1953ZM115.792 123.563L107.155 121.031L107.155
121.031L115.792 123.563ZM113.283 128.798L105.902 123.648L96.0296
137.798H113.283V128.798ZM54.7167 128.798V137.798H71.9704L62.0977
123.648L54.7167 128.798ZM52.2081 123.563L60.8447 121.031L60.8447 121.031L52.2081
123.563ZM41.8417 88.1953L33.2051 90.7267L33.2051 90.7267L41.8417
88.1953ZM45.7961 71.0243L52.4556 77.0783L52.4556 77.0783L45.7961
71.0243ZM70.2858 44.0856L76.9453 50.1396L80.8945 45.7955L78.5103 40.4306L70.2858
44.0856ZM105.939 47.7406C107.267 44.7523 108 41.4498 108 38H90C90 38.8832 89.8148
39.6991 89.4897 40.4306L105.939 47.7406ZM113.864 48.4708L104.374 38.0315L91.0547
50.1396L100.545 60.5789L113.864 48.4708ZM96.7027 79.3284L113.843 60.6014L100.565
48.4483L83.4248 67.1754L96.7027 79.3284ZM96.609 77.2092C97.2201 77.7685 97.2621
78.7173 96.7027 79.3284L83.4248 67.1754C77.2722 73.8975 77.7338 84.3345 84.4559
90.4871L96.609 77.2092ZM94.4898 77.3029C95.0491 76.6918 95.9979 76.6499 96.609
77.2092L84.4559 90.4871C91.178 96.6397 101.615 96.1781 107.768 89.456L94.4898
77.3029ZM110.698 59.5943L94.4898 77.3029L107.768 89.456L123.976 71.7474L110.698
59.5943ZM128.863 64.9702L123.997 59.6168L110.678 71.7249L115.544 77.0783L128.863
64.9702ZM134.795 90.7267C137.442 81.6942 135.195 71.9349 128.863 64.9702L115.544
77.0783C117.655 79.3999 118.404 82.653 117.522 85.6638L134.795 90.7267ZM124.429
126.094L134.795 90.7267L117.522 85.6638L107.155 121.031L124.429 126.094ZM120.664
133.948C122.307 131.594 123.591 128.951 124.429 126.094L107.155 121.031C106.874
121.992 106.446 122.869 105.902 123.648L120.664 133.948ZM116
119.798H113.283V137.798H116V119.798ZM139 142.798C139 130.096 128.703 119.798
116 119.798V137.798C118.761 137.798 121 140.037 121 142.798H139ZM116
165.798C128.703 165.798 139 155.501 139 142.798H121C121 145.56 118.761 147.798 116
147.798V165.798ZM54 165.798H116V147.798H54V165.798ZM31 142.798C31 155.501
41.2974 165.798 54 165.798V147.798C51.2386 147.798 49 145.56 49 142.798H31ZM54
119.798C41.2975 119.798 31 130.096 31 142.798H49C49 140.037 51.2386 137.798 54
137.798V119.798ZM54.7167 119.798H54V137.798H54.7167V119.798ZM43.5714
126.094C44.4088 128.951 45.6929 131.594 47.3357 133.948L62.0977 123.648C61.5539
122.869 61.1262 121.992 60.8447 121.031L43.5714 126.094ZM33.2051 90.7267L43.5714
126.094L60.8447 121.031L50.4784 85.6638L33.2051 90.7267ZM39.1366 64.9702C32.8051
71.9349 30.5576 81.6942 33.2051 90.7267L50.4784 85.6638C49.5959 82.653 50.3451
79.3999 52.4556 77.0783L39.1366 64.9702ZM63.6264 38.0315L39.1366 64.9702L52.4556
77.0783L76.9453 50.1396L63.6264 38.0315ZM60 38C60 41.4498 60.7334 44.7523 62.0614
47.7406L78.5103 40.4306C78.1852 39.6991 78 38.8832 78 38H60ZM84 14C70.7452 14 60
24.7452 60 38H78C78 34.6863 80.6863 32 84 32V14ZM108 38C108 24.7452 97.2548 14 84
14V32C87.3137 32 90 34.6863 90 38H108Z" />
```

```
    <path fill-rule="evenodd" clip-rule="evenodd" d="M99 38C99 40.1665 98.5407 42.2257
97.7142 44.0856L107.204 54.5248L90.0638 73.2519C87.2671 76.3074 87.477 81.0515
90.5325 83.8481C93.588 86.6448 98.3321 86.435 101.129 83.3795L117.337
65.6709L122.204 71.0243C126.425 75.6674 127.923 82.1736 126.158 88.1953L115.792
123.563C115.233 125.471 114.377 127.231 113.283 128.798H116C123.732 128.798 130
135.066 130 142.798C130 150.53 123.732 156.798 116 156.798H54C46.268 156.798 40
150.53 40 142.798C40 135.066 46.268 128.798 54 128.798H54.7167C53.6234 127.231
52.7675 125.471 52.2081 123.563L41.8417 88.1953C40.0767 82.1736 41.5751 75.6674
45.7961 71.0243L70.2858 44.0856C69.4593 42.2257 69 40.1665 69 38C69 29.7157 75.7157
23 84 23C92.2843 23 99 29.7157 99 38Z" />
  </g>

  <g id="queen">
    <path fill="var(--stroke)" d="M98.3201 39.3222L89.7589 36.5462C88.2688 41.1416
90.6725 46.0937 95.2048 47.7659L98.3201 39.3222ZM71.6799 39.3222L74.7952
47.7659C79.3275 46.0937 81.7312 41.1416 80.2411 36.5462L71.6799 39.3222ZM54.2398
58V67C58.658 67 62.4228 63.7931 63.1253 59.4312L54.2398 58ZM23.8059 83.7931L32.42
81.1862L32.42 81.1862L23.8059 83.7931ZM34.7006 119.793L26.0864 122.4L26.0864
122.4L34.7006 119.793ZM45.4311 132.146L51.1198 139.12C53.5226 137.16 54.7472
134.101 54.3609 131.025C53.9746 127.948 52.0318 125.287 49.2191 123.982L45.4311
132.146ZM124.913 131.982L120.97 123.892C118.183 125.25 116.292 127.947 115.963
131.029C115.635 134.112 116.917 137.147 119.355 139.061L124.913 131.982ZM135.299
119.793L126.685 117.186L126.685 117.186L135.299 119.793ZM146.194 83.7932L137.58
81.1862L137.58 81.1862L146.194 83.7932ZM115.76 58L106.875 59.4312C107.577 63.7931
111.342 67 115.76 67V58ZM90 35C90 35.5543 89.9131 36.0707 89.7589 36.5462L106.881
42.0983C107.61 39.8513 108 37.4624 108 35H90ZM85 30C87.7614 30 90 32.2386 90
35H108C108 22.2975 97.7026 12 85 12V30ZM80 35C80 32.2386 82.2386 30 85
30V12C72.2975 12 62 22.2975 62 35H80ZM80.2411 36.5462C80.0869 36.0707 80 35.5543
80 35H62C62 37.4624 62.3902 39.8513 63.1188 42.0983L80.2411 36.5462ZM63.1253
59.4312C63.7988 55.2498 67.5604 50.4352 74.7952 47.7659L68.5647 30.8786C57.1112
35.1044 47.3444 44.2138 45.3544 56.5688L63.1253 59.4312ZM54.2398
49H42.9485V67H54.2398V49ZM42.9485 49C23.4932 49 9.55633 67.7789 15.1917
86.4001L32.42 81.1862C30.2825 74.123 35.5689 67 42.9485 67V49ZM15.1917
86.4001L26.0864 122.4L43.3148 117.186L32.42 81.1862L15.1917 86.4001ZM26.0864
122.4C28.5395 130.506 34.3182 136.911 41.643 140.31L49.2191 123.982C46.429 122.687
44.2438 120.256 43.3148 117.186L26.0864 122.4ZM39.7423 125.172C34.5896 129.375
31.2741 135.804 31.2741 143H49.2741C49.2741 141.44 49.978 140.051 51.1198
139.12L39.7423 125.172ZM31.2741 143C31.2741 155.703 41.5715 166 54.2741
166V148C51.5127 148 49.2741 145.761 49.2741 143H31.2741ZM54.2741
166H116.274V148H54.2741V166ZM116.274 166C128.977 166 139.274 155.703 139.274
143H121.274C121.274 145.761 119.036 148 116.274 148V166ZM139.274 143C139.274
135.648 135.815 129.099 130.471 124.904L119.355 139.061C120.54 139.991 121.274
141.407 121.274 143H139.274ZM128.856 140.073C135.943 136.619 141.516 130.321
143.914 122.4L126.685 117.186C125.777 120.186 123.67 122.576 120.97 123.892L128.856
140.073ZM143.914 122.4L154.808 86.4001L137.58 81.1862L126.685 117.186L143.914
122.4ZM154.808 86.4001C160.444 67.7789 146.507 49 127.052 49V67C134.431 67 139.718
74.123 137.58 81.1862L154.808 86.4001ZM127.052 49H115.76V67H127.052V49ZM95.2048
47.7659C102.44 50.4352 106.201 55.2498 106.875 59.4312L124.646 56.5688C122.656
44.2138 112.889 35.1044 101.435 30.8786L95.2048 47.7659Z" />
    <path fill-rule="evenodd" clip-rule="evenodd" d="M98.3201 39.3222C98.7615 37.961 99
36.5084 99 35C99 27.268 92.732 21 85 21C77.268 21 71 27.268 71 35C71 36.5084 71.2385
```

```
37.961 71.6799 39.3222C62.3358 42.7698 55.5716 49.7318 54.2398 58H42.9485C29.5311 58
19.9194 70.951 23.8059 83.7931L34.7006 119.793C36.3917 125.381 40.3736 129.799
45.4311 132.146C42.2838 134.713 40.2741 138.622 40.2741 143C40.2741 150.732 46.5421
157 54.2741 157H116.274C124.006 157 130.274 150.732 130.274 143C130.274 138.528
128.177 134.545 124.913 131.982C129.807 129.597 133.647 125.254 135.299
119.793L146.194 83.7932C150.081 70.951 140.469 58 127.052 58H115.76C114.428 49.7318
107.664 42.7698 98.3201 39.3222Z" />
  </g>

  <g id="king">
    <path fill="var(--stroke)" d="M71.3981 29V38C74.4851 38 77.3566 36.4179 79.0059
33.8085C80.6552 31.1991 80.8521 27.9265 79.5275 25.1382L71.3981 29ZM55.5392
47V56C59.196 56 62.4893 53.7874 63.8715 50.4019C65.2537 47.0165 64.4501 43.1312
61.8383 40.5718L55.5392 47ZM16.8101 72.5915L25.4512 70.0754L25.4512
70.0754L16.8101 72.5915ZM30.4959 119.592L39.137 117.075L39.137 117.075L30.4959
119.592ZM44.3251 133.266L50.7933 139.524C52.9992 137.244 53.8418 133.971 53.011
130.909C52.1801 127.848 49.7986 125.449 46.7427 124.597L44.3251 133.266ZM126.438
133.253L123.999 124.59C120.947 125.449 118.572 127.851 117.747 130.913C116.923
133.974 117.77 137.244 119.978 139.52L126.438 133.253ZM140.22 119.592L148.861
122.108L148.861 122.108L140.22 119.592ZM153.906 72.5915L145.265 70.0753L145.265
70.0753L153.906 72.5915ZM115.236 47L108.937 40.5718C106.325 43.1312 105.522
47.0165 106.904 50.4019C108.286 53.7874 111.579 56 115.236 56V47ZM99.3773
29L91.2479 25.1382C89.9233 27.9265 90.1202 31.1991 91.7695 33.8085C93.4188 36.4179
96.2903 38 99.3773 38V29ZM79.3877 24.5C79.3877 23.6716 80.0593 23 80.8877
23V5C70.1181 5 61.3877 13.7304 61.3877 24.5H79.3877ZM79.5275 25.1382C79.4407
24.9555 79.3877 24.7465 79.3877 24.5H61.3877C61.3877 27.4743 62.0598 30.3168 63.2688
32.8618L79.5275 25.1382ZM71.3981 20H62.8877V38H71.3981V20ZM62.8877 20C52.1181
20 43.3877 28.7304 43.3877 39.5H61.3877C61.3877 38.6716 62.0593 38 62.8877
38V20ZM43.3877 39.5C43.3877 44.9545 45.639 49.8994 49.2401 53.4282L61.8383
40.5718C61.5498 40.2891 61.3877 39.9216 61.3877 39.5H43.3877ZM55.5392
38H36.0125V56H55.5392V38ZM36.0125 38C16.6845 38 2.7653 56.5505 8.16895
75.1077L25.4512 70.0754C23.4015 63.0364 28.6812 56 36.0125 56V38ZM8.16895
75.1077L21.8548 122.108L39.137 117.075L25.4512 70.0754L8.16895 75.1077ZM21.8548
122.108C24.7108 131.916 32.4229 139.291 41.9074 141.936L46.7427 124.597C43.1399
123.592 40.2185 120.789 39.137 117.075L21.8548 122.108ZM37.8569 127.008C33.861
131.139 31.3877 136.79 31.3877 143H49.3877C49.3877 141.646 49.9143 140.433 50.7933
139.524L37.8569 127.008ZM31.3877 143C31.3877 155.703 41.6852 166 54.3877
166V148C51.6263 148 49.3877 145.761 49.3877 143H31.3877ZM54.3877
166H116.388V148H54.3877V166ZM116.388 166C129.09 166 139.388 155.703 139.388
143H121.388C121.388 145.761 119.149 148 116.388 148V166ZM139.388 143C139.388
136.779 136.906 131.119 132.898 126.987L119.978 139.52C120.859 140.429 121.388
141.643 121.388 143H139.388ZM128.876 141.917C138.33 139.256 146.012 131.893 148.861
122.108L131.579 117.075C130.5 120.781 127.59 123.579 123.999 124.59L128.876
141.917ZM148.861 122.108L162.547 75.1077L145.265 70.0753L131.579 117.075L148.861
122.108ZM162.547 75.1077C167.951 56.5504 154.032 38 134.704 38V56C142.035 56
147.315 63.0364 145.265 70.0753L162.547 75.1077ZM134.704
38H115.236V56H134.704V38ZM121.535 53.4282C125.136 49.8994 127.388 44.9545
127.388 39.5H109.388C109.388 39.9216 109.226 40.2891 108.937 40.5718L121.535
53.4282ZM127.388 39.5C127.388 28.7304 118.657 20 107.888 20V38C108.716 38 109.388
38.6716 109.388 39.5H127.388ZM107.888 20H99.3773V38H107.888V20ZM91.3877
24.5C91.3877 24.7465 91.3347 24.9555 91.2479 25.1382L107.507 32.8618C108.716 30.3168
```

```
109.388 27.4743 109.388 24.5H91.3877ZM89.8877 23C90.7161 23 91.3877 23.6716 91.3877
24.5H109.388C109.388 13.7304 100.657 5 89.8877 5V23ZM80.8877
23H89.8877V5H80.8877V23Z" />
    <path fill-rule="evenodd" clip-rule="evenodd" d="M80.8877 14C75.0887 14 70.3877
18.701 70.3877 24.5C70.3877 26.1104 70.7502 27.6361 71.3981 29H62.8877C57.0887 29
52.3877 33.701 52.3877 39.5C52.3877 42.438 53.5944 45.0942 55.5392
47H36.0125C22.6829 47 13.0834 59.7934 16.8101 72.5915L30.4959 119.592C32.4647
126.353 37.7814 131.441 44.3251 133.266C41.8876 135.786 40.3877 139.218 40.3877
143C40.3877 150.732 46.6557 157 54.3877 157H116.388C124.12 157 130.388 150.732
130.388 143C130.388 139.211 128.883 135.774 126.438 133.253C132.96 131.417 138.256
126.337 140.22 119.592L153.906 72.5915C157.633 59.7934 148.033 47 134.704
47H115.236C117.181 45.0942 118.388 42.438 118.388 39.5C118.388 33.701 113.687 29
107.888 29H99.3773C100.025 27.6361 100.388 26.1104 100.388 24.5C100.388 18.701
95.6867 14 89.8877 14H80.8877Z" />
  </g>
</svg>
```

CSS

```
:root {
  --border-width: calc(var(--diameter-tile) / 60);
  --diameter-board: min(85vw, 85vh);
  --diameter-tile: calc(1 / 8 * var(--diameter-board));
  --edge-width: calc((min(100vw, 100vh) - var(--diameter-board)) * 0.3);
  --color-danger: tomato;
  --color-success: #1d83e0;
  --color-white: #f0f0f0;
  --color-black: #222;
  --color-board-hue: 30;
  --color-board-sat: 40%;
  --color-shadow: hsl(var(--color-board-hue), var(--color-board-sat), 50%);
  --color-shadow-lighter: hsl(var(--color-board-hue), var(--color-board-sat), 55%);
  --transition-ease: cubic-bezier(0.25, 1, 0.5, 1);
  --color-background: var(--color-black);
}

aside {
  display: flex;
  justify-content: space-between;
  left: 0;
  position: absolute;
  top: calc(var(--edge-width) * -0.55);
  transform: translateY(-50%);
  width: 100%;
  z-index: 999;
}
aside div {
  align-items: center;
  color: white;
  display: flex;
```

```
}
aside div > * {
  align-items: center;
  display: flex;
}
aside div > * + * {
  margin-left: calc(var(--border-width) * 2);
}
aside div h3,
aside div label {
  font-size: calc(var(--edge-width) * 0.3);
  height: calc(var(--edge-width) * 0.3);
  line-height: 1;
  margin-bottom: 0;
  margin-top: 0;
  text-transform: uppercase;
}
aside div label {
  cursor: pointer;
}
aside div input {
  left: -99999px;
  position: absolute;
}

aside div input + * {
  opacity: 0.5;
}
aside div input:checked + * {
  font-weight: bold;
  opacity: 1;
}

aside div svg {
  height: calc(var(--edge-width) * 0.5);
  width: auto;
}

html,
body {
  height: 100%;
}

body {
  background: var(--color-background);
  overflow: hidden;
  transition: background-color 250ms ease-in-out;
}

#view {
  background: var(--color-shadow-lighter);
```

```css
  box-shadow: 0 0 0 calc(var(--border-width) * 3) var(--color-shadow-lighter),
    0 0 0 var(--edge-width) var(--color-shadow);
  height: var(--diameter-board);
  margin: calc((100vh - var(--diameter-board)) * 0.5)
    calc((100vw - var(--diameter-board)) * 0.5);
  position: relative;
  width: var(--diameter-board);
}

.board {
  display: flex;
  flex-direction: column-reverse;
  height: 100%;
  width: 100%;
}

.board .row {
  display: flex;
  height: var(--diameter-tile);
  width: 100%;
}

.perspective-black .board .row {
  flex-direction: row-reverse;
}

.perspective-black .board {
  flex-direction: column;
}

.board .row .tile {
  background-color: currentcolor;
  border: none;
  box-shadow: inset 0 0 0 var(--border-width) var(--color-shadow-lighter);
  display: flex;
  flex-direction: column;
  height: var(--diameter-tile);
  justify-content: space-between;
  padding: 0;
  position: relative;
  transition: background-color 350ms var(--transition-ease);
  width: var(--diameter-tile);
}

.perspective-black .board .row:nth-child(even) .tile:nth-child(odd),
.perspective-black .board .row:nth-child(odd) .tile:nth-child(even),
.perspective-white .board .row:nth-child(even) .tile:nth-child(even),
.perspective-white .board .row:nth-child(odd) .tile:nth-child(odd) {
  color: hsl(var(--color-board-hue), var(--color-board-sat), 62%);
}
```

```
.perspective-black .board .row:nth-child(even) .tile:nth-child(even),
.perspective-black .board .row:nth-child(odd) .tile:nth-child(odd),
.perspective-white .board .row:nth-child(even) .tile:nth-child(odd),
.perspective-white .board .row:nth-child(odd) .tile:nth-child(even) {
  color: hsl(var(--color-board-hue), var(--color-board-sat), 70%);
}
.board .row .tile.highlight-active {}
.board .row .tile.highlight-capture {}
.board .row .tile.highlight-move {}

.board .row .tile .move,
.board .row .tile .moves,
.board .row .tile .captures {
  box-sizing: border-box;
  display: flex;
  flex-wrap: wrap;
  height: var(--diameter-tile);
  justify-content: center;
  left: 0;
  padding: calc(var(--diameter-tile) * 0.025);
  position: absolute;
  top: 0;
  width: var(--diameter-tile);
  z-index: 9;
}

.board .row .tile .move,
.board .row .tile .moves {
  align-content: center;
  align-items: center;
}
.board .row .tile .captures {
  align-items: flex-start;
  justify-content: space-between;
}
.board .row .tile:not(.occupied) .captures {
  align-items: center;
  justify-content: center;
}

.board .row .tile > div > svg {
  --stroke: transparent;
  box-sizing: border-box;
  height: var(--di);
  line-height: var(--di);
  width: var(--di);
}

.board .row .tile .move svg {
  --di: calc(var(--diameter-tile) / 4);
  --fill: var(--color-shadow);
}

.board .row .tile .moves svg,
.board .row .tile .captures svg {
  --di: calc(var(--diameter-tile) / 4);
  --fill: var(--color-shadow);
  opacity: 0.4;
}

.board .row .tile.occupied .captures svg { position: absolute; }
.board .row .tile.occupied .captures svg:nth-child(1) { top: 0; left: 0; }
.board .row .tile.occupied .captures svg:nth-child(2) { top: 0; right: 0; }
.board .row .tile.occupied .captures svg:nth-child(3) { bottom: calc(var(--di) * 0.1); left: 0; }
.board .row .tile.occupied .captures svg:nth-child(4) { bottom: calc(var(--di) * 0.1); right: 0; }
.board .row .tile.occupied .captures svg:nth-child(5) { top: calc(50% - var(--di) * 0.55); left: 0;
}
.board .row .tile.occupied .captures svg:nth-child(6) { top: calc(50% - var(--di) * 0.55); right:
0; }
.board .row .tile.occupied .captures svg:nth-child(7) { top: 0; left: calc(50% - var(--di) * 0.5); }
.board .row .tile.occupied .captures svg:nth-child(8) { bottom: calc(var(--di) * 0.1); left:
calc(50% - var(--di) * 0.5); }

.touching .board .row .tile .moves,
.touching .board .row .tile .captures,
.turn-black .board .row .tile .moves .white,
.turn-black .board .row .tile .captures .white,
.turn-white .board .row .tile .moves .black,
.turn-white .board .row .tile .captures .black {
  display: none;
}

.board .row .tile[class*="highlight-"] .moves,
.board .row .tile[class*="highlight-"] .captures {
  display: none;
}

button:focus {
  outline: none;
  position: relative;
  z-index: 9;
}

svg {
  --fill: var(--color-black);
  --stroke: var(--color-shadow);
  fill: var(--fill);
}

svg.white { --fill: var(--color-white); }
svg.black { --fill: var(--color-black); }
```

```
.pieces {
  display: block;
  height: var(--diameter-board);
  left: 0;
  pointer-events: none;
  position: absolute;
  top: 0;
  width: var(--diameter-board);
  z-index: 99;
}

.pieces .piece.white {
  --pos-row: -1;
}
.pieces .piece.black {
  --pos-row: 8;
}
.pieces .piece {
  --pos-col: 3.5;
  --scale: 0;
  --transition-delay: 0ms;
  --transition-duration: 200ms;
  bottom: 0;
  display: block;
  height: var(--diameter-tile);
  position: absolute;
  left: 0;
  transform: translate(
      calc(var(--pos-col) * 100%),
      calc(var(--pos-row) * -100%)
    )
    translateZ(0);
  transform-origin: 50% 50%;
  transition: all var(--transition-duration) var(--transition-ease)
    var(--transition-delay);
  width: var(--diameter-tile);
}
.perspective-black .pieces .piece {
  transform: translate(
      calc((7 - var(--pos-col)) * 100%),
      calc((7 - var(--pos-row)) * -100%)
    )
    translateZ(0);
}
.pieces .piece svg {
  display: block;
  left: 50%;
  opacity: 1;
  position: absolute;
  top: 50%;
  transform: translate(-50%, -50%) translateZ(0) scale(var(--scale));
  transform-origin: 50% 50%;
  transition: transform var(--transition-duration) var(--transition-ease),
    fill var(--transition-duration) var(--transition-ease),
    opacity var(--transition-duration) var(--transition-ease);
}
.turn-white .pieces .piece:not(.highlight-capture) svg.black,
.turn-black .pieces .piece:not(.highlight-capture) svg.white,
.turn-black .pieces .piece:not(.can-move):not(.can-capture) svg.black,
.turn-white .pieces .piece:not(.can-move):not(.can-capture) svg.white {
  --stroke: transparent;
  opacity: 0.8;
}

@keyframes wobble {
  0%, 50%, 100% { transform: translate(-50%, -50%) translateZ(0) scale(1) rotate(0deg); }
  25% { transform: translate(-50%, -50%) translateZ(0) scale(1.1) rotate(-2deg); }
  75% { transform: translate(-50%, -50%) translateZ(0) scale(1.1) rotate(2deg); }
}
.pieces .piece.highlight-active svg {
  animation: wobble 500ms linear infinite;
  --stroke: var(--color-success);
}

.pieces .piece.highlight-capture svg {
  --stroke: var(--color-danger);
}

.piece svg {
  --svg-di: calc(var(--diameter-tile) * 0.666);
  display: block;
  font-weight: bold;
  height: var(--svg-di);
  left: 50%;
  line-height: var(--svg-di);
  position: absolute;
  stroke-linejoin: round;
  text-align: center;
  top: 50%;
  transform: translate(-50%, -50%);
  width: var(--svg-di);
```

JS

```
console.clear();

// TODO: url based

type MoveType = "CANCEL" | "CAPTURE" | "INVALID" | "MOVE" | "TOUCH";
// prettier-ignore
type PieceIdBlack = "A8" | "B8" | "C8" | "D8" | "E8" | "F8" | "G8" | "H8" |
                    "A7" | "B7" | "C7" | "D7" | "E7" | "F7" | "G7" | "H7";
// prettier-ignore
type PieceIdWhite = "A2" | "B2" | "C2" | "D2" | "E2" | "F2" | "G2" | "H2" |
                    "A1" | "B1" | "C1" | "D1" | "E1" | "F1" | "G1" | "H1";
type PieceId = PieceIdWhite | PieceIdBlack;
type PieceType = "PAWN" | "ROOK" | "KNIGHT" | "BISHOP" | "QUEEN" | "KING";
type PlayerId = "WHITE" | "BLACK";
type PositionColumn = "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H";
type PositionRow = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8";

interface ActivateResponse {
  activePieceId?: PieceId;
  capturedPieceId?: PieceId;
  captures?: Position[];
  castledId?: PieceId;
  moves?: Position[];
  type: MoveType;
}
interface Options {
  captures: Position[];
  moves: Position[];
}
interface PieceData {
  id: PieceId;
  type: PieceType;
  player: PlayerId;
}
interface PiecePositionOnBoard extends Position {
  active: true;
}
interface PiecePositionOffBoard {
  active: false;
  row?: undefined;
  col?: undefined;
}
interface Position {
  row: PositionRow;
  col: PositionColumn;
  capture?: Position;
  castles?: PositionColumn;
  _moves?: number;
  _promoted?: boolean;
```

```
}

type BoardPieces = { [K in PieceId]: HTMLElement };
type BoardState = { [K in PositionRow]?: { [K in PositionColumn]?: PieceId } };
type BoardTiles = {
  [K in PositionRow]: { [K in PositionColumn]: HTMLElement };
};
type PiecePosition = PiecePositionOnBoard | PiecePositionOffBoard;
type Pieces = { [K in PieceId]: Piece };
type PieceDirResponse = Position | undefined;
type PiecePositions = { [K in PieceId]: PiecePosition };
type PiecesToTiles = { [K in PieceId]?: Position[] };
type TilesToPieces = {
  [K in PositionRow]: { [K in PositionColumn]: PieceId[] };
};
type TileRelation = "FRIEND" | "ENEMY" | "BLANK";

type ConstraintArguments = {
  moveIndex: number;
  piece: Piece;
  pieces: Pieces;
  piecePositions: PiecePositions;
  state: BoardState;
  kingCastles?: (king: Piece) => Position[];
};

type ResultingChecksArguments = {
  piece: Piece;
  location: Position;
  capture: boolean;
  moveIndex: number;
};

let PIECE_DIR_CALC = 0;

class Utils {
  static colToInt(col: PositionColumn): number {
    return Board.COLS.indexOf(col);
  }
  static rowToInt(row: PositionRow): number {
    return Board.ROWS.indexOf(row);
  }
  static intToCol(int: number): PositionColumn {
    return Board.COLS[int];
  }
  static intToRow(int: number): PositionRow {
    return Board.ROWS[int];
  }

  static getPositionsFromShortCode(shortCode: string): PiecePositions {
    const positions = Utils.getInitialPiecePositions();
    const overrides = {};
    const defaultPositionMode = shortCode.charAt(0) === "X";
    if (defaultPositionMode) {
      shortCode = shortCode.slice(1);
    }
    shortCode.split(",").forEach((string) => {
      const promoted = string.charAt(0) === "P";
      if (promoted) {
        string = string.slice(1);
      }
      if (defaultPositionMode) {
        const inactive = string.length === 3;
        const id = string.slice(0, 2);
        const col = inactive ? undefined : string.charAt(2);
        const row = inactive ? undefined : string.charAt(3);
        const moves = string.charAt(4) || "1";
        overrides[id] = {
          col,
          row,
          active: !inactive,
          _moves: parseInt(moves),
          _promoted: promoted,
        };
      } else {
        const moved = string.length >= 4;
        const id = string.slice(0, 2);
        const col = string.charAt(moved ? 2 : 0);
        const row = string.charAt(moved ? 3 : 1);
        const moves = string.charAt(4) || moved ? "1" : "0";
        overrides[id] = { col, row, active: true, _moves: parseInt(moves), _promoted: promoted };
      }
    });
    for (let id in positions) {
      if (overrides[id]) {
        positions[id] = overrides[id];
      } else {
        positions[id] = defaultPositionMode ? positions[id] : { active: false };
      }
    }
    return positions;
  }

  static getInitialBoardPieces(parent: HTMLElement, pieces: Pieces): BoardPieces {
    const boardPieces = {};
    const container = document.createElement("div");
    container.className = "pieces";
    parent.appendChild(container);
    for (let pieceId in pieces) {
      const boardPiece = document.createElement("div");
      boardPiece.className = `piece ${pieces[pieceId].data.player.toLowerCase()}`;
      boardPiece.innerHTML = pieces[pieceId].shape();
```

```
      container.appendChild(boardPiece);
      boardPieces[pieceId] = boardPiece;
    }
    return boardPieces as BoardPieces;
  }

  static getInitialBoardTiles(parent: HTMLElement, handler: (params: Position) => void):
BoardTiles {
    const tiles = { 1: {}, 2: {}, 3: {}, 4: {}, 5: {}, 6: {}, 7: {}, 8: {} };
    const board = document.createElement("div");
    board.className = "board";
    parent.appendChild(board);
    for (let i = 0; i < 8; i++) {
      const row = document.createElement("div");
      row.className = "row";
      board.appendChild(row);
      for (let j = 0; j < 8; j++) {
        const tile = document.createElement("button");
        tile.className = "tile";
        const r = Utils.intToRow(i);
        const c = Utils.intToCol(j);
        tile.addEventListener("click", () => handler({ row: r, col: c }));
        row.appendChild(tile);
        tiles[r][c] = tile;
      }
    }
    return tiles as BoardTiles;
  }

  static getInitialBoardState(construct = () => undefined): any {
    const blankRow = () => ({
      A: construct(),
      B: construct(),
      C: construct(),
      D: construct(),
      E: construct(),
      F: construct(),
      G: construct(),
      H: construct(),
    });
    return {
      1: { ...blankRow() },
      2: { ...blankRow() },
      3: { ...blankRow() },
      4: { ...blankRow() },
      5: { ...blankRow() },
      6: { ...blankRow() },
      7: { ...blankRow() },
      8: { ...blankRow() },
    };
  }
```

```
static getInitialPiecePositions(): PiecePositions {
  return {
    A8: { active: true, row: "8", col: "A" },
    B8: { active: true, row: "8", col: "B" },
    C8: { active: true, row: "8", col: "C" },
    D8: { active: true, row: "8", col: "D" },
    E8: { active: true, row: "8", col: "E" },
    F8: { active: true, row: "8", col: "F" },
    G8: { active: true, row: "8", col: "G" },
    H8: { active: true, row: "8", col: "H" },
    A7: { active: true, row: "7", col: "A" },
    B7: { active: true, row: "7", col: "B" },
    C7: { active: true, row: "7", col: "C" },
    D7: { active: true, row: "7", col: "D" },
    E7: { active: true, row: "7", col: "E" },
    F7: { active: true, row: "7", col: "F" },
    G7: { active: true, row: "7", col: "G" },
    H7: { active: true, row: "7", col: "H" },
    A2: { active: true, row: "2", col: "A" },
    B2: { active: true, row: "2", col: "B" },
    C2: { active: true, row: "2", col: "C" },
    D2: { active: true, row: "2", col: "D" },
    E2: { active: true, row: "2", col: "E" },
    F2: { active: true, row: "2", col: "F" },
    G2: { active: true, row: "2", col: "G" },
    H2: { active: true, row: "2", col: "H" },
    A1: { active: true, row: "1", col: "A" },
    B1: { active: true, row: "1", col: "B" },
    C1: { active: true, row: "1", col: "C" },
    D1: { active: true, row: "1", col: "D" },
    E1: { active: true, row: "1", col: "E" },
    F1: { active: true, row: "1", col: "F" },
    G1: { active: true, row: "1", col: "G" },
    H1: { active: true, row: "1", col: "H" },
  };
}

static getInitialPieces(): Pieces {
  return {
    A8: new Piece({ id: "A8", player: "BLACK", type: "ROOK" }),
    B8: new Piece({ id: "B8", player: "BLACK", type: "KNIGHT" }),
    C8: new Piece({ id: "C8", player: "BLACK", type: "BISHOP" }),
    D8: new Piece({ id: "D8", player: "BLACK", type: "QUEEN" }),
    E8: new Piece({ id: "E8", player: "BLACK", type: "KING" }),
    F8: new Piece({ id: "F8", player: "BLACK", type: "BISHOP" }),
    G8: new Piece({ id: "G8", player: "BLACK", type: "KNIGHT" }),
    H8: new Piece({ id: "H8", player: "BLACK", type: "ROOK" }),
    A7: new Piece({ id: "A7", player: "BLACK", type: "PAWN" }),
    B7: new Piece({ id: "B7", player: "BLACK", type: "PAWN" }),
    C7: new Piece({ id: "C7", player: "BLACK", type: "PAWN" }),
      D7: new Piece({ id: "D7", player: "BLACK", type: "PAWN" }),
      E7: new Piece({ id: "E7", player: "BLACK", type: "PAWN" }),
      F7: new Piece({ id: "F7", player: "BLACK", type: "PAWN" }),
      G7: new Piece({ id: "G7", player: "BLACK", type: "PAWN" }),
      H7: new Piece({ id: "H7", player: "BLACK", type: "PAWN" }),
      A2: new Piece({ id: "A2", player: "WHITE", type: "PAWN" }),
      B2: new Piece({ id: "B2", player: "WHITE", type: "PAWN" }),
      C2: new Piece({ id: "C2", player: "WHITE", type: "PAWN" }),
      D2: new Piece({ id: "D2", player: "WHITE", type: "PAWN" }),
      E2: new Piece({ id: "E2", player: "WHITE", type: "PAWN" }),
      F2: new Piece({ id: "F2", player: "WHITE", type: "PAWN" }),
      G2: new Piece({ id: "G2", player: "WHITE", type: "PAWN" }),
      H2: new Piece({ id: "H2", player: "WHITE", type: "PAWN" }),
      A1: new Piece({ id: "A1", player: "WHITE", type: "ROOK" }),
      B1: new Piece({ id: "B1", player: "WHITE", type: "KNIGHT" }),
      C1: new Piece({ id: "C1", player: "WHITE", type: "BISHOP" }),
      D1: new Piece({ id: "D1", player: "WHITE", type: "QUEEN" }),
      E1: new Piece({ id: "E1", player: "WHITE", type: "KING" }),
      F1: new Piece({ id: "F1", player: "WHITE", type: "BISHOP" }),
      G1: new Piece({ id: "G1", player: "WHITE", type: "KNIGHT" }),
      H1: new Piece({ id: "H1", player: "WHITE", type: "ROOK" }),
    };
  }
}

class Shape {
  static shape(player: string, piece: string) {
    return `<svg class="${player}" width="170" height="170" viewBox="0 0 170 170"
fill="none" xmlns="http://www.w3.org/2000/svg">
      <use href="#${piece}" />
    </svg>`;
  }
  static shapeBishop(player: string) {
    return Shape.shape(player, "bishop");
  }
  static shapeKing(player: string) {
    return Shape.shape(player, "king");
  }
  static shapeKnight(player: string) {
    return Shape.shape(player, "knight");
  }
  static shapePawn(player: string) {
    return Shape.shape(player, "pawn");
  }
  static shapeQueen(player: string) {
    return Shape.shape(player, "queen");
  }
  static shapeRook(player: string) {
    return Shape.shape(player, "rook");
  }
}
```

```
class Constraints {
  static generate(args: ConstraintArguments, resultingChecks?: (args:
ResultingChecksArguments) => PiecePosition[]): Options {
    let method;
    const { piecePositions, piece } = args;
    if (piecePositions[piece.data.id].active) {
      switch (piece.data.type) {
        case "BISHOP":
          method = Constraints.constraintsBishop;
          break;
        case "KING":
          method = Constraints.constraintsKing;
          break;
        case "KNIGHT":
          method = Constraints.constraintsKnight;
          break;
        case "PAWN":
          method = Constraints.constraintsPawn;
          break;
        case "QUEEN":
          method = Constraints.constraintsQueen;
          break;
        case "ROOK":
          method = Constraints.constraintsRook;
          break;
      }
    }
    const result = method ? method(args) : { moves: [], captures: [] };
    if (resultingChecks) {
      const moveIndex = args.moveIndex + 1;
      result.moves = result.moves.filter((location) => !resultingChecks({ piece, location, capture:
false, moveIndex }).length);
      result.captures = result.captures.filter((location) => !resultingChecks({ piece, location,
capture: true, moveIndex }).length);
    }
    return result;
  }

  static constraintsBishop(args: ConstraintArguments): Options {
    return Constraints.constraintsDiagonal(args);
  }

  static constraintsDiagonal(args: ConstraintArguments): Options {
    const response = { moves: [], captures: [] };
    const { piece } = args;

    Constraints.runUntil(piece.dirNW.bind(piece), response, args);
    Constraints.runUntil(piece.dirNE.bind(piece), response, args);
    Constraints.runUntil(piece.dirSW.bind(piece), response, args);
    Constraints.runUntil(piece.dirSE.bind(piece), response, args);
    return response;
  }

  static constraintsKing(args: ConstraintArguments): Options {
    const { piece, kingCastles, piecePositions } = args;
    const moves = [];
    const captures = [];
    const locations = [
      piece.dirN(1, piecePositions),
      piece.dirNE(1, piecePositions),
      piece.dirE(1, piecePositions),
      piece.dirSE(1, piecePositions),
      piece.dirS(1, piecePositions),
      piece.dirSW(1, piecePositions),
      piece.dirW(1, piecePositions),
      piece.dirNW(1, piecePositions),
    ];
    if (kingCastles) {
      const castles = kingCastles(piece);
      castles.forEach((position) => moves.push(position));
    }
    locations.forEach((location) => {
      const value = Constraints.relationshipToTile(location, args);
      if (value === "BLANK") {
        moves.push(location);
      } else if (value === "ENEMY") {
        captures.push(location);
      }
    });
    return { moves, captures };
  }

  static constraintsKnight(args: ConstraintArguments): Options {
    const { piece, piecePositions } = args;
    const moves = [];
    const captures = [];
    const locations = [
      piece.dir(1, 2, piecePositions),
      piece.dir(1, -2, piecePositions),
      piece.dir(2, 1, piecePositions),
      piece.dir(2, -1, piecePositions),
      piece.dir(-1, 2, piecePositions),
      piece.dir(-1, -2, piecePositions),
      piece.dir(-2, 1, piecePositions),
      piece.dir(-2, -1, piecePositions),
    ];
    locations.forEach((location) => {
      const value = Constraints.relationshipToTile(location, args);
      if (value === "BLANK") {
        moves.push(location);
      } else if (value === "ENEMY") {
        captures.push(location);
      }
    });
    return { moves, captures };
  }

  static constraintsOrthangonal(args: ConstraintArguments): Options {
    const { piece } = args;
    const response = { moves: [], captures: [] };

    Constraints.runUntil(piece.dirN.bind(piece), response, args);
    Constraints.runUntil(piece.dirE.bind(piece), response, args);
    Constraints.runUntil(piece.dirS.bind(piece), response, args);
    Constraints.runUntil(piece.dirW.bind(piece), response, args);

    return response;
  }

  static constraintsPawn(args: ConstraintArguments): Options {
    const { piece, piecePositions } = args;
    const moves = [];
    const captures = [];
    const locationN1 = piece.dirN(1, piecePositions);
    const locationN2 = piece.dirN(2, piecePositions);

    if (Constraints.relationshipToTile(locationN1, args) === "BLANK") {
      moves.push(locationN1);
      if (!piece.moves.length && Constraints.relationshipToTile(locationN2, args) ===
  "BLANK") {
        moves.push(locationN2);
      }
    }

    [
      [piece.dirNW(1, piecePositions), piece.dirW(1, piecePositions)],
      [piece.dirNE(1, piecePositions), piece.dirE(1, piecePositions)],
    ].forEach(([location, enPassant]) => {
      const standardCaptureRelationship = Constraints.relationshipToTile(location, args);
      const enPassantCaptureRelationship = Constraints.relationshipToTile(enPassant, args);
      if (standardCaptureRelationship === "ENEMY") {
        captures.push(location);
      } else if (piece.moves.length > 0 && enPassantCaptureRelationship === "ENEMY") {
        const enPassantRow = enPassant.row === (piece.playerWhite() ? "5" : "4");
        const other = Constraints.locationToPiece(enPassant, args);
        if (enPassantRow && other && other.data.type === "PAWN") {
          if (other.moves.length === 1 && other.moves[0] === args.moveIndex - 1) {
            location.capture = { ...enPassant };
            captures.push(location);
          }
        }
      }
    });
    return { moves, captures };
  }

  static constraintsQueen(args: ConstraintArguments): Options {
    const diagonal = Constraints.constraintsDiagonal(args);
    const orthagonal = Constraints.constraintsOrthangonal(args);
    return {
      moves: diagonal.moves.concat(orthagonal.moves),
      captures: diagonal.captures.concat(orthagonal.captures),
    };
  }

  static constraintsRook(args: ConstraintArguments): Options {
    return Constraints.constraintsOrthangonal(args);
  }

  static locationToPiece(location: Position, args: ConstraintArguments): Piece | undefined {
    if (!location) {
      return undefined;
    }
    const { state, pieces } = args;
    const row = state[location.row];
    const occupyingId = row === undefined ? undefined : row[location.col];
    return pieces[occupyingId];
  }

  static relationshipToTile(location: Position, args: ConstraintArguments): TileRelation |
undefined {
    if (!location) {
      return undefined;
    }
    const { piece } = args;
    const occupying = Constraints.locationToPiece(location, args);
    if (occupying) {
      return occupying.data.player === piece.data.player ? "FRIEND" : "ENEMY";
    } else {
      return "BLANK";
    }
  }

  static runUntil(locationFunction: (integer: number, positions: PiecePositions) =>
PieceDirResponse, response: Options, args: ConstraintArguments) {
    const { piecePositions } = args;

    let inc = 1;
    let location = locationFunction(inc++, piecePositions);
    while (location) {
      let abort = false;
      const relations = Constraints.relationshipToTile(location, args);
      if (relations === "ENEMY") {
        response.captures.push(location);
        abort = true;
      } else if (relations === "FRIEND") {
        abort = true;
      } else {
        response.moves.push(location);
      }
      if (abort) {
        location = undefined;
      } else {
        location = locationFunction(inc++, piecePositions);
      }
    }
  }
}

class Piece {
  data: PieceData;
  moves = [];
  promoted = false;
  updateShape = false;

  constructor(data: PieceData) {
    this.data = data;
  }

  get orientation() {
    return this.data.player === "BLACK" ? -1 : 1;
  }

  dirN(steps: number, positions: PiecePositions): PieceDirResponse {
    return this.dir(steps, 0, positions);
  }
  dirS(steps: number, positions: PiecePositions): PieceDirResponse {
    return this.dir(-steps, 0, positions);
  }
  dirW(steps: number, positions: PiecePositions): PieceDirResponse {
    return this.dir(0, -steps, positions);
  }
  dirE(steps: number, positions: PiecePositions): PieceDirResponse {
    return this.dir(0, steps, positions);
  }
  dirNW(steps: number, positions: PiecePositions): PieceDirResponse {
    return this.dir(steps, -steps, positions);
  }
  dirNE(steps: number, positions: PiecePositions): PieceDirResponse {
    return this.dir(steps, steps, positions);
  }
  dirSW(steps: number, positions: PiecePositions): PieceDirResponse {
    return this.dir(-steps, -steps, positions);
  }
  dirSE(steps: number, positions: PiecePositions): PieceDirResponse {
    return this.dir(-steps, steps, positions);
  }
  dir(stepsRow: number, stepsColumn: number, positions: PiecePositions): PieceDirResponse {
    PIECE_DIR_CALC++;
    const row = Utils.rowToInt(positions[this.data.id].row) + this.orientation * stepsRow;
    const col = Utils.colToInt(positions[this.data.id].col) + this.orientation * stepsColumn;
    if (row >= 0 && row <= 7 && col >= 0 && col <= 7) {
      return { row: Utils.intToRow(row), col: Utils.intToCol(col) };
    }
    return undefined;
  }

  move(moveIndex: number) {
    this.moves.push(moveIndex);
  }

  options(
    moveIndex: number,
    state: BoardState,
    pieces: Pieces,
    piecePositions: PiecePositions,
    resultingChecks?: (args: ResultingChecksArguments) => PiecePosition[],
    kingCastles?: (king: Piece) => Position[]
  ): Options {
    return Constraints.generate({ moveIndex, state, piece: this, pieces, piecePositions,
kingCastles }, resultingChecks);
  }

  playerBlack() {
    return this.data.player === "BLACK";
  }
  playerWhite() {
    return this.data.player === "WHITE";
  }

  promote(type: PieceType = "QUEEN") {
    this.data.type = type;
    this.promoted = true;
    this.updateShape = true;
  }

  shape() {
    const player = this.data.player.toLowerCase();
    switch (this.data.type) {
      case "BISHOP":
        return Shape.shapeBishop(player);
      case "KING":
        return Shape.shapeKing(player);
      case "KNIGHT":
```

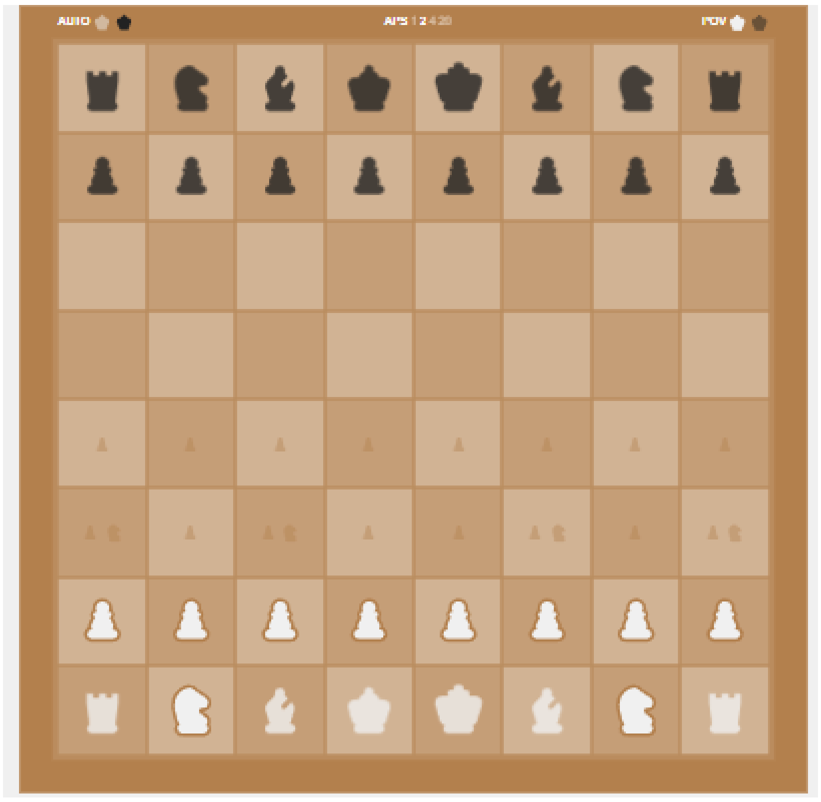
```
      return Shape.shapeKnight(player);
    case "PAWN":
      return Shape.shapePawn(player);
    case "QUEEN":
      return Shape.shapeQueen(player);
    case "ROOK":
      return Shape.shapeRook(player);
   }
  }
}

class Board {
  checksBlack: PiecePosition[] = [];
  checksWhite: PiecePosition[] = [];
  piecesTilesCaptures: PiecesToTiles = {};
  piecesTilesMoves: PiecesToTiles = {};
  tilesPiecesBlackCaptures: TilesToPieces = Utils.getInitialBoardState(() => []);
  tilesPiecesBlackMoves: TilesToPieces = Utils.getInitialBoardState(() => []);
  tilesPiecesWhiteCaptures: TilesToPieces = Utils.getInitialBoardState(() => []);
  tilesPiecesWhiteMoves: TilesToPieces = Utils.getInitialBoardState(() => []);

  pieceIdsBlack: PieceIdBlack[] = [];
  pieceIdsWhite: PieceIdWhite[] = [];
  piecePositions: PiecePositions;
  pieces: Pieces;
  state: BoardState = Utils.getInitialBoardState() as BoardState;

  static COLS: PositionColumn[] = ["A", "B", "C", "D", "E", "F", "G", "H"];
  static ROWS: PositionRow[] = ["1", "2", "3", "4", "5", "6", "7", "8"];

  constructor(pieces: Pieces, piecePositions: PiecePositions) {
    this.pieces = pieces;
    for (let id in pieces) {
      if (pieces[id].playerWhite()) {
        this.pieceIdsWhite.push(id as PieceIdWhite);
      } else {
        this.pieceIdsBlack.push(id as PieceIdBlack);
      }
    }
    this.initializePositions(piecePositions);
  }

  initializePositions(piecePositions: PiecePositions) {
    this.piecePositions = piecePositions;
    this.initializeState();
    this.piecesUpdate(0);
  }

  initializeState() {
    for (let pieceId in this.pieces) {
      const { row, col, active, _moves, _promoted } = this.piecePositions[pieceId];
    if (_moves) {
      delete this.piecePositions[pieceId]._moves;
      // TODO: come back to this
      // this.pieces[pieceId].moves = new Array(_moves);
    }
    if (_promoted) {
      delete this.piecePositions[pieceId]._promoted;
      this.pieces[pieceId].promote();
    }
    if (active) {
      this.state[row] = this.state[row] || [];
      this.state[row][col] = pieceId;
    }
  }
}

kingCastles(king: Piece): Position[] {
  const castles = [];
  // king has to not have moved
  if (king.moves.length) {
    return castles;
  }
  const kingIsWhite = king.playerWhite();
  const moves = kingIsWhite ? this.tilesPiecesBlackMoves : this.tilesPiecesWhiteMoves;
  const checkPositions = (row: PositionRow, rookCol: PositionColumn, castles: Position[]) =>
{
    const cols: PositionColumn[] = rookCol === "A" ? ["D", "C", "B"] : ["F", "G"];
    // rook has to not have moved
    const rookId = `${rookCol}${row}`;
    const rook = this.pieces[rookId];
    const { active } = this.piecePositions[rookId];
    if (active && rook.moves.length === 0) {
      let canCastle = true;
      cols.forEach((col) => {
        // each tile has to be empty
        if (this.state[row][col]) {
          canCastle = false;
          // each tile cant be in the path of the other team
        } else if (moves[row][col].length) {
          canCastle = false;
        }
      });
      if (canCastle) {
        castles.push({ col: cols[1], row, castles: rookCol });
      }
    }
  };
  const row = kingIsWhite ? "1" : "8";
  if (!this.pieces[`A${row}`].moves.length) {
    checkPositions(row, "A", castles);
  }
```

```
    if (!this.pieces[`H${row}`].moves.length) {
      checkPositions(row, "H", castles);
    }
    return castles;
  }

  kingCheckStates(kingPosition: PiecePosition, captures: TilesToPieces, piecePositions:
PiecePositions): PiecePosition[] {
    const { col, row } = kingPosition;
    return captures[row][col].map((id) => piecePositions[id]).filter((pos) => pos.active);
  }

  pieceCalculateMoves(
    pieceId: PieceId,
    moveIndex: number,
    state: BoardState,
    piecePositions: PiecePositions,
    piecesTilesCaptures: PiecesToTiles,
    piecesTilesMoves: PiecesToTiles,
    tilesPiecesCaptures: TilesToPieces,
    tilesPiecesMoves: TilesToPieces,
    resultingChecks?: (args: ResultingChecksArguments) => PiecePosition[],
    kingCastles?: (king: Piece) => Position[]
  ) {
    const { captures, moves } = this.pieces[pieceId].options(moveIndex, state, this.pieces,
piecePositions, resultingChecks, kingCastles);
    piecesTilesCaptures[pieceId] = Array.from(captures);
    piecesTilesMoves[pieceId] = Array.from(moves);
    captures.forEach(({ col, row }) => tilesPiecesCaptures[row][col].push(pieceId));
    moves.forEach(({ col, row }) => tilesPiecesMoves[row][col].push(pieceId));
  }

  pieceCapture(piece: Piece) {
    const pieceId = piece.data.id;
    const { col, row } = this.piecePositions[pieceId];
    this.state[row][col] = undefined;
    delete this.piecePositions[pieceId].col;
    delete this.piecePositions[pieceId].row;
    this.piecePositions[pieceId].active = false;
  }

  pieceMove(piece: Piece, location: Position) {
    const pieceId = piece.data.id;
    const { row, col } = this.piecePositions[pieceId];
    this.state[row][col] = undefined;
    this.state[location.row][location.col] = pieceId;
    this.piecePositions[pieceId].row = location.row;
    this.piecePositions[pieceId].col = location.col;
    if (piece.data.type === "PAWN" && (location.row === "8" || location.row === "1")) {
      piece.promote();
    }
  }

  piecesUpdate(moveIndex: number) {
    this.tilesPiecesBlackCaptures = Utils.getInitialBoardState(() => []);
    this.tilesPiecesBlackMoves = Utils.getInitialBoardState(() => []);
    this.tilesPiecesWhiteCaptures = Utils.getInitialBoardState(() => []);
    this.tilesPiecesWhiteMoves = Utils.getInitialBoardState(() => []);

    this.pieceIdsBlack.forEach((id) =>
      this.pieceCalculateMoves(
        id,
        moveIndex,
        this.state,
        this.piecePositions,
        this.piecesTilesCaptures,
        this.piecesTilesMoves,
        this.tilesPiecesBlackCaptures,
        this.tilesPiecesBlackMoves,
        this.resultingChecks.bind(this),
        this.kingCastles.bind(this)
      )
    );
    this.pieceIdsWhite.forEach((id) =>
      this.pieceCalculateMoves(
        id,
        moveIndex,
        this.state,
        this.piecePositions,
        this.piecesTilesCaptures,
        this.piecesTilesMoves,
        this.tilesPiecesWhiteCaptures,
        this.tilesPiecesWhiteMoves,
        this.resultingChecks.bind(this),
        this.kingCastles.bind(this)
      )
    );

    this.checksBlack = this.kingCheckStates(this.piecePositions.E1,
this.tilesPiecesBlackCaptures, this.piecePositions);
    this.checksWhite = this.kingCheckStates(this.piecePositions.E8,
this.tilesPiecesWhiteCaptures, this.piecePositions);
  }

  resultingChecks({ piece, location, capture, moveIndex }: ResultingChecksArguments) {
    const tilesPiecesCaptures = Utils.getInitialBoardState(() => []);
    const tilesPiecesMoves = Utils.getInitialBoardState(() => []);
    const piecesTilesCaptures = {};
    const piecesTilesMoves = {};
    const state = JSON.parse(JSON.stringify(this.state));
    const piecePositions = JSON.parse(JSON.stringify(this.piecePositions));
    if (capture) {
      const loc = location.capture || location;
      const capturedId = state[loc.row][loc.col];
      if (this.pieces[capturedId].data.type === "KING") {
        // this is a checking move
      } else {
        delete piecePositions[capturedId].col;
        delete piecePositions[capturedId].row;
        piecePositions[capturedId].active = false;
      }
    }
    const pieceId = piece.data.id;
    const { row, col } = piecePositions[pieceId];
    state[row][col] = undefined;
    state[location.row][location.col] = pieceId;
    piecePositions[pieceId].row = location.row;
    piecePositions[pieceId].col = location.col;

    const ids = piece.playerWhite() ? this.pieceIdsBlack : this.pieceIdsWhite;
    const king = piece.playerWhite() ? piecePositions.E1 : piecePositions.E8;
    ids.forEach((id) =>
      this.pieceCalculateMoves(id, moveIndex, state, piecePositions, piecesTilesCaptures,
piecesTilesMoves, tilesPiecesCaptures, tilesPiecesMoves)
    );
    return this.kingCheckStates(king, tilesPiecesCaptures, piecePositions);
  }

  tileEach(callback: (position: Position, piece?: Piece, pieceMoves?: Position[], pieceCaptures?:
Position[]) => void) {
    Board.ROWS.forEach((row) => {
      Board.COLS.forEach((col) => {
        const piece = this.tileFind({ row, col });
        const moves = piece ? this.piecesTilesMoves[piece.data.id] : undefined;
        const captures = piece ? this.piecesTilesCaptures[piece.data.id] : undefined;
        callback({ row, col }, piece, moves, captures);
      });
    });
  }

  tileFind({ row, col }: Position): Piece | undefined {
    const id = this.state[row][col];
    return this.pieces[id];
  }

  toShortCode() {
    const positionsAbsolute = [];
    const positionsDefaults = [];
    for (let id in this.piecePositions) {
      const { active, col, row } = this.piecePositions[id];
      const pos = `${col}${row}`;
      const moves = this.pieces[id].moves;
      const promotedCode = this.pieces[id].promoted ? "P" : "";
```

```
     const movesCode = moves > 9 ? "9" : moves > 1 ? moves.toString() : "";
     if (active) {
       positionsAbsolute.push(`${promotedCode}${id}${id === pos ? "" :
pos}${movesCode}`);
       if (id !== pos || moves > 0) {
         positionsDefaults.push(`${promotedCode}${id}${pos}${movesCode}`);
       }
     } else {
       if (id !== "BQ" && id !== "WQ") {
         positionsDefaults.push(`${promotedCode}${id}X`);
       }
     }
   }
   const pA = positionsAbsolute.join(",");
   const pD = positionsDefaults.join(",");
   return pA.length > pD.length ? `X${pD}` : pA;
 }
}

class Game {
 active: Piece | null = null;
 activePieceOptions: Position[] = [];
 board: Board;
 moveIndex = 0;
 moves = [];
 turn: PlayerId;

 constructor(pieces: Pieces, piecePositions: PiecePositions, turn: PlayerId = "WHITE") {
   this.turn = turn;
   this.board = new Board(pieces, piecePositions);
 }

 activate(location: Position): ActivateResponse {
   const tilePiece = this.board.tileFind(location);
   if (tilePiece && !this.active && tilePiece.data.player !== this.turn) {
     this.active = null;
     return { type: "INVALID" };
     // a piece is active rn
   } else if (this.active) {
     const activePieceId = this.active.data.id;
     this.active = null;
     const validatedPosition = this.activePieceOptions.find((option) => option.col ===
location.col && option.row === location.row);
     const positionIsValid = !!validatedPosition;
     this.activePieceOptions = [];
     const capturePiece: Piece | undefined = validatedPosition?.capture ?
this.board.tileFind(validatedPosition.capture) : tilePiece;

     // a piece is on the tile
     if (capturePiece) {
       const capturedPieceId = capturePiece.data.id;
```

```
      // cancelling the selected piece on invalid location
      if (capturedPieceId === activePieceId) {
        return { type: "CANCEL" };
      } else if (positionIsValid) {
        // capturing the selected piece
        this.capture(activePieceId, capturedPieceId, location);
        return {
          type: "CAPTURE",
          activePieceId,
          capturedPieceId,
          captures: [location],
        };
        // cancel
      } else if (capturePiece.data.player !== this.turn) {
        return { type: "CANCEL" };
      } else {
        // proceed to TOUCH or CANCEL
      }
    } else if (positionIsValid) {
      // moving will return castled if that happens (only two move)
      const castledId = this.move(activePieceId, location);
      return { type: "MOVE", activePieceId, moves: [location], castledId };
      // invalid spot. cancel.
    } else {
      return { type: "CANCEL" };
    }
  }

  // no piece selected or new CANCEL + TOUCH
  if (tilePiece) {
    const tilePieceId = tilePiece.data.id;
    const moves = this.board.piecesTilesMoves[tilePieceId];
    const captures = this.board.piecesTilesCaptures[tilePieceId];
    if (!moves.length && !captures.length) {
      return { type: "INVALID" };
    }
    this.active = tilePiece;
    this.activePieceOptions = moves.concat(captures);
    return { type: "TOUCH", captures, moves, activePieceId: tilePieceId };
    // cancelling
  } else {
    this.activePieceOptions = [];
    return { type: "CANCEL" };
  }
}

capture(capturingPieceId: PieceId, capturedPieceId: PieceId, location: Position) {
  const captured = this.board.pieces[capturedPieceId];
  this.board.pieceCapture(captured);
  this.move(capturingPieceId, location, true);
}
```

```
  handleCastling(piece: Piece, location: Position): PieceId | undefined {
    if (
      piece.data.type !== "KING" ||
      piece.moves.length ||
      location.row !== (piece.playerWhite() ? "1" : "8") ||
      (location.col !== "C" && location.col !== "G")
    ) {
      return;
    }

    return `${location.col === "C" ? "A" : "H"}${location.row}` as PieceId;
  }

  move(pieceId: PieceId, location: Position, capture = false): PieceId | undefined {
    const piece = this.board.pieces[pieceId];
    const castledId = this.handleCastling(piece, location);
    piece.move(this.moveIndex);
    if (castledId) {
      const castled = this.board.pieces[castledId];
      castled.move(this.moveIndex);
      this.board.pieceMove(castled, { col: location.col === "C" ? "D" : "F", row: location.row });
      this.moves.push(`${pieceId}O${location.col}${location.row}`);
    } else {
      this.moves.push(`${pieceId}${capture ? "x" : ""}${location.col}${location.row}`);
    }
    this.moveIndex++;
    this.board.pieceMove(piece, location);
    this.turn = this.turn === "WHITE" ? "BLACK" : "WHITE";
    this.board.piecesUpdate(this.moveIndex);
    const state = this.moveResultState();
    console.log(state);
    if (!state.moves && !state.captures) {
      alert(state.stalemate ? "Stalemate!" : `${this.turn === "WHITE" ? "Black" : "White"}
Wins!`);
    }
    return castledId;
  }

  moveResultState() {
    let movesWhite = 0;
    let capturesWhite = 0;
    let movesBlack = 0;
    let capturesBlack = 0;
    this.board.tileEach(({ row, col }) => {
      movesWhite += this.board.tilesPiecesWhiteMoves[row][col].length;
      capturesWhite += this.board.tilesPiecesWhiteCaptures[row][col].length;
      movesBlack += this.board.tilesPiecesBlackMoves[row][col].length;
      capturesBlack += this.board.tilesPiecesBlackCaptures[row][col].length;
    });
    const activeBlack = this.board.pieceIdsBlack.filter((pieceId) =>
this.board.piecePositions[pieceId].active).length;
    const activeWhite = this.board.pieceIdsWhite.filter((pieceId) =>
this.board.piecePositions[pieceId].active).length;
    const moves = this.turn === "WHITE" ? movesWhite : movesBlack;
    const captures = this.turn === "WHITE" ? capturesWhite : capturesBlack;
    const noMoves =  movesWhite + capturesWhite + movesBlack + capturesBlack === 0;
    const checked = !!this.board[this.turn === "WHITE" ? "checksBlack" :
"checksWhite"].length;
    const onlyKings = activeBlack === 1 && activeWhite === 1;
    const stalemate =  onlyKings || noMoves || ((moves + captures === 0) && !checked);
    const code = this.board.toShortCode();
    return { turn: this.turn, checked, moves, captures, code, stalemate };
  }

  randomMove(): Position {
    if (this.active) {
      if (this.activePieceOptions.length) {
        const { col, row } = this.activePieceOptions[Math.floor(Math.random() *
this.activePieceOptions.length)];
        return { col, row };
      } else {
        const {col, row} = this.board.piecePositions[this.active.data.id];
        return { col, row };
      }
    } else {
      const ids: PieceId[] = this.turn === "WHITE" ? this.board.pieceIdsWhite :
this.board.pieceIdsBlack;
      const positions = ids.map((pieceId: PieceId) => {
        const moves = this.board.piecesTilesMoves[pieceId];
        const captures = this.board.piecesTilesCaptures[pieceId];
        return (moves.length || captures.length) ? this.board.piecePositions[pieceId] : undefined;
      }).filter((position) => position?.active);
      const remaining = positions[Math.floor(Math.random() * positions.length)];
      const { col, row } = remaining || { col: "E", row: "1" };
      return { col, row };
    }
  }
}

class View {
  element: HTMLElement;
  game: Game;
  pieces: BoardPieces;
  tiles: BoardTiles;

  constructor(element: HTMLElement, game: Game, perspective?: PlayerId) {
    this.element = element;
    this.game = game;
    this.setPerspective(perspective || this.game.turn);
    this.tiles = Utils.getInitialBoardTiles(this.element, this.handleTileClick.bind(this));
    this.pieces = Utils.getInitialBoardPieces(this.element, this.game.board.pieces);
    this.drawPiecePositions();
  }

  drawActivePiece(activePieceId: PieceId) {
    const { row, col } = this.game.board.piecePositions[activePieceId];
    this.tiles[row][col].classList.add("highlight-active");
    this.pieces[activePieceId].classList.add("highlight-active");
  }

  drawCapturedPiece(capturedPieceId: PieceId) {
    const piece = this.pieces[capturedPieceId];
    piece.style.setProperty("--transition-delay", "var(--transition-duration)");
    piece.style.removeProperty("--pos-col");
    piece.style.removeProperty("--pos-row");
    piece.style.setProperty("--scale", "0");
  }

  drawPiecePositions(moves: Position[] = [], moveInner: string = "") {
    document.body.style.setProperty("--color-background", `var(--color-
${this.game.turn.toLowerCase()}`);
    const other = this.game.turn === "WHITE" ? "turn-black" : "turn-white";
    const current = this.game.turn === "WHITE" ? "turn-white" : "turn-black";
    this.element.classList.add(current);
    this.element.classList.remove(other);
    if (moves.length) {
      this.element.classList.add("touching");
    } else {
      this.element.classList.remove("touching");
    }

    const key = (row, col) => `${row}-${col}`;
    const moveKeys = moves.map(({ row, col }) => key(row, col));
    this.game.board.tileEach(({ row, col }, piece, pieceMoves, pieceCaptures) => {
      const tileElement = this.tiles[row][col];
      const move = moveKeys.includes(key(row, col)) ? moveInner : "";
      const format = (id, className) => this.game.board.pieces[id].shape();
      tileElement.innerHTML = `
        <div class="move">${move}</div>
        <div class="moves">
          ${this.game.board.tilesPiecesBlackMoves[row][col].map((id) => format(id,
"black")).join("")}
          ${this.game.board.tilesPiecesWhiteMoves[row][col].map((id) => format(id,
"white")).join("")}
        </div>
        <div class="captures">
          ${this.game.board.tilesPiecesBlackCaptures[row][col].map((id) => format(id,
"black")).join("")}
          ${this.game.board.tilesPiecesWhiteCaptures[row][col].map((id) => format(id,
"white")).join("")}
        </div>
      `;
      if (piece) {
        tileElement.classList.add("occupied");
        const pieceElement = this.pieces[piece.data.id];
        pieceElement.style.setProperty("--pos-col", Utils.colToInt(col).toString());
        pieceElement.style.setProperty("--pos-row", Utils.rowToInt(row).toString());
        pieceElement.style.setProperty("--scale", "1");
        pieceElement.classList[pieceMoves?.length ? "add" : "remove"]("can-move");
        pieceElement.classList[pieceCaptures?.length ? "add" : "remove"]("can-capture");
        if (piece.updateShape) {
          piece.updateShape = false;
          pieceElement.innerHTML = piece.shape();
        }
      } else {
        tileElement.classList.remove("occupied");
      }
    });
  }

  drawPositions(moves: Position[], captures: Position[]) {
    moves?.forEach(({ row, col }) => {
      this.tiles[row][col].classList.add("highlight-move");
      this.pieces[this.game.board.tileFind({ row, col })?.data.id]?.classList.add("highlight-
move");
    });
    captures?.forEach(({ row, col, capture }) => {
      if (capture) {
        row = capture.row;
        col = capture.col;
      }
      this.tiles[row][col].classList.add("highlight-capture");
      this.pieces[this.game.board.tileFind({ row, col })?.data.id]?.classList.add("highlight-
capture");
    });
  }

  drawResetClassNames() {
    document.querySelectorAll(".highlight-active").forEach((element) =>
element.classList.remove("highlight-active"));
    document.querySelectorAll(".highlight-capture").forEach((element) =>
element.classList.remove("highlight-capture"));
    document.querySelectorAll(".highlight-move").forEach((element) =>
element.classList.remove("highlight-move"));
  }

  handleTileClick(location: Position) {
    const { activePieceId, capturedPieceId, moves = [], captures = [], type } =
this.game.activate(location);

    this.drawResetClassNames();
    if (type === "TOUCH") {
      const enPassant = captures.find((capture) => !!capture.capture);
      const passingMoves = enPassant ? moves.concat([enPassant]) : moves;
      this.drawPiecePositions(passingMoves, this.game.board.pieces[activePieceId].shape());
    } else {
      this.drawPiecePositions();
    }

    if (type === "CANCEL" || type === "INVALID") {
      return;
    }

    if (type === "MOVE" || type === "CAPTURE") {
    } else {
      this.drawActivePiece(activePieceId);
    }
    if (type === "TOUCH") {
      this.drawPositions(moves, captures);
    } else if (type === "CAPTURE") {
      this.drawCapturedPiece(capturedPieceId);
    }
    // crazy town
    // this.setPerspective(this.game.turn);
  }

  setPerspective(perspective: PlayerId) {
    const other = perspective === "WHITE" ? "perspective-black" : "perspective-white";
    const current = perspective === "WHITE" ? "perspective-white" : "perspective-black";
    this.element.classList.add(current);
    this.element.classList.remove(other);
  }
}

class Control {
  game: Game;
  inputSpeedAsap: HTMLInputElement = document.getElementById("speed-asap") as
HTMLInputElement;
  inputSpeedFast: HTMLInputElement = document.getElementById("speed-fast") as
HTMLInputElement;
  inputSpeedMedium: HTMLInputElement = document.getElementById("speed-medium") as
HTMLInputElement;
  inputSpeedSlow: HTMLInputElement = document.getElementById("speed-slow") as
HTMLInputElement;
  inputRandomBlack: HTMLInputElement = document.getElementById("black-random") as
HTMLInputElement;
  inputRandomWhite: HTMLInputElement = document.getElementById("white-random") as
HTMLInputElement;
  inputPerspectiveBlack: HTMLInputElement = document.getElementById("black-
perspective") as HTMLInputElement;
  inputPerspectiveWhite: HTMLInputElement = document.getElementById("white-
perspective") as HTMLInputElement;
  view: View;
  constructor(game: Game, view: View) {
    this.game = game;
    this.view = view;
    this.inputPerspectiveBlack.addEventListener("change",
this.updateViewPerspective.bind(this));
    this.inputPerspectiveWhite.addEventListener("change",
this.updateViewPerspective.bind(this));
    this.updateViewPerspective();
  }

  get speed() {
    if (this.inputSpeedAsap.checked) {
      return 50;
    }
    if (this.inputSpeedFast.checked) {
      return 250;
    }
    if (this.inputSpeedMedium.checked) {
      return 500;
    }
    if (this.inputSpeedSlow.checked) {
      return 1000;
    }
  }

  autoplay() {
    const input = this.game.turn === "WHITE" ? this.inputRandomWhite :
this.inputRandomBlack;
    if (!input.checked) {
      setTimeout(this.autoplay.bind(this), this.speed);
      return;
    }
    const position = this.game.randomMove();
    this.view.handleTileClick(position);
    setTimeout(this.autoplay.bind(this), this.speed);
  }

  updateViewPerspective() {
    this.view.setPerspective(this.inputPerspectiveBlack.checked ? "BLACK" : "WHITE");
  }
}

const DEMOS = {
  castle1: "XD8B3,B1X,C1X,D1X,F1X,G1X",
  castle2: "XD8B3,B1X,C1X,C2X,D1X,F1X,G1X",
  castle3: "XD8E3,B1X,C1X,F2X,D1X,F1X,G1X",
  promote1: "E1,E8,C2C7",
  promote2: "E1,E8E7,PC2C8",
  start: "XE7E6,F7F5,D2D4,E2E5",
  test2: "C8E2,E8,G8H1,D7E4,H7H3,PA2H7,PB2G7,D2D6,E2E39,A1H2,E1B3",
  test: "C8E2,E8,G8H1,D7E4,H7H3,D1H7,PB2G7,D2D6,E2E39,A1H2,E1B3",
};

const initialPositions = Utils.getInitialPiecePositions();
// const initialPositions = Utils.getPositionsFromShortCode(DEMOS.castle1);
const initialTurn = "WHITE";
const perspective = "WHITE";
const game = new Game(Utils.getInitialPieces(), initialPositions, initialTurn);
const view = new View(document.getElementById("board"), game, perspective);
const control = new Control(game, view);

control.autoplay();
```



3. Tilting Maze
HTML

```
<div id="center">
  <div id="game">
    <div id="maze">
      <div id="end"></div>
    </div>
    <div id="joystick">
      <div class="joystick-arrow"></div>
      <div class="joystick-arrow"></div>
      <div class="joystick-arrow"></div>
      <div class="joystick-arrow"></div>
      <div id="joystick-head"></div>
    </div>
    <div id="note">
      Click the joystick to start!
      <p>Move every ball to the center. Ready for hard mode? Press H</p>
    </div>
  </div>
</div>
<a id="youtube" href="https://youtu.be/bTk6dcAckuI" target="_top">
  <span>See how this game was made</span>
</a>
<div id="youtube-card">
  How to simulate ball movement in a maze with JavaScript
</div>
```

CSS

```
body {
  /* https://coolors.co/f06449-ede6e3-7d82b8-36382e-613f75  */
  --background-color: #ede6e3;
  --wall-color: #36382e;
  --joystick-color: #210124;
  --joystick-head-color: #f06449;
  --ball-color: #f06449;
  --end-color: #7d82b8;
  --text-color: #210124;

  font-family: "Segoe UI", Tahoma, Geneva, Verdana, sans-serif;
  background-color: var(--background-color);
}

html,
body {
  height: 100%;
  margin: 0;
}

#center {
  display: flex;
  align-items: center;
  justify-content: center;
  height: 100%;
}

#game {
  display: grid;
  grid-template-columns: auto 150px;
  grid-template-rows: 1fr auto 1fr;
  gap: 30px;
  perspective: 600px;
}

#maze {
  position: relative;
  grid-row: 1 / -1;
  grid-column: 1;
  width: 350px;
  height: 315px;
  display: flex;
  justify-content: center;
  align-items: center;
}

#end {
  width: 65px;
  height: 65px;
  border: 5px dashed var(--end-color);
  border-radius: 50%;
}

#joystick {
  position: relative;
  background-color: var(--joystick-color);
  border-radius: 50%;
  width: 50px;
  height: 50px;
  display: flex;
  align-items: center;
  justify-content: center;
  margin: 10px 50px;
  grid-row: 2;
}

#joystick-head {
  position: relative;
  background-color: var(--joystick-head-color);
  border-radius: 50%;
  width: 20px;
  height: 20px;
  cursor: grab;

  animation-name: glow;
  animation-duration: 0.6s;
  animation-iteration-count: infinite;
  animation-direction: alternate;
  animation-timing-function: ease-in-out;
  animation-delay: 4s;
}

@keyframes glow {
  0% {
    transform: scale(1);
  }
  100% {
    transform: scale(1.2);
  }
}

.joystick-arrow:nth-of-type(1) {
  position: absolute;
  bottom: 55px;

  width: 0;
  height: 0;
  border-left: 10px solid transparent;
  border-right: 10px solid transparent;

  border-bottom: 10px solid var(--joystick-color);
}

.joystick-arrow:nth-of-type(2) {
  position: absolute;
  top: 55px;

  width: 0;
  height: 0;
  border-left: 10px solid transparent;
  border-right: 10px solid transparent;

  border-top: 10px solid var(--joystick-color);
}

.joystick-arrow:nth-of-type(3) {
  position: absolute;
  left: 55px;

  width: 0;
  height: 0;
  border-top: 10px solid transparent;
  border-bottom: 10px solid transparent;

  border-left: 10px solid var(--joystick-color);
}

.joystick-arrow:nth-of-type(4) {
  position: absolute;
  right: 55px;

  width: 0;
  height: 0;
  border-top: 10px solid transparent;
  border-bottom: 10px solid transparent;

  border-right: 10px solid var(--joystick-color);
}
```

```
#note {
  grid-row: 3;
  grid-column: 2;
  text-align: center;
  font-size: 0.8em;
  color: var(--text-color);
  transition: opacity 2s;
}

a:visited {
  color: inherit;
}

.ball {
  position: absolute;
  margin-top: -5px;
  margin-left: -5px;
  border-radius: 50%;
  background-color: var(--ball-color);
  width: 10px;
  height: 10px;
}

.wall {
  position: absolute;
  background-color: var(--wall-color);
  transform-origin: top center;
  margin-left: -5px;
}

.wall::before,
.wall::after {
  display: block;
  content: "";
  width: 10px;
  height: 10px;
  background-color: inherit;
  border-radius: 50%;
  position: absolute;
}

.wall::before {
  top: -5px;
}

.wall::after {
  bottom: -5px;
}

.black-hole {
  position: absolute;
  margin-top: -9px;
  margin-left: -9px;
  border-radius: 50%;
  background-color: black;
  width: 18px;
  height: 18px;
}

#youtube,
#youtube-card {
  display: none;
}

@media (min-height: 425px) {
  /** Youtube logo by https://codepen.io/alvaromontoro */
  #youtube {
    z-index: 2;
    display: block;
    width: 100px;
    height: 70px;
    position: absolute;
    bottom: 20px;
    right: 20px;
    background: red;
    border-radius: 50% / 11%;
    transform: scale(0.8);
    transition: transform 0.5s;
  }

  #youtube:hover,
  #youtube:focus {
    transform: scale(0.9);
  }

  #youtube::before {
    content: "";
    display: block;
    position: absolute;
    top: 7.5%;
    left: -6%;
    width: 112%;
    height: 85%;
    background: red;
    border-radius: 9% / 50%;
  }

  #youtube::after {
    content: "";
    display: block;
    position: absolute;
    top: 20px;
    left: 40px;
    width: 45px;
    height: 30px;
    border: 15px solid transparent;
    box-sizing: border-box;
    border-left: 30px solid white;
  }

  #youtube span {
    font-size: 0;
    position: absolute;
    width: 0;
    height: 0;
    overflow: hidden;
  }

  #youtube:hover + #youtube-card {
    display: block;
    position: absolute;
    bottom: 12px;
    right: 10px;
    padding: 25px 130px 25px 25px;
    width: 300px;
    background-color: white;
  }
}
```

JS

```
/*

If you want to know how this game works, you can find a source code walkthrough video here:
https://youtu.be/bTk6dcAckuI

Follow me on twitter for more: https://twitter.com/HunorBorbely

*/

Math.minmax = (value, limit) => {
  return Math.max(Math.min(value, limit), -limit);
};

const distance2D = (p1, p2) => {
  return Math.sqrt((p2.x - p1.x) ** 2 + (p2.y - p1.y) ** 2);
};

// Angle between the two points
const getAngle = (p1, p2) => {
  let angle = Math.atan((p2.y - p1.y) / (p2.x - p1.x));
  if (p2.x - p1.x < 0) angle += Math.PI;
  return angle;
};

// The closest a ball and a wall cap can be
const closestItCanBe = (cap, ball) => {
  let angle = getAngle(cap, ball);

  const deltaX = Math.cos(angle) * (wallW / 2 + ballSize / 2);
  const deltaY = Math.sin(angle) * (wallW / 2 + ballSize / 2);

  return { x: cap.x + deltaX, y: cap.y + deltaY };
};

// Roll the ball around the wall cap
const rollAroundCap = (cap, ball) => {
  // The direction the ball can't move any further because the wall holds it back
  let impactAngle = getAngle(ball, cap);

  // The direction the ball wants to move based on it's velocity
  let heading = getAngle(
    { x: 0, y: 0 },
    { x: ball.velocityX, y: ball.velocityY }
  );

  // The angle between the impact direction and the ball's desired direction
  // The smaller this angle is, the bigger the impact
  // The closer it is to 90 degrees the smoother it gets (at 90 there would be no collision)
  let impactHeadingAngle = impactAngle - heading;

  // Velocity distance if not hit would have occurred
  const velocityMagnitude = distance2D(
    { x: 0, y: 0 },
    { x: ball.velocityX, y: ball.velocityY }
  );
  // Velocity component diagonal to the impact
  const velocityMagnitudeDiagonalToTheImpact =
    Math.sin(impactHeadingAngle) * velocityMagnitude;

  // How far should the ball be from the wall cap
  const closestDistance = wallW / 2 + ballSize / 2;

  const rotationAngle = Math.atan(
    velocityMagnitudeDiagonalToTheImpact / closestDistance
  );

  const deltaFromCap = {
    x: Math.cos(impactAngle + Math.PI - rotationAngle) * closestDistance,
    y: Math.sin(impactAngle + Math.PI - rotationAngle) * closestDistance
  };

  const x = ball.x;
  const y = ball.y;
  const velocityX = ball.x - (cap.x + deltaFromCap.x);
  const velocityY = ball.y - (cap.y + deltaFromCap.y);
  const nextX = x + velocityX;
  const nextY = y + velocityY;

  return { x, y, velocityX, velocityY, nextX, nextY };
};

// Decreases the absolute value of a number but keeps it's sign, doesn't go below abs 0
const slow = (number, difference) => {
  if (Math.abs(number) <= difference) return 0;
  if (number > difference) return number - difference;
  return number + difference;
};

const mazeElement = document.getElementById("maze");
const joystickHeadElement = document.getElementById("joystick-head");
const noteElement = document.getElementById("note"); // Note element for instructions and
game won, game failed texts

let hardMode = false;
let previousTimestamp;
let gameInProgress;
let mouseStartX;
let mouseStartY;
let accelerationX;
let accelerationY;
let frictionX;
let frictionY;

const pathW = 25; // Path width
const wallW = 10; // Wall width
const ballSize = 10; // Width and height of the ball
const holeSize = 18;

const debugMode = false;

let balls = [];
let ballElements = [];
let holeElements = [];

resetGame();

// Draw balls for the first time
balls.forEach(({ x, y }) => {
  const ball = document.createElement("div");
  ball.setAttribute("class", "ball");
  ball.style.cssText = `left: ${x}px; top: ${y}px; `;

  mazeElement.appendChild(ball);
  ballElements.push(ball);
});

// Wall metadata
const walls = [
  // Border
  { column: 0, row: 0, horizontal: true, length: 10 },
  { column: 0, row: 0, horizontal: false, length: 9 },
  { column: 0, row: 9, horizontal: true, length: 10 },
  { column: 10, row: 0, horizontal: false, length: 9 },

  // Horizontal lines starting in 1st column
  { column: 0, row: 6, horizontal: true, length: 1 },
  { column: 0, row: 8, horizontal: true, length: 1 },

  // Horizontal lines starting in 2nd column
  { column: 1, row: 1, horizontal: true, length: 2 },
  { column: 1, row: 7, horizontal: true, length: 1 },

  // Horizontal lines starting in 3rd column
  { column: 2, row: 2, horizontal: true, length: 2 },
  { column: 2, row: 4, horizontal: true, length: 1 },
  { column: 2, row: 5, horizontal: true, length: 1 },
  { column: 2, row: 6, horizontal: true, length: 1 },

  // Horizontal lines starting in 4th column
  { column: 3, row: 3, horizontal: true, length: 1 },
  { column: 3, row: 8, horizontal: true, length: 3 },

  // Horizontal lines starting in 5th column
  { column: 4, row: 6, horizontal: true, length: 1 },

  // Horizontal lines starting in 6th column
  { column: 5, row: 2, horizontal: true, length: 2 },
  { column: 5, row: 7, horizontal: true, length: 1 },

  // Horizontal lines starting in 7th column
  { column: 6, row: 1, horizontal: true, length: 1 },
  { column: 6, row: 6, horizontal: true, length: 2 },

  // Horizontal lines starting in 8th column
  { column: 7, row: 3, horizontal: true, length: 2 },
  { column: 7, row: 7, horizontal: true, length: 2 },

  // Horizontal lines starting in 9th column
  { column: 8, row: 1, horizontal: true, length: 1 },
  { column: 8, row: 2, horizontal: true, length: 1 },
  { column: 8, row: 3, horizontal: true, length: 1 },
  { column: 8, row: 4, horizontal: true, length: 2 },
  { column: 8, row: 8, horizontal: true, length: 2 },
  // Vertical lines after the 1st column
  { column: 1, row: 1, horizontal: false, length: 2 },
  { column: 1, row: 4, horizontal: false, length: 2 },

  // Vertical lines after the 2nd column
  { column: 2, row: 2, horizontal: false, length: 2 },
  { column: 2, row: 5, horizontal: false, length: 1 },
  { column: 2, row: 7, horizontal: false, length: 2 },

  // Vertical lines after the 3rd column
  { column: 3, row: 0, horizontal: false, length: 1 },
  { column: 3, row: 4, horizontal: false, length: 1 },
  { column: 3, row: 6, horizontal: false, length: 2 },

  // Vertical lines after the 4th column
  { column: 4, row: 1, horizontal: false, length: 2 },
  { column: 4, row: 6, horizontal: false, length: 1 },

  // Vertical lines after the 5th column
  { column: 5, row: 0, horizontal: false, length: 2 },
  { column: 5, row: 6, horizontal: false, length: 1 },
  { column: 5, row: 8, horizontal: false, length: 1 },

  // Vertical lines after the 6th column
  { column: 6, row: 4, horizontal: false, length: 1 },
  { column: 6, row: 6, horizontal: false, length: 1 },

  // Vertical lines after the 7th column
  { column: 7, row: 1, horizontal: false, length: 4 },
  { column: 7, row: 7, horizontal: false, length: 2 },

  // Vertical lines after the 8th column
  { column: 8, row: 2, horizontal: false, length: 1 },
  { column: 8, row: 4, horizontal: false, length: 2 },

  // Vertical lines after the 9th column
  { column: 9, row: 1, horizontal: false, length: 1 },
  { column: 9, row: 5, horizontal: false, length: 2 }
].map((wall) => ({
  x: wall.column * (pathW + wallW),
  y: wall.row * (pathW + wallW),
  horizontal: wall.horizontal,
  length: wall.length * (pathW + wallW)
}));

// Draw walls
walls.forEach(({ x, y, horizontal, length }) => {
  const wall = document.createElement("div");
  wall.setAttribute("class", "wall");
  wall.style.cssText = `
      left: ${x}px;
      top: ${y}px;
      width: ${wallW}px;
      height: ${length}px;
      transform: rotate(${horizontal ? -90 : 0}deg);
    `;

  mazeElement.appendChild(wall);
});

const holes = [
  { column: 0, row: 5 },
  { column: 2, row: 0 },
  { column: 2, row: 4 },
  { column: 4, row: 6 },
  { column: 6, row: 2 },
  { column: 6, row: 8 },
  { column: 8, row: 1 },
  { column: 8, row: 2 }
].map((hole) => ({
  x: hole.column * (wallW + pathW) + (wallW / 2 + pathW / 2),
  y: hole.row * (wallW + pathW) + (wallW / 2 + pathW / 2)
}));

joystickHeadElement.addEventListener("mousedown", function (event) {
  if (!gameInProgress) {
    mouseStartX = event.clientX;
    mouseStartY = event.clientY;
    gameInProgress = true;
    window.requestAnimationFrame(main);
    noteElement.style.opacity = 0;
    joystickHeadElement.style.cssText = `
        animation: none;
        cursor: grabbing;
      `;
  }
});

window.addEventListener("mousemove", function (event) {
  if (gameInProgress) {
    const mouseDeltaX = -Math.minmax(mouseStartX - event.clientX, 15);
    const mouseDeltaY = -Math.minmax(mouseStartY - event.clientY, 15);

    joystickHeadElement.style.cssText = `
        left: ${mouseDeltaX}px;
        top: ${mouseDeltaY}px;
        animation: none;
        cursor: grabbing;
      `;

    const rotationY = mouseDeltaX * 0.8; // Max rotation = 12
    const rotationX = mouseDeltaY * 0.8;
    mazeElement.style.cssText = `
        transform: rotateY(${rotationY}deg) rotateX(${-rotationX}deg)
      `;

    const gravity = 2;
    const friction = 0.01; // Coefficients of friction

    accelerationX = gravity * Math.sin((rotationY / 180) * Math.PI);
    accelerationY = gravity * Math.sin((rotationX / 180) * Math.PI);
    frictionX = gravity * Math.cos((rotationY / 180) * Math.PI) * friction;
    frictionY = gravity * Math.cos((rotationX / 180) * Math.PI) * friction;
  }
});

window.addEventListener("keydown", function (event) {
  // If not an arrow key or space or H was pressed then return
  if (![" ", "H", "h", "E", "e"].includes(event.key)) return;

  // If an arrow key was pressed then first prevent default
  event.preventDefault();

  // If space was pressed restart the game
  if (event.key == " ") {
    resetGame();
    return;
  }

  // Set Hard mode
  if (event.key == "H" || event.key == "h") {
    hardMode = true;
    resetGame();
    return;
  }

  // Set Easy mode
  if (event.key == "E" || event.key == "e") {
    hardMode = false;
    resetGame();
    return;
  }
});

function resetGame() {
  previousTimestamp = undefined;
  gameInProgress = false;
  mouseStartX = undefined;
  mouseStartY = undefined;
  accelerationX = undefined;
  accelerationY = undefined;
  frictionX = undefined;
```

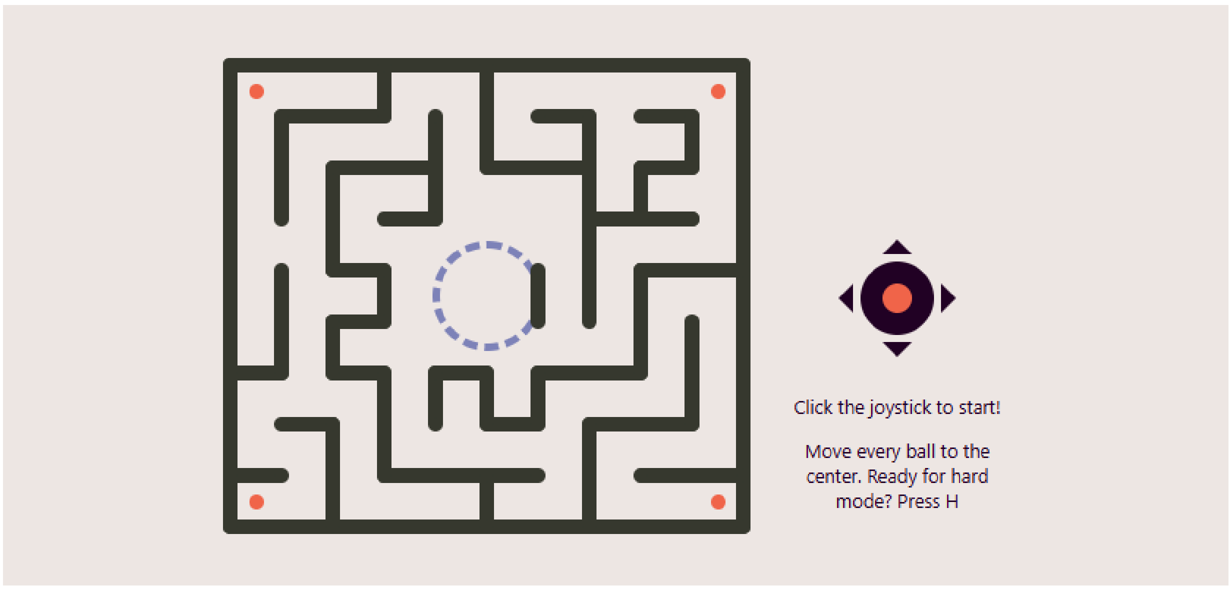
```
frictionY = undefined;

mazeElement.style.cssText = `
    transform: rotateY(0deg) rotateX(0deg)
  `;

joystickHeadElement.style.cssText = `
    left: 0;
    top: 0;
    animation: glow;
    cursor: grab;
  `;

if (hardMode) {
  noteElement.innerHTML = `Click the joystick to start!
      <p>Hard mode, Avoid black holes. Back to easy mode? Press E</p>`;
} else {
  noteElement.innerHTML = `Click the joystick to start!
      <p>Move every ball to the center. Ready for hard mode? Press H</p>`;
}
noteElement.style.opacity = 1;

balls = [
  { column: 0, row: 0 },
  { column: 9, row: 0 },
  { column: 0, row: 8 },
  { column: 9, row: 8 }
].map((ball) => ({
  x: ball.column * (wallW + pathW) + (wallW / 2 + pathW / 2),
  y: ball.row * (wallW + pathW) + (wallW / 2 + pathW / 2),
  velocityX: 0,
  velocityY: 0
}));

if (ballElements.length) {
  balls.forEach(({ x, y }, index) => {
    ballElements[index].style.cssText = `left: ${x}px; top: ${y}px; `;
  });
}

// Remove previous hole elements
holeElements.forEach((holeElement) => {
  mazeElement.removeChild(holeElement);
});
holeElements = [];

// Reset hole elements if hard mode
if (hardMode) {
  holes.forEach(({ x, y }) => {
    const ball = document.createElement("div");
    ball.setAttribute("class", "black-hole");
```

```
      ball.style.cssText = `left: ${x}px; top: ${y}px; `;

      mazeElement.appendChild(ball);
      holeElements.push(ball);
    });
  }
}

function main(timestamp) {
  // It is possible to reset the game mid-game. This case the look should stop
  if (!gameInProgress) return;

  if (previousTimestamp === undefined) {
    previousTimestamp = timestamp;
    window.requestAnimationFrame(main);
    return;
  }

  const maxVelocity = 1.5;

  // Time passed since last cycle divided by 16
  // This function gets called every 16 ms on average so dividing by 16 will result in 1
  const timeElapsed = (timestamp - previousTimestamp) / 16;

  try {
    // If mouse didn't move yet don't do anything
    if (accelerationX != undefined && accelerationY != undefined) {
      const velocityChangeX = accelerationX * timeElapsed;
      const velocityChangeY = accelerationY * timeElapsed;
      const frictionDeltaX = frictionX * timeElapsed;
      const frictionDeltaY = frictionY * timeElapsed;

      balls.forEach((ball) => {
        if (velocityChangeX == 0) {
          // No rotation, the plane is flat
          // On flat surface friction can only slow down, but not reverse movement
          ball.velocityX = slow(ball.velocityX, frictionDeltaX);
        } else {
          ball.velocityX = ball.velocityX + velocityChangeX;
          ball.velocityX = Math.max(Math.min(ball.velocityX, 1.5), -1.5);
          ball.velocityX =
            ball.velocityX - Math.sign(velocityChangeX) * frictionDeltaX;
          ball.velocityX = Math.minmax(ball.velocityX, maxVelocity);
        }

        if (velocityChangeY == 0) {
          // No rotation, the plane is flat
          // On flat surface friction can only slow down, but not reverse movement
          ball.velocityY = slow(ball.velocityY, frictionDeltaY);
        } else {
          ball.velocityY = ball.velocityY + velocityChangeY;
  ball.velocityY =
    ball.velocityY - Math.sign(velocityChangeY) * frictionDeltaY;
  ball.velocityY = Math.minmax(ball.velocityY, maxVelocity);
}

// Preliminary next ball position, only becomes true if no hit occurs
// Used only for hit testing, does not mean that the ball will reach this position
ball.nextX = ball.x + ball.velocityX;
ball.nextY = ball.y + ball.velocityY;

if (debugMode) console.log("tick", ball);

walls.forEach((wall, wi) => {
  if (wall.horizontal) {
    // Horizontal wall

    if (
      ball.nextY + ballSize / 2 >= wall.y - wallW / 2 &&
      ball.nextY - ballSize / 2 <= wall.y + wallW / 2
    ) {
      // Ball got within the strip of the wall
      // (not necessarily hit it, could be before or after)

      const wallStart = {
        x: wall.x,
        y: wall.y
      };
      const wallEnd = {
        x: wall.x + wall.length,
        y: wall.y
      };

      if (
        ball.nextX + ballSize / 2 >= wallStart.x - wallW / 2 &&
        ball.nextX < wallStart.x
      ) {
        // Ball might hit the left cap of a horizontal wall
        const distance = distance2D(wallStart, {
          x: ball.nextX,
          y: ball.nextY
        });
        if (distance < ballSize / 2 + wallW / 2) {
          if (debugMode && wi > 4)
            console.warn("too close h head", distance, ball);

          // Ball hits the left cap of a horizontal wall
          const closest = closestItCanBe(wallStart, {
            x: ball.nextX,
            y: ball.nextY
          });
          const rolled = rollAroundCap(wallStart, {
```

```
        x: closest.x,
        y: closest.y,
        velocityX: ball.velocityX,
        velocityY: ball.velocityY
      });

      Object.assign(ball, rolled);
    }
  }

  if (
    ball.nextX - ballSize / 2 <= wallEnd.x + wallW / 2 &&
    ball.nextX > wallEnd.x
  ) {
    // Ball might hit the right cap of a horizontal wall
    const distance = distance2D(wallEnd, {
      x: ball.nextX,
      y: ball.nextY
    });
    if (distance < ballSize / 2 + wallW / 2) {
      if (debugMode && wi > 4)
        console.warn("too close h tail", distance, ball);

      // Ball hits the right cap of a horizontal wall
      const closest = closestItCanBe(wallEnd, {
        x: ball.nextX,
        y: ball.nextY
      });
      const rolled = rollAroundCap(wallEnd, {
        x: closest.x,
        y: closest.y,
        velocityX: ball.velocityX,
        velocityY: ball.velocityY
      });

      Object.assign(ball, rolled);
    }
  }

  if (ball.nextX >= wallStart.x && ball.nextX <= wallEnd.x) {
    // The ball got inside the main body of the wall
    if (ball.nextY < wall.y) {
      // Hit horizontal wall from top
      ball.nextY = wall.y - wallW / 2 - ballSize / 2;
    } else {
      // Hit horizontal wall from bottom
      ball.nextY = wall.y + wallW / 2 + ballSize / 2;
    }
    ball.y = ball.nextY;
    ball.velocityY = -ball.velocityY / 3;
```

```
      if (debugMode && wi > 4)
        console.error("crossing h line, HIT", ball);
    }
  }
} else {
  // Vertical wall

  if (
    ball.nextX + ballSize / 2 >= wall.x - wallW / 2 &&
    ball.nextX - ballSize / 2 <= wall.x + wallW / 2
  ) {
    // Ball got within the strip of the wall
    // (not necessarily hit it, could be before or after)

    const wallStart = {
      x: wall.x,
      y: wall.y
    };
    const wallEnd = {
      x: wall.x,
      y: wall.y + wall.length
    };

    if (
      ball.nextY + ballSize / 2 >= wallStart.y - wallW / 2 &&
      ball.nextY < wallStart.y
    ) {
      // Ball might hit the top cap of a horizontal wall
      const distance = distance2D(wallStart, {
        x: ball.nextX,
        y: ball.nextY
      });
      if (distance < ballSize / 2 + wallW / 2) {
        if (debugMode && wi > 4)
          console.warn("too close v head", distance, ball);

        // Ball hits the left cap of a horizontal wall
        const closest = closestItCanBe(wallStart, {
          x: ball.nextX,
          y: ball.nextY
        });
        const rolled = rollAroundCap(wallStart, {
          x: closest.x,
          y: closest.y,
          velocityX: ball.velocityX,
          velocityY: ball.velocityY
        });

        Object.assign(ball, rolled);
      }
    }
```

```
      if (
        ball.nextY - ballSize / 2 <= wallEnd.y + wallW / 2 &&
        ball.nextY > wallEnd.y
      ) {
        // Ball might hit the bottom cap of a horizontal wall
        const distance = distance2D(wallEnd, {
          x: ball.nextX,
          y: ball.nextY
        });
        if (distance < ballSize / 2 + wallW / 2) {
          if (debugMode && wi > 4)
            console.warn("too close v tail", distance, ball);

          // Ball hits the right cap of a horizontal wall
          const closest = closestItCanBe(wallEnd, {
            x: ball.nextX,
            y: ball.nextY
          });
          const rolled = rollAroundCap(wallEnd, {
            x: closest.x,
            y: closest.y,
            velocityX: ball.velocityX,
            velocityY: ball.velocityY
          });

          Object.assign(ball, rolled);
        }
      }

      if (ball.nextY >= wallStart.y && ball.nextY <= wallEnd.y) {
        // The ball got inside the main body of the wall
        if (ball.nextX < wall.x) {
          // Hit vertical wall from left
          ball.nextX = wall.x - wallW / 2 - ballSize / 2;
        } else {
          // Hit vertical wall from right
          ball.nextX = wall.x + wallW / 2 + ballSize / 2;
        }
        ball.x = ball.nextX;
        ball.velocityX = -ball.velocityX / 3;

        if (debugMode && wi > 4)
          console.error("crossing v line, HIT", ball);
      }
    }
  }
});

// Detect is a ball fell into a hole
if (hardMode) {
        holes.forEach((hole, hi) => {
          const distance = distance2D(hole, {
            x: ball.nextX,
            y: ball.nextY
          });

          if (distance <= holeSize / 2) {
            // The ball fell into a hole
            holeElements[hi].style.backgroundColor = "red";
            throw Error("The ball fell into a hole");
          }
        });
      }

      // Adjust ball metadata
      ball.x = ball.x + ball.velocityX;
      ball.y = ball.y + ball.velocityY;
    });

    // Move balls to their new position on the UI
    balls.forEach(({ x, y }, index) => {
      ballElements[index].style.cssText = `left: ${x}px; top: ${y}px; `;
    });
  }

  // Win detection
  if (
    balls.every(
      (ball) => distance2D(ball, { x: 350 / 2, y: 315 / 2 }) < 65 / 2
    )
  ) {
    noteElement.innerHTML = `Congrats, you did it!
      ${!hardMode ? "<p>Press H for hard mode</p>" : ""}
      <p>
        Follow me
        <a href="https://twitter.com/HunorBorbely" , target="_top"
          >@HunorBorbely</a
        >
      </p>`;
    noteElement.style.opacity = 1;
    gameInProgress = false;
  } else {
    previousTimestamp = timestamp;
    window.requestAnimationFrame(main);
  }
} catch (error) {
  if (error.message == "The ball fell into a hole") {
    noteElement.innerHTML = `A ball fell into a black hole! Press space to reset the game.
      <p>
        Back to easy? Press E
      </p>`;
      noteElement.style.opacity = 1;
      gameInProgress = false;
    } else throw error;
  }
}
```



Click the joystick to start!

Move every ball to the center. Ready for hard mode? Press H