Trashbin Toss Game

Atharva Sachin Khadgi
a.khadgi001@umb.edu
University of Massachusetts Boston

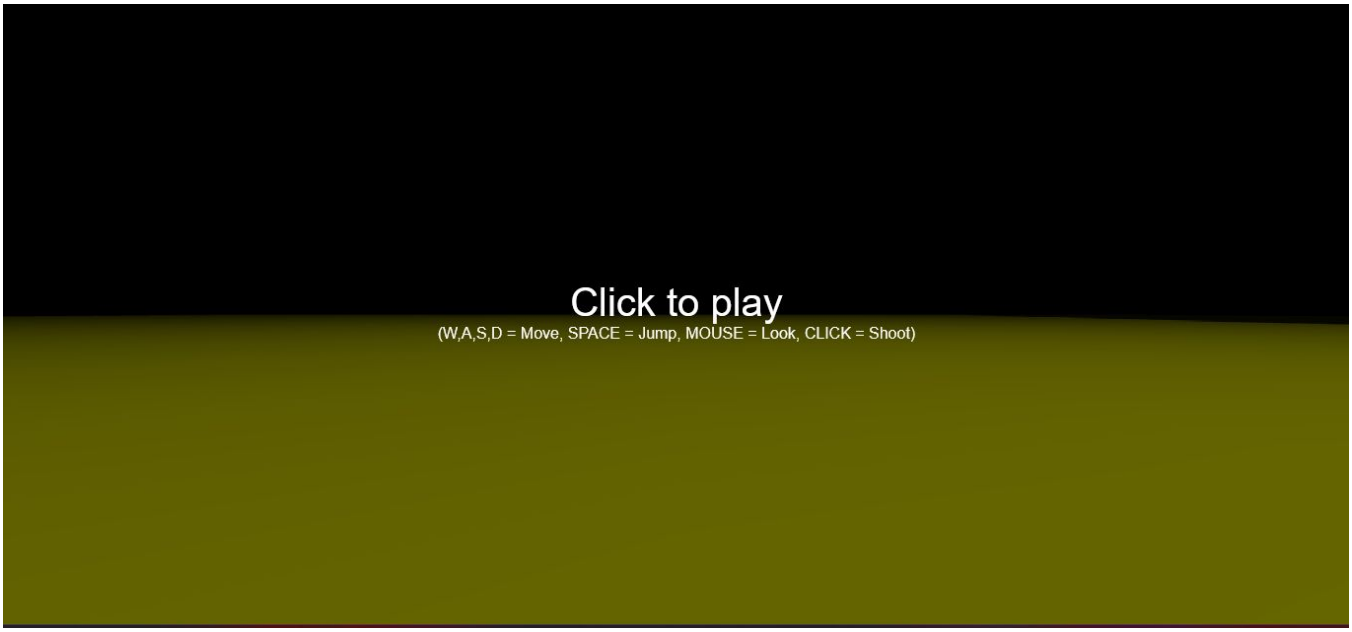

Figure 1: Starting state of the game

ABSTRACT

This project implements the classic "paper ball in trashcan" game. The users are given the freedom to move around the bounded map, which also has a trashcan, the aim of this game is to let users throw a paper ball on click, with the obvious aim being that it should land inside the bin.

KEYWORDS

WebGL, Visualization, game, simulation, HTML.

ACM Reference Format:
Atharva Sachin Khadgi. 2022. Trashbin Toss Game. In *CS460: Computer Graphics at UMass Boston, Fall 2022*. Boston, MA, USA, 3 pages. https://CS460.org

1 INTRODUCTION

This project includes a lot of concepts that were covered in cs460 class, moreover, I have made it as realistic as possible, which means that this simulation also has rules of physics that apply in the real world, i.e the trash can tips over on impact, collisions, etc.

2 RELATED WORK

There were similar projects implemented before which served as inspiration for this project, even though the idea of the project was something I conceptualized on my own, I did refer to a number of state-of-the-art projects to get an idea about how to execute this project. [1] served as the main inspiration, altho there were a number of code pens that I referred to for styling purposes.

3 METHOD

Here are the general steps that were taken to create a three.js game with a paperball trashcan:

The development environment was set up by installing the necessary tools, such as a text editor, a local server, and a browser.

A new HTML file was created and the three.js library was included in the head of the HTML file. This was done by adding a script tag with the src attribute set to the three.js library file.

A canvas element was created in the body of the HTML file. This was used to render the 3D graphics.

In the JavaScript file, a new scene and a camera were created. The scene is where all of the 3D objects were placed, and the camera was used to view the scene from a specific perspective.

CS460, Fall 2022, Boston, MA

ACM ISBN 1337.
https://CS460.org

A trashcan model was created using Three.js geometry and material objects. The THREE.CylinderGeometry class was used to create the cylinder shape of the trashcan, and the THREE.MeshBasicMaterial class was used to give it a simple, flat color.

A paperball model was created using Three.js geometry and material objects. The THREE.SphereGeometry class was used to create the sphere shape of the paperball, and the THREE.MeshBasicMaterial class was used to give it a simple, flat color.

The trashcan and paperball models were added to the scene. This was done by calling the add method on the scene object and passing in the models as arguments.

A rendering loop was set up using the requestAnimationFrame function. This ensured that the scene was constantly re-drawn at a specific frame rate.

Within the rendering loop, the position of the paperball model was updated based on user input or other game logic. This was done by modifying the position property of the paperball model.

Collisions between the paperball and trashcan models were checked using the THREE.Raycaster class. If a collision was detected, the game state and score were updated accordingly.

The scene was rendered using the render method of the WebGLRenderer object. This drew the 3D graphics to the canvas element.

3.1 Implementation

Please tell the reader how you implemented the project. You can include code snippets that you want to highlight. Don't include the whole code.

```
var contactNormal = new CANNON.Vec3(); // Normal in the contact, pointing "out" of whatever the player touched
var upAxis = new CANNON.Vec3(0,1,0);
cannonBody.addEventListener("collide",function(e){
    var contact = e.contact;

    // contact.bi and contact.bj are the colliding bodies, and contact.ni is the collision normal.
    // We do not yet know which one is which! Let's check.
    if(contact.bi.id == cannonBody.id)  // bi is the player body, flip the contact normal
        contact.ni.negate(contactNormal);
    else
        contactNormal.copy(contact.ni); // bi is something else. Keep the normal as it is

    // If contactNormal.dot(upAxis) is between 0 and 1, we know that the contact normal is somewhat in the up direction
    if(contactNormal.dot(upAxis) > 0.5) // Use a "good" threshold value between 0 and 1 here!
        canJump = true;
});
```

Figure 2: Function for collissions

```
var velocity = cannonBody.velocity;

var PI_2 = Math.PI / 2;

var onMouseMove = function ( event ) {

    if ( scope.enabled === false ) return;

    var movementX = event.movementX || event.mozMovementX || event.webkitMovementX || 0;
    var movementY = event.movementY || event.mozMovementY || event.webkitMovementY || 0;

    yawObject.rotation.y -= movementX * 0.002;
    pitchObject.rotation.x -= movementY * 0.002;

    pitchObject.rotation.x = Math.max( - PI_2, Math.min( PI_2, pitchObject.rotation.x ) );
```

Figure 3: Function for velocity

3.2 Milestones

I will describe the journey of the development of this website right from it's conceptual inception to its final implementation.

```
// Moves the camera to the Cannon.js object position and adds velocity to the object if the run key is down
var inputVelocity = new THREE.Vector3();
var euler = new THREE.Euler();
this.update = function ( delta ) {

    if ( scope.enabled === false ) return;

    delta *= 0.1;

    inputVelocity.set(0,0,0);

    if ( moveForward ){
        inputVelocity.z = -velocityFactor * delta;
    }
    if ( moveBackward ){
        inputVelocity.z = velocityFactor * delta;
    }

    if ( moveLeft ){
        inputVelocity.x = -velocityFactor * delta;
    }
    if ( moveRight ){
        inputVelocity.x = velocityFactor * delta;
    }
```

Figure 4: Camera with respect to object position and velocity to the object

```
// Convert velocity to world coordinates
euler.x = pitchObject.rotation.x;
euler.y = yawObject.rotation.y;
euler.order = "XYZ";
quat.setFromEuler(euler);
inputVelocity.applyQuaternion(quat);
//quat.multiplyVector3(inputVelocity);

// Add to the object
velocity.x += inputVelocity.x;
velocity.z += inputVelocity.z;

yawObject.position.copy(cannonBody.position);
```

Figure 5: velocity to world coordinates

```
// Setup our world
world = new CANNON.World();
world.quatNormalizeSkip = 0;
world.quatNormalizeFast = false;

var solver = new CANNON.GSSolver();

world.defaultContactMaterial.contactEquationStiffness = 1e9;
world.defaultContactMaterial.contactEquationRelaxation = 4;

solver.iterations = 7;
solver.tolerance = 0.1;
var split = true;
if(split)
    world.solver = new CANNON.SplitSolver(solver);
else
    world.solver = solver;

world.gravity.set(0,-20,0);
world.broadphase = new CANNON.NaiveBroadphase();
```

Figure 6: setup world

3.2.1 Milestone 1. I wanted to simulate simple, day-to-day, known to all daily tasks or something trivial right in the browser. I then jotted down a list of possible games that I used to play during my childhood, that list had some games like tic-tac-toe, Bingo, Housie, etc. I then narrowed down the list and selected this as my project.

```
// Update ball positions
for(var i=0; i<balls.length; i++){
    ballMeshes[i].position.copy(balls[i].position);
    ballMeshes[i].quaternion.copy(balls[i].quaternion);
}

// Update box positions
for(var i=0; i<boxes.length; i++){
    boxMeshes[i].position.copy(boxes[i].position);
    boxMeshes[i].quaternion.copy(boxes[i].quaternion);
}
```

Figure 7: Ball and box postion

3.2.2 Milestone 2. I wanted the user to roam freely and explore a bit of the small world that I planed to create, hence I allowed movements and jumping in an otherwise simple game of throwing paper balls in a trashcan.

3.2.3 Milestone 3. I wanted my "world" to come to life and make everything feel realistic, so I made sure to implement laws of physics in this virtual world of mine and allow objects to react upon collisions, stopping due to friction, etc.

3.2.4 Milestone 4. Worked on the project's core elements and implemented them, the boxes, paper balls, the ground, their dimensions, colors, textures, etc. were made in this step.

3.2.5 Milestone 5. Polished my work and added some final touches. Mostly aesthetic enhancements.

3.3 Challenges

This section describes the challenges I faced. and tackled while working on this project.

- Challenge 1: Determining the paper ball direction and velocity was quite challenging.
- Challenge 2: Maintaining camera movement while object positioning was also difficult.
- Challenge 3: I am currently trying to generate shapes with cavities that will represent a trashcan

4 RESULTS

Figure 8: End result

5 CONCLUSIONS

In conclusion, the three.js game with a paperball trashcan was a successful project that demonstrated the capabilities of the three.js library. The development process involved setting up a development environment, creating a scene and camera, creating 3D models for the trashcan and paperball, adding the models to the scene, setting up a rendering loop, updating the position of the paperball based on user input or other game logic, checking for collisions between the paperball and trashcan, and rendering the scene. These steps allowed for the creation of a functional and interactive game that could be enjoyed by players. Overall, the use of three.js allowed for the creation of a visually appealing and engaging game that made use of 3D graphics and physics.

REFERENCES

[1] https://shohei-pf.work/3d/basket/object.html
[2] https //discourse.threejs.org/t/basketball-shooting-game/22744
[3] schteppe / https://github.com/schteppe
[4] mrdoob / http://mrdoob.com/