



UNIVERSITÉ PARIS SACLAY

VIRTUAL AND MIXED REALITY

MASTER OF RESEARCH IN ARTIFICIAL INTELLIGENCE AND  
MACHINE VISION

---

## Lab session

Using and Programming Virtual Reality Headsets

---

*Student:*

Yann Terrom  
Dat Nguyen  
Giuseppe Ricciardi  
Chakib Gamir

*Supervisor:*

Jean-Yves Didier

*Master:*

M2MMVAI

Academic Year  
2023-2024

## Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	Team Members . . . . .	4
1.2	Tower of Hanoi Rules . . . . .	4
1.3	Technical Framework . . . . .	4
<b>2</b>	<b>Environment Implementation</b>	<b>5</b>
2.1	Basic Game Elements: Disks and Structure . . . . .	5
2.1.1	Disk Class . . . . .	5
2.1.2	MainStructure Class . . . . .	7
2.1.3	Board Class . . . . .	9
2.2	3D Background Environment . . . . .	12
2.3	Background Music . . . . .	13
2.4	Final Environment . . . . .	15
<b>3</b>	<b>Physics Implementation</b>	<b>15</b>
3.1	Drag and Drop Implementation with DragControls . . . . .	15
<b>4</b>	<b>Rule Implementation</b>	<b>16</b>
4.1	Collision Detection . . . . .	16
4.2	Drag-and-Drop Interaction . . . . .	17

## List of Figures

1	Instance of Disk . . . . .	7
2	Instance of MainStructure . . . . .	9
3	Instance of Board . . . . .	12
4	Part of the 3D Background . . . . .	13
5	Final Environment with 3 Disks, Main Structure, Background, and Music. . . . .	15

## Listings

1	JS Class for Disks . . . . .	5
2	JS Example for Disk . . . . .	7
3	JS Class for MainStructure . . . . .	7
4	JS Example for MainStructure . . . . .	9
5	JS Class for Board . . . . .	9
6	JS Example for Board . . . . .	11
7	JS Code for Including 3D Background . . . . .	12
8	JS code for adding Audio . . . . .	13
9	JS Class for Drag Control . . . . .	15
10	Collision Detection Function . . . . .	16
11	Drag-and-Drop Interaction . . . . .	17

## 1 Introduction

This report provides an overview of the extended reality (XR) game development project undertaken by the team consisting of Yann Terrom, Dat Nguyen, Giuseppe Ricciardi, and Chakib Gamir. The primary objective of the project was to design and implement a 3D game for virtual reality headsets, such as Oculus, focusing on the classic puzzle known as the Tower of Hanoi.

### 1.1 Team Members

- Yann Terrom - Contributed to the project's structural development, including setting up the project using npm. Worked on creating fundamental game objects such as disks, the main game structure, background music, and the 3D environment.
- Dat Nguyen - Collaborated on object selection and played a crucial role in refining the physics of the game, incorporating elements like gravity and collision detection.
- Giuseppe Ricciardi - Collaborated on object selection and played a crucial role in refining the physics of the game, incorporating elements like gravity and collision detection.
- Chakib Gamir - Contributed to the project's structural development, including setting up the project using npm. Worked on creating fundamental game objects such as disks, the main game structure, background music, and the 3D environment.

After establishing individual contributions, the team members collaborated to develop the game logic, including defining the rules and overall gameplay.

### 1.2 Tower of Hanoi Rules

The Tower of Hanoi is a mathematical puzzle that involves three pegs and a number of disks of different sizes. The puzzle starts with the disks in a neat stack in ascending order of size on one peg, the smallest at the top. The objective is to move the entire stack to another peg, obeying the following rules:

1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or on an empty peg.
3. No disk may be placed on top of a smaller disk.

The game is typically played with three pegs, but the number of pegs can vary. The minimal number of moves required to solve a Tower of Hanoi puzzle with  $n$  disks is  $2^n - 1$ .

### 1.3 Technical Framework

The Tower of Hanoi extended reality game was developed with a focus on immersive experiences using the Oculus headset. The team employed JavaScript as the primary programming language for its versatility and wide compatibility. The project's structure was facilitated by Vite, a build tool that enhances the development workflow. Additionally, Three.js, a popular JavaScript library, was utilized for managing the 3D aspects of the game, providing a robust foundation for creating interactive and visually appealing virtual environments.

This combination of technologies allowed the team to seamlessly integrate the game with the Oculus headset, providing users with an engaging and immersive experience. The subsequent sections of this report will delve into the specific details of the game's implementation, covering both the technical and creative aspects of our Tower of Hanoi extended reality project.

## 2 Environment Implementation

To create a captivating Tower of Hanoi extended reality game, the development of the environment involves several crucial elements. At its core, the game requires the creation of disks and a structure to hold them. This basic setup lays the foundation for the Tower of Hanoi puzzle. Building upon this foundation, the team explored ways to enhance the user experience by incorporating a 3D background environment and background music.

### 2.1 Basic Game Elements: Disks and Structure

The fundamental components of the Tower of Hanoi game are the disks and the structure that supports them. Disks of varying sizes are essential for representing the puzzle's elements, and a solid structure provides the foundation for the puzzle-solving process. In our implementation, we used Three.js, a powerful JavaScript library, to model and render these components in the virtual space. This allowed for seamless interaction and realistic visuals, contributing to an engaging gaming experience.

Now, let's delve into the implementation details of the `Disk` and `MainStructure` classes, showcasing how these classes contribute to the creation and management of the Tower of Hanoi game elements.

#### 2.1.1 Disk Class

To represent the disks within the Tower of Hanoi game, we designed a versatile and reusable `Disk` class using JavaScript and the Three.js library. The `Disk` class encapsulates the properties and behaviors of a disk with a central hole, contributing to the visual and interactive aspects of the game.

**Class Structure** The `Disk` class is structured to be flexible and easy to use, allowing the creation of disks with various specifications. The constructor function initializes the disk with parameters such as radius, hole radius, height, and the file path for the disk's texture image.

```
1 /**
2  * Class representing a disk with a hole in the center.
3 */
4 export class Disk {
5   /**
6    * Create a disk with a hole.
7    * @param {number} radius - The radius of the disk.
8    * @param {number} holeRadius - The hole radius of the disk (default 0.5).
9    * @param {number} height - The height of the disk (default 0.5).
10   * @param {string} texturePath - The file path for the disk texture image.
11   * @param {number} mass - The mass of the disk (default 1).
12 */
13 constructor(radius, holeRadius = 0.1, height = 0.1, texturePath, mass = 1) {
14   this.radius = radius;
15   this.holeRadius = holeRadius;
16   this.height = height;
17   this.texturePath = texturePath;
18   this.mass = mass;
19   this.createDisk();
20   this.setDefaultPosition();
21 }
22
23 /**
24  * Create the geometry and material for the disk.
25 */
26 createDisk() {
27   const loader = new THREE.TextureLoader();
28   const texture = loader.load(this.texturePath);
29
30   // Create the geometry of the disk
31   const diskShape = new THREE.Shape();
32   diskShape.moveTo(0, 0);
```

```

34     diskShape.absarc(0, 0, this.radius, 0, Math.PI * 2, false);
35
36     // Create the geometry of the hole
37     const holeShape = new THREE.Path();
38     holeShape.moveTo(0, 0);
39     holeShape.absarc(0, 0, this.holeRadius, 0, Math.PI * 2, true);
40
41     // Subtract the hole from the disk geometry
42     diskShape.holes.push(holeShape);
43
44     // Use ExtrudeGeometry to create the extruded geometry
45     const extrudeSettings = {
46         depth: this.height, // Extrusion thickness
47         bevelEnabled: false, // No bevel to get a flat shape
48     };
49
50     const diskGeometry = new THREE.ExtrudeGeometry(diskShape, extrudeSettings);
51
52
53     const diskMaterial = new THREE.MeshBasicMaterial({ map: texture });
54
55
56     // Create the mesh
57     this.mesh = new THREE.Mesh(diskGeometry, diskMaterial);
58     this.mesh.userData.physics = { mass: this.mass };
59
60     // Add edges with a line material
61     const edgesGeometry = new THREE.EdgesGeometry(diskGeometry);
62     const edgesMaterial = new THREE.LineBasicMaterial({ color: 0x000000 });
63     const edges = new THREE.LineSegments(edgesGeometry, edgesMaterial);
64
65     // Add the edges to the main mesh
66     this.mesh.add(edges);
67 }
68
69 setDefaultPosition() {
70     this.mesh.rotation.x = -Math.PI / 2; // -90deg rotation X axis
71     this.mesh.position.set(0, 0, 0);
72 }
73
74 /**
75 * Set the position of the disk.
76 * @param {number} x - The x-coordinate.
77 * @param {number} y - The y-coordinate.
78 * @param {number} z - The z-coordinate.
79 */
80 setPosition(x, y, z) {
81     this.mesh.position.set(x, y, z);
82 }
83
84 /**
85 * Set the rotation of the disk.
86 * @param {number} x - The rotation around the x-axis in radians.
87 * @param {number} y - The rotation around the y-axis in radians.
88 * @param {number} z - The rotation around the z-axis in radians.
89 */
90 setRotation(x, y, z) {
91     this.mesh.rotation.set(x, y, z);
92 }
93
94 /**
95 * Get the height of the disk.
96 * @returns {number} The height of the disk.
97 */
98 getHeightDisk() {
99     return this.height ;
100 }
101 }
```

Listing 1: JS Class for Disks

**Geometry and Material Creation** The `createDisk` method is responsible for generating the geometry and material for the disk. It uses Three.js to create a disk shape and a hole shape, subtracts the hole from the disk geometry, and then utilizes ExtrudeGeometry to create the final 3D geometry. A texture is applied to the material for a realistic appearance, and edges are added to enhance visualization.

**Positioning and Rotation** The `setDefaultPosition` method establishes the default rotation and position for the disk. Additionally, the class provides methods (`setPosition` and `setRotation`) for dynamically setting the position and rotation of the disk during runtime.

**Height Retrieval** To retrieve the height of the disk, the `getHeightDisk` method is available, facilitating interaction with other elements in the Tower of Hanoi game.

This modular approach allows for the easy integration of disks into the game, providing a foundation for creating an immersive and visually appealing Tower of Hanoi extended reality experience.

```
1 // Example Usage
2 const disk1 = new Disk(1, 0.2, 0.1, 'path/to/texture.jpg');
3 disk1.setPosition(0, 0, 0);
```

Listing 2: JS Example for Disk

In this example, a disk is instantiated with specific parameters, and its position is set to (0, 0, 0). The `Disk` class provides a clean and efficient way to manage the visual aspects of the Tower of Hanoi game.

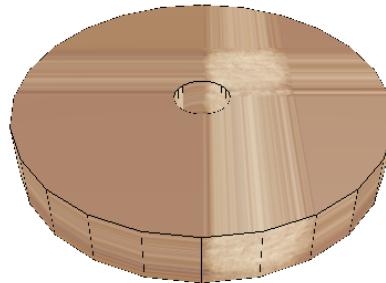


Figure 1: Instance of Disk

### 2.1.2 MainStructure Class

The `MainStructure` class is responsible for creating the central structure in the Tower of Hanoi game, consisting of a base rectangle and three cylinders. Let's focus on the `createMainStructure` method, which initializes and positions these components.

**Class Structure** The `createMainStructure` method is a crucial part of the `MainStructure` class, where the geometry and material for the main structure are generated. Here's a detailed breakdown of the method:

```
1 export class MainStructure {
2   /**
3    * Create the main structure with a base rectangle and three cylinders on top.
4    * @param {number} baseWidth - The width of the base rectangle.
5 }
```

```

6   * @param {number} baseDepth - The depth of the base rectangle.
7   * @param {number} baseHeight - The height of the base rectangle.
8   * @param {number} cylinderRadius - The radius of the cylinders.
9   * @param {number} cylinderHeight - The height of the cylinders.
10  * @param {string} baseTexturePath - The file path for the base texture image.
11  * @param {string} cylinderTexturePath - The file path for the cylinder texture
12  *      image.
13  */
14 constructor(baseWidth, baseDepth, baseHeight, cylinderRadius, cylinderHeight,
15             baseTexturePath, cylinderTexturePath) {
16     this.baseWidth = baseWidth;
17     this.baseDepth = baseDepth;
18     this.baseHeight = baseHeight;
19     this.cylinderRadius = cylinderRadius;
20     this.cylinderHeight = cylinderHeight;
21     this.baseTexturePath = baseTexturePath;
22     this.cylinderTexturePath = cylinderTexturePath;
23
24     this.createMainStructure();
25     this.setDefaultPosition();
26 }
27 /**
28 * Create the geometry and material for the main structure.
29 */
30 createMainStructure() {
31     const loader = new THREE.TextureLoader();
32
33     // Calculate the extended width of the base rectangle
34     const extendedWidth = this.baseWidth + 2 * this.cylinderRadius;
35
36     // Calculate the width of each section
37     const sectionWidth = this.baseWidth / 3;
38
39     // Load textures
40     const baseTexture = loader.load(this.baseTexturePath);
41     const cylinderTexture = loader.load(this.cylinderTexturePath);
42
43     // Create the base rectangle geometry with extended sides
44     const baseGeometry = new THREE.BoxGeometry(extendedWidth, this.baseHeight, this
45         .baseDepth);
46     const baseMaterial = new THREE.MeshBasicMaterial({ map: baseTexture });
47     this.baseMesh = new THREE.Mesh(baseGeometry, baseMaterial);
48     this.baseMesh.userData.physics = { mass: 1 }; // Set mass for the base
49
50     // Create the cylinder geometry
51     const cylinderGeometry = new THREE.CylinderGeometry(this.cylinderRadius, this.
52         cylinderRadius, this.cylinderHeight, 32);
53     const cylinderMaterial = new THREE.MeshBasicMaterial({ map: cylinderTexture });
54
55     // Create cylinders and position them in the middle of each section
56     this.cylinder1 = new THREE.Mesh(cylinderGeometry, cylinderMaterial);
57     this.cylinder1.position.set(-extendedWidth / 2 + this.cylinderRadius +
58         sectionWidth / 2, this.baseHeight / 2 + this.cylinderHeight / 2, 0);
59     this.cylinder1.userData.physics = { mass: 1 }; // Set mass for the cylinder
60
61     this.cylinder2 = new THREE.Mesh(cylinderGeometry, cylinderMaterial);
62     this.cylinder2.position.set(-extendedWidth / 2 + this.cylinderRadius +
63         sectionWidth + sectionWidth / 2, this.baseHeight / 2 + this.cylinderHeight
64         / 2, 0);
65     this.cylinder2.userData.physics = { mass: 1 }; // Set mass for the cylinder
66
67     // Create a group to hold all the objects
68     this.mainStructure = new THREE.Group();

```

```

67     this.mainStructure.add(this.baseMesh);
68     this.mainStructure.add(this.cylinder1);
69     this.mainStructure.add(this.cylinder2);
70     this.mainStructure.add(this.cylinder3);
71   }
72 // Getter and Setter ...

```

Listing 3: JS Class for MainStructure

- **Texture Loading:** The method starts by loading textures for both the base rectangle and the cylinders using the Three.js `TextureLoader`.
- **Base Rectangle Creation:** It creates a base rectangle geometry (`baseGeometry`) with extended sides to accommodate the cylinders. The material for the base (`baseMaterial`) is mapped with the loaded base texture.
- **Cylinder Creation:** Three cylinders (`cylinder1`, `cylinder2`, and `cylinder3`) are created with appropriate positions. The cylinders are positioned in the middle of each section of the extended base rectangle.
- **Physics Properties:** Physics properties (mass) are set for both the base and cylinders, which is useful for physics simulations.
- **Grouping Objects:** All created objects, including the base rectangle and cylinders, are grouped together using a Three.js `Group` (`mainStructure`).

This method ensures that the main structure is created with the correct geometry, material, and positioning, laying the foundation for an integral part of the Tower of Hanoi extended reality game.

```

1 // Example Usage
2 const mainStructure = new MainStructure(5, 1, 0.1, 0.05, 0.7, 'path/to/texture_base
3 .jpg', 'path/to/texture_cylinder.jpg')
mainStructure.setPosition(0, 0, 0);

```

Listing 4: JS Example for MainStructure



Figure 2: Instance of MainStructure

### 2.1.3 Board Class

To manage the overall presentation and interaction in the Tower of Hanoi game, we implemented the `Board` class using JavaScript and the Three.js library. The `Board` class encompasses the visual representation of the game board, including buttons for essential actions and dynamic text to display moves and status.

**Class Structure** The `Board` class is designed to provide a customizable game board with default parameters for color, height, and width. It includes buttons for actions such as exit, reset, and play/pause, each with associated text for user guidance. The class maintains a count of moves and status information to enhance the gaming experience.

```

1 export class Board {
2   constructor(color = 'grey', boardHeight = 1, boardWidth = 1) {
3     this.color = color;
4     this.boardHeight = boardHeight;

```

```
5     this.boardWidth = boardWidth;
6     this.moves = 0; // Initialize moves
7     this.status = 'valid'; // Initialize status
8
9     // Create the board geometry and material
10    const boardGeometry = new THREE.PlaneGeometry(this.boardWidth, this.
11        boardHeight);
12    const boardMaterial = new THREE.MeshBasicMaterial({ color: this.color, side:
13        THREE.DoubleSide });
14    this.board = new THREE.Mesh(boardGeometry, boardMaterial);
15
16    // Create buttons
17    this.createButtons();
18    this.createText();
19 }
20
21 createButtons() {
22     // Define button dimensions
23     const buttonWidth = 0.6;
24     const buttonHeight = 0.2;
25     const buttonDepth = 0.1;
26     const buttonGeometry = new THREE.BoxGeometry(buttonWidth, buttonHeight,
27         buttonDepth);
28
29     // Example: Exit Button
30     const exitMaterial = new THREE.MeshBasicMaterial({ color: 0xff0000 }); // red
31         color
32     this.exit = new THREE.Mesh(buttonGeometry, exitMaterial);
33     this.exit.position.set(-this.boardWidth / 2 + buttonWidth, this.boardHeight /
34         2 - buttonHeight, 0.1);
35
36     // Example: Reset Button
37     const resetMaterial = new THREE.MeshBasicMaterial({ color: 0x0000ff }); // blue
38         color
39     this.reset = new THREE.Mesh(buttonGeometry, resetMaterial);
40     this.reset.position.set(-this.boardWidth / 2 + 3 * buttonWidth, this.
41         boardHeight / 2 - buttonHeight, 0.1);
42
43     const playMaterial = new THREE.MeshBasicMaterial({ color: 0x0000ff }); // blue
44         color
45     this.play = new THREE.Mesh(buttonGeometry, playMaterial);
46     this.play.position.set(-this.boardWidth / 2 + 5 * buttonWidth, this.
47         boardHeight / 2 - buttonHeight, 0.1);
48
49     const loader = new FontLoader();
50     loader.load('./assets/helvetiker_regular.typeface.json', (font)=>{
51         const textOptions = {
52             font: font,
53             size: 0.05,
54             height: 0.05,
55         };
56         // Create text geometry for each button
57         const exitTextGeometry = new TextGeometry('Exit', textOptions);
58         const resetTextGeometry = new TextGeometry('Reset', textOptions);
59         const play_pauseTextGeometry = new TextGeometry('Play/Pause', textOptions);
60
61         // Create meshes for the text
62         const textMaterial = new THREE.MeshBasicMaterial({ color: 0xffffffff });
63         const exitText = new THREE.Mesh(exitTextGeometry, textMaterial);
64         const resetText = new THREE.Mesh(resetTextGeometry, textMaterial);
65         const playPauseText = new THREE.Mesh(play_pauseTextGeometry, textMaterial);
66
67         // Position the text on the buttons
68         exitText.position.set(-0.1, -0.05, 0.05); // Adjust these values as needed
69         resetText.position.set(-0.1, -0.05, 0.05);
70         playPauseText.position.set(-0.1, -0.05, 0.05);
71
72         // Add the text to the buttons
73     }
```

```

66     console.log(this.exit)
67     this.exit.add(exitText);
68     this.reset.add(resetText);
69     this.play.add(playPauseText);
70   });
71
72   this.board.add(this.exit);
73   this.board.add(this.reset);
74   this.board.add(this.play);
75 }
76
77 createText() {
78   const loader = new FontLoader();
79   loader.load('./assets/helvetiker_regular.typeface.json', (font) => {
80     this.font = font; // Store the font for later use
81     this.updateMovesAndStatus(0, 'valid'); // Initial text setup
82   });
83 }
84
85 updateMovesAndStatus(moves, invalid) {
86   this.moves = moves;
87   if (invalid == true){
88     this.status = 'invalid';
89   }
90   else{
91     this.status = 'valid';
92   }
93
94   const textOptions = {
95     font: this.font,
96     size: 0.3,
97     height: 0.05,
98   };
99
100  // Update moves text
101  if (this.movesText) this.board.remove(this.movesText); // Remove old text
102  const movesTextGeometry = new TextGeometry('Moves: ' + this.moves,
103    textOptions);
104  const movesTextMaterial = new THREE.MeshBasicMaterial({ color: 0xffffffff });
105  this.movesText = new THREE.Mesh(movesTextGeometry, movesTextMaterial);
106  this.movesText.position.set(-this.boardHeight / 2, 0, 0);
107  this.board.add(this.movesText);
108
109  // Update status text
110  if (this.statusText) this.board.remove(this.statusText); // Remove old text
111  const statusTextGeometry = new TextGeometry('Status: ' + this.status,
112    textOptions);
113  const statusTextMaterial = new THREE.MeshBasicMaterial({ color: 0xffffffff });
114  this.statusText = new THREE.Mesh(statusTextGeometry, statusTextMaterial);
115  this.statusText.position.set(-this.boardHeight / 2, -this.boardWidth / 4, 0);
116  this.board.add(this.statusText);
117 }
118 }

```

Listing 5: JS Class for Board

**Button and Text Creation** The `createButtons` method generates interactive buttons on the board, such as exit, reset, and play/pause. These buttons include 3D geometries with associated materials for visual appeal. Text labels are added to the buttons using the Three.js library's capabilities.

**Text Display and Updates** The `createText` method initializes the font for later use in displaying text on the board. The `updateMovesAndStatus` method dynamically updates and displays the move count and game status. This interactive text provides valuable feedback to the player during gameplay.

```
1 // Example Usage
```

```
2 const gameBoard = new Board('blue', 2, 2);
3 gameBoard.updateMovesAndStatus(5, false);
```

Listing 6: JS Example for Board

In this example, a game board is instantiated with specific parameters, and the moves and status are updated during gameplay. The `Board` class enhances the Tower of Hanoi game interface, combining visual elements and interactivity for an engaging user experience.

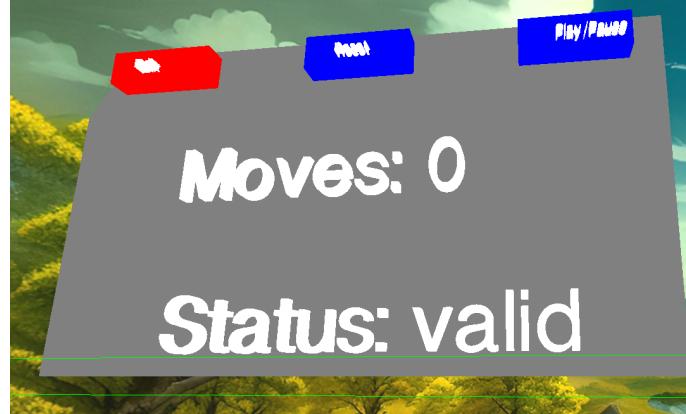


Figure 3: Instance of Board

## 2.2 3D Background Environment

To enhance the immersive quality of the Tower of Hanoi game, we implemented a 3D background environment using the `GLTFLoader` in `Three.js`. This addition introduces an extra layer to the scene, providing a sense of depth and presence for a more engaging visual experience.

```
1 setUpBackground() {
2     const loader = new GLTFLoader();
3     loader.load('../assets/models/back.glb',
4         (gltf) => {
5             const model = gltf.scene;
6             model.position.z = 1;
7             this.scene.add(model);
8         },
9         (xhr) => {
10             // Progress callback (optional)
11             // console.log((xhr.loaded / xhr.total * 100) + '% loaded');
12         },
13         (error) => {
14             console.error('Error loading the model:', error);
15         }
16     );
17 }
```

Listing 7: JS Code for Including 3D Background

### Code Explanation:

- **Loader Initialization:** An instance of the `GLTFLoader` is created, which is a loader specifically designed for loading models in the glTF format, a popular format for 3D models.
- **Model Loading:** The loader loads the 3D model from the specified file path ('`../assets/models/back.glb`'). Upon successful loading, the provided callback function is executed, receiving a glTF object (`gltf`).

- **Scene Placement:** The 3D model is retrieved from the glTF object (`gltf.scene`). The model's position is set to have a positive z-coordinate (`model.position.z = 1`), placing it slightly in front of the camera in the scene.
- **Adding to Scene:** The model is added to the main scene (`this.scene`), integrating it as part of the overall 3D environment.
- **Progress Callback (Optional):** During the loading process, a progress callback is provided (commented out in this case). It can be used to track the loading progress if needed.
- **Error Handling:** If there is an error during the loading process, an error message is logged to the console (`console.error('Error loading the model:', error)`).

This code segment ensures that the 3D background model is loaded and positioned appropriately within the Three.js scene, contributing to the immersive quality of the Tower of Hanoi game.

Here is a part of the background:



Figure 4: Part of the 3D Background

### 2.3 Background Music

To enrich the gaming experience, we introduced a carefully curated background soundtrack to the Tower of Hanoi extended reality game. The background music enhances the immersive atmosphere, providing players with a multi-sensory experience that complements the gameplay.

```

1  setUpAudio() {
2      // Create an audio listener attached to the camera
3      const listener = new THREE.AudioListener();
4      this.camera.add(listener);
5
6      // Create a global audio object
7      const audio = new THREE.Audio(listener);
8
9      // Create an AudioLoader
10     const audioLoader = new THREE.AudioLoader();
11
12     // Load the soundtrack
13     audioLoader.load('../assets/sakura.mp3', (buffer) => {
14         // Set the audio buffer and configure playback properties
15         audio.setBuffer(buffer);
16         audio.setLoop(true); // Loop the soundtrack
17         audio.setVolume(2); // Adjust the volume (2 times the normal volume)
18         audio.play(); // Start playing the soundtrack
19     });
20 }
```

Listing 8: JS code for adding Audio

**Code Explanation:**

- **Audio Listener Creation:** An audio listener is created using `THREE.AudioListener()`, and it is attached to the camera to ensure that the audio is spatialized based on the camera's position.
- **Global Audio Object:** A global audio object (`audio`) is created using the audio listener.
- **AudioLoader Initialization:** An `AudioLoader` is instantiated to load the soundtrack asynchronously.
- **Soundtrack Loading:** The `audioLoader.load()` method loads the soundtrack from the specified file path ('`..../assets/sakura.mp3`'). Upon successful loading, the provided callback function is executed, receiving an `AudioBuffer` (`buffer`).
- **Audio Configuration:** The audio buffer is set for the `audio` object, and playback properties are configured. The soundtrack is set to loop indefinitely, and its volume is adjusted (2 times the normal volume).
- **Start Playback:** The `audio.play()` method is called to start playing the background soundtrack.

This code segment ensures the seamless integration of background music into the Tower of Hanoi extended reality game, contributing to a more immersive and enjoyable gaming experience.

## 2.4 Final Environment



Figure 5: Final Environment with 3 Disks, Main Structure, Background, and Music.

## 3 Physics Implementation

### 3.1 Drag and Drop Implementation with DragControls

To enhance user interaction, we implemented the drag and drop functionality using DragControls in Three.js. This feature allows users to select and move game objects seamlessly within the 3D environment.

```
1 export class DraggablesManager {
2   constructor(objects, camera, renderer, scene) {
3     this.objects = objects;
4     this.dragControls = new DragControls(this.objects, camera, renderer.domElement)
5       ;
6
7   //this.dragControls.addEventListener('hoveron', this.onHoverOn.bind(this));
8   //this.dragControls.addEventListener('hoveroff', this.onHoverOff.bind(this));
9 }
10
11 addObject(object) {
12   this.objects.push(object);
13   this.dragControls.setObjects(this.objects);
14 }
15
16 onHoverOn(event) {
17   // Add your logic when hovering on an object
18   const hoveredObject = event.object;
19   console.log('Hovered on:', hoveredObject);
20 }
21
22 onHoverOff(event) {
```

```

23     // Add your logic when hovering off an object
24     const unhoveredObject = event.object;
25     console.log('Hovered off:', unhoveredObject);
26   }
27 }
```

Listing 9: JS Class for Drag Control

### Code Explanation:

- **DragControls Initialization:** We initialized the DragControls to enable the drag and drop functionality. This included creating an instance of DragControls, associating it with the camera and the array of draggable objects.
- **Event Handling:** We set up event listeners to handle the start, drag, and end events triggered by DragControls. These events enabled us to update the position of the dragged object during interaction.
- **Selectable Objects:** We defined the array of selectable objects to include the disks in the Tower of Hanoi game. These objects became draggable and responsive to user input.

This implementation provides an intuitive way for users to interact with the Tower of Hanoi game by selecting and moving objects using drag and drop functionality.

## 4 Rule Implementation

In the Tower of Hanoi game, the implementation of rules is crucial for ensuring a challenging and engaging experience. One of the key functions responsible for enforcing game rules is the checkCollisions function.

### 4.1 Collision Detection

The checkCollisions function is responsible for detecting collisions between disks during gameplay. It utilizes the bounding box technique to determine if any two disks intersect. The bounding box for each disk is updated based on its current position and orientation in the 3D space.

```

1 function checkCollisions() {
2   scene.disk1BB.copy(scene.disks[0].mesh.geometry.boundingBox).applyMatrix4(scene
3     .disks[0].mesh.matrixWorld);
4   scene.disk2BB.copy(scene.disks[1].mesh.geometry.boundingBox).applyMatrix4(scene
5     .disks[1].mesh.matrixWorld);
6   scene.disk3BB.copy(scene.disks[2].mesh.geometry.boundingBox).applyMatrix4(scene
7     .disks[2].mesh.matrixWorld);
8
9   // Check for collisions between Disk 1 and Disk 2
10  if (scene.disk1BB.intersectsBox(scene.disk2BB)) {
11    if (scene.disks[0].mesh.position.y > scene.disks[1].mesh.position.y) {
12      Invalid = true;
13    }
14  }
15
16  // Check for collisions between Disk 1 and Disk 3
17  if (scene.disk1BB.intersectsBox(scene.disk3BB)) {
18    if (scene.disks[0].mesh.position.y > scene.disks[2].mesh.position.y) {
19      Invalid = true;
20    }
21  }
22  // Check for collisions between Disk 2 and Disk 3
23  if (scene.disk2BB.intersectsBox(scene.disk3BB)) {
24    if (scene.disks[1].mesh.position.y > scene.disks[2].mesh.position.y) {
      Invalid = true;
    }
  }
}
```

25 }

Listing 10: Collision Detection Function

In the code snippet above, the function checks for collisions between each pair of disks (Disk 1 and Disk 2, Disk 1 and Disk 3, Disk 2 and Disk 3) and updates the game state accordingly.

This collision detection mechanism adds a layer of complexity to the Tower of Hanoi game, requiring players to adhere to the rules and make strategic moves to successfully complete the puzzle.

## 4.2 Drag-and-Drop Interaction

Enabling a seamless drag-and-drop interaction for the Tower of Hanoi game enhances the user experience and allows players to make strategic moves. The implementation involves utilizing event listeners and controls provided by the Three.js library.

```
1 scene.controls.addEventListener( 'drag', function(event){  
2     if (selected?.body.getCollisionFlags() === 2) {  
3         const { x, y } = pointer  
4         const speed = 0.03  
5         const movementX = (x - prev.x) * speed  
6         const movementZ = (y - prev.y) * -speed  
7         selected.position.x += movementX  
8         selected.position.z += movementZ  
9         selected.body.needsUpdate = true  
10        prev = { x, y }  
11    }  
12    selectedObject = event.object  
13    render();  
14 } );  
15  
16 document.addEventListener( 'pointermove', onPointerMove );  
17 document.addEventListener( 'pointerdown', onPointerDown );  
18  
19 scene.controls.addEventListener('dragstart', function (event) {  
20     // Disable orbit control during drag  
21     makingMove = true;  
22     orbitControls.enabled = false;  
23     selectedObject = event.object;  
24  
25     // Check for intersecting disks during drag start  
26     const intersects_disks = raycaster.intersectObjects( scene.disks_mashes, true )  
27     ;  
28     if ( intersects_disks.length > 0 ) {  
29         intersects_disks[0].object.material.color.set( 0xffb3b3 ); // Change color  
30         of selected object  
31         let disk_onTop = hasDisksOnTop(intersects_disks[0].object, scene.  
32             disks_mashes);  
33         console.log("disk on top variable: ",disk_onTop);  
34  
35         if(disk_onTop === false){  
36             selected = intersects_disks[0].object;  
37             console.log("selected: ",selected.body.getCollisionFlags());  
38             selected.body.setCollisionFlags(2); // Set collision flag for the  
39                 dragged disk  
40         }  
41     }  
42 };  
43  
44 scene.controls.addEventListener('dragend', function (event) {  
45     // Enable orbit control after drag ends  
46     makingMove = false;  
47     orbitControls.enabled = true;  
48     const selectedObject = event.object;  
49  
50     // Check if the dragged object is a disk  
51     if (selectedObject.body.getCollisionFlags() === 2){  
52         movesMade +=1; // Increment move count  
53     }  
54 }
```

```

50     selectedObject.material.color.setHex( CURRENTCOLOR );
51     selectedObject.body.setCollisionFlags(0); // Reset collision flag
52
53     // When the object is dropped, check if it is on the table
54     // If on the table, set the collision flag to 1 (static)
55     selectedObject.body.on.collision((otherObject, event) => {
56         if (otherObject.name === 'table') selectedObject.body.setCollisionFlags(1);
57     })
58
59     selected = null; // Reset selected object
60 });
61

```

Listing 11: Drag-and-Drop Interaction

In the provided code snippet, the drag-and-drop interaction is handled through event listeners and controls. The drag, dragstart, and dragend events are utilized to manage the state of the interaction, enable/disable orbit control, and update relevant parameters.

The drag-and-drop feature enhances player control and adds an interactive layer to the Tower of Hanoi game.

### 4.3 Rendering and Game State Update

The rendering process in the Tower of Hanoi game plays a crucial role in updating the game state, including the move count and game status. The `render` function is responsible for orchestrating these updates, ensuring a smooth and responsive gaming experience.

```

1 function render(event) {
2     // Check for collisions if not in the process of making a move
3     if (makingMove == false) {
4         checkCollisions();
5     }
6
7     // Update physics simulation
8     physics.update(clock.getDelta() * 1000);
9     physics.updateDebugger();
10
11    // Render the scene
12    scene.renderer.render(scene.scene, scene.camera);
13
14    // Update the game board with the latest move count and status
15    scene.gameBoard.updateMovesAndStatus(movesMade, Invalid);
16 }
17
18 // Initial update of the game board state
19 scene.gameBoard.updateMovesAndStatus(movesMade, Invalid);

```

Listing 12: Rendering and Game State Update

In the code snippet above, the `render` function is described as the central hub for updating the game state during the rendering process. It checks for collisions, updates the physics simulation, renders the scene, and ensures that the game board reflects the latest move count and status.

This integration of rendering and game state update contributes to a responsive and visually informative Tower of Hanoi gaming experience.