SAPIENZA
UNIVERSITÀ DI ROMA

DIAG
Dipartimento di Ingegneria
informatica, automatica e gestionale
Antonio Ruberti

3D Maze Mastery: An Interactive Adventure with Three.js

Course Title: Interactive Graphic

Professor: Paolo Russo

Session: 2023-24

28 June 2024

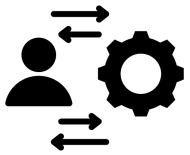
Syed Saad Hasan

Student ID: 2106512

Program: Artificial Intelligence and Robotics

Introduction:

Overview



Innovative Approach: The Maze Game is an interactive 3D maze navigation game developed using WebXR and Three.js. It enhances spatial awareness and problem-solving skills.

Purpose

Purpose:

Immerse players in a procedurally generated maze, where they navigate through complex pathways with realistic lighting and smooth controls, enhancing their spatial awareness and problem-solving skills.

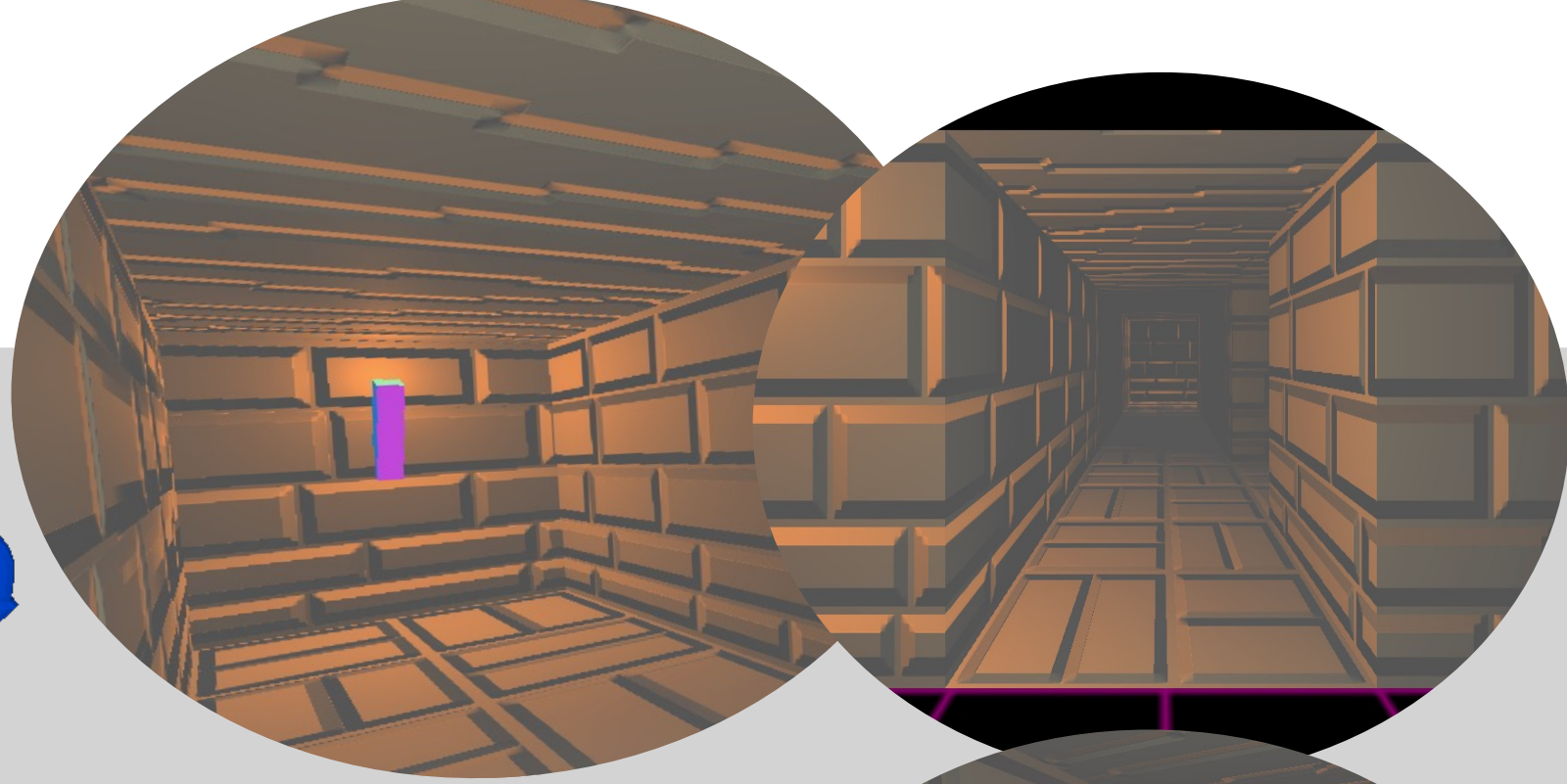
Technologies Used:

Three.js:
For overall code implementation

WebXR:
Immersive VR experiences in web applications.

WebGL via Three.js to implement different shading techniques (Flat, Gouraud, and Phong) to enhance visual realism.

Game Overview:



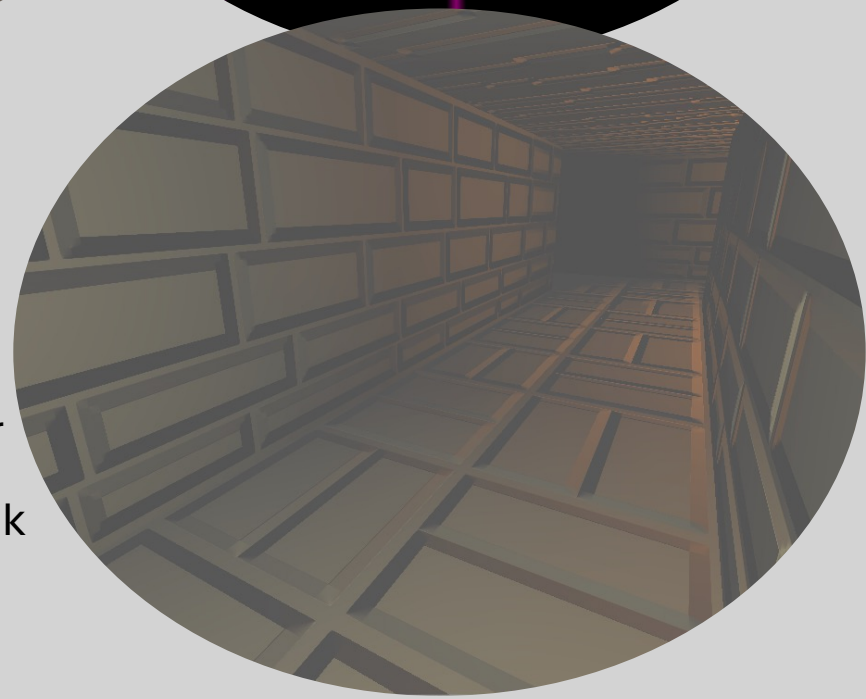
TextStudio

MAZE GENRARTION

- **Depth-First Search (DSF)** utilize for maze.
- **Code Implementation:** generateMaze, getUnvisitedNeighbors, removeWall

INTERACTIVE CONTROL

- **Mechanism of Control:** for queries and documents.
- **Keyborad Input Management:** InputManager
- **Mouse & Pointer Locker Control**: PointerLock
- **Vritual Reality Control:** VRontrols



Game Demonstration:

- Starting the Game
- Navigationg Through the Maze
- Interacting with Objects
- Overall Project Code Structure

Game Live Demo Link Demo

Rendering Technology:

Scene - Camera - Renderer

Introduction to Three.js:

- ✓ Generating 3D graphics
- ✓ Create Visual representation .
- ✓ Utilizes WebGL for render (*WebGLRenderer* class)

How Three.js Use Rendering 3D Graphics:

- ✓ Scene Management:
- ✓ Realistic Lighting.
- ✓ Texture Mapping.

Code Snippet: Setting up the Scene, Camera and Renderer:

- ✓ Scene, Camera, Renderer, Animation Loop, Object Addition

Scene Initialization:

```javascript
const scene = new THREE.Scene();
scene.background = new THREE.Color(0x000000); // Set background color
```

Camera Setup:

```javascript
// Camera Setup
const camera = new THREE.PerspectiveCamera(
    75,
    window.innerWidth / window.innerHeight,
    0.1,
    1000
);
camera.position.z = 5;
```

Renderer Initialization:

```javascript
const renderer = new THREE.WebGLRenderer();
renderer.setSize(window.innerWidth, window.innerHeight);
document.body.appendChild(renderer.domElement);
```

Animation Loop:

```javascript
function animate() {
    requestAnimationFrame(animate);
    renderer.render(scene, camera);
}
animate();
```

Shading Technique:

Type of Shading Used:

- ✓ Flat Shading
- ✓ Gouraud Shading
- ✓ Phong Shading

How Shading Enhance Visual realism

- ✓ Realistic Lighting
- ✓ Depth and Detail
- ✓ Immersion

Code Snippet: Implementastion Shader in three.js

Flat Shading

```javascript
const materialFlat = new THREE.MeshBasicMaterial({
    color: 0xff0000,
    flatShading: true
});
const cubeFlat = new THREE.Mesh(geometry, materialFlat);
scene.add(cubeFlat);
```

Gourand Shading

```javascript
const materialGouraud = new THREE.MeshLambertMaterial({
    color: 0x00ff00
});
const cubeGouraud = new THREE.Mesh(geometry, materialGouraud);
scene.add(cubeGouraud);
```

Phong Shading

```javascript
const materialPhong = new THREE.MeshPhongMaterial({
    color: 0x0000ff,
    shininess: 100
});
const cubePhong = new THREE.Mesh(geometry, materialPhong);
scene.add(cubePhong);
```

Animation:

Type of Animation Used:

keyframe Animation, Physics simulation

Example:

Animating Object in Game

Code Snippet:

Keyframe Animation Implementation

Use AnimationMixer

```javascript
const mixer = new THREE.AnimationMixer(movingBox);
const action = mixer.clipAction(clip);
action.play();
```

Object Initialization

```javascript
const boxGeometry = new THREE.BoxGeometry(1, 1, 1);
const boxMaterial = new THREE.MeshBasicMaterial({ color: 0x00ff00 });
const movingBox = new THREE.Mesh(boxGeometry, boxMaterial);
scene.add(movingBox);
```

Animation in rendering loop

```javascript
const clock = new THREE.Clock();

function animate() {
    requestAnimationFrame(animate);
    const delta = clock.getDelta();
    mixer.update(delta);
    renderer.render(scene, camera);
}
animate();
```

Keyframe track

```javascript
const positionKF = new THREE.VectorKeyframeTrack(
    '.position',
    [0, 1, 2], // Times
    [0, 0, 0, 0, 5, 0, 0, 10, 0] // Values for x, y, z positions
);
const duration = 3; // Duration of the animation
const clip = new THREE.AnimationClip('move', duration, [positionKF]);
```

Code Expla

- ✓ **Structure of the project:** Key f
- ✓ **Specific methods and classes**
  management, user input handlir

"The project structure consists

, sets

logic

es ga

loadin

javascript

assetmanager.js

```javascript
// Asset manager class
var AssetManager = function() {
  this.textures = {};
  this.textureLoader = new THREE.TextureLoader();
};

// Load texture method
AssetManager.prototype.loadTexture = function(name, path) {
  const texture = this.textureLoader.load(path, () => {
    console.log(`${name} texture loaded.`);
  });
  this.textures[name] = texture;
};

// Get texture method
AssetManager.prototype.getTexture = function(name) {
  return this.textures[name];
};

// Usage example
const assets = new AssetManager();
assets.loadTexture('brick', 'textures/brick.jpg');
const brickTexture = assets.getTexture('brick');
```

Mazegame.js

```javascript
// Maze generation function
function generateMaze(width, height) {
  const maze = new Array(height).fill().map(() => new Array(width).fill(0));
  const stack = [];
  let currentCell = { x: 0, y: 0 };
  maze[currentCell.y][currentCell.x] = 1;
  stack.push(currentCell);

  while (stack.length > 0) {
    const neighbors = getUnvisitedNeighbors(currentCell, maze);
    if (neighbors.length > 0) {
      const nextCell = neighbors[Math.floor(Math.random() * neighbors.length)]
      removeWall(currentCell, nextCell, maze);
      stack.push(nextCell);
      currentCell = nextCell;
      maze[currentCell.y][currentCell.x] = 1;
    } else {
      currentCell = stack.pop();
    }
  }
  return maze;
}

// Get unvisited neighbors
function getUnvisitedNeighbors(cell, maze) {
  const { x, y } = cell;
  const neighbors = [];
  if (x > 0 && maze[y][x - 1] === 0) neighbors.push({ x: x - 1, y });
  if (x < maze[0].length - 1 && maze[y][x + 1] === 0) neighbors.push({ x: x +
  if (y > 0 && maze[y - 1][x] === 0) neighbors.push({ x, y: y - 1 });
  if (y < maze.length - 1 && maze[y + 1][x] === 0) neighbors.push({ x, y: y +
  return neighbors;
}

// Remove wall between cells
function removeWall(cell1, cell2, maze) {
  const x = (cell1.x + cell2.x) / 2;
  const y = (cell1.y + cell2.y) / 2;
  maze[y][x] = 1;
}
```

ode Snip

MAZEGAME_3D

- js
  - assetmanager.js
  - inputmanager.js
  - main.js
  - mazegame.js
  - pointerlock.js
  - torch.js
  - utils.js
  - xrcontrols.js
- lib
  - mazegen.js
  - three.min.js
  - THREEx.FullScreen.js
  - tools.js

Main.js

```
nd renderer

veCamera(
nerHeight, 0.1, 1000

derer();
, window.innerHeight);
.domElement);
```

Project Requirements:

How the game integrates Rendering, shading and Animation

Integration?

- ✓ **Rendering**
- ✓ **Shading**
- ✓ **Animation**

Keyframe animation
for objects & physics-based
animations

Overview of Requirements

Integrates rendering, shading, and animation technologies discussed in class, using either plain WebGL or Three.js. If using Three.js, the project must be more complex, such as a video game or a complex simulation, to achieve the same grade as a WebGL project due to Three.js's higher-level abstractions.

three.js

WebGL™

Q&A:

Feel Free to ask the Question
Future Recommandations

Special thanks to the
Professor Paolo Russa