Student's name: Piyaporn Puangprasert (pp712)

Course: Introduction to Computer Graphics (CS523)

Instructor: Prof. Mridul Aanjaneya

Date: December 10, 2025

Titel: 3D Graduation Journey: A WenGL Game for Visualizing Academic Progress

1. Project Goal

The goal of this project is to help students design and develop a lightweight 3D WebGL-based game that allows students to visualize and track their academic journey toward graduation. This game will present each completed course as a glowing badge, and together these badges form a staircase that represents progress toward degree completion. The player's objective is to collect badges (classes) and ascend to the top of the starcase. The last level is graduation.

Based on my own experience. I am a full-time worker at Rutgers (Housekeeper) who just completed a business degree and am interested in a computer science degree. Last semester was my first semester, and I failed the Algorithms CS512. I printed out all my choices I could enroll in next semester, but out of luck. Half of my classes were a prerequisite with CS512. Half of them were full because there is a limited number of students. This Challenge also includes:

- No more than 10 semesters taken (Yes, I did).
- No more than 2 C grades are allowed (Can not make it yet)
- At least 12 Classes (36 credits) (Yes, I did)
- Only classes that open before 4 PM, Monday to Friday, that I can take (Can not make it yet)
- 2 Theses or 2 non-thesis papers (Yes, I did)
-

2. Description, Object, and Purpose

This project combines two ideas: the 3D badge collector and the 3D Graduation Path Visualizer in a single WebGL game. Students can visualize their completed and pending courses in the interactive 3D space while learning the principles of computer graphics.

- Goal: Collect 36 credits (12 Courses)

- Rules: Student must satisfy the General Degree constraints (2 Essay or Thesis + 2 Category A + 2 Category B +2 Category A or B + 4 (others)

How to play:

1. Save the code: copy the code block below and save it as an .html file
2. Open: run it in Chrome, Firefox, Edge
3. Controls:
    a. Arrow Keys/WASD: move student avatar
    b. Mouse: Click “Start” to begin

3. References and Theoretical Foundation

There are 2 academic studies about the potential of WebGL for 3D rendering and gaming:

1. Ru Miao et al.(2025). “3D Geographic Scenes Visualization Based on WebGL.” IEEE Xplore.
    a. This paper introduces WebGL’s glTF rendering pipeline and discusses occlusion culling and scene prefetching for efficient 3D rendering.
    b. Its framework for 3D visualization in web browsers inspired the optimization and structure of this project’s rendering engine.
2. Dedi Prayudi and Nurul Hamdi (2023). “Development of a Video Games With WebGL Format That Allows You to Play Video Games Without Need for a Device with High Specifications” Brilliance Journal of AI Research.
    a. This study demonstrates how WebGL can deliver high-performance 3D games accessible to users with low-end devices.
    b. It supports this project’s goal to ensure accessibility and smooth performance for all students.

4. Associated Challenges

This project involves the following technical and design challenges:

- In the real situation, I ask the TA about “ Four from the list of courses acceptable for MSCS credit: approved courses from other departments, additional Category A or B courses, CS Seminars, acceptable undergraduate courses, OR independent study. Note 1: At most two of the four courses may be independent study. Note 2: Refer to the SGS and Global Policies below for online courses. Note 3: non-MSCS course registration requires office (email) approval based on extraordinary circumstances.” requirement. He said that if you take 12 courses

from our department, you can graduate. Then, I just design the courses when students take 2(Essay or Thesis) + 2(Category A) + 2(Category B) + 2(Category A or B) = 8, and anything from our other 4 courses, so students will graduate (for now). I will update the code in the future.

5. Conclusion

This project will combine education visualization with computer graphics game development. By leveraging WebGL, the 3D Graduation Journey game can benefit students to experience their academic progress in a game-like environment. Supported by research on WebGL rendering and low-spec hardware compatibility, this project aims to make learning both visual and engaging.

Output:

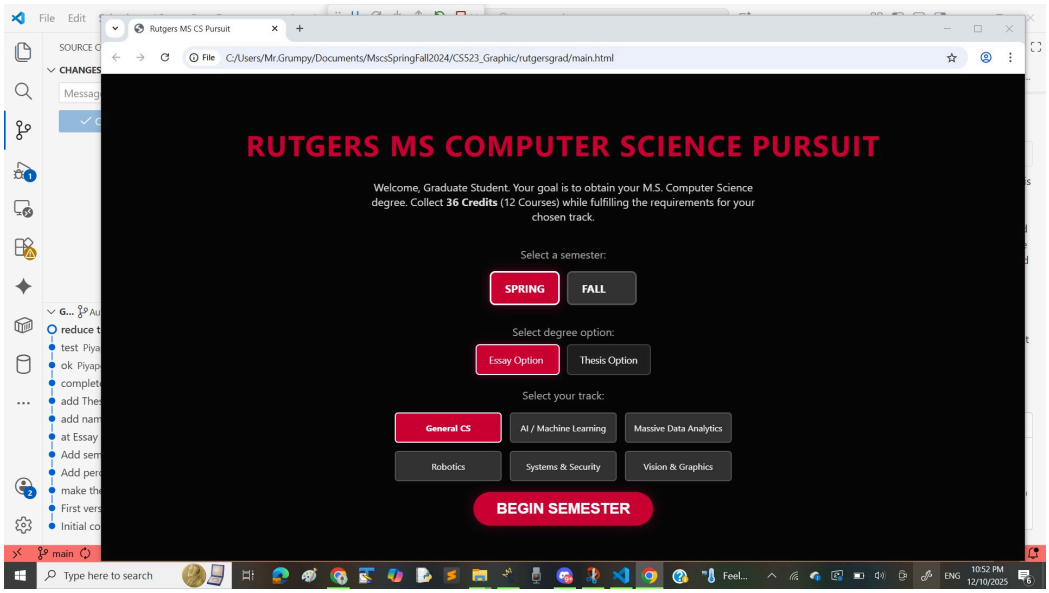

Figure 1: Student chooses Semester, Essay or Thesis, and Major

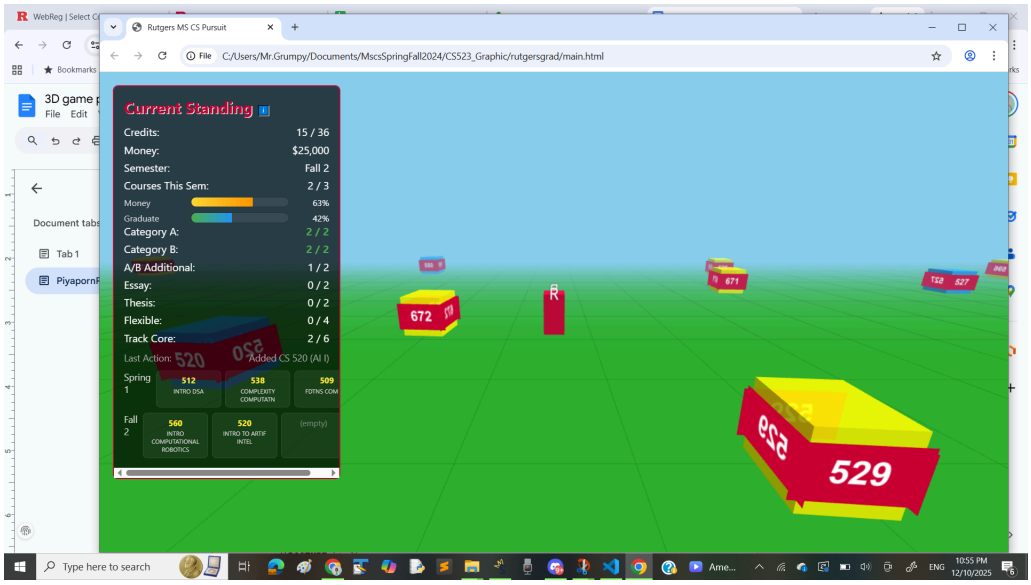

Figure 2: The Student in the red box moves with the arrow front, back, forward, and backward.

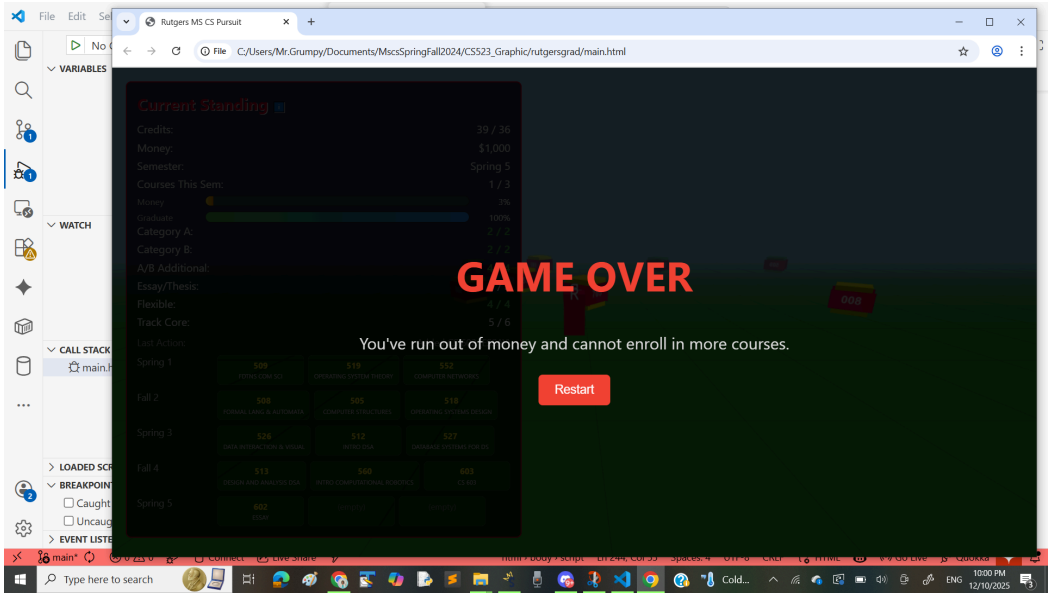

Figure 3: Game over if the student runs out of money

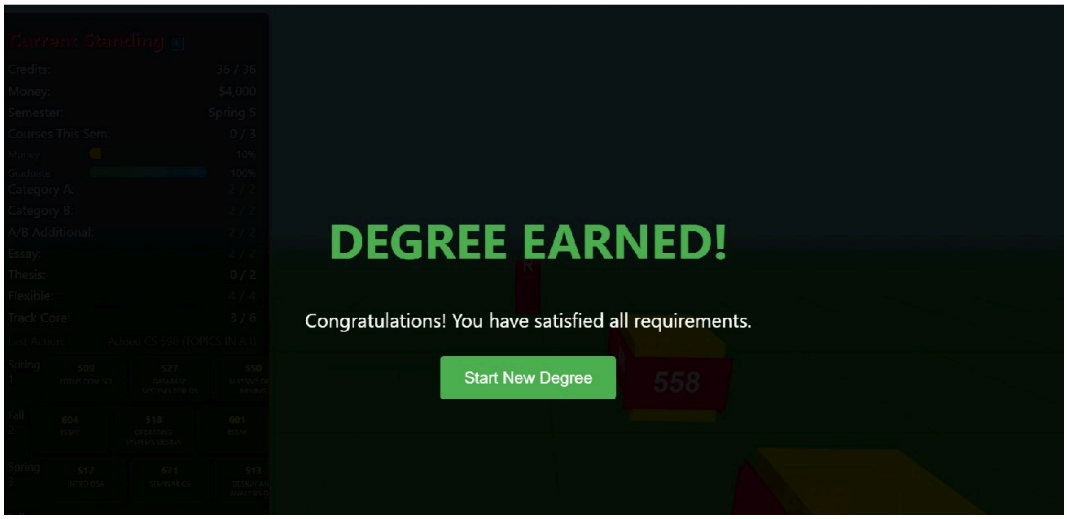

Figure 4: The Student wins the game

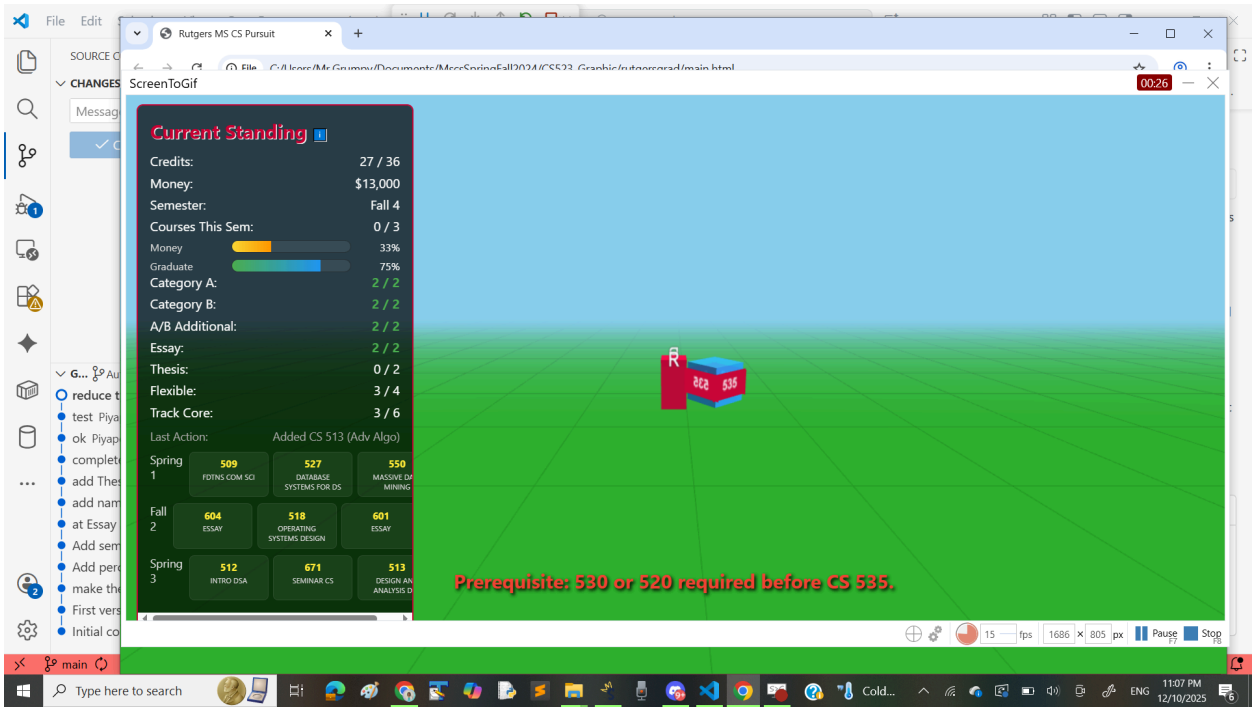

Figure 6: When some courses show prerequisites

Github: https://github.com/nanpiyaporn/rutgersgrad

Video folder: https://drive.google.com/drive/folders/1NjziBcPBOQhfJF9L3zLTJEHqtEOVphu2?usp=sharing

Video link 1:
https://drive.google.com/file/d/1aVyNO_AvZ2h5-sbpQuK00LdzKLfMALWk/view?usp=drive_link

Video link 2:
https://drive.google.com/file/d/1YCjiUWnJZqC8p8ft6yI60SC4GOB8Q-Jx/view?usp=drive_link

Code:

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Rutgers MS CS Pursuit</title>

    <style>

        body { margin: 0; overflow: hidden; font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif; background-color: #202020; color: white; }

        #ui-layer { position: absolute; top: 0; left: 0; width: 100%; height: 100%; pointer-events: none; }

        #hud { position: absolute; top: 20px; left: 20px; font-size: 16px; background: rgba(0,0,0,0.7); padding: 15px; border-radius: 8px; border: 2px solid #cc0033; display: none; width: 25vw; max-width: 420px; overflow: auto; box-sizing: border-box; }

        #hud h2 { margin: 0 0 10px 0; color: #cc0033; text-shadow: 1px 1px 0 #fff; }

        .stat-row { display: flex; justify-content: space-between; min-width: 0; margin-bottom: 5px; }

        .valid { color: #4caf50; font-weight: bold; }

        .invalid { color: #f44336; }
```

```
        #start-screen { position: absolute; top: 0; left: 0; width: 100%;
height: 100%; background: rgba(10,10,10,0.95); display: flex;
flex-direction: column; align-items: center; justify-content: flex-start;
padding-top: 48px; pointer-events: auto; z-index: 10; }

        #start-screen h1 { color: #cc0033; font-size: 42px;
text-transform: uppercase; letter-spacing: 2px; margin-bottom: 8px; }

        #start-screen p { font-size: 16px; max-width: 600px; text-align:
center; color: #ddd; margin-bottom: 18px; }


        .track-grid { display: grid; grid-template-columns: repeat(3,
1fr); gap: 12px; margin-bottom: 16px; }

        .track-btn { background: #333; border: 2px solid #555; color:
white; padding: 12px; cursor: pointer; border-radius: 5px; font-size:
13px; transition: all 0.15s; text-align: center; }

        .track-btn:hover { border-color: #cc0033; background: #404040;
transform: translateY(-2px); }

        .track-btn.selected { background: #cc0033; border-color: #fff;
font-weight: bold; }


        .semester-grid { display: grid; grid-template-columns: repeat(2,
1fr); gap: 10px; margin-bottom: 12px; }

        .semester-btn { background: #333; border: 3px solid #555; color:
white; padding: 12px 20px; cursor: pointer; border-radius: 8px; font-size:
16px; font-weight: bold; transition: all 0.2s; text-transform: uppercase;
}

        .semester-btn:hover { border-color: #cc0033; background: #404040;
transform: translateY(-2px); }

        .semester-btn.selected { background: #cc0033; border-color: #fff;
box-shadow: 0 0 12px rgba(204,0,51,0.55); }


        /* Degree option buttons (Essay / Thesis) */
```

```
        .option-grid { display: grid; grid-template-columns: repeat(2, 1fr); gap: 10px; margin-bottom: 12px; }

        .option-btn { background: #222; border: 2px solid #444; color: white; padding: 12px 18px; cursor: pointer; border-radius: 6px; font-size: 14px; transition: all 0.2s; text-transform: none; }

        .option-btn:hover { border-color: #cc0033; background: #333; transform: translateY(-2px); }

        .option-btn.selected { background: #cc0033; border-color: #fff; box-shadow: 0 0 10px rgba(204,0,51,0.45); }


        #start-btn { background: #cc0033; color: white; border: none; padding: 12px 34px; font-size: 22px; cursor: pointer; border-radius: 40px; font-weight: bold; text-transform: uppercase; box-shadow: 0 0 14px rgba(204,0,51,0.35); margin-bottom: 18px; }

        #start-btn:hover { background: #e6003a; transform: scale(1.04); }


        #message-area { position: absolute; bottom: 100px; width: 100%; text-align: center; font-size: 24px; font-weight: bold; text-shadow: 2px 2px 4px #000; pointer-events: none; transition: opacity 0.5s; opacity: 0; }


        #win-screen { display: none; position: absolute; top: 0; left: 0; width: 100%; height: 100%; background: rgba(0,0,0,0.9); flex-direction: column; align-items: center; justify-content: center; z-index: 20; pointer-events: auto; }

        #win-screen h1 { color: #4caf50; font-size: 60px; margin-bottom: 20px; }

        #win-screen p { font-size: 24px; margin-bottom: 30px; }

        #restart-btn { background: #4caf50; color: white; border: none; padding: 15px 30px; font-size: 20px; cursor: pointer; border-radius: 5px; }
```

```
        /* Rutgers Red: #cc0033 */

        /* Progress bar styles */

        .progress-row { display:flex; align-items:center; gap:10px;
margin-top:6px; }

        .progress-label { width:90px; font-size:13px; color:#ddd; }

        .progress-bar { flex:1; height:14px;
background:rgba(255,255,255,0.08); border-radius:8px; overflow:hidden;
border:1px solid rgba(0,0,0,0.3); }

        .progress-fill { height:100%; width:0%;
background:linear-gradient(90deg,#4caf50,#2196f3); transition:width 300ms
ease; }

        .money-fill { background: linear-gradient(90deg,#fdd835,#ff9800);
}

        .percent-text { width:50px; text-align:right; font-size:13px;
color:#fff; }

        /* Semester tracker (per-semester boxes) */

        .sem-tracker { margin-top:10px; display:flex;
flex-direction:column; gap:8px; }

        .sem-row { display:flex; align-items:flex-start; gap:8px; }

        .sem-title { width:80px; color:#ddd; font-size:14px; }

        .course-box { background: rgba(255,255,255,0.06); border:1px solid
rgba(255,255,255,0.08); padding:6px 8px; border-radius:6px;
min-width:80px; text-align:center; font-size:12px; color:#fff; }

        .course-id { font-weight:bold; color:#ffeb3b; display:block; }

        .course-name { font-size:9px; color:#ddd; display:block;
margin-top:2px; }

    </style>

</head>

<body>
```

```
    <div id="ui-layer">
        <div id="start-screen">
            <h1>Rutgers MS Computer Science Pursuit</h1>
            <p>Welcome, Graduate Student. Your goal is to obtain your M.S. Computer Science degree. Collect <strong>36 Credits</strong> (12 Courses) while fulfilling the requirements for your chosen track.</p>

            <p style="margin-bottom: 15px; color: #aaa; font-size: 16px;">Select a semester:</p>
            <div class="semester-grid">
                <div class="semester-btn selected" data-semester="spring">Spring</div>
                <div class="semester-btn" data-semester="fall">Fall</div>
            </div>

            <p style="margin-bottom: 8px; color: #aaa; font-size: 16px;">Select degree option:</p>
            <div class="option-grid">
                <div class="option-btn selected" data-option="essay">Essay Option</div>
                <div class="option-btn" data-option="thesis">Thesis Option</div>
            </div>

            <p style="margin-top: 6px; margin-bottom: 15px; color: #aaa; font-size: 16px;">Select your track:</p>
            <div class="track-grid">
                <div class="track-btn selected" data-track="general">General CS</div>
```

```
                <div class="track-btn" data-track="aiml">AI / Machine
Learning</div>
                <div class="track-btn" data-track="data">Massive Data
Analytics</div>
                <div class="track-btn"
data-track="robotics">Robotics</div>
                <div class="track-btn" data-track="syssec">Systems &
Security</div>
                <div class="track-btn" data-track="vision">Vision &
Graphics</div>
            </div>

            <button id="start-btn">Begin Semester</button>
        </div>

        <div id="hud">
            <h2>Current Standing <span style="font-size:14px;
cursor:help;" title="Collect courses to earn credits and complete your
degree requirements."> i </span></h2>
            <div class="stat-row"><span>Credits:</span> <span
id="credit-count">0 / 36</span></div>
            <div class="stat-row"><span>Money:</span> <span
id="money-count">$40,000</span></div>
            <div class="stat-row"><span>Semester:</span> <span
id="semester-display">-</span></div>
            <div class="stat-row"><span>Courses This Sem:</span> <span
id="picks-count">0 / 3</span></div>
            <div class="progress-row">
                <div class="progress-label">Money</div>
```

```
                <div class="progress-bar"><div id="money-bar-fill"
class="progress-fill money-fill"></div></div>

                <div id="money-percent-text"
class="percent-text">100%</div>

            </div>

            <div class="progress-row">

                <div class="progress-label">Graduate</div>

                <div class="progress-bar"><div id="grad-bar-fill"
class="progress-fill"></div></div>

                <div id="grad-percent-text" class="percent-text">0%</div>

            </div>

            <div class="stat-row"><span>Category A:</span> <span
id="cat-a-count">0 / 2</span></div>

            <div class="stat-row"><span>Category B:</span> <span
id="cat-b-count">0 / 2</span></div>

            <div class="stat-row"><span>A/B Additional:</span> <span
id="cat-ab-add-count">0 / 4</span></div>

            <div class="stat-row"><span>Essay:</span> <span
id="essay-count">0 / 2</span></div>

            <div class="stat-row"><span>Thesis:</span> <span
id="thesis-count">0 / 2</span></div>

            <div class="stat-row"><span>Flexible:</span> <span
id="flex-count">0 / 4</span></div>

            <div class="stat-row"><span>Track Core:</span> <span
id="track-core-count">0 / 6</span></div>

            <div class="stat-row" style="margin-top:10px; font-size:14px;
color:#aaa;">Last Action: <span id="last-action">-</span></div>

            <div id="semester-tracker" class="sem-tracker">

                <!-- Filled dynamically with per-semester rows -->

            </div>
```

```
        </div>

        <div id="message-area">Course Collected!</div>

        <div id="win-screen">
            <h1>DEGREE EARNED!</h1>
            <p>Congratulations! You have satisfied all requirements.</p>
            <button id="restart-btn" onclick="location.reload()">Start New Degree</button>
        </div>

        <div id="game-over-screen" style="display:none; position: absolute; top: 0; left: 0; width:100%; height:100%; background: rgba(0,0,0,0.85); z-index:30; flex-direction: column; align-items: center; justify-content: center; pointer-events: auto;">
            <h1 style="color:#f44336; font-size:60px; margin-bottom:20px;">GAME OVER</h1>
            <p style="font-size:24px; color:#ddd; margin-bottom:30px;">You've run out of money and cannot enroll in more courses.</p>
            <button onclick="location.reload()" style="background:#f44336; color:white; border:none; padding:12px 24px; font-size:18px; border-radius:6px;">Restart</button>
        </div>
    </div>

    <script src="https://cdnjs.cloudflare.com/ajax/libs/three.js/r128/three.min.js"></script>
```

```
    <script>
        // --- DATA & CONFIG ---
        const TRACKS = {
            "general": {
                name: "General CS",
                core: ["501", "512", "513", "507", "518", "519", "515",
"520", "530", "527", "539"] // Flexible buckets
            },
            "aiml": {
                name: "AI / Machine Learning",
                core: ["581", "501", "512", "513", "543", "550", "520",
"530", "535", "536"]
            },
            "data": {
                name: "Massive Data Analytics",
                core: ["501", "512", "513", "527", "539", "543", "550",
"526"]
            },
            "robotics": {
                name: "Robotics",
                core: ["501", "560", "512", "513", "520", "530", "534"]
            },
            "syssec": {
                name: "Systems & Security",
                core: ["507", "518", "512", "513", "519", "552", "520",
"530"]
            },
```

```
            "vision": {
                name: "Vision & Graphics",
                core: ["581", "501", "512", "513", "518", "527", "539",
"520", "530", "526"]
            }
        };

        // Course Database (Simplified for Game)
        // Cat A: Math/Algo, Cat B: Systems/Apps
        const COURSES = [
            // Category A
            { id: "501", name: "Math Fnd", cat: "A" },
            { id: "512", name: "Algo", cat: "A" },
            { id: "513", name: "Adv Algo", cat: "A" },
            { id: "581", name: "Stats", cat: "A" },
            { id: "509", name: "Complexity", cat: "A" },
            // Category B
            { id: "507", name: "Comp Arch", cat: "B" },
            { id: "518", name: "OS Design", cat: "B" },
            { id: "519", name: "OS Theory", cat: "B" },
            { id: "515", name: "Prog Lang", cat: "B" },
            { id: "520", name: "AI I", cat: "B" },
            { id: "530", name: "AI II", cat: "B" },
            { id: "527", name: "DB Sys", cat: "B" },
            { id: "539", name: "DB Theory", cat: "B" },
            { id: "535", name: "Pattern Rec", cat: "B" },
```

```
            { id: "536", name: "Mach Learn", cat: "B" },
            { id: "560", name: "Robotics", cat: "B" },
            { id: "552", name: "Networks", cat: "B" },
            { id: "543", name: "MDS", cat: "B" },
            { id: "550", name: "MDM", cat: "B" },
            { id: "526", name: "DIVA", cat: "B" },
            { id: "534", name: "Vision", cat: "B" },
            // Electives (Generic)
            { id: "601", name: "Ind. Study", cat: "E" },
            { id: "671", name: "Seminar", cat: "E" },
            { id: "XXX", name: "Approved Other Dept Course", cat: "E" }
        ];

        // Semester availability lists (IDs) " I took all 800-level courses out!"
         const SPRING_COURSES = ["509","512","513","514","519","520","523","526","527","530","533","534","535","536","538","545","550","552","553","558","562","580","590","602","605","660","670","671","684","699","702","705"];
         const FALL_COURSES = ["505","508","512","513","515","518","520","523","525","527","529","530","535","550","560","580","596","598","599","601","604","660","664","671","672","701","704"];

         // Course name mapping
         const COURSE_NAMES = {
             "501": "MATH FOUND OF DS",
             "505": "COMPUTER STRUCTURES",
```

```
"508": "FORMAL LANG & AUTOMATA",
"509": "FDTNS COM SCI",
"512": "INTRO DSA",
"513": "DESIGN AND ANALYSIS DSA",
"514": "DSGN & ANL DS&ALGOR II",
"515": "PROG LANG AND COMPILERS I",
"518": "OPERATING SYSTEMS DESIGN",
"519": "OPERATING SYSTEM THEORY",
"520": "INTRO TO ARTIF INTEL",
"523": "COMPUTER GRAPHICS",
"525": "BRAIN INSPIRED COMPUTING",
"526": "DATA INTERACTION & VISUAL",
"527": "DATABASE SYSTEMS FOR DS",
"529": "COMPUTATN AL GEOMETRY",
"530": "PRINCIPLES OF AI",
"533": "NATURAL LANGUAGE",
"534": "COMPUTER VISION",
"535": "MACHINE LEARNING I",
"536": "MACHINE LEARNING II",
"538": "COMPLEXITY COMPUTATN",
"545": "DISTRIBUTED SYSTEMS",
"550": "MASSIVE DATA MINING",
"552": "COMPUTER NETWORKS",
"553": "DESIGN OF INTERNET SERVICES",
"558": "QUANT COMP PRO/SYST",
"560": "INTRO COMPUTATIONAL ROBOTICS",
```

```
"562": "ADVANCED ROBOTICS",

"580": "TOPICS IN COM IN BIO",

"590": "SOCIALLY COGNIZANT ROBOTICS",

"596": "TOPICS IN CS",

"598": "TOPICS IN A I",

"599": "DESIGN SOC COG ROBOT",

"601": "ESSAY",

"602": "ESSAY",

//"603": "BUSINESS DATA MANAGEMENT",

"604": "ESSAY",

"605": "ESSAY",

//"609": "INFO TECH FOR MGRS", 22:198:609

//"643": "INFORMATION SECURITY",

//"644": "DATA MINING",

//"660": "BUSINESS ANALYTICS PROGRAMMING",

//"664": "BUSINESS APPLICATION IN MACHINE LEARNING",

"670": "INFORMATION TECH STRATEGY",

"671": "SEMINAR CS",

"672": "SEMINAR CS",

//"684": "SPECIAL TOPICS IS", 26:198:684

"699": "NON-THESIS STUDY",

"XXX": "APPROVED OTHER DEPARTMENT COURSE",

"701": "THESIS",

"702": "THESIS",

"704": "THESIS",

"705": "THESIS",
```

```
            //"800": "MATRICULATION CONTINUED",
            //"811": "GRADUATE FELLOWSHIP",
            //"844": "RESEARCH INTERNSHIP",
            //"855": "GRADUATE TRAINEESHIP",
            //"866": "GRADUATE ASSISTANTSHIP",
            //"867": "PART GA APPOINTMENT",
            //"877": "TEACHING ASSISTANTSHIP",
            //"878": "PART TA APPOINTMENT"
        };


        // Prerequisite mapping. Each key maps to an array of option-sets.
        // Each option-set is an array of course IDs that must ALL be taken (AND).
        // Multiple option-sets represent OR between them.
        const PREREQS = {
            // 512 before 513
            "513": [["512"]],
            // 513 before 514
            "514": [["513"]],
            // 519 before 545
            "545": [["519"]],
            // 534,535,536 require 530 OR 520
            "534": [["530"],["520"]],
            "535": [["530"],["520"]],
            "536": [["530"],["520"]]
        };
```

```
        function hasPrereqs(courseId) {
            const req = PREREQS[courseId];
            if (!req) return true; // no prereqs
            // req is array of option-sets. Return true if any option-set
fully satisfied
            for (let optionSet of req) {
                let ok = true;
                for (let need of optionSet) {
                    if (!state.collected.includes(need)) { ok = false;
break; }
                }
                if (ok) return true;
            }
            return false;
        }

        function prereqMessage(courseId) {
            const req = PREREQS[courseId];
            if (!req) return '';
            // If single optionSet with single element -> simple message
            if (req.length === 1 && req[0].length === 1) {
                return `You must take CS ${req[0][0]} before CS
${courseId}.`;
            }
            // Otherwise build OR message
            const orParts = req.map(set => set.join(' & '));
```

```
            return `Prerequisite: ${orParts.join(' or ')} required before
CS ${courseId}.`;
        }

        // Course category helpers
        const ESSAY_COURSES = ["601", "602", "604", "605"];
        // Include thesis course IDs that should be restricted when Essay
option is chosen
        const THESIS_COURSES = ["701", "702", "704", "705"];
        // Independent study courses that count toward the 4 flexible
MSCS-acceptable courses
        const INDEPENDENT_STUDY = ["699"];
        const CAT_A_COURSES = ["501", "509", "512", "513", "514", "581"];
        const CAT_B_COURSES = ["507", "515", "518", "519", "520", "523",
"525", "526", "527", "529", "530", "534", "535", "536", "538", "545",
"550", "552", "553", "558", "560", "562"];

        function getCourseCategory(courseId) {
            if (ESSAY_COURSES.includes(courseId)) return 'essay';
            if (THESIS_COURSES.includes(courseId)) return 'thesis';
            if (CAT_A_COURSES.includes(courseId)) return 'A';
            if (CAT_B_COURSES.includes(courseId)) return 'B';
            return 'flex'; // flexible/other
        }

        let selectedTrack = "general";
        let selectedSemester = "spring";
        let selectedDegreeOption = "essay"; // 'essay' or 'thesis'
```

```
        const STARTING_MONEY = 40000;

        let state = {

            credits: 0,

            money: STARTING_MONEY,

            catA: 0,

            catB: 0,

            trackCore: 0,

            collected: [],

            semesterCourses: [], // array of arrays: courses picked per
semester

            catACore: 0, // min 2

            catBCore: 0, // min 2

            // Additional A/B courses: total and how many are strict A/B
(not essay/thesis)

            catABAdditional: 0, // total additional A/B (track how many
A/B beyond cores)

            catABAdditionalStrict: 0, // of the above, how many are strict
A/B courses (need >=2)

            // Track how many essay/thesis courses have been used to
satisfy the additional A/B slots (max 2)

            essayThesisUsedForAB: 0,

            essayCount: 0, // essay courses picked

            thesisCount: 0, // thesis courses picked

            flexibleCourses: 0, // other courses acceptable for MSCS
credit (need 4)

            independentStudyCount: 0, // at most 2 of the 4 flexible may
be independent study

            essayThesisCompleted: false // whether essay or thesis
requirement met
```

```
        };

        // Game constants
        const COURSE_COST = 3000;
        state.gameOver = false;

        // --- THREE.JS SETUP ---
        const scene = new THREE.Scene();
        scene.background = new THREE.Color(0x87CEEB); // Sky blue
        scene.fog = new THREE.Fog(0x87CEEB, 10, 50);

        const camera = new THREE.PerspectiveCamera(75, window.innerWidth /
window.innerHeight, 0.1, 1000);
        camera.position.set(0, 5, 10);
        camera.lookAt(0, 0, 0);

        const renderer = new THREE.WebGLRenderer({ antialias: true });
        renderer.setSize(window.innerWidth, window.innerHeight);
        renderer.shadowMap.enabled = true;
        document.body.appendChild(renderer.domElement);

        // Lighting
        const ambientLight = new THREE.AmbientLight(0xffffff, 0.6);
        scene.add(ambientLight);
        const dirLight = new THREE.DirectionalLight(0xffffff, 0.8);
        dirLight.position.set(10, 20, 10);
```

```
        dirLight.castShadow = true;
        scene.add(dirLight);

        // Ground (The Campus)
        const groundGeo = new THREE.PlaneGeometry(200, 200);
        const groundMat = new THREE.MeshStandardMaterial({ color: 0x228b22
});
        const ground = new THREE.Mesh(groundGeo, groundMat);
        ground.rotation.x = -Math.PI / 2;
        ground.receiveShadow = true;
        scene.add(ground);

        // Grid Helper
        const grid = new THREE.GridHelper(200, 50, 0x000000, 0x000000);
        grid.material.opacity = 0.2;
        grid.material.transparent = true;
        scene.add(grid);

        // Player (The Student)
        const playerGeo = new THREE.BoxGeometry(1, 2, 1);
        const playerMat = new THREE.MeshStandardMaterial({ color: 0xcc0033
}); // Rutgers Red
        const player = new THREE.Mesh(playerGeo, playerMat);
        player.position.y = 1;
        player.castShadow = true;
        scene.add(player);
```

```
        // Add an 'R' logo on the player's chest using a canvas texture
        (function addPlayerR() {
            const rCanvas = document.createElement('canvas');
            rCanvas.width = 256; rCanvas.height = 256;
            const rCtx = rCanvas.getContext('2d');
            rCtx.clearRect(0, 0, rCanvas.width, rCanvas.height);

            // Draw white 'R' on transparent background so cube red shows
behind it
            rCtx.fillStyle = 'white';
            // Big bold font for visibility
            rCtx.font = 'bold 180px Arial';
            rCtx.textAlign = 'center';
            rCtx.textBaseline = 'middle';
            rCtx.fillText('R', rCanvas.width / 2, rCanvas.height / 2 + 6);

            const rTex = new THREE.CanvasTexture(rCanvas);
            const rMat = new THREE.MeshBasicMaterial({ map: rTex,
transparent: true, side: THREE.DoubleSide });
            const rGeo = new THREE.PlaneGeometry(0.9, 1.2);

            // Front chest label
            const rLabel = new THREE.Mesh(rGeo, rMat);
            rLabel.position.set(0, 1, 0.51);
            rLabel.renderOrder = 999;
            player.add(rLabel);
```

```
            // Back label so 'R' shows from behind as well
            const rLabelBack = rLabel.clone();
            rLabelBack.rotation.y = Math.PI;
            rLabelBack.position.set(0, 1, -0.51);
            player.add(rLabelBack);
        })();

        // Collectibles Group
        const collectibles = [];

        function spawnCourse(x, z) {
            // Pick a course appropriate for the current semester when
available
            let courseData;
            // Small chance to spawn an 'Approved Other Dept' box (XXX)
            if (Math.random() < 0.08) {
                courseData = { id: 'XXX', name: COURSE_NAMES['XXX'] ||
'Approved Other Dept Course', cat: 'E' };
            }
            try {
                const allowed = (state.currentSemester === 'spring') ?
SPRING_COURSES : (state.currentSemester === 'fall') ? FALL_COURSES : null;
                let pool = allowed && allowed.length > 0 ? allowed.slice()
: null;
                // Filter out courses not allowed by selected degree
option
                if (pool) {
                    pool = pool.filter(id => {
```

```
                        if (selectedDegreeOption === 'thesis' &&
ESSAY_COURSES.includes(id)) return false;
                        if (selectedDegreeOption === 'essay' &&
THESIS_COURSES.includes(id)) return false;
                        return true;
                    });
                    // fallback to original allowed list if filtering
removes everything
                    if (pool.length === 0) pool = allowed.slice();
                }

                if (pool && pool.length > 0) {
                    const id = pool[Math.floor(Math.random() *
pool.length)];
                    const fullName = COURSE_NAMES[id] || `CS ${id}`;
                    courseData = COURSES.find(c => c.id === id) || { id:
id, name: fullName, cat: 'E' };
                } else {
                    // fallback: random from all courses but still respect
degree option
                    const globalPool = COURSES.map(c => c.id).filter(id =>
{
                        if (selectedDegreeOption === 'thesis' &&
ESSAY_COURSES.includes(id)) return false;
                        if (selectedDegreeOption === 'essay' &&
THESIS_COURSES.includes(id)) return false;
                        return true;
                    });
```

```
                    const randId = globalPool.length ?
globalPool[Math.floor(Math.random() * globalPool.length)] :
COURSES[Math.floor(Math.random() * COURSES.length)].id;

                    const fullName = COURSE_NAMES[randId] || `CS
${randId}`;

                    courseData = COURSES.find(c => c.id === randId) || {
id: randId, name: fullName, cat: 'E' };

                }

            } catch (e) {

                courseData = COURSES[Math.floor(Math.random() *
COURSES.length)];

            }


            // Geometry

            const geo = new THREE.BoxGeometry(1.5, 1.5, 1.5);


            // Material - Color code by Category

            let color = 0xffff00; // Elective/Other (Yellow)

            if (courseData.cat === 'A') color = 0x4caf50; // Green

            if (courseData.cat === 'B') color = 0x2196f3; // Blue

            // Special color for Approved Other Dept box

            if (courseData.id === 'XXX') color = 0x9c27b0; // Purple


            // Check if it's a core requirement for current track (Make it
glow/distinct)

            const isCore =
TRACKS[selectedTrack].core.includes(courseData.id);

            if (isCore) {
```

```
                // Gold tint if it's a core requirement we need
            }

            const mat = new THREE.MeshStandardMaterial({ color: color,
opacity: 0.9, transparent: true });
            const mesh = new THREE.Mesh(geo, mat);

            mesh.position.set(x, 1.5, z);
            mesh.userData = { ...courseData, isCore: isCore };

            // Add floating animation data
            mesh.userData.floatOffset = Math.random() * Math.PI * 2;

            // Simple text label (using canvas texture for course name)
            const canvas = document.createElement('canvas');
            // Larger canvas for crisper, bigger text
            canvas.width = 512; canvas.height = 256;
            const ctx = canvas.getContext('2d');
            // Rutgers red background with white text for strong contrast
            ctx.fillStyle = '#cc0033';
            ctx.fillRect(0,0,canvas.width,canvas.height);
            ctx.fillStyle = 'white';
            // Bigger font so the course number is prominent
            ctx.font = 'bold 140px Arial';
            ctx.textAlign = 'center';
            ctx.textBaseline = 'middle';
```

```
            ctx.fillText(courseData.id, canvas.width/2, canvas.height/2);

            const tex = new THREE.CanvasTexture(canvas);
            const labelGeo = new THREE.PlaneGeometry(1.8, 1.0);
            const labelMat = new THREE.MeshBasicMaterial({ map: tex,
transparent: true, side: THREE.DoubleSide });

            // Front label
            const labelFront = new THREE.Mesh(labelGeo, labelMat);
            labelFront.position.z = 0.76;
            mesh.add(labelFront);

            // Back label (rotated)
            const labelBack = new THREE.Mesh(labelGeo, labelMat);
            labelBack.rotation.y = Math.PI;
            labelBack.position.z = -0.76;
            mesh.add(labelBack);

            // Right label
            const labelRight = new THREE.Mesh(labelGeo, labelMat);
            labelRight.rotation.y = -Math.PI/2;
            labelRight.position.x = 0.76;
            mesh.add(labelRight);

            // Left label
            const labelLeft = new THREE.Mesh(labelGeo, labelMat);
```

```
            labelLeft.rotation.y = Math.PI/2;
            labelLeft.position.x = -0.76;
            mesh.add(labelLeft);

            scene.add(mesh);
            collectibles.push(mesh);
        }

        // Refresh all spawned courses to match current semester
        function refreshSpawnedCourses() {
            // remove old
            for (let i = collectibles.length - 1; i >= 0; i--) {
                const m = collectibles[i];
                scene.remove(m);
                collectibles.splice(i, 1);
            }
            // spawn new set for the current semester
            for (let i = 0; i < 30; i++) {
                spawnCourse((Math.random() - 0.5) * 80, (Math.random() -
0.5) * 80);
            }
        }

        // --- GAMEPLAY LOGIC ---
         const keys = { w:false, a:false, s:false, d:false, ArrowUp:false,
ArrowLeft:false, ArrowDown:false, ArrowRight:false, " ":false };
```

```
        window.addEventListener('keydown', (e) => { keys[e.key] = true;
});

        window.addEventListener('keyup', (e) => { keys[e.key] = false;
});


        // Menu Logic
        const semesterBtns = document.querySelectorAll('.semester-btn');
        semesterBtns.forEach(btn => {
            btn.addEventListener('click', () => {
                semesterBtns.forEach(b => b.classList.remove('selected'));
                btn.classList.add('selected');
                selectedSemester = btn.getAttribute('data-semester');
            });
        });

        // Degree option buttons (Essay / Thesis)
        const degreeBtns = document.querySelectorAll('.option-btn');
        degreeBtns.forEach(btn => {
            btn.addEventListener('click', () => {
                degreeBtns.forEach(b => b.classList.remove('selected'));
                btn.classList.add('selected');
                selectedDegreeOption = btn.getAttribute('data-option');
            });
        });

        const trackBtns = document.querySelectorAll('.track-btn');
        trackBtns.forEach(btn => {
```

```
            btn.addEventListener('click', () => {
                trackBtns.forEach(b => b.classList.remove('selected'));
                btn.classList.add('selected');
                selectedTrack = btn.getAttribute('data-track');
            });
        });

        document.getElementById('start-btn').addEventListener('click', () => {
            document.getElementById('start-screen').style.display = 'none';
            document.getElementById('hud').style.display = 'block';
            // Initialize semester state
            state.currentSemester = selectedSemester; // 'spring' or 'fall'
            state.picksThisSemester = 0; // courses picked this semester
            state.semestersElapsed = 1; // starting semester counts as 1
            // initialize semesterCourses with first empty semester
            state.semesterCourses = [[]];
            // Log the selected semester, degree option and track
            console.log(`Starting ${selectedSemester.toUpperCase()} semester, ${selectedDegreeOption.toUpperCase()} option, track: ${TRACKS[selectedTrack].name}`);
            // Spawn courses appropriate for starting semester
            refreshSpawnedCourses();
            updateHUD();
            animate();
        });
```

```
        function showMessage(text, isGood) {
            const el = document.getElementById('message-area');
            el.innerText = text;
            el.style.color = isGood ? '#4caf50' : '#f44336';
            el.style.opacity = 1;
            setTimeout(() => { el.style.opacity = 0; }, 2000);
        }

        function updateHUD() {
            document.getElementById('credit-count').innerText =
`${state.credits} / 36`;
            // Format money with commas
            const moneyEl = document.getElementById('money-count');
            if (moneyEl) {
                moneyEl.innerText =
`$${state.money.toString().replace(/\B(?=(\d{3})+(?!\d))/g, ",")}`;
                moneyEl.className = state.money <= 0 ? 'invalid' : '';
            }

            // Update percent bars
            const moneyPercent = Math.max(0, Math.round((state.money /
(STARTING_MONEY || 1)) * 100));
            const moneyFill = document.getElementById('money-bar-fill');
            const moneyPctText =
document.getElementById('money-percent-text');
            if (moneyFill) moneyFill.style.width = `${moneyPercent}%`;
```

```
            if (moneyPctText) moneyPctText.innerText = `${moneyPercent}%`;


            const gradPercent = Math.min(100, Math.round((state.credits /
36) * 100));
            const gradFill = document.getElementById('grad-bar-fill');
            const gradPctText =
document.getElementById('grad-percent-text');
            if (gradFill) gradFill.style.width = `${gradPercent}%`;
            if (gradPctText) gradPctText.innerText = `${gradPercent}%`;
            // Semester display
            const semEl = document.getElementById('semester-display');
            if (semEl) {
                if (state.currentSemester) {
                    semEl.innerText =
`${state.currentSemester.charAt(0).toUpperCase() +
state.currentSemester.slice(1)} ${state.semestersElapsed || 0}`;
                } else {
                    semEl.innerText = '-';
                }
            }
            const picksEl = document.getElementById('picks-count');
            if (picksEl) picksEl.innerText = `${state.picksThisSemester ||
0} / 3`;
            // Render semester tracker
            const tracker = document.getElementById('semester-tracker');
            if (tracker) {
                tracker.innerHTML = '';
                const rows = state.semesterCourses || [];
```

```
                for (let i = 0; i < rows.length; i++) {
                    const semName = (i % 2 === 0) ? 'Spring' : 'Fall';
                    const rowDiv = document.createElement('div');
                    rowDiv.className = 'sem-row';

                    const title = document.createElement('div');
                    title.className = 'sem-title';
                    title.innerText = `${semName} ${i+1}`;
                    rowDiv.appendChild(title);

                    const boxesContainer = document.createElement('div');
                    boxesContainer.style.display = 'flex';
                    boxesContainer.style.gap = '6px';

                    const semesterCourses = rows[i] || [];
                    // show up to 3 boxes; if fewer, show empty
placeholders
                    for (let j = 0; j < 3; j++) {
                        const box = document.createElement('div');
                        box.className = 'course-box';
                        if (semesterCourses[j]) {
                            const idSpan = document.createElement('span');
                            idSpan.className = 'course-id';
                            idSpan.innerText = semesterCourses[j].id;
                            const nameSpan =
document.createElement('span');
                            nameSpan.className = 'course-name';
```

```
                            nameSpan.innerText = semesterCourses[j].name;
                            box.appendChild(idSpan);
                            box.appendChild(nameSpan);
                        } else {
                            box.innerHTML = '<span
style="opacity:0.35;">(empty)</span>';
                        }
                        boxesContainer.appendChild(box);
                    }

                    rowDiv.appendChild(boxesContainer);
                    tracker.appendChild(rowDiv);
                }
            }

            const catAEl = document.getElementById('cat-a-count');
            catAEl.innerText = `${state.catACore} / 2`;
            catAEl.className = state.catACore >= 2 ? 'valid' : '';

            const catBEl = document.getElementById('cat-b-count');
            catBEl.innerText = `${state.catBCore} / 2`;
            catBEl.className = state.catBCore >= 2 ? 'valid' : '';

            const catABEl = document.getElementById('cat-ab-add-count');
            if (catABEl) {
                catABEl.innerText = `${state.catABAdditionalStrict} / 2`;
```

```
                catABEl.className = state.catABAdditionalStrict >= 2 ?
'valid' : '';
            }

            const essayEl = document.getElementById('essay-count');
            if (essayEl) {
                essayEl.innerText = `${state.essayCount} / 2`;
                essayEl.className = state.essayCount >= 2 ? 'valid' : '';
            }
            const thesisEl = document.getElementById('thesis-count');
            if (thesisEl) {
                thesisEl.innerText = `${state.thesisCount} / 2`;
                thesisEl.className = state.thesisCount >= 2 ? 'valid' :
'';
            }

            const flexEl = document.getElementById('flex-count');
            if (flexEl) {
                flexEl.innerText = `${state.flexibleCourses} / 4`;
                flexEl.className = state.flexibleCourses >= 4 ? 'valid' :
'';
            }

            const coreEl = document.getElementById('track-core-count');
            coreEl.innerText = `${state.trackCore} / 6`;
            coreEl.className = state.trackCore >= 6 ? 'valid' : '';
```

```
            // Win Condition Check: 36 credits + all category requirements
met
            if (state.credits >= 36 && state.catACore >= 2 &&
state.catBCore >= 2 &&
                state.catABAdditionalStrict >= 2 && (state.essayCount >= 2
|| state.thesisCount >= 2)) {
                document.getElementById('win-screen').style.display =
'flex';
                // Stop loop handled by logic
            }
        }

        function collectCourse(mesh, index) {
            if (state.gameOver) return;
            // Check funds first
            if (state.money < COURSE_COST) {
                showMessage(`Not enough money to enroll!`, false);
                triggerGameOver();
                return;
            }
            const data = mesh.userData;
            // Enforce prerequisites
            if (!hasPrereqs(data.id)) {
                showMessage(prereqMessage(data.id), false);
                return;
            }
```

```
            // Enforce degree-option restrictions
            if (selectedDegreeOption === 'thesis' &&
ESSAY_COURSES.includes(data.id)) {
                showMessage(`You selected Thesis option — cannot take
Essay courses like CS ${data.id}.`, false);
                return;
            }
            if (selectedDegreeOption === 'essay' &&
THESIS_COURSES.includes(data.id)) {
                showMessage(`You selected Essay option — cannot take
Thesis courses like CS ${data.id}.`, false);
                return;
            }

            // Logic: Cannot retake course
            if (state.collected.includes(data.id)) {
                showMessage(`Already took CS ${data.id}!`, false);
                return; // Don't remove, just bounce or ignore
            }

            state.collected.push(data.id);
            // Record this course under the current semester's list
            const semIndex = (state.semestersElapsed || 1) - 1;
            if (!state.semesterCourses) state.semesterCourses = [];
            if (!state.semesterCourses[semIndex])
state.semesterCourses[semIndex] = [];
            const fullName = COURSE_NAMES[data.id] || data.name || `CS
${data.id}`;
```

```
            state.semesterCourses[semIndex].push({ id: data.id, name: fullName });

            // Track course by category for requirement fulfillment
            const cat = getCourseCategory(data.id);
            if (cat === 'A') {
                if (state.catACore < 2) {
                    state.catACore++;
                } else if (state.catABAdditionalStrict < 2) {
                    state.catABAdditional++;
                    state.catABAdditionalStrict++;
                } else if (state.flexibleCourses < 4) {
                    state.flexibleCourses++;
                }
            } else if (cat === 'B') {
                if (state.catBCore < 2) {
                    state.catBCore++;
                } else if (state.catABAdditionalStrict < 2) {
                    state.catABAdditional++;
                    state.catABAdditionalStrict++;
                } else if (state.flexibleCourses < 4) {
                    state.flexibleCourses++;
                }
            } else if (cat === 'essay') {
                state.essayCount++;
                // Essay counts toward its own requirement; if flexible slots remain, count there
```

```
                if (state.flexibleCourses < 4) state.flexibleCourses++;
            } else if (cat === 'thesis') {
                state.thesisCount++;
                // Thesis counts toward its own requirement; if flexible slots remain, count there
                if (state.flexibleCourses < 4) state.flexibleCourses++;

            } else {
                // Flexible / other courses. Enforce independent-study limit (at most 2 of the 4)
                if (INDEPENDENT_STUDY.includes(data.id)) {
                    if (state.independentStudyCount < 2 && state.flexibleCourses < 4) {
                        state.flexibleCourses++;
                        state.independentStudyCount++;
                    } else {
                        // independent study doesn't count beyond limit
                        showMessage(`Independent study limit reached; CS ${data.id} won't count toward the 4 flexible MSCS credits.`, false);
                    }
                } else {
                    if (state.flexibleCourses < 4) state.flexibleCourses++;
                }
            }
            // Charge for the course
            state.money -= COURSE_COST;
            if (state.money < 0) state.money = 0;
```

```
            state.credits += 3;

            if (data.cat === 'A') state.catA++;
            if (data.cat === 'B') state.catB++;

            // Track Requirement Check
            const trackReqs = TRACKS[selectedTrack].core;
            if (trackReqs.includes(data.id)) {
                state.trackCore++;
                showMessage(`Picked up CS ${data.id} (Core)!`, true);
            } else {
                showMessage(`Picked up CS ${data.id} (Elective)`, true);
            }

            document.getElementById('last-action').innerText = `Added CS
${data.id} (${data.name})`;

            // Remove from scene
            scene.remove(mesh);
            collectibles.splice(index, 1);

            // Spawn a new one to keep field populated
            spawnCourse((Math.random()-0.5)*80, (Math.random()-0.5)*80);
            // Track picks for current semester and advance when needed
            state.picksThisSemester = (state.picksThisSemester || 0) + 1;
```

```
    updateHUD();

    // If money ran out exactly after purchase, game over
    if (state.money <= 0) {
        showMessage('You have no money left!', false);
        triggerGameOver();
    }

    if (state.picksThisSemester >= 3) {
        // Advance to next semester
        advanceSemester();
    }
}

function triggerGameOver() {
    if (state.gameOver) return;
    state.gameOver = true;
    const el = document.getElementById('game-over-screen');
    if (el) el.style.display = 'flex';
}

function advanceSemester() {
    // increase semester count and switch term
    state.semestersElapsed = (state.semestersElapsed || 1) + 1;
    if (state.semestersElapsed > 10) {
        showMessage('Maximum semesters reached!', false);
```

```
                triggerGameOver();
                return;
            }
            // toggle semester
            state.currentSemester = (state.currentSemester === 'spring') ?
'fall' : 'spring';
            state.picksThisSemester = 0;
            // refresh available courses for the new semester
            refreshSpawnedCourses();
            updateHUD();
            showMessage(`Now ${state.currentSemester.toUpperCase()}
semester`, true);
        }

        // --- ANIMATION LOOP ---
        let playerVelY = 0;
        const speed = 0.3;

        function animate() {
            requestAnimationFrame(animate);

            // If game over, freeze game (no input or pickups)
            if (state.gameOver) {
                renderer.render(scene, camera);
                return;
            }
```

```
            // Player Movement
            if (keys['w'] || keys['ArrowUp']) player.position.z -= speed;
            if (keys['s'] || keys['ArrowDown']) player.position.z +=
speed;
            if (keys['a'] || keys['ArrowLeft']) player.position.x -=
speed;
            if (keys['d'] || keys['ArrowRight']) player.position.x +=
speed;

            // Jumping
            if (keys[' '] && player.position.y <= 1) {
                playerVelY = 0.3;
            }

            player.position.y += playerVelY;
            playerVelY -= 0.015; // Gravity
            if (player.position.y < 1) {
                player.position.y = 1;
                playerVelY = 0;
            }

            // Camera Follow
            camera.position.x = player.position.x;
            camera.position.z = player.position.z + 15;
            camera.lookAt(player.position);

            // Animate Collectibles & Collision Detection
```

```
            const playerBox = new THREE.Box3().setFromObject(player);

            for (let i = collectibles.length - 1; i >= 0; i--) {
                const mesh = collectibles[i];
                mesh.rotation.y += 0.02;
                mesh.position.y = 1.5 + Math.sin(Date.now() * 0.003 +
mesh.userData.floatOffset) * 0.3;

                const box = new THREE.Box3().setFromObject(mesh);
                if (playerBox.intersectsBox(box)) {
                    collectCourse(mesh, i);
                }
            }

            renderer.render(scene, camera);
        }

        // Handle Resize
        window.addEventListener('resize', () => {
            camera.aspect = window.innerWidth / window.innerHeight;
            camera.updateProjectionMatrix();
            renderer.setSize(window.innerWidth, window.innerHeight);
        });

    </script>
</body>
```

```
</html>
```

References

Miao, R., Song, J., & Zhu, Y. (2017). 3D geographic scenes visualization based on WebGL. *2017 6th International Conference on Agro-Geoinformatics*, 1–6. https://doi.org/10.1109/agro-geoinformatics.2017.8046999

Prayudi, D., & Hamdi, N. (2023). Development of video games with webgl format that allows you to play video games without need for a device with high specifications. *Brilliance: Research of Artificial Intelligence*, *3*(2), 117–121. https://doi.org/10.47709/brilliance.v3i2.3004