CSCI 310 GUI & GAME

Project: 3D Game

Goal: This project challenges you to design and develop a 3D interactive game experience using modern web technologies. The goal is to build a functional, engaging, and visually appealing game while adhering to core principles of game development and good coding practices. You have the flexibility to choose your game's theme and mechanics, such as a shooter, puzzle, exploration, or simulation game, but must meet the fundamental requirements outlined below.

Core Technologies
This project requires the use of the Three.js library for all 3D rendering, scene management, and object creation.

Incorporating a physics engine (like Cannon.js or Rapier.js, for example) is highly needed for some types of games. You need to have proper rationale for not using a physics engine.

Requirements:

Your game should start from a game story which set up the theme of the play. Therefore, you need to include a game story as a beginning point of your development.

Regardless of your chosen game type, every interactive 3D game shares fundamental components. Consider these abstractions as you design and build:

- **Visuals & World Building**
    - 3D Scene Setup: Establish a Three.js scene, including a camera for perspective (e.g., first-person, third-person, or top-down), and a renderer to display your game.
    - Lighting: Implement appropriate lighting to illuminate your 3D world and objects. Consider ambient light, directional light, or point lights to set the mood and provide visual depth.
    - 3D Objects: Create and manage various 3D objects within your scene. These could represent:
    - A player character or controlled entity.
    - Environmental Elements (e.g., ground, walls, background elements like stars or buildings).
    - Interactive Elements (e.g., collectible items, targets, enemies).
    - Materials & Textures: Apply materials and colors to your 3D objects to define their appearance. You can experiment with basic colors or advanced materials for more realism.

- **Interaction & Controls**

- Player Input: Implement mechanisms for player input, typically using keyboard events (e.g., WASD or arrow keys for movement, spacebar for actions). Mouse input for camera control or aiming can also be included.
- Object Manipulation: Allow the player to interact with objects in the world, whether through direct movement of the player character, pushing objects, firing projectiles, triggering events or etc.

- **Physics & Movement**
  - Movement Logic: Implement movement for your game objects. For the player, this might involve applying force, velocity, or directly setting position. For other objects, it could be basic animations or physics-driven motion.
  - Collision Detection: Detect when objects in your game interact (collide) with each other. This is crucial for actions like hitting targets, collecting items, or preventing objects from passing through walls.
  - Response to Collisions: Define how your game responds to collisions (e.g., objects bouncing, disappearing, triggering events, or taking damage).

- **Game Flow**
  - Initialization: The game should initialize smoothly upon loading, setting up the scene, objects, and any necessary physics environments.
  - Game Loop: Implement a continuous animation loop (requestAnimationFrame) to update the game state, physics, object positions, and render the scene each frame.
  - Reset Mechanism: Provide a way to reset the game to its initial state (e.g., by pressing a specific key like 'R'), allowing for replayability.
  - Start/End States: Consider how your game begins (e.g., immediate start) and how it might conclude (e.g., achieving a goal, running out of lives). While full overlay UIs are not required for this simplified approach, a simple console log or on-screen text for "Game Over" is sufficient.

- **Scoring & Feedback**
  - Scoring: This could be a score based on targets hit, items collected, or time survived.
  - Visual Feedback: Provide clear visual or audio feedback to the player (e.g., objects disappearing upon impact, score updates, or simple color changes).

You are allowed to improve the sample games, but please ensure you can satisfy the above requirements and you need to improve sufficiently to improve the quality of the game.

NOTE: if you need to host your game in a domain. There are two options for doing that.

(1) Host on the github

(2) Docker (suggested): you can use the docker file on the sample code folder.

Grading Policy (100pts + Max 20pts)

- Fulfill above requirements (50pts)
- Visuals & Design (15pts):
- Three.js and Physics engine Implementation (20pts)
- Quality of the Game (15pts)
- Bonus (Max 20pts) – Creativity and Fun

Presentation (20pts)

- Prepare for a 8 minute presentation + 2 minute Q&A.
- Include your game story.
- Show how you develop the game.
    - Ideation
    - Development
    - Each of the team member's tasks and responsibilities.
- Demonstrate your game and show you follow the requirements

Developer Notebook (30pts)

- Follow the specification and document your development well.

Each item will be graded based on:

- Well-completed (100%)
- Completed (85-100%)
- Developing (75-85%)
- Progressing (60-75%)
- Under-developed (under 60%)
- Your personal grades will also be calculated based on individual contributions on the team project.

Submission:

- Submit all your codes, your developer notebook, and presentation to BrightSpace.
    - If your game needs a domain, please include the running environment for me to check.