**1. Introduction**

This expanded report documents the development of a **snooker application** written in JavaScript using **p5.js** (for rendering and user interaction) and **matter.js** (for 2D physics simulation). The assignment, derived from **CM2030 – Graphics Programming**, requires implementing:

1. A realistically proportioned snooker table (ratio **2:1**).
2. Balls with correct **physics** properties and **pocket** detection.
3. A **cue** system controlled by user input.
4. Multiple **game modes** (standard vs. random setups).
5. Additional functionalities like collision prompts and re-spotting colored balls.

This report also includes references to **key parts of the code**, short explanatory notes on how certain functions operate, and instructions for **how to play** the game.

**2. Implementation Overview**

**2.1 Table Dimensions and Ratios**

* The table is created in code via the Table class. For example:

js

Copy code

let tableX = 200;

let tableY = 50;

let tableW = 1000;

let tableH = 500;

table = new Table(tableX, tableY, tableW, tableH, pocketSize, engine);

This ensures a **2:1 ratio** (length = 1000, width = 500), matching the real-life ratio of **12 ft × 6 ft**.

* The **ball diameter** is defined as tableH / 36 to approximate the 2-inch ball diameter:

js

Copy code

let ballDiameter = tableH / 36; // e.g., 500 / 36 ≈ 13.9px

And the **pocket** diameter is 1.5 \* ballDiameter, per the assignment instructions.

**2.2 Drawing the Table and Pockets**

* The Table class (shown partially below) uses p5.js functions (rect(), arc(), etc.) to draw:
  1. A **wooden boundary** around the cloth.
  2. A **green cloth** rectangle for the playable area.
  3. The **baulk line** and the “**D**” semi-circle.
  4. **Six pockets** (four corners + two middle pockets) using Bodies.circle(..., { isSensor: true }) in matter.js for detection.

js

Copy code

setupPockets() {

let r = this.pocketSize / 2;

// Create corner pockets + mid side pockets

// Each is a static sensor circle in Matter.js

...

}

* **Cushions** are instantiated as Cushion objects, each a static Matter.js rectangle with high restitution (restitution: 1.0) to bounce balls realistically.

**2.3 Ball Arrays and Multiple Modes**

* All balls (15 reds, 6 colors, and the cue ball) are stored in a single array named balls. Each ball is represented by a custom Ball class, which:

js

Copy code

class Ball {

constructor(x, y, diameter, color, engine, label) {

this.body = Bodies.circle(x, y, diameter / 2, { ... });

...

}

draw() { /\* uses p5 ellipse(...) with ball color \*/ }

}

* **Three primary modes** are triggered via key presses (1, 2, and 3), coded in the keyPressed() function:
  1. **Mode 1** (key=1): Standard triangular arrangement (like classic snooker).

js

Copy code

function setupStandardPositions(d) {

// Place colors in usual positions, then stack reds in a triangle

}

* 1. **Mode 2** (key=2): Random placement of reds only, with standard color ball positions.
  2. **Mode 3** (key=3): Completely random placement of both reds and colored balls.
* A **fourth mode** (key=4) is also supported, introducing **advanced rules** and an **AI** (see [2.6 Extension](#extension-ai-opponent)).

**2.4 Physics with Matter.js**

* Each ball is a **Matter.js body** with friction, frictionAir, and restitution:

js

Copy code

this.body = Bodies.circle(x, y, diameter / 2, {

restitution: 0.9,

friction: 0.01,

frictionAir: 0.005,

label: label

});

* **Cushions** (using the Cushion class) are rectangles with isStatic=true and restitution=1.0. They cause **bounce** when balls collide:

js

Copy code

let opts = { isStatic: true, restitution: 1.0, label: 'Cushion' };

this.body = Bodies.rectangle(x, y, width, height, opts);

* Collisions are handled by the event:

js

Copy code

Events.on(engine, 'collisionStart', handleCollision);

The handleCollision() function checks if a ball was potted (ball-pockets collision), or if the cue ball collided with a cushion or other balls.

**2.5 Cue Mechanics**

* A dedicated Cue class controls:
  1. **Angle**: Calculated from the cue ball’s position to the mouse location (via setAngleFromMouse(mx, my)).
  2. **Power**: Adjusted via a vertical power bar (drawn on the left side of the canvas).
  3. **Striking**: Pressing SPACE calls cue.triggerStrike(), which animates the cue forward and applies a force to the cue ball.

js

Copy code

triggerStrike() {

if (!this.ball || this.ball.isPotted) return;

this.isStriking = true; // animation logic in update()

}

When fully “pushed forward,” it calls applyStrikeForce(), which applies a force (magnitude proportional to the user-set power) to the cue ball in matter.js.

**2.6 Potting Logic**

* **Red balls**: If a red ball’s center is close enough to a pocket sensor, it is removed from the balls array (or set isPotted=true).
* **Colored balls**: If a color is potted, it is typically re-spotted on its original position (unless advanced mode logic changes that).
* **Cue ball**: If potted before any red, it must be placed again in the “D” area.
* The relevant code block is in handleCollision(), which detects potting events:

js

Copy code

function handleCollision(event) {

let pairs = event.pairs;

...

// if (distance < ballRadius + pocketRadius) => isPotted

}

**3. Collision Detection & Prompting**

The collision function (handleCollision()) logs collisions for the **cue ball** specifically:

* **cue-red**
* **cue-colour**
* **cue-cushion**

Example snippet:

js

Copy code

if (labels.includes('Cue') && audioCollision.length > 0) {

// Play collision sound if cue ball is involved

}

...

if (ballObj.label.startsWith('Red')) {

// Increase player score for a red, remove it from the table, etc.

}

Additionally, the application plays distinct sounds for collisions and potting (loaded during preload()).

**4. Commentary on Design and Extensions**

**4.1 How to Play the Game**

1. **Launching the App**:
   * Load the index.html in a browser (which references p5.js, p5.sound.js, matter.js, and your JavaScript files).
2. **Choose a Mode**:
   * Press **‘1’** for **standard snooker** ball arrangement.
   * Press **‘2’** for **random reds**.
   * Press **‘3’** for **random reds & random colors**.
   * Press **‘4’** for **advanced rules** + optional AI demonstration.
3. **Place the Cue Ball**:
   * If not yet placed, **click inside the “D” semi-circle** on the left side of the baulk line.
   * The cue ball will appear where you clicked, assuming your click is within the valid half-circle boundary.
4. **Aiming the Cue**:
   * **Move the mouse** around the table to rotate the cue around the cue ball.
5. **Adjust Power**:
   * A **power bar** is drawn on the left side.
   * **Click and drag** inside the bar to set a power level between 0% and 100%.
6. **Strike the Ball**:
   * Once satisfied with the angle and power, **press SPACE**.
   * The cue animates forward and hits the cue ball, sending it towards the targeted direction.
7. **Scoring and Potting**:
   * Red balls are removed from the table if potted (or re-spotted in advanced modes).
   * Colored balls are re-spotted or tallied according to the mode’s rules.
   * If the **cue ball** goes in a pocket, it must be placed again in the “D.” If it happens before potting a red, the current turn may switch (in advanced logic).
8. **Other Controls**:
   * Press **‘R’** to **show/hide** the rules overlay on screen.

**4.2 Random Modes**

* For the random modes (key=2 or key=3), the positioning logic is handled by functions like:

js

Copy code

function getRandomPosition(diam) { ... }

which randomly picks (x, y) within table bounds, checking if it overlaps any existing ball or pocket. The process repeats until a valid position is found.

**4.3 Unique Extension: AI Opponent (Mode 4)**

* In **Mode 4**, the code instantiates:

js

Copy code

aiDebugger = new AIDebugger();

aiEngine = new AIEngine(aiDebugger, cue, balls, table, snookerRules);

* The AI attempts to pot the appropriate “ball on” or execute a safety. This goes beyond normal assignment requirements, showcasing **innovative** gameplay.

**4.4 Code Reusability & OOP**

* Each major component is in its own class (Table, Ball, Cue, etc.), satisfying strong **object-oriented design** principles.
* Key data (positions, velocities, angles) is stored in these objects, not in global scope. Functions like respot(x, y) or checkPotted() belong to Ball objects, boosting clarity.

**5. Conclusion**

This snooker application successfully fulfills the **CM2030** requirements:

* **Realistic Table**: 2:1 ratio with correct pockets, baulk line, and “D”.
* **Physics**: Balls and cushions use matter.js with friction, restitution, and collision detection.
* **Cue Mechanics**: A dynamic cue controlled by mouse (angle) and keyboard (space to strike).
* **Modes**:
  + **Mode 1**: Standard layout.
  + **Mode 2** & **3**: Randomized ball setups.
  + **Mode 4**: Advanced rule demo with an optional AI extension.
* **Collision Detection**: Distinguishes among cue-cushion, cue-red, cue-color collisions; logs potting events and re-spots balls appropriately.
* **User Interaction**: Manual placement of the cue ball in the “D,” adjustable shot power, and re-placing the cue ball if potted prematurely.