Discrete Mathematics Simulator

Luka Cassar

Running the Game

To run the game, ensure the following prerequisites are met:

- Node.js: Version 22.15.1 or higher, available at https://nodejs.org.
- Express.js: Version 5.1.0, run **npm install express** inside terminal.

Start the server by running **npm start** inside terminal, and keep the window open.

Access the game at **http://localhost:3000/game.html** – you will be greeted with a main menu.

Save state by:

(1) unlocking an upgrade,

(2) finishing the minigame, or

(3) returning to the main menu (click ”Main Menu”).

Refresh the page. Verify points, upgrades, and achievements persist.

Example: Unlock Predicate Logic, refresh, and confirm the unlock button is hidden and points are retained.

Points earned via ”Solve Question” or idle generation are not saved until one of these actions is performed. To retain progress, ensure you trigger a save before refreshing.

Restarting the Server:

In the terminal, hit **CTRL+C** and type **Y** if prompted for confirmation. Then, run **npm start** again. Your data will be lost (game functions via in-memory storage).

UI Design Decisions

The UI consists of three main sections: the main menu, game screen, and minigame screen, styled with style.css and controlled via scripts.js.

Buttons (e.g., ”Play,” ”Solve Question,” ”Return to Menu”) use high-contrast colors (white text on green backgrounds) and hover effects which change the colour of the button for an aesthetic look. For instance:

```
.button:hover {
  background-color: #3d723f;
  border-color: #2e5a30;
}
```

Clicking ”Solve Question” triggers a cooldown animation (grayed-out button) and a floating ”+X” points indicator, reinforcing player actions. Notifications (e.g., ”Predicate Logic unlocked!”) appear briefly at the top, styled as:

```
.notification {
  background: #4caf50;
  color: white;
  font-size: 16px;
  font-weight: bold;
  padding: 8px 12px;
  border-radius: 5px;
  margin-bottom: 10px;
  opacity: 0;
  animation: notify 2s ease-in-out forwards;
}
```

To support assistive technologies (e.g., screen readers), role attributes and ARIA properties are used:

• Buttons: Upgrade buttons include aria-label for clarity,

e.g., <ul id="nav-menu" aria-label="navigation menu">

Game Mechanics and Implementation

Point Generation:

- Manual: Clicking ”Solve Question” earns **pointsPerSolve** (default 1) points every **solveCooldown** (default 600ms).
- Idle: Every 5 seconds, **baseGenerationRate** (default 2) points are added.
- Boosts: A 3x point boost is applied for **minigameScore** seconds after the minigame, and a 2.5x boost occurs every 60 seconds for 10 seconds after upgrading Relations.
- Bonuses: Chance to get 1000 points every 60 seconds. This is available after unlocking Predicate Logic, being a **1%** chance at that stage. Unlocking Set Theory makes it **10%**, Relations makes it **20%** and Classifying Relations makes it **40%**

solveQuestion handles clicks, applying cooldown and updating points:

```
function solveQuestion() {
  if (!cooldown) {
    let pointsGained = pointBoostActive ? pointsPerSolve * 3 : pointsPerSolve;
    points += pointsGained;
    showPointsIndicator(pointsGained);
    updatePoints();
    checkAchievements();
    cooldown = true;
    let solveButton = document.querySelector('.game-button-container .button');
    solveButton.classList.add('cooldown');
    setTimeout(function () {
      cooldown = false;
      solveButton.classList.remove('cooldown');
```

```
    }, solveCooldown);
  }
}
```

- Predicate Logic: Unlock (100 points) enables a 3-point-per-click upgrade (250 points).
- Set Theory: Unlock (1000 points) enables an idle rate increase (1500 points, +1 every 5 seconds).
- Relations: Unlock (5000 points) enables a periodic 2.5x boost (7500 points).
- Classifying Relations: Unlock (20000 points) enables a secret bonus (triple your points or reduces cooldown to 100ms, costing 30000 points)

Minigame:

- Players click circles (pink elephant image) spawning at increasing rates over 30 seconds.
- The score (**minigameScore**) determines the duration of a 3x point boost.

startMinigame spawns circles with dynamic intervals

```
function spawnCircle() {
    const container = document.getElementById('minigame-container');
    const circle = document.createElement('div');
    circle.className = 'minigame-circle';
    const maxX = container.clientWidth - 80;
    const maxY = container.clientHeight - 80;
    const x = Math.random() * maxX;
    const y = Math.random() * maxY;
    circle.style.left = `${x}px`;
    circle.style.top = `${y}px`;
    circle.onclick = () => {
      minigameScore++;
      circle.remove();
```

```
    };
    container.appendChild(circle);
    setTimeout(() => circle.remove(), 1300);
  }
```

Achievements:

- Six achievements: three for points (100, 1000, 10000) and three for upgrades (1, 2, 3)

  checkAchievements updates unlocked and notifies once:

```
function checkAchievements() {
  achievements.forEach(achievement => {
    if (!achievement.unlocked) {
      if (achievement.points && points >= achievement.points) {
        achievement.unlocked = true;
        notify(`${achievement.name} unlocked: ${achievement.description}`);
      } else if (achievement.upgrades && totalUpgrades >= achievement.upgrades) {
        achievement.unlocked = true;
        notify(`${achievement.name} unlocked: ${achievement.description}`);
      }
    }
  });
  updateAchievementsDisplay();
}
```

API Calls and Server Operations

The game uses an Express server (server.js) to manage state via two API endpoints: POST /gameState and GET /gameState/:playerId. State is stored in-memory, resetting on server restart.

POST /gameState:

- Purpose: Saves the game state for a player.
- Payload:

  playerId: String (e.g., ”player-rgz3b7pca”)

  points, minigameScore, baseGenerationRate, predicateUpgrades, pointsPerSolve,solveCooldown, totalUpgrades: Numbers.

  predicateUnlocked, setUnlocked, relationsUnlocked, classifyingUnlocked: Booleans.

  achievements: Array of objects with id (string) and unlocked (boolean).

- Operation:

  Validates types and ranges (e.g., points >= 0, achievements is an array).

  Stores state in gameState[playerId]. ∗ Returns { message: 'Game state saved' } or 400 (Invalid data).

GET /gameState/:playerId:

- Purpose: Retrieves the saved state for a player.
- Response: Returns the state object or 404 (Game state not found).
- Operation: Looks up gameState[playerId] and returns it.

Showcasing the game via a YouTube Video:

https://www.youtube.com/watch?v=FZZ8aDCsS6Y