

Circuit Puzzle Game (RC Timing & Drag-and-Drop)

Group 08
January 14, 2026

Abstract

This report presents the design and implementation of the **Circuit Puzzle Game**, an educational JavaFX-based application designed to help students visualize and understand basic electrical circuit concepts. Players construct electrical circuits by dragging components—such as wires, resistors, and capacitors—onto a grid board. The system simulates the circuit behavior (calculating total resistance, capacitance, and voltage drops) and evaluates whether the configuration meets the level objectives.

The project emphasizes **Object-Oriented Programming (OOP)** principles, including abstraction, inheritance, polymorphism, and modular design. UML Use Case and Class diagrams are employed to articulate the system structure and behavioral logic.

I. Assignment of Members

1. Nguyen Hoang Anh

- **Student ID:** 202416773
- **Role:** Requirement analysis, System design (Use Case, Class Diagrams), Java implementation (UI, Simulation), Documentation.

2. Pham Duy Nguyen

- **Student ID:** 202416730
- **Role:** Requirement analysis, System design (Use Case, Class Diagrams), Java implementation (Game Logic, UI), Documentation.

3. Do Thanh Trung

- **Student ID:** 202416833
- **Role:** Requirement analysis, System design (Use Case Diagram), Java implementation (Game Logic), Documentation.

4. Le Viet Anh

- **Student ID:** 202416774
- **Role:** Requirement analysis, System design (Use Case, Class Diagrams), Java implementation (Components), Documentation.

Statement of Academic Integrity:

All source code and concepts presented in this project are self-developed or adapted with full understanding. No unauthorized copying has occurred.

II. Mini-Project Description

1. Project Requirements

The Circuit Puzzle Game is designed to meet the following functional requirements:

- **Level Selection:** Users can select between Series or Parallel circuit challenges.
- **Component Toolbox:** A graphical inventory containing electrical components.
- **Interactive Board:** Enables drag-and-drop placement of components onto a grid.
- **Simulation Engine:** Simulates circuit behavior in real-time based on the topology of placed components.
- **Result Evaluation:** Validates circuit integrity and calculates performance metrics (e.g., lighting duration of the bulb).

2. Use Case Diagram and Explanation

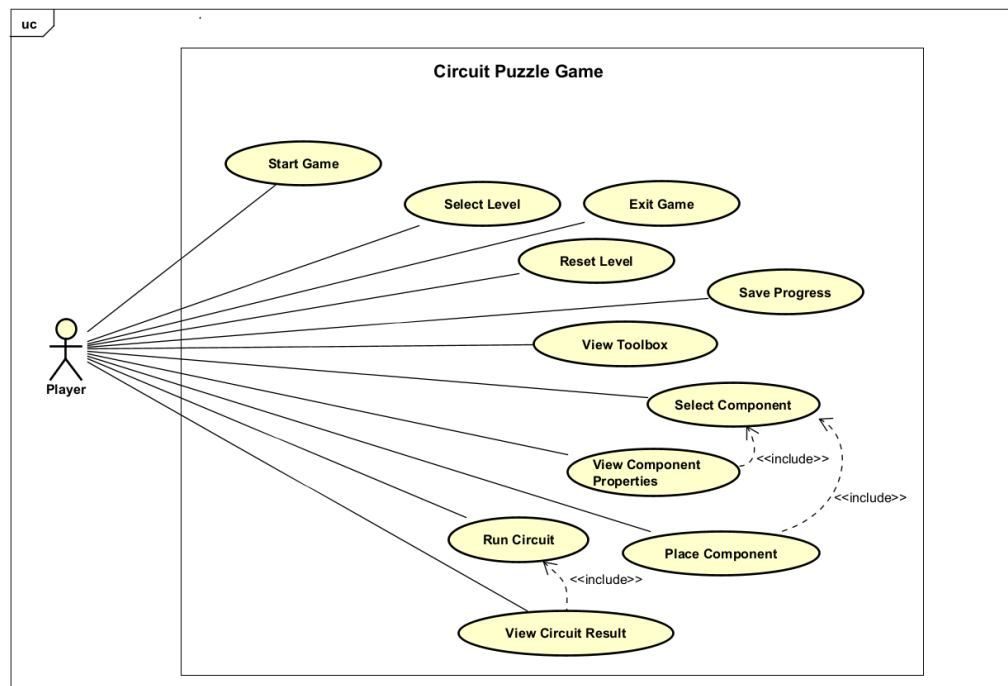


Figure 1: Use Case Diagram of Circuit Puzzle Game

2.1. Actors

- **Player:** The primary user who interacts with the system to solve circuit puzzles.

2.2. Main Use Cases

- **Start Game:** Initializes the game session.
- **Select Level:** Allows the player to choose a Series or Parallel circuit mode.
- **View Toolbox / Select Component:** Browsing and selecting items from the inventory.
- **Place Component:** dragging a component onto the grid board.
- **View Properties:** Inspecting attributes like resistance or capacitance.
- **Run Circuit:** Triggers the simulation engine.
- **View Result:** Displays success/failure status and metrics.
- **Reset/Exit:** Clears the board or terminates the application.

2.3. Use Case Relationships

- Select Component **includes** View Component Properties.
- Run Circuit **includes** View Circuit Result.
- Play Tutorial **extends** Start Game.

III. Design

This section details the software architecture of the **Circuit Puzzle Game**. The design adheres to OOP principles, organized into logical packages.

1. General Class Diagram

Figure 2 illustrates the high-level architecture. The system is divided into three primary packages: **Component**, **Board**, and **Utils**.

- **Component:** Contains the abstract base class and concrete circuit elements (Resistors, Wires, Sources).
- **Board:** Manages the grid state and evaluation logic (SeriesBoard, ParallelBoard).
- **Utils:** Provides helper functions for connectivity logic and GUI rendering.

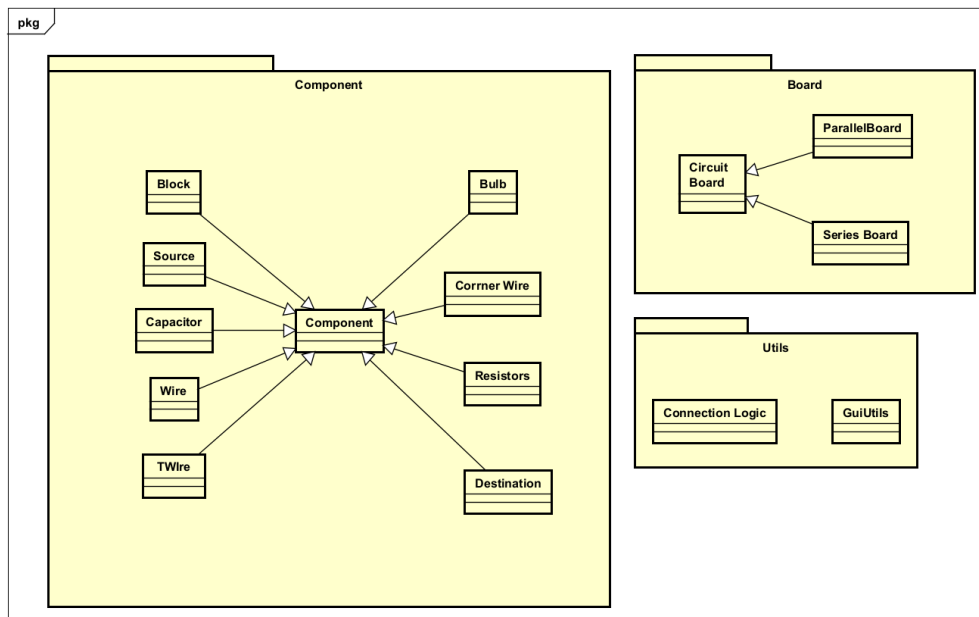


Figure 2: General Class Diagram

2. Detailed Class Diagrams

2.1. Component Package

The core of this package is the abstract class **Component**, which defines shared attributes:

- **voltage, current**: Electrical properties.
- **isLocked**: Prevents removal of fixed level elements.
- **rotationDegree**: Orientation state for wires.

Concrete implementations include **Wire** (conductors), **Resistor/Capacitor** (load elements), and **Source/Bulb** (I/O elements). This structure allows the board to treat all elements polymorphically.

2.2. Board Package

The **CircuitBoard** class acts as the container for the 2D grid. Key responsibilities include:

- **placeComponent(r, c, comp)**: Validates and inserts a component.
- **calculateTotalResistance()**: Abstract method implemented by subclasses.

SeriesBoard and **ParallelBoard** extend **CircuitBoard** to implement topology-specific physics calculations.

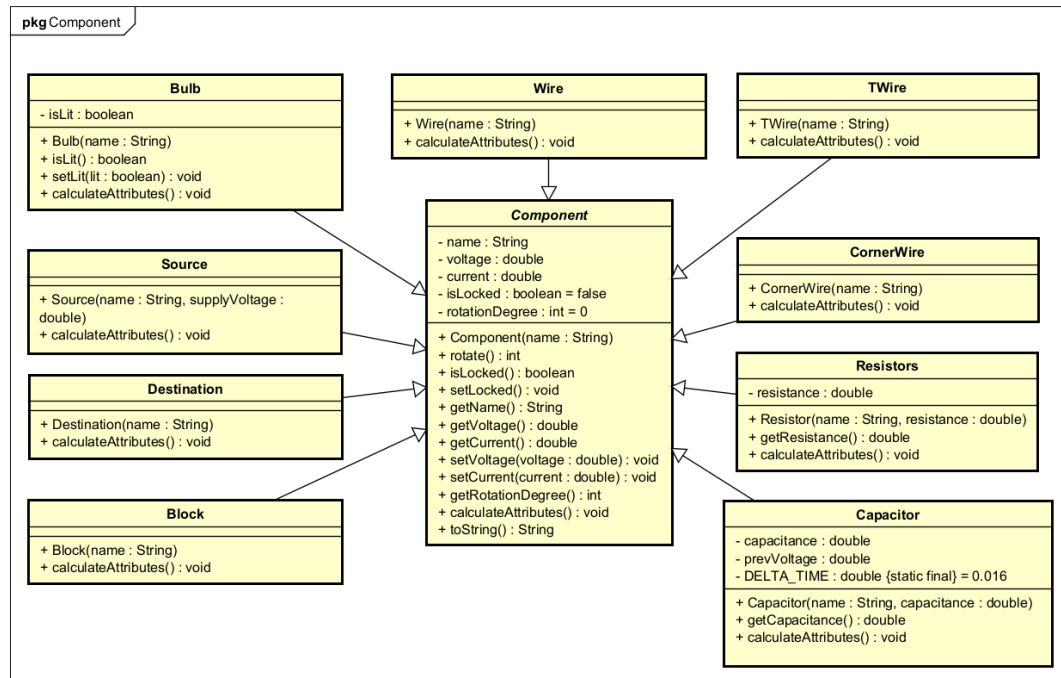


Figure 3: Detailed Component Package

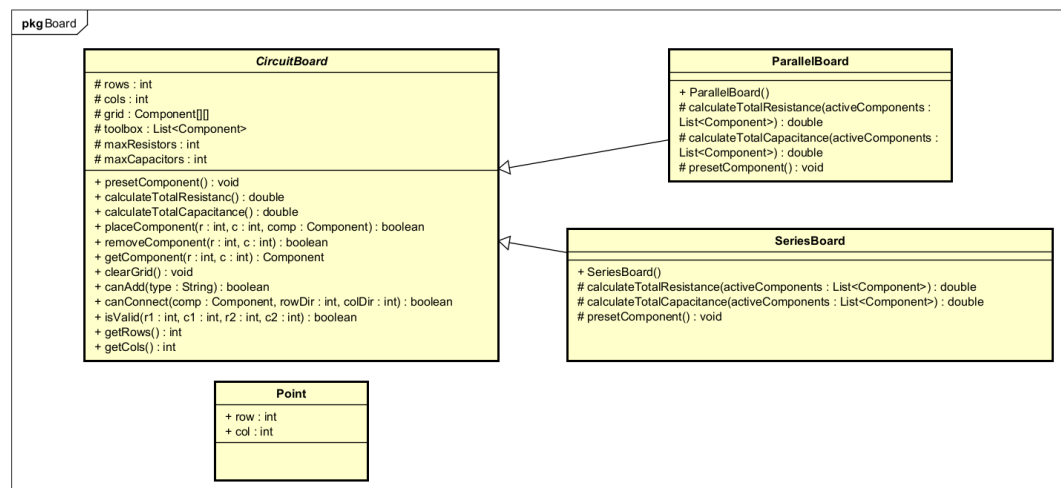


Figure 4: Detailed Board Package

2.3. Utils Package

The **Utils** package contains stateless helper classes:

- **ConnectionLogic**: Uses graph traversal (BFS/DFS) to verify if the Source connects to the Destination.
- **GuiUtils**: Handles visual rendering of valid connection points.

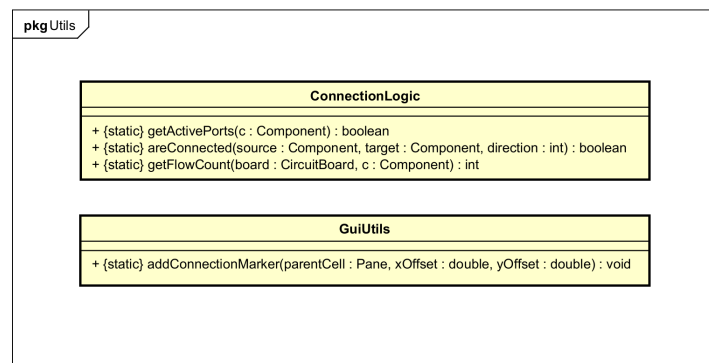


Figure 5: Utils Package

3. Design Relationships & Logic

3.1. Inheritance & Polymorphism

The system leverages inheritance to decouple the game logic from specific component types. For instance, the `CircuitBoard` iterates over a list of `Component` objects to calculate voltage drops, without needing to know if a specific object is a `Resistor` or a `Bulb` at compile time.

3.2. Circuit Evaluation Logic

The simulation engine calculates total resistance (R_{eq}) and capacitance (C_{eq}) based on the board type:

- **Series Circuit:**

$$R_{eq} = \sum_i R_i \quad ; \quad \frac{1}{C_{eq}} = \sum_i \frac{1}{C_i}$$

- **Parallel Circuit:**

$$\frac{1}{R_{eq}} = \sum_i \frac{1}{R_i} \quad ; \quad C_{eq} = \sum_i C_i$$

The bulb lighting duration (t) is derived from the RC time constant (τ):

$$\tau = R_{eq} \times C_{eq}$$



IV. Conclusion

The Circuit Puzzle Game successfully integrates OOP principles to create an interactive educational tool. By utilizing abstraction and polymorphism, the system is designed to be extensible, allowing for future addition of complex components (e.g., transistors or inductors) with minimal changes to the core engine.