# Content

1. Allocation ........ 3

1. 1. Resources—Liu Tengyi (Prepare the image resources) ........ 3

1. 2. Entity—Zhou Bohan (User entity + Data Access Layer) ........ 3

1. 3. Service + UI Implementation—Tu Yuxiang + Ouyang Xiaojun ........ 3

1. 4. UI Design + UI Mainframe Implementation—Wang Zixuan ........ 4

2. Design of the system ........ 4

2. 1. Flowchart ........ 4

2. 2. Main Classes ........ 6

2. 2. 1. Project Structure and Functionality Description ........ 6

2. 2. 2. Project Workflow Description ........ 8

3. Design Highlights Analysis ........ 9

3. 1. Adaptive Design for Colorblind Users ........ 9

3. 1. 1. Diverse Color Schemes ........ 9

3. 1. 2. Adaptive Background Images ........ 9

3. 1. 3. Semantic Color Definitions ........ 9

3. 1. 4. Dynamic Color Switching Mechanism ........ 9

3. 2. Child-Friendly Interaction Design ........ 10

3. 2. 1. Specialized Button Components ........ 10

3. 2. 2. Interactive Animation Effects ........ 10

3. 2. 3. Visual Progress Tracking ........ 10

4. User-manual ........ 10

- 4. 1. Introduction ........ 10
- 4. 2. System Requirements ........ 11
- 4. 3. Getting Started ........ 11
- 4. 4. Application Interface ........ 11
- 4. 5. Accessibility Features ........ 11
- 4. 6. Learning Modules ........ 12
- 4. 7. Scoring System ........ 13
- 4. 8. Tips for Teachers/Parents ........ 13
- 4. 9. Ending a Session ........ 13
- 4. 10. Troubleshooting ........ 13

5. Utilisation of AI ........ 14
- 5. 1. Colorblind-Friendly Color System Implementation ........ 14
- 5. 2. Java Drawing Functionality (Geometric Visualization) ........ 14
- 5. 3. Python Angle Generation Script ........ 15

6. Reflection on your experience ........ 16
- 6. 1. My First Java GUI Project Experience ........ 16
- 6. 2. What Went Well ........ 16
- 6. 3. What Could Be Improved ........ 16

7. Appendix(Prompts and Answers) ........ 17
- 7. 1. Colorblind-Friendly Color System Implementation ........ 17
- 7. 2. Java Drawing Functionality (Geometric Visualization) ........ 18
- 7. 3. Python Angle Generation Script ........ 22

# 1. Allocation

## 1.1. Resources—Liu Tengyi (Prepare the image resources)

Liu Tengyi was mainly responsible for the preparation of questions and answers: first, including the screenshot of question images for task1, bonus1, and bonus2; for the angle images of task2, he used a Python script to automatically generate the required 35 images; for task3 and task4 which required randomly generating question parameters and producing corresponding answer images and analysis, he completed the code for generating answer analysis images covering 8 scenarios for these two tasks; for the answer analysis of bonus1 and bonus2, he manually drew each one and saved them in the resources folder after organization.

## 1.2. Entity—Zhou Bohan (User entity + Data Access Layer)

Zhou Bohan was primarily responsible for establishing the user entity and implementing the data access layer: he created the User class, utilizing static variables and static methods to track users' game progress; additionally, he implemented the functionality of the data access layer, which involved encapsulating question data and corresponding image information from the resource layer into Java, while also developing foundational answer validation features to support subsequent service layer invocations.

## 1.3. Service + UI Implementation—Tu Yuxiang + Ouyang Xiaojun

### Tu Yuxiang:

At the service layer, responsible for task3, task4, bonus1, and bonus2

At the UI layer, responsible for implementing task3, task4, bonus1, and bonus2 interfaces

### Ouyang Xiaojun:

At the service layer, responsible for task1 and task2

At the UI layer, responsible for implementing task1 and task2 interfaces

By invoking classes from the data access layer and comprehending specific business logic, they accomplished the implementation of concrete business functionalities for each task. Additionally, by comprehending and applying the Swing library, they successfully implemented the GUI interface.

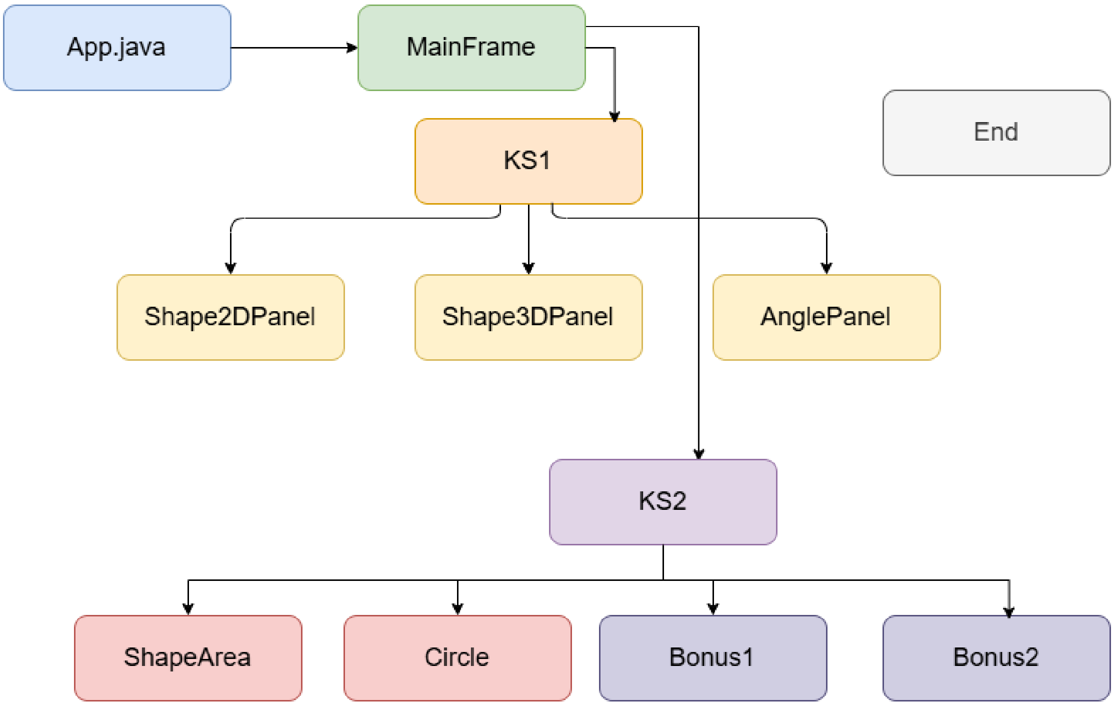
### 1. 4. UI Design + UI Mainframe Implementation—Wang Zixuan

Wang Zixuan was responsible for the overall UI design, including the style of the main interface, the architectural design of interface switching, and the specific styling of each individual interface. For colorblind accessibility, three sets of color schemes were designed along with color scheme switching buttons. Additionally, meticulous design was applied to components such as buttons, and multiple UI prototypes were created to facilitate the implementation of UI code.

Apart from these, Wang Zixuan also implemented the Mainframe UI interface and correpsonding UI utils such as the ColorScheme, KidButton and so on, which establishes a solid foundation for the UI interfaces within this mini-project.

## 2. Design of the system

### 2. 1. Flowchart



1. Project Initiation

The project starts with [App.java], which serves as the entry point of the application, responsible for initializing and launching the program.

2. Main Interface Display

In [App.java], the [MainFrame] is called. This class represents the main framework of the application and is used to host various functional panels in the subsequent stages.

3. Phase 1 Functionality (KS1)

Within [MainFrame], the user enters the first functional module, known as `KS1`.

Functional Branches in KS1:

Shape2DPanel: Handles operations related to two-dimensional shapes.

Shape3DPanel: Manages tasks involving three-dimensional objects.

AnglePanel: Deals with functionalities related to angles.

4. Phase 2 Functionality (KS2)

From any of the three panels in `KS1` ([Shape2DPanel], [Shape3DPanel], or [AnglePanel]), the user can proceed to the second stage of functionality, referred to as `KS2`.

Functional Branches in KS2:

ShapeArea: Calculates the area of different shapes.

Circle: Provides specific operations related to circles.

Bonus1 and Bonus2: Additional bonus modules offering extra features.

5. End of the Process

Once all operations are completed, the process reaches the `End` state, indicating that the program has finished execution.

6. Module Functions:

The KS1 phase primarily provides foundational functionalities for handling shapes and angle-related tasks.

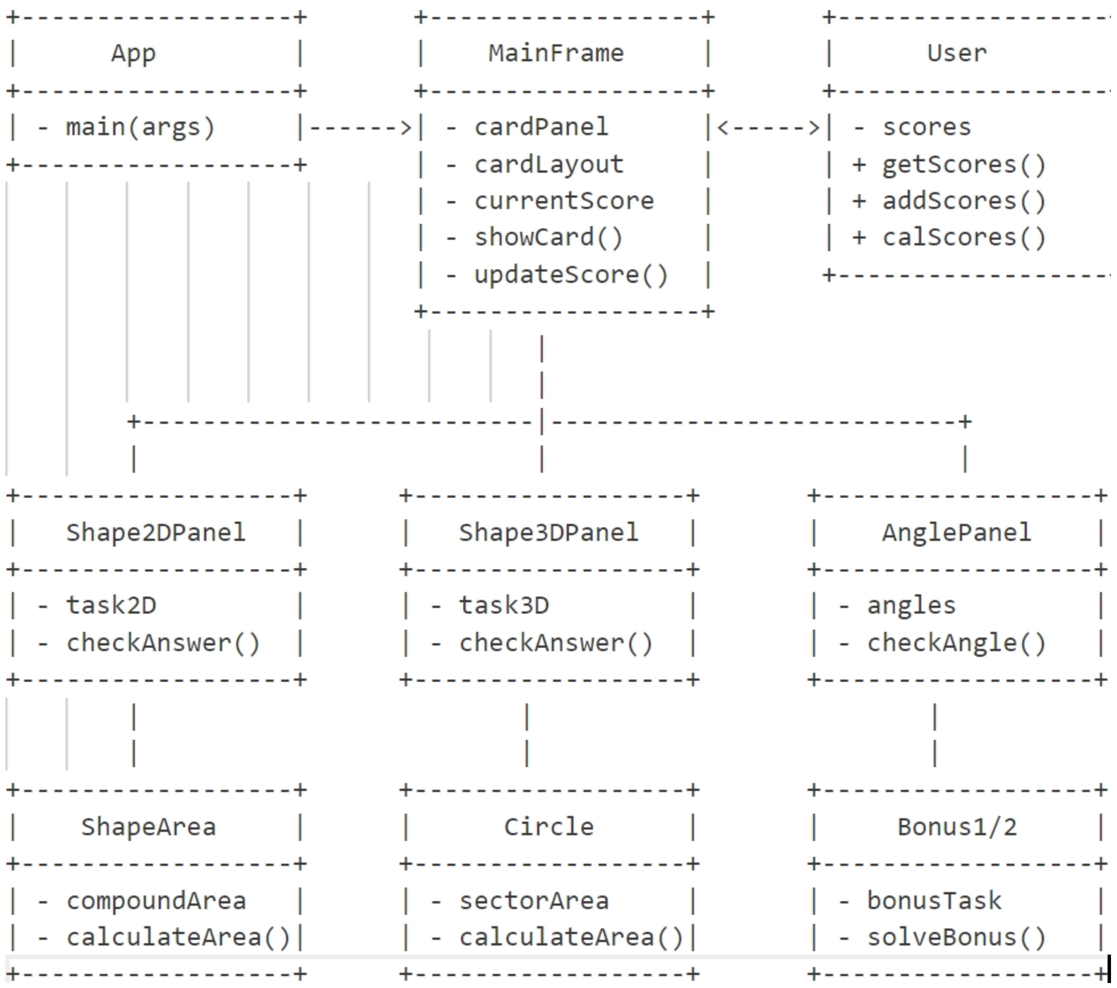
The KS2 phase builds upon KS1 by offering deeper and more specific computations of shape properties, along with additional features.

This flowchart clearly illustrates the modular design of the project and the step-by-step implementation of its functionalities, aiding in the understanding of each stage’s purpose and the overall structure of the project.

7. Summary of the Workflow:

The primary workflow begins with [App.java], transitions through [MainFrame] into the `KS1` phase, where users can choose to work with [Shape2DPanel], [Shape3DPanel], or [AnglePanel]. Following this, users may proceed to the `KS2` phase for more detailed calculations such as shape areas, circle-related operations, or bonus functions. The flow ultimately concludes at `End`.

## 2. 2. Main Classes



```
+------------------+      +------------------+      +------------------
|       App        |      |    MainFrame     |      |       User
+------------------+      +------------------+      +------------------
| - main(args)     |------>| - cardPanel      |<----->| - scores
+------------------+      | - cardLayout     |      | + getScores()
                          | - currentScore   |      | + addScores()
                          | - showCard()     |      | + calScores()
                          | - updateScore()  |      +------------------
                          +------------------+
                                 |
                                 |
        +------------------------|-------------------------+
        |                        |                         |
+------------------+     +------------------+     +------------------+
|   Shape2DPanel   |     |   Shape3DPanel   |     |    AnglePanel    |
+------------------+     +------------------+     +------------------+
| - task2D         |     | - task3D         |     | - angles         |
| - checkAnswer()  |     | - checkAnswer()  |     | - checkAngle()   |
+------------------+     +------------------+     +------------------+
        |                        |                         |
        |                        |                         |
+------------------+     +------------------+     +------------------+
|    ShapeArea     |     |      Circle      |     |     Bonus1/2     |
+------------------+     +------------------+     +------------------+
| - compoundArea   |     | - sectorArea     |     | - bonusTask      |
| - calculateArea()|     | - calculateArea()|     | - solveBonus()   |
+------------------+     +------------------+     +------------------+
```

### 2. 2. 1. Project Structure and Functionality Description

1. App Class

Function: [main(args)]: The entry point of the program, responsible for starting the application.

2. MainFrame Class

Attributes:

[cardPanel]: A card panel used for switching between different task panels.

[cardLayout]: A card layout manager.

[currentScore]: The current score of the user.

Methods:

[showCard()]: Displays a specific task panel.

[updateScore()]: Updates the user's score.

3. User Class

Attributes:

[scores]: A list of the user's scores.

Methods:

[getScores()]: Retrieves the user's scores.

[addScores()]: Adds new scores to the user's score list.

[calScores()]: Calculates the total score of the user.

4. Shape2DPanel Class

Attributes:

`task2D`: A 2D shape task.

Methods:

[checkAnswer()]: Checks if the user's answer to the 2D shape task is correct.

5. Shape3DPanel Class

Attributes:

`task3D`: A 3D shape task.

Methods:

[checkAnswer()]: Checks if the user's answer to the 3D shape task is correct.

6. AnglePanel Class

Attributes:

`angles`: An angle task.

Methods:

`checkAngle()`: Checks if the user's answer to the angle task is correct.

7. ShapeArea Class

Attributes:

`compoundArea`: A compound area.

Methods:

[calculateArea()]: Calculates the area of a shape.

8. Circle Class

Attributes:

`sectorArea`: The area of a sector.

Methods:

[calculateArea()]: Calculates the area of a circle or sector.

9. Bonus1/2 Class

Attributes:

`bonusTask`: A bonus task.

Methods:

`solveBonus()`: Solves the bonus task.

### 2.2.2. Project Workflow Description

1. Application Startup:

   The application starts through the [main(args)] method in the [App] class.

2. Main Interface Display:

   The [MainFrame] class initializes and displays the main interface, which includes a [cardPanel] for switching between different task panels.

3. Task Panel Display:

   Based on the user's selection, the [showCard()]method in [MainFrame] displays the corresponding task panel ([Shape2DPanel], [Shape3DPanel], or [AnglePanel]).

4. Task Execution and Verification:

   The user completes tasks within the task panels.

   The respective task panels ([Shape2DPanel], [Shape3DPanel], or [AnglePanel]) call the appropriate verification methods ([checkAnswer()]or `checkAngle()`) to check the correctness of the user's answers.

5. Score Update:

   If the answer is correct, the [updateScore()]method in [MainFrame] updates the user's score.

6. Score Management:

   The [User] class manages the user's scores, providing methods such as [getScores()], [addScores()], and [calScores()]to retrieve, add, and calculate scores.

7. Additional Features:

   The [ShapeArea] and [Circle] classes provide functionality for calculating the areas of shapes.

   The `Bonus1/2` classes offer additional task-solving features.

Summary:this project is a graphical learning application that tests and enhances users' knowledge of shapes through various task panels (2D shapes, 3D shapes, and angle tasks). The core logic is managed by the [MainFrame] class, while user scores are maintained by the [User] class. Specific task verification and shape calculations are handled by the respective panel classes and service classes. The design is modular with clear responsibilities, making it easy to extend and maintain.

# 3. Design Highlights Analysis

## 3. 1. Adaptive Design for Colorblind Users

### 3. 1. 1. Diverse Color Schemes

The Shapeville application implements three distinct color schemes to accommodate users with different visual abilities:

Standard Vision Mode: Uses conventional color combinations including vibrant blues, cyans, and greens to create an engaging interface for typical users.

Red-Green Colorblind Mode: Specifically designed for deuteranopia/protanopia users, avoiding red-green contrasts and primarily using blue and yellow-orange combinations.

Blue-Yellow Colorblind Mode: Designed for tritanopia users, employing purple and red as primary hues to prevent blue-yellow confusion.

### 3. 1. 2. Adaptive Background Images

The application provides specially designed background images for each color scheme, automatically adjusting based on the user's selected visual mode.

### 3. 1. 3. Semantic Color Definitions

The design utilizes semantic color naming (e.g., "primary", "success") rather than hard-coded color values, enabling seamless theme switching while maintaining consistent information hierarchy across all color schemes.

### 3. 1. 4. Dynamic Color Switching Mechanism

The application features one-click color scheme switching with comprehensive UI synchronization:

Automatic button text updates indicating current scheme

Dynamic background image adjustment

Instantaneous UI element recoloring

### 3. 2. Child-Friendly Interaction Design

#### 3. 2. 1. Specialized Button Components

Custom-designed buttons incorporate multiple child-friendly elements:
Rounded Corners: 20px border radius for soft edges
Playful Typography: Comic Sans MS font for informal appeal
Oversized Targets: Larger touch areas for developing motor skills
Concise Labels: Simple, direct text for easy comprehension

#### 3. 2. 2. Interactive Animation Effects

Engaging micro-interactions enhance child engagement:
Smooth scaling animations on hover
Immediate visual feedback mechanisms
Attention-sustaining motion design

#### 3. 2. 3. Visual Progress Tracking

Clear status visualization helps children understand task completion:
Color-coded task indicators (completed/incomplete)
Achievement reinforcement through visual cues
Progress visualization supporting learning motivation

## 4. User-manual

### 4. 1. Introduction

Shapeville is an interactive educational application designed to help children aged 5-10 learn geometry concepts through engaging activities. The software offers a colorful, accessible environment where young learners can explore 2D shapes, 3D shapes, angles, areas, and more.

### 4. 2. System Requirements

Operating System: Windows 7 or newer, macOS 10.12 or newer, Linux
Java Runtime Environment (JRE) 8 or newer
Minimum Screen Resolution: 1024 x 768
Memory: 256MB RAM minimum

### 4. 3. Getting Started

1. Use IDEs such as vscode, idea, eclipse to run the program "App.java";
   Or you run a terminal in current directory, then run following cmd:
   ```
   java -cp out/production/MiniProject com.geometry.App
   ```
2. The main menu will appear showing two learning paths:
   Key Stage 1 (KS1): Basic shape and angle recognition (for younger children)
   Key Stage 2 (KS2): More advanced concepts like area and perimeter (for older children)

### 4. 4. Application Interface

The Shapeville interface consists of:
Top Area: Shows the "Color Mode" button for accessibility settings
Main Content Area: Displays the current learning activity
Bottom Status Bar: Shows your progress score and task completion status
Navigation Buttons: Appear at the bottom of most screens to help you move between sections

### 4. 5. Accessibility Features

Shapeville is designed to be accessible to all children, including those with color vision deficiencies:

1. Color Schemes: The application provides three different color schemes that can be toggled using the "Color Mode" button:
   Normal Vision: Standard color palette with vibrant colors
   Red-Green Colorblind: Special color scheme for those with red-green color vision deficiency
   Blue-Yellow Colorblind: Adapted color scheme for blue-yellow color vision

deficiency

2. Each time you press the "Color Mode" button, the application will cycle to the next color scheme.

## 4. 6. Learning Modules

### Key Stage 1 (KS1)

**2D Shapes**

View and learn various 2D shapes

Identify shapes shown on screen by typing their names

Receive instant feedback on your answers

You have 3 attempts for each shape

**3D Shapes**

Learn about three-dimensional shapes and their properties

Identify 3D shapes from their images

Similar to 2D shapes, you have 3 attempts for each answer

**Angles**

Learn different types of angles (acute, right, obtuse, straight, reflex)

Practice identifying angle types

Interactive angle visualizations help reinforce concepts

### Key Stage 2 (KS2)

**Shape Area**

Learn how to calculate areas of various shapes (rectangles, triangles, parallelograms, trapeziums)

Practice solving area problems with randomly generated parameters

Visual representations help understand area formulas

**Circle**

Learn about circle properties, including area and circumference

Practice calculations using both radius and diameter

Visual illustrations demonstrate the relationship between dimensions and calculations

### Bonus Modules

**Compound Area**: Calculate areas of complex shapes made from multiple simple shapes

**Sector Area**: Learn about circle sectors and how to calculate their areas

## 4.7. Scoring System

Shapeville tracks your progress with a points-based scoring system:

Each correct answer earns points based on:

- Task difficulty (Basic or Advanced)
- Number of attempts used (fewer attempts = more points)

Basic level activities award 1-3 points per correct answer

Advanced level activities award 2-6 points per correct answer

Your current score appears in the progress bar at the bottom of the screen

Task completion status is shown with colored indicators

## 4.8. Tips for Teachers/Parents

1. Start with KS1: For young children (5-7), begin with the Key Stage 1 modules
2. Progression: Move to Key Stage 2 once the child is comfortable with shape recognition
3. Supervision: Younger children may need assistance with typing answers
4. Explanations: Use the visual representations as teaching aids to explain concepts
5. Accessibility: Adjust the color scheme if your child has any color vision deficiency

## 4.9. Ending a Session

1. Click the "End" button in the bottom right corner of the screen
2. A summary dialog will show the total points earned during the session
3. Click "OK" or press the "Enter" to close the application

## 4.10. Troubleshooting

Application won't start: Ensure you have Java installed on your computer

Text appears cut off: Try resizing the window to see if it helps

Colors don't match descriptions: Try cycling through the color modes using the "Color Mode" button

Thank you for using Shapeville to explore the world of geometry with your young learners!

# 5. Utilisation of AI

## 5. 1. Colorblind-Friendly Color System Implementation

During the development of the Shapeville geometry learning application, AI-assisted tools played a crucial role in designing and implementing a comprehensive colorblind adaptation system. Through interaction with AI tools, I was able to:

Research colorblindness types and characteristics: AI helped me understand the visual characteristics of red-green colorblindness (most common) and blue-yellow colorblindness, providing scientific basis for the adaptation design.

Generate color combinations: Based on AI-provided colorblind-friendly color theory, I implemented three complete color schemes:

Blue as a substitute for green in red-green colorblind mode

Yellow-orange for warning indicators

Purple as primary color and cyan for success indicators in blue-yellow colorblind mode

Verify color contrast: AI tools assisted in verifying the contrast and distinguishability of each color scheme, ensuring clear differentiation of interface elements for different colorblind users.

Implement dynamic switching mechanism: With AI suggestions, I created a semantic color reference system enabling seamless switching between different color schemes.

AI tools significantly enhanced my understanding of colorblind users' needs and helped design a professional color adaptation system that would otherwise require extensive specialized research and testing.

## 5. 2. Java Drawing Functionality (Geometric Visualization)

For the geometric visualization module, AI tools notably improved development efficiency and code quality:

Complex drawing algorithms: AI assisted in implementing sophisticated Java 2D drawing code for various shapes including parallelograms, triangles, rectangles, and trapezoids.

Mathematical formula visualization: AI tools helped design visual representations of area calculation formulas, including:

Fractional notation

Arrow markers

Dynamic calculation results

Making teaching effects more intuitive.

Coordinate transformation and scaling: AI provided algorithms for handling different scale values and coordinate transformations, ensuring correct display of shapes under various parameters.

Color coordination and consistency: AI assisted in creating drawing schemes that maintain recognizability across different color modes, ensuring graphic elements remain clear in colorblind modes.

### 5. 3. Python Angle Generation Script

For the angle learning module, AI tools assisted in developing a Python script for batch angle image generation:

Matplotlib drawing code generation: AI helped write Python code using Matplotlib library to create precise angle visualizations.

Special angle handling: AI provided specialized visualization code for different angle types (acute, right, obtuse, straight, and reflex angles), each with unique representations.

Batch generation and file management: AI implemented looping structures and filesystem operations for batch image generation, enabling easy creation of 36 angle images from 0° to 350°.

Visual consistency: AI ensured all generated angle images maintained consistent visual style and educational elements, unifying the application's overall design language.

AI-assisted tools played a pivotal role in the Shapeville project, particularly in three core aspects:

The use of these tools not only shortened the development cycle but also notably improved the application's accessibility and educational effectiveness, enabling Shapeville to better meet the geometry learning needs of children aged 5-10. AI input proved essential for achieving the project's educational goals and inclusive design principles.

# 6. Reflection on your experience

## 6. 1. My First Java GUI Project Experience

At first, this project really surprised me because it wasn’t a typical CRUD (Create, Read, Update, Delete) project like I expected. I felt a little lost at the beginning, but after carefully reading the Product Requirements Document, I started building the project structure. Instead of using the MVC (Model-View-Controller) architecture common in web projects, I went with a simpler three-layer design:

Data Layer – Handles user data, question images, and answers.
Service Layer – Processes the core functions.
UI Layer – Manages the GUI (Graphical User Interface).

## 6. 2. What Went Well

AI is advancing so fast! For this project, I used Cursor (an AI coding assistant) and Claude 3.7 Sonnet (a powerful AI model). They helped me with a lot of things—from project structure to solving tricky problems. Sometimes, AI gave me amazing solutions, but other times, it was a bit unpredictable. Still, I learned a lot about AI-assisted programming.

I also gained real-world experience with GUI programming. Before, I only knew about events and listeners from PowerPoint slides. Now, I actually used .addEventListener() with anonymous inner classes and lambda functions. It turns out, Java GUI has a lot in common with front-end web development (HTML, CSS, JS)—just different syntax, but the core ideas are similar!

## 6. 3. What Could Be Improved

High Coupling Between Layers.At first, I didn’t design interfaces properly, so different parts of the project depended too much on each other. Next time, I’ll use interfaces to reduce dependencies.

Manual Testing (No Unit Tests). I tested everything by hand, running the program over and over. That’s not efficient! In future projects, I’ll learn unit testing (like JUnit) to make sure my code works correctly before running the whole program.

# 7. Appendix(Prompts and Answers)

## 7. 1. Colorblind-Friendly Color System Implementation

### Prompt

I want to add a color mode switcher to my app so people with color blindness can use it easily. But first, I need to know: What types of color blindness are most common?

### Answer

Adding Color Mode Switching to Shapeville App

I want to explore how to add color mode options to Shapeville. First, let's learn about different types of color blindness.

Current App Colors:

Background: Light blue (rgb 177,208,239)

Buttons: Blue shades (rgb 52,119,219 and 81,222,232)

Hover colors: Orange (255,140,0) and pink (255,140,150)

Shape colors: Light red, green, blue, and yellow

Color Blindness Mode Design:

1. Supported Types:

We'll support 3 main types:

Red-green colorblind (most common - affects ~8% of boys/0.5% of girls)

Includes: Protanopia (red weakness) & Deuteranopia (green weakness)

Blue-yellow colorblind (less common - Tritanopia)

Full colorblind (very rare - needs high contrast mode)

2. Switching Button:

Add a color mode button/menu in the main screen

3. Special Color Settings:

Create different color rules for each type

4. How Switching Works:

Add a color change method in the main code

5. Shape Helpers:

Add patterns/textures to shapes to make them easier to tell apart

6. Button Helpers:

Put small shape icons on buttons too

7. Remember Preferences:

Save each user's favorite color mode when they start the app

Why This Works:

✓ Doesn't just use color - adds shapes/patterns too

✓ Special colors for each type of color blindness

✓ Easy-to-use switch button

✓ Makes shapes clearer with extra markers

✓ Remembers user choices

✓ Keeps the app looking good while helping everyone

This design makes Shapeville better for colorblind kids learning geometry, while keeping it fun and easy to use for everyone!

### Note

I ultimately implemented the color mode switching solution mentioned above, adopting five recommendations from the answer: 1, 2, 3, 4, and 7. The color schemes correspond to three modes: normal + red-green colorblind + blue-yellow colorblind.

## 7. 2. Java Drawing Functionality (Geometric Visualization)

### Prompt

According to the interaction requirements, I need to display many images in the GUI, such as this parallelogram area calculation diagram. There are two issues: First, how should I draw these diagrams (I'm not very familiar with drawing)? Second, how should I store these images in advance and then display them in the GUI?

### Answer

For displaying the parallelogram area calculation diagram and other shapes, there are two solutions:

Solution 1: Using Java drawing functionality

Graphic drawing

/**

Parallelogram class

```
*/
public class Parallelogram extends Shape {
@Override
public void draw(Graphics g, int x, int y, int width, int height) {
// Set offset to determine the slant of the parallelogram
int offset = width / 4;
  int[] xPoints = {x + offset, x + width + offset, x + width, x};
  int[] yPoints = {y, y, y + height, y + height};
  // Draw parallelogram outline
  g.drawPolygon(xPoints, yPoints, 4);
  // Use different colors to mark base and height
  Graphics2D g2d = (Graphics2D) g;
  // Save original color
  Color originalColor = g2d.getColor();
  // Draw base (yellow)
  g2d.setColor(Color.ORANGE);
  g2d.setStroke(new BasicStroke(2));
  g2d.drawLine(x, y + height, x + width, y + height);
  g2d.drawString("Base", x + width/2 - 15, y + height + 15);
  // Draw height (purple)
  g2d.setColor(new Color(148, 0, 211)); // Purple
  g2d.drawLine(x + width/2, y, x + width/2, y + height);
  g2d.drawString("Height", x + width/2 + 5, y + height/2);
  // Restore original color
  g2d.setColor(originalColor);
  // Draw area formula
  g2d.drawString("Area = Base × Height", x + width + 20, y + height/2);
}
}
```

Solution 2: Using pre-prepared images

Image storage structure

```
src/
 └─── resources/
 └─── images/
 ├─── shapes/
```

```
│   ├── circle.png
│   ├── triangle.png
│   ├── square.png
│   ├── rectangle.png
│   └── parallelogram.png
├── angles/
│   ├── acute_angle.png
│   ├── right_angle.png
│   └── obtuse_angle.png
└── formulas/
├── circle_area.png
├── triangle_area.png
├── rectangle_area.png
└── parallelogram_area.png
```

Image loading and display

```
/**
Image loader utility class
*/
public class ImageLoader {
/**
Load image
@param path Image path
@return Image object
*/
public static BufferedImage loadImage(String path) {
BufferedImage img = null;
try {
img = ImageIO.read(ImageLoader.class.getResourceAsStream(path));
} catch (IOException e) {
System.err.println("Failed to load image: " + path);
e.printStackTrace();
}
return img;
}
}
```

```
/**
Shape image panel
*/
public class ShapeImagePanel extends JPanel {
private BufferedImage image;
private String description;
public ShapeImagePanel(String imagePath, String description) {
this.image = ImageLoader.loadImage(imagePath);
this.description = description;
setPreferredSize(new Dimension(400, 300));
}
@Override
protected void paintComponent(Graphics g) {
super.paintComponent(g);
// Draw image
if (image != null) {
g.drawImage(image, 10, 10, getWidth() - 20, getHeight() - 60, this);
}
  // Draw description text
  g.setFont(new Font("SimSun", Font.PLAIN, 14));
  g.setColor(Color.BLACK);
  g.drawString(description, 10, getHeight() - 30);
}
}
```

## Note

For tasks 3 and 4, I ultimately adopted the first solution mentioned in the answer - using Java's java.awt.geom.Line2D and java.awt.geom.Path2D to dynamically generate required diagrams based on user input. The system draws real-time graphics that meet the requirements and displays both the answers and relevant explanations directly on the diagrams.Since I'm not very experienced with drawing graphics, and AI excels at this kind of front-end UI coding, I used AI assistance to generate these image-rendering codes. The AI produced the initial code, which I then continuously reviewed and fine-tuned to meet our specific requirements.

## 7. 3. Python Angle Generation Script

### Prompt

Now write a simple ipynb script using matplotlib to batch generate all angle images (@angle.ipynb). The style and appearance should match what's shown in the image, but without any text - only the angles need to be included.

### Answer

Let me create a regular Python file instead, which you can then copy into a Jupyter Notebook：

```
import matplotlib.pyplot as plt
import numpy as np
import os
import math
from matplotlib.patches import Arc, Rectangle, FancyArrowPatch, Circle
# Create output directory
output_dir = 'angles'
if not os.path.exists(output_dir):
    os.makedirs(output_dir)
def create_angle_image(angle_degrees, filename, figsize=(5, 5), dpi=100):
    """Create and save angle image
    Parameters:
        angle_degrees: Angle value (0-360)
        filename: Output filename
        figsize: Figure size
        dpi: Resolution
    """
    # Convert angle to radians
    angle_rad = math.radians(angle_degrees)
    # Create figure
    fig, ax = plt.subplots(figsize=figsize)
    # Set axis limits
    ax.set_xlim(-1.5, 1.5)
    ax.set_ylim(-1.5, 1.5)
    # Hide axes
    ax.axis('off')
    # Set background color - white
    ax.set_facecolor('white')
    # First line segment is always horizontal reference line
    arrow_length = 1.0
    arrow_props = dict(arrowstyle='->', linewidth=2, color='#4682b4')
    # Determine angle type and draw accordingly
    if angle_degrees == 0 or angle_degrees == 360:  # 0 or 360 degrees
        # Horizontal right line
        arrow = FancyArrowPatch((0, 0), (arrow_length, 0), **arrow_props)
        ax.add_patch(arrow)
    elif angle_degrees == 90:  # Right angle
        # Horizontal right line
        arrow1 = FancyArrowPatch((0, 0), (arrow_length, 0), **arrow_props)
        # Vertical up line
        arrow2 = FancyArrowPatch((0, 0), (0, arrow_length), **arrow_props)
        ax.add_patch(arrow1)
        ax.add_patch(arrow2)
        # Add right angle symbol
        rect = Rectangle((0, 0), 0.15, 0.15, linewidth=0, edgecolor='none', facecolor='#b0d0e0')
        ax.add_patch(rect)
    elif angle_degrees == 180:  # Straight angle
        # Horizontal right line
        arrow1 = FancyArrowPatch((0, 0), (arrow_length, 0), **arrow_props)
        # Horizontal left line
        arrow2 = FancyArrowPatch((0, 0), (-arrow_length, 0), **arrow_props)
        ax.add_patch(arrow1)
        ax.add_patch(arrow2)
        # Add midpoint marker
        circle = Circle((0, 0), 0.05, facecolor='#4682b4')
        ax.add_patch(circle)
        # Add angle arc
        arc = Arc((0, 0), 0.4, 0.4, theta1=0, theta2=180, linewidth=1.5, color='#4682b4')
```

```
            ax.add_patch(arc)
        elif 0 < angle_degrees < 90:    # Acute angle
            # Horizontal right line
            arrow1 = FancyArrowPatch((0, 0), (arrow_length, 0), **arrow_props)
            # Slanted line
            end_x = arrow_length * math.cos(angle_rad)
            end_y = arrow_length * math.sin(angle_rad)
            arrow2 = FancyArrowPatch((0, 0), (end_x, end_y), **arrow_props)
            ax.add_patch(arrow1)
            ax.add_patch(arrow2)
            # Add angle arc
            arc = Arc((0, 0), 0.4, 0.4, theta1=0, theta2=angle_degrees, linewidth=0,
facecolor='#b0d0e0', alpha=0.7)
            ax.add_patch(arc)
        elif 90 < angle_degrees < 180:    # Obtuse angle
            # Horizontal right line
            arrow1 = FancyArrowPatch((0, 0), (arrow_length, 0), **arrow_props)
            # Slanted line
            end_x = arrow_length * math.cos(angle_rad)
            end_y = arrow_length * math.sin(angle_rad)
            arrow2 = FancyArrowPatch((0, 0), (end_x, end_y), **arrow_props)
            ax.add_patch(arrow1)
            ax.add_patch(arrow2)
            # Add angle arc
            arc = Arc((0, 0), 0.4, 0.4, theta1=0, theta2=angle_degrees, linewidth=0,
facecolor='#b0d0e0', alpha=0.7)
            ax.add_patch(arc)
        elif 180 < angle_degrees < 360:    # Reflex angle
            # Horizontal right line
            arrow1 = FancyArrowPatch((0, 0), (arrow_length, 0), **arrow_props)
            # Slanted line
            end_x = arrow_length * math.cos(angle_rad)
            end_y = arrow_length * math.sin(angle_rad)
            arrow2 = FancyArrowPatch((0, 0), (end_x, end_y), **arrow_props)
            ax.add_patch(arrow1)
            ax.add_patch(arrow2)
            # Add angle arc
            arc = Arc((0, 0), 0.4, 0.4, theta1=0, theta2=angle_degrees, linewidth=0, facecolor='#b0d0e0', alpha=0.7)
            ax.add_patch(arc)
        # Save image
        plt.savefig(filename, dpi=dpi, bbox_inches='tight', transparent=False)
        plt.close(fig)
    # Generate 36 angle images (0-350 degrees, every 10 degrees)
    for degree in range(0, 360, 10):
        filename = f"{output_dir}/angle_{degree}.png"
        create_angle_image(degree, filename)
        print(f"Generated {degree} degree angle image")
    print("\nAll angle images generated!")
```

## Note

For the task of generating 35 angle images from 0 to 360 degrees at 10-degree intervals, the AI was able to accomplish it effortlessly. Therefore, after reviewing and running this Python script, I made almost no modifications and directly applied the generated images in the program.