IMPROVE INTERACTIVE PATTERN MATCHING SKILLS USING DYNAMIC PUZZLES

Harshana W.C.

IT21175466

BSc (Hons) in Information Technology Specializing in Information Technology

Department of Information Technology

Sri Lanka Institute of Information Technology
Sri Lanka

4/11/2025

DECLARATION

“I declare that this is my own work and this dissertation does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to Sri Lanka Institute of Information Technology, the nonexclusive right to reproduce and distribute my dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Date: 2025/04/11

ABSTRACT

Children with Nonverbal Learning Disability (NVLD) often face significant challenges in interpreting visual-spatial information, understanding object relationships, and applying nonverbal reasoning. To address these learning barriers, this project introduces a mobile-based, puzzle-driven application specifically designed to enhance visual-spatial skills in children aged 10–13 with NVLD. The system integrates AI-assisted puzzle generation, real-time performance tracking, and dynamic difficulty adjustment to ensure that each user experiences a learning environment tailored to their ability level.

The application allows users to either capture images or generate them digitally, which are then split into puzzle segments with varying complexity based on past performance data. The module tracks key performance metrics—such as accuracy, completion time, and hint usage—to automatically adapt the next puzzle's structure. Children receive immediate visual feedback through an intuitive drag-and-drop interface, supported by gamified rewards and hints to maintain engagement.

Testing and evaluation with a small user group revealed improvements in accuracy, reduced hint reliance, and faster puzzle completion times over multiple sessions. The project demonstrates how an adaptive, puzzle-based learning platform can play a meaningful role in developing cognitive and spatial reasoning skills in NVLD learners. Future work will focus on scaling the system, deepening analytics, and integrating collaborative features to enhance learning outcomes further.

Keywords:

NVLD, Puzzle-Based Learning, Adaptive Difficulty, Visual-Spatial Skills, Mobile Application, Educational Technology, Cognitive Development, Special Education, Gamification, Human-Computer Interaction

ACKNOWLEDGMENT

I would like to extend my heartfelt gratitude to my supervisor, Mrs. Lokesha Weerasinghe, and co-supervisor, Ms. Malithi Nawarathne, for their unwavering support and guidance throughout my undergraduate research. As this project addresses Nonverbal Learning Disorder (NVLD) in children aged 10–13, the insights and expertise from both the fields of technology and developmental disorders were crucial.

My sincere appreciation goes to External Supervisor Ms. Shalindi Pandithakoralage and social worker Ms. Sathushika Fernando for their invaluable support in bridging these fields, helping me gain a deeper understanding of NVLD and its impact on children. I am also deeply grateful to the doctors, and other resource people who generously shared their knowledge and experiences. A special thanks to the children who actively participated in the activities, contributing valuable insights that enriched this project.

Finally, I would like to express my gratitude to everyone who contributed to this project, directly or indirectly — my teammates, family, and friends — whose encouragement and support made this journey possible.

Table of Contents

1. INTRODUCTION

Nonverbal Learning Disability (NVLD) is a developmental disorder marked by below average visual-spatial reasoning and fine motor coordination skills but significantly strong verbal skills. NVLD makes cognitive processing difficult for children, which impacts their ability to navigate the spatial world as well as to understand social clues and academic subjects that require cognitive learning. Even though these children have good verbal comprehension, they struggle in pattern recognition, spatial awareness, and motor coordination skills into their daily life—obstacles that can have a major impact in academic success and overall cognitive development.

To address these challenges, our larger project seeks to develop an all-in-one mobile application that targets core cognitive, pattern recognition and motor coordination skills in children with NVLD. Our team of four members is undertaking a project that includes four parts which are separate yet interlinked. These elements collectively serve as an integrated application capable of providing adaptive, interactive, and developmentally appropriate exercises.

In Dynamic Puzzle Based Module uses AI-generated images and captures images from mobile camera and make puzzles using those images with real-time difficulty adjustments to develop pattern recognition skills and problem-solving strategies with special attention to children with NVLD. Children are encouraged to drag and drop and make the complete image. According to children's performance, an advanced python algorithm will adjust the image splitting pattern and generate image prompt to create puzzle images with missing parts.

1.1. Scope of the Report

This document primarily focuses on the technical, pedagogical, and design aspects of the puzzle-based module.

It includes:

• **Background & Literature Review**: A summary of the previous work on NVLD targeted interventions, puzzle-based education, and adaptive gamification.

• **System Requirements & Design**: Detailed puzzle mechanics specifications, initial user interface designs and data models for tracking performance.

• **Methodology**: A step-by-step overview on the implementation of AI based difficulty modification alongside the tools and frameworks (Python based logic, Flutter front-end, etc.) used.

• **Testing & Evaluation**: Output from user trials or simulations—accuracy improvements, error rate reductions, and levels of engagement—contrasting baseline results against final outputs.

• **Discussion & Limitations**: Reflections on potential refinements, constraints, and ethical considerations in designing puzzle tasks for children with NVLD.

• **Conclusion**: Summary of findings together with suggestions for continuous iterations and potential extensions.

1.1.1. Structure and Relation to the other Components

This module handles visual-spatial and pattern recognition skills but can also combine with the larger ecosystem. We have a finite amount of data on user progress, e.g. time to solve puzzles, hints used, DDA and so on, which we share

across the whole platform. It models the speed at which a learner can complete horizontal arrangement puzzles (indicative of spatial tasks) integrated with an architecture whereby if a learner's spatial task is troublesome and they slow down significantly, the app itself can adjust all other activities by introducing complementing exercises or adjusting the difficulty of reading/vocabulary tasks accordingly.

1.2 Background and Literature Review

Nonverbal Learning Disability (NVLD) is a cognitive processing disruption characterized by a marked discrepancy between strong verbal skills and impaired visual-spatial processing, motor coordination, and multitasking. This presents challenges in daily situations, including reading people's body language, understanding math concepts, information in school tasks, and spatial relationships (e.g., shapes, distance, orientation of an object, etc.). However, in traditional classroom settings, instructional approaches often do not accommodate the specialized supports needed to enable children with NVLD to absorb and apply visual or spatial concepts, given the typical emphasis on verbal and written forms of instruction.

Fortunately, the primary objective of this project is to help mitigate those gaps, via Dynamic Difficulty Adjustment (DDA) and age-based User Interface representations. This can be done with dynamic difficulty adjustment: monitoring user performance in real time, tracking the correct and incorrect moves, the time spent completing tasks, the usage of hints, etc. and recalibrating automatically the complexity of the task based on the learner's current needs. This method maintains tasks at an intermediate level to avoid them being overly easy or too difficult and allows for continued engagement and enhanced skill acquisition, which in turn supports learning.

1.2.1. The Impact of Dynamic Difficulty Adjustment (DDA) on Learning

Brown et al [1]. This real-time adaptation of games to a user's skill level has proven to lead to increased levels of engagement and decreased levels of frustration, as shown by those who presented early concepts of realizing DDA in the context of educational games. Garrison et al [4]. who furthered this concept, emphasizing how adaptive systems can assist with tailored feedback, providing learners immediate response on the correctness of answers, as well as aspects that require attention. Yet, the majority focus on generalized reading or math skills rather than Edu corner behavioral-focused and visual-spatial interventions aimed at children with NVLD.

DDA is an integral component pertaining to customization of puzzle-based activities for users with Nonverbal Learning Disability in the system continually weighs correct/incorrect placements and other utilized hints while dynamically calibrating puzzle difficulties up or down as needed. For instance, if the child struggles with these splitting sectors in piles they've multiple hints and make too many mistakes the puzzle intelligently downsizes its splits sections or makes easier patterns. In turn, children who execute this work accurately and efficiently are given extra, more complex tasks, for example, fitting pieces into an irregular-shaped puzzle, or handling additional pieces.

1.2.2. Importance of Age-Appropriate Interface Design

Children aged approximately 10 to 13 years represent a key developmental period when more sophisticated reading tasks, math concepts, and problem-solving skills take hold (Lee et al. [2]). Cognitive Load, Navigational Ease, and Visual Clarity (Williams et al. [3]). For students with NVLD, and for visual or spatial overloaded learners, a clutter screen can throw learning off track significantly. This implies interface elements need to be instinctive: clear symbols or short content or drag-and-drop for puzzles. Everything needs to be as smooth as you can in the way in which

students can concentrate on understanding and training spatial concepts with getting diverted or confused by the app's framework.

It also stresses wisely mixing gamification elements (e.g. helping alert dialogs, scores or hints) while performing learning activities to prevent children from excessive stimulation and aligning them with key learning goals. For example, hints in puzzle tasks can be configured to show as tiny pop-ups with minimal text and one explanatory icon. It ensures that children respect how far their eyes can go while also ensuring that they will have different designs that mimic their interests.

1.2.3. Puzzle-Based Approaches for NVLD Interventions

Research increasingly points to the effectiveness of puzzle-based exercises for children with NVLD. Such exercises directly target visual-spatial reasoning—an area of consistent weakness for these learners (Williams et al. [3]). Puzzles provided opportunities for hands-on experimentation, immediate feedback, and repeated practice in spatial manipulation, unlike static worksheets. Puzzle tasks can dramatically increase a learner's intrinsic motivation by providing real-time challenges in a game-like environment that compels the learner to tackle persistent obstacles in the learning process.

As explained, puzzle-based techniques become even more powerful when combined with dynamic difficulty. Children remain within a zone of "optimal challenge": not so easy they become bored, but not so hard they feel discouraged. That formative, real-time recalibrating nurtures particularly large gains in spatial awareness, pattern recognition and logical problem-solving skills that might also transfer to navigating physical space more skillfully or analyzing visual data in science or math.

1.2.4. Purpose of the Image-Based Puzzle Game Visual-Spatial Module

This report's main goal is to elucidate the image-based spatial skills and pattern recognition skills development element of the puzzle, which utilizes advanced AI Image manipulation to support children with NVLD and improve their spatial skills. In particular, this module creates puzzles whose level of difficulty (e.g., number of pieces, difficulty level, image prompt) adapts dynamically to each child's performance in real time.

Key features include:

• Dynamic Difficulty Adjustment (DDA) based on user performance metrics (correct/incorrect placements, hint usage, and time-to-complete, auto fill usage, score, image prompt, difficulty level) to make sure tasks are suitably challenging, but not too frustrating.

• Strengthening Visual-Spatial Concepts: Object location (Where should the image piece need to be placed, above, below, left, right), pattern completion, and motor coordination.

• Personalized Engagement: Usages of gaming elements (difficulty level and auto filling helping system, attractive sound clips and colorful user interface usually used in game development to engage users) that valuable for them over time, path recognition and journey implementations and system in which people can receive bonuses or rewards for each step they learn.

By creating a system that calculates and adapts puzzles based on a child's ability, we've essentially provided an education method which recognizes the variances in how children learn similar guiding solutions were previously applied to our complete app.

1.3. Expected Outcomes

This dynamic difficulty adjustment puzzle component aims to: - Create a progressive and adaptive puzzle experience for the child and help to improve spatial thinking level of the child and help parents or doctors to understand about NVLD level of the child.

- Enhance Spatial Reasoning: Aid a child in mentally moving shapes around, seeing the split pattern between various image pieces and whole image.
- Problem solving and Self helping decision: The hint system tracks the child's progress when doing drag and drop activity. If a child gets stuck, the system automatically asks for some help. Child can decide at what time I want some help.
- Facilitate Self-Efficacy: A child's strong belief that he/she can overcome challenges that are traditionally perceived as complex/unachievable ensures that the child's learning is based on intrinsic motivation.
- Hybrid Intervention Strategies: Could gather data around NVLD learners' response to incremental spatial challenges to potentially inform some combination of occupational or therapeutic intervention strategies.

1.4. Research Gap

While multiple studies confirm the importance of adaptive learning (Brown et al. [1]) and age-tailored interfaces (Lee et al. [2]; Williams et al. [3]), but none fully integrate these strengths into a puzzle form specifically for NVLD children. Existing research tends to focus on learning disabilities more generally, or on partial aspects of adaptivity such as dynamic difficulty adjustment or the presence of a child-friendly user interface rather than integrating the elements into a single holistic educational tool.

1.4.1. Comparison of Existing Research with Proposed Puzzle-Based System

Features / Focus	Research 1 (Brown et al. [1])	Research 2 (Lee et al. [2])	Research 3 (Williams et al. [3])	Proposed Puzzle-Based System
Performance Based Dynamic Difficulty Adjustment (DDA)	✗	✗	✗	✓
Age-Specific Interface Design impacts engagement and learning	✗	✓	✓	✓
Focus Educational applications and effectiveness of real time adjustment in image tasks.	✓	✗	✓	✓
Focus on strategies designed to attract and sustain interest in cognitive exercises for children.	✗	✗	✓	✓
Auto filling Helping System	✗	✗	✗	✓
Mobile App	✗	✗	✗	✓

1. Brown et al. [1] – Established the advantages of dynamic difficulty but did not address NVLD or puzzle-specific designs.

2. Lee et al. [2] – Explored age-specific UI considerations but lacked deeper adaptive feedback loops.

3. Williams et al. [3] – Investigated adaptive interfaces for special needs children; however, visual-spatial puzzle tasks were not a focus.

4. Proposed Puzzle-Based System – Merges Dynamic Difficulty Adjustment (DDA), auto fill helping system, puzzle-based learning, and an age-appropriate interface while specifically targeting NVLD children and improve their thinking patterns.

1.4.2. Summary of Identified Gaps

Limited NVLD-Specific Solutions: Many tools lack the fine-grained tailoring necessary for NVLD learners, who need explicit support in visual-spatial tasks and real-time adjustments.

Inadequate Integration of DDA and Puzzles: While DDA has been studied extensively, synergy with Real time capture image and create puzzles, AI-driven puzzle generation and drag-and-drop mechanics remains underexplored.

3. Minimal Emphasis on Age-Appropriate, Gamified UI: Although Lee et al. [2] and Williams et al. [3] propose user-friendly designs, they do not address NVLD learners' heightened sensitivity to cluttered or visually dense interfaces.

4. Lack of Real-Time Tracking and auto filling system: Garrison et al. [4] highlights real-time adaptation but seldom involves interactive puzzle auto filling helping system that modifies difficulty on-the-fly.

Bridging these gaps, the proposed puzzle-based system integrates dynamic difficulty adjustment, auto filling helping system, age-appropriate user interface, and gamified puzzle-solving to offer a holistic learning environment for children with NVLD. By leveraging insights on real-time difficulty tuning and puzzle generation, this approach aims to accelerate visual-spatial skill development while maintaining a

child-friendly, rewarding experience. The subsequent sections detail how these concepts are designed, implemented, and evaluated within the overall application.

1.5. Importance and Gap in Research

While many educational apps target more general reading or mathematics deficits, very few target the spatial understanding deficits that often define NVLD. Children with NVLD typically show a gap between their verbal abilities (which might emerge early and stutter to a quick advance) and their spatial reasoning (which trails behind), so "one-size-fits-all" solutions are usually not successful. Standard puzzles or game-based interventions typically do not:

1. Adjust puzzle complexity in real time based on children performance data and hint and helping system usages.

2. Integrate puzzle-solving with explicit NVLD-friendly supports (for example, hints, partial auto-fills, or integrated reinforcement strategies).

3. Provide structured data collection to track progression in visual-spatial skills over time.

4. Capture real time camera images and split them accordingly puzzle levels, then make puzzles from those images real time.

To mitigate these gaps, this initiative integrates AI-Based Puzzle Generation and real-time analytics to create a personalized, on-demand approach that naturally accommodates a child's individual learning process.

2. OBJECTIVES

2.1. Main Objective

The primary objective of this component—derived from the document—is to create an interactive and personalized puzzle-based gamming learning module for children aged 10–13 with Nonverbal Learning Disability (NVLD), which integrates:

- Photos taken real-time by mobile phone camera or generate child friendly images using AI.
- Automatically split images into puzzle parts according to children's performance.
- Dynamic difficulty adjustments based on user performance metrics by an advanced python algorithm.

The goal of this module is to promote visual-spatial reasoning, develop pattern recognition skills, and increase overall cognitive involvement in those children who normally have a less good performance in nonverbal tasks.

2.2. Specific Objectives

1. Generate or pick real-time image

• Introduce a feature to choose an option from Generate image from AI or pick a real-time image from a mobile phone's camera.

2. Image Splitting into a puzzle

• Create feature split image into similar pieces (Ex- 2x2, 3x3, 4x4) in Level 1 and, Split images into random non-similar sized images (1x2, 3,5, 2,3) in Level 2.

• Create algorithms that divide images into their corresponding missing components for drag and drop use for NVLD to keep interaction age-appropriate and clear for NVLD children.

3. Adaptive Difficulty Adjustment (Back-End Logic)

• Implement a Python-based advanced algorithm that dynamically adjusts the complexity of the puzzle (number of segments, different sizes, image prompt) in real-time.

• Monitor correct moves, wrong moves, and hints used to dynamically increase/decrease task difficulty on-the-fly to avoid either boredom or frustration.

4. User Interface and Gamification

• For the target age group (10–13), structural guidelines for a child-friendly interface thus incorporate large buttons, intuitive navigation, and self-help prompts.

• Add gamification elements (scores, sound clips, cool animations) to encourage students to keep completing our puzzles while avoiding complex UX in their apps for neurodiverse learners with NVLD.

5. Real-Time Feedback and Hint System

• Display on-screen feedback whenever the child drags and drops a puzzle piece (correct placement, incorrect placement), similar to ’s autofill or hint logic.

• Enable an option for hint and auto filling that helps a child complete some parts of the puzzle if he/she gets stuck while recording all hint triggers for adapting difficulty.

6. Performance Data Capture and Reporting

• Record the completion times, accuracy and hint usage of each user's puzzle, using the structured format proposed in, for subsequent analysis and improvement of the system.

• To provide analysis/visualization of student performance across multiple sessions, within the framework of an integrated design approach across team modules by generating charts and generating PDFs.

7. Evaluation and Iterative Improvement

• Collect qualitative feedback related to puzzle difficulty, user interface comprehensibility, and overall engagement from children, parents, or therapists, and adjust the component accordingly.

3. METHODOLOGY

This part describes the general procedure used to develop an interactive puzzle-based module to improve the visual-spatial and cognitive skills for 10–13 years old children with Nonverbal Learning Disability (NVLD). The approach includes system architecture, AI mechanics, data processing, and implementation information that direct puzzle-generation, difficulty migration, and performance monitoring.

3.1. System Overview and Architecture

The puzzle module operates through a three-tiered structure consisting of a client-side interface, a backend layer responsible for logic and data storage, and a dynamic difficulty adjustment system to modify puzzle complexity in real time.

1. Client/Frontend (Mobile Application)

• Built with Flutter to ensure cross-platform (Android/iOS) compatibility.

• Offers a drag-and-drop environment that will seem familiar to children seeing puzzle pieces and putting them on a puzzle board.

• Offers functionalities for:

- Take pictures from camera or generate images from AI to create puzzles.
- Show the puzzle difficulty like the number and shape of pieces.
- Offering hints or partial auto-fills to support struggling users.

2. Backend (API + Business Logic)

• An Advance python DDA algorithm processes requests from the client:

- Receives performance data (correct/incorrect placements, time taken, hint usage, difficulty level).

- Runs the DDA algorithm that determines how complex the next puzzle will be.

• Stores user progress and puzzle configurations in the MongoDB database for long-term tracking and personalized adjustments.

3. Adaptive Feedback System

• Dynamically adjusts puzzles based on user performance metrics.

• Adjust puzzle parameters (number of pieces, split count, image prompt, time taken, hint usage) to keep the challenge at an acceptable level (neither trivially easy nor intractably difficult).

• Sends the updated puzzle specifications to the client.

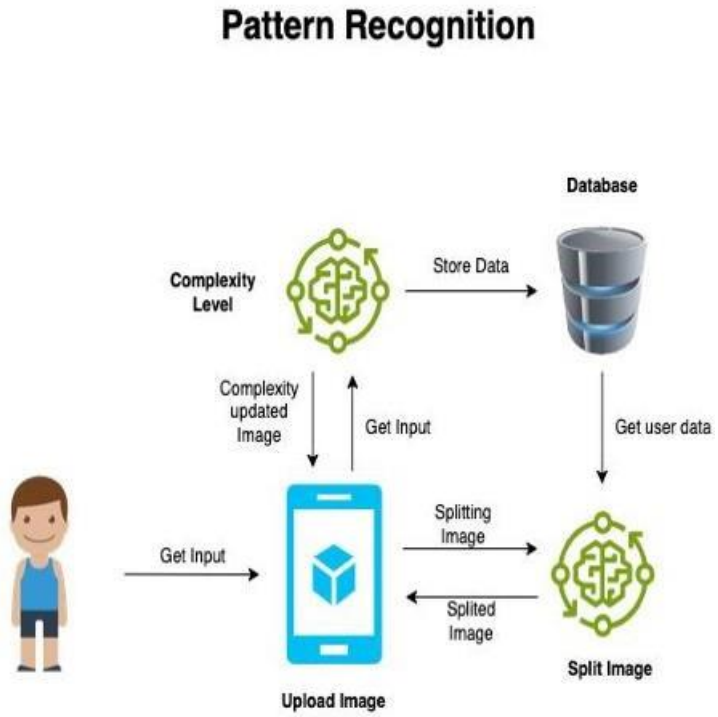


Figure 3.1 – High-Level Architectural Diagram depicting the client, backend algorithm, and database communication flow.

3.2. Project Requirements

This section outlines the essential requirements of the puzzle-based learning module aimed at supporting children with Nonverbal Learning Disability (NVLD). The requirements are informed by the project objectives, user needs, and the methodology detailed in the previous sections.

3.2.1. Functional Requirements

1. Puzzle Creation and Levels

• FR1.1: The system shall allow users to either capture an image via the camera or generate an image (using an AI prompt) to be transformed into a puzzle.

• FR1.2: The puzzle creation process shall split images into segments in at least two main modes:

• Level 1 (similar-size segments)

• Level 2 (irregular segments)

• FR1.3: The system shall provide feedback on puzzle segmentation, visually indicating puzzle piece boundaries or shapes before starting the puzzle if requested.

2. Drag-and-Drop Interaction

• FR2.1: Each puzzle piece shall be draggable, and the target board shall accept correct pieces within a defined position threshold.

• FR2.2: The system shall highlight or snap puzzle pieces into place when dropped correctly, providing immediate visual or auditory feedback.

3. Adaptive Difficulty Adjustment

• FR3.1: The backend shall monitor performance metrics (correct/wrong placements, time taken, hint usage) for each session.

• FR3.2: The puzzle difficulty (e.g., number of segments, segment shape complexity) shall automatically increase if a child's performance exceeds defined success thresholds (accuracy, minimal hints).

• FR3.3: The puzzle difficulty shall automatically decrease if user performance suggests high struggle (excessive time, multiple wrong moves, frequent hint usage).

4. Hint and Auto-Fill Features

• FR4.1: The system should provide a hint function that shows a partial overlay of the correct puzzle layout or helps auto-place a limited number of pieces.

• FR4.2: The user's hint usage must be tracked (e.g., increment a "hint count") to inform future difficulty adjustments.

5. Data Capture and User Profiling

• FR5.1: The system shall record each session’s data (correct moves, wrong moves, hint usage, session completion time) in a database under the respective user profile.

• FR5.2: The system shall maintain a session history, enabling educators or parents to view the child’s progress over multiple sessions.

6. Reporting and Analytics

• FR6.1: A summary screen shall appear at the end of each puzzle, displaying time taken, accuracy rate, and hints used.

• FR6.2: The system should allow exporting or viewing performance reports PDFs and charts to track improvement in puzzle completion times and reduced hint usage over time.

7. User Interface and Experience

• FR7.1: The puzzle interface shall be age-appropriate, incorporating large icons, clear visuals, and minimal text to accommodate NVLD-related challenges.

• FR7.2: The demo needs to display concise instructions or tooltips for each feature (capture image, generate puzzle, drag-and-drop, hints), ensuring intuitive navigation.

8. Error Handling and Fallbacks

• FR8.1: If puzzle generation fails (because of invalid image or capture failing), the system shall revert to a default puzzle to ensure the user can continue practicing.

• FR8.2: During network or server issues, the module shall provide an offline mode with locally cached puzzle sets, if feasible.

3.2.2. Non-Functional Requirements

1. Performance Requirements

• NFR1.1: The system should render each puzzle (including image splitting and drag-and-drop setup) within 2 seconds on standard mid-range devices.

• NFR1.2: Adaptive difficulty adjustments (backend logic calls) shall occur in real time, ensuring minimal lag (< 1 second) in updating puzzle settings between sessions.

2. Scalability and Availability

• NFR2.1: The backend API shall handle simultaneous requests from multiple users without significant performance degradation, supporting the possibility of classroom usage.

• NFR2.2: Database structures (session data) must accommodate large volumes of puzzle completion records without loss or corruption of data.

3. Usability and Accessibility

• NFR3.1: The application interface shall adhere to basic accessibility guidelines (high-contrast colors, readable fonts, large buttons) to accommodate children with NVLD or other reading challenges.

• NFR3.2: Instructional text shall be concise and, where possible, supplemented by icons or voice prompts to reduce textual overload.

4. Security and Privacy

• NFR4.1: All user data (puzzle progress, personal info) shall be transmitted and stored securely, following relevant data protection regulations.

• NFR4.2: Only authorized personnel (teachers, parents) shall access user progress reports; authentication measures must be in place to protect sensitive data.

5. Maintainability and Extensibility

• NFR5.1: The codebase (frontend, backend) shall be modular, permitting easy updates to puzzle logic, Algorithms, or interface components without major disruptions.

6. Reliability

• NFR6.1: The system shall recover from unexpected crashes or device restarts by resuming from the last saved puzzle state or offering a prompt to restart.

• NFR6.2: Puzzle data updates (hints, correct moves) must be atomically committed to avoid data inconsistency when switching from one puzzle to another.

Together, these functional and non-functional requirements provide a comprehensive framework for what the puzzle-based module should do, how users will interact with it, and the standards it must meet in terms of quality. Functional requirements describe behaviors that are core to the application (generation of puzzles, adaptive difficulty, capturing of data), and non-functional requirements describe targets for performance, principles of usability, and criteria for security / reliability. Building from the ground up ensures that the system not only satisfies educational and therapeutic solutions to working with kids with NVLD, but also provides a solid, intuitive, and secure framework.

3.3. Data Preprocessing

Accurate adaptive learning depends on accurate, high-quality data about a child's performance. This means that the puzzle system collects raw user inputs and then cleans and transforms that data into meaningful metrics.

1. Image Preprocessing

• All uploaded or AI images are resized to have a common resolution (300× screenPixelScale) to match the image scales by app install device's screen's resolution.

```
215
216     final screenPixelScale = MediaQuery.of(context).devicePixelRatio;
217     final imageSize = (300 * screenPixelScale).toInt();
218
```

Figure 3.1 – Adjust the image resolution according to screen.

2. Data Collection

• Correct Moves: Number of correct moves are tracked by "_movesMade" variable.

• Wrong Moves: Number of wrong moves are tracked by "consecutiveWrongMoves" variable.

```
void _onPiecePlaced(JigsawPiece piece, Offset pieceDropPosition) async {
  _totalMoves++; // Increment total moves
  final RenderBox box =
      _boardWidgetKey.currentContext?.findRenderObject() as RenderBox;
  final boardPosition = box.localToGlobal(Offset.zero);
  final targetPosition =
      boardPosition.translate(piece.boundary.left, piece.boundary.top);

  const threshold = 48.0;
  final distance = (pieceDropPosition - targetPosition).distance;

  if (distance < threshold) {
    setState(() {
      _currentPiece = piece;
      pieceOnPool.remove(piece);
      _movesMade++; // Correct move made
      _consecutiveWrongMoves = 0; // Reset wrong move counter
    });
```

Figure 3.2 – Tracking correct and wrong moves.

• Hint Usage: The number of times the child clicks the "hint" button or requests to auto fill up a certain piece of the puzzle.

• Completion Time: Time spent completing the current puzzle, or the average time per piece.

3. Feature Engineering

• Success Rate = Correct Moves / (Correct + Wrong Moves)

• Used to gauge user mastery of a current puzzle's complexity.

• Hint Coefficient: Weighted measure capturing how often hints are used.

e.g., hint_coefficient = 1 + (0.5 * number_of_hints)

• Time Normalization: Convert raw time to a “per-piece” metric to compare short vs. large puzzles fairly.

Optional advanced features:

- Streak Tracking (consecutive successful placements without hints).
- Difficulty Tiers (Low, Medium, High) stored as numeric levels for quick reference.

4. Data Storage

• The whole session’s data is stored against a userID in a NoSQL (MongoDB) or relational database, linking puzzle sessions to user ids.

• A record consists of puzzle specifications (level, piece count), performance metrics (correct/wrong moves, hints), and timestamps.

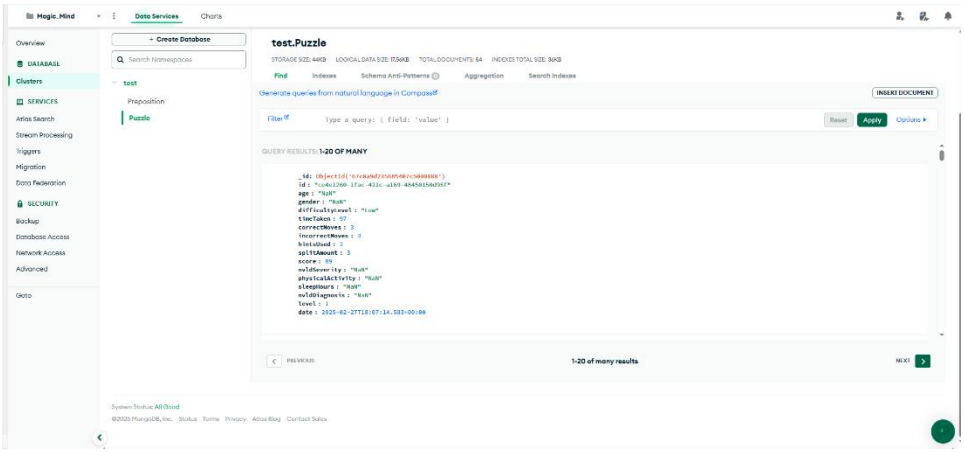


Figure 3.2.4 – Store child’s activity details

3.4. System Design and Implementation

3.4.1. Technology Stack

A puzzle-based module introduced many frameworks and tools with a prerequisite of resource-effective real-time communication and scalable backend to make it accessible across multi-platforms. A summary of the key technologies used is as follows:

1. Frontend (Client)

Flutter:

- Chosen for its cross-platform capabilities (Android & iOS).
- Provides widgets for intuitive drag-and-drop puzzle interactions.
- Offers straightforward handling of animations, gesture detection, and device camera access.

2. Backend Services

- Python with Flask:
- Handles all of the business logic behind how puzzles are generated, how difficulty levels work, how data is validated.
- Collects various performance metrics from the client (number of correct moves, hint usage) to re-evaluate the complexity of the puzzle.
- Returns updated puzzle configurations as JSON.
- Database – MongoDB
- Backend server – AWS E2C Instance
- User profiles, puzzle session logs as well as the accuracy (correct box checks) and hints used.
- Facilitates fast read/write operations crucial for real-time difficulty adaptation.

3. AI Components

- Image Generation – Dezgo Stable Diffusion API

4. System Design and Testing Methodology:

The system design uses a Flask and Python-based backend API to handle real-time performance data and adjust the difficulty of the puzzle dynamically.

Backend Implementation

API Design: - Input data is collected by the flask API through an HTTP POST request, processed and difficulty adjustments sent back.

Dynamic Difficulty Adjustment: - Input data is collected by the flask API through an HTTP POST request, processed and difficulty adjustments sent back.

```
Future<void> _adjestDifficulity(
    int correctM, int wrongM, int hintUsage) async {
  final response = await http.post(
    Uri.parse('$ML_API/adjust-difficulty'),
    headers: {'Content-Type': 'application/json'},
    body: json.encode({
      "correct_moves": correctM,
      "wrong_moves": wrongM,
      "hint_usage": hintUsage,
      "current_split_count": correctM
    }),
  );

  if (response.statusCode == 200) {
    final responseData = json.decode(response.body);

    String difficulty = responseData['difficulty'];
    String prompt = responseData['image_prompt'];

    setState(() {
      dificulityLevel = difficulty;
      imagePrompt = prompt;
      isLoading = false;
    });

    int factor = responseData['new_split_count'];

    await saveString("l1_gen_factor", factor.toString());
    await saveString("l1_gen_level", difficulty.toString());
    await saveString("l1_gen_difficulty", "1");
```

Figure 3.4.1.4 – http request to backend

5. Supporting Libraries and Packages

• http flutter package

• Handles API requests GET/POST to the Python backend for puzzle data and performance metrics.

• Shared Preferences Flutter package

• Local caching of user settings (difficulty level, last puzzle state) for offline continuity.

• Charting/Visualization Libraries

• Renders real-time or session-based performance graphs for immediate feedback or post-session reporting.

Figure 3.4.1.1 – Backend Libraries

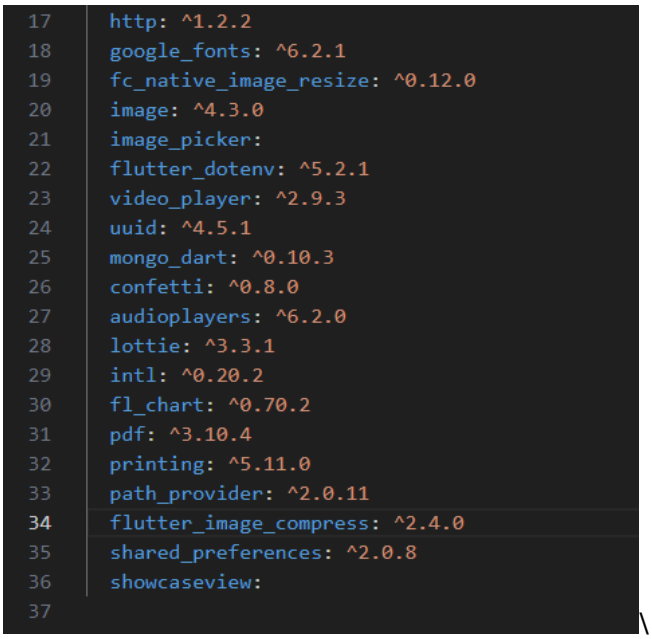


Figure 3.2.1.2 – Frontend Packages

3.5. System Diagrams

This subsection provides high-level visual representations of the module's architecture, data flow, and interactions among its core components.

3.5.1 High-Level Architecture Diagram

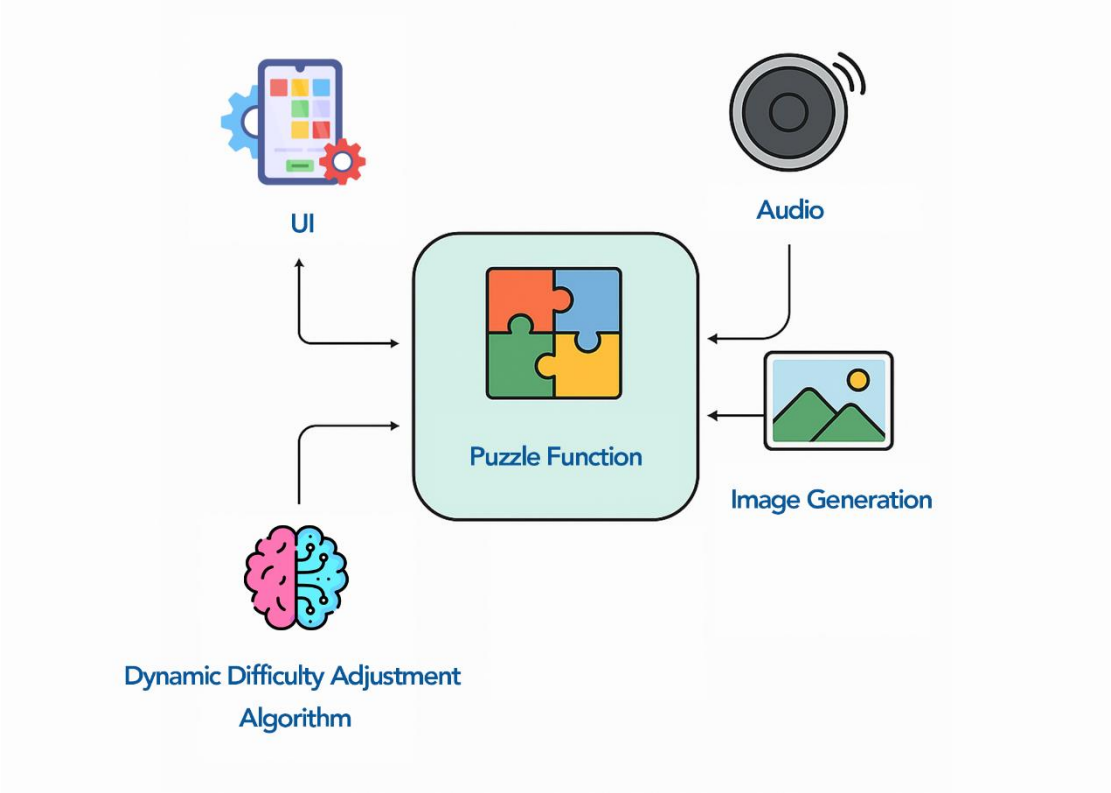


Figure 3.3.4 – Module Architecture Diagram

1. User Device (Mobile App)

• The child interacts with the puzzle UI in Flutter.

• Puzzle tasks (drag-and-drop, hints) are handled locally unless updates are needed from the backend.

2. Backend API (Python)

- Receives data points like (correct_moves, wrong_moves, hint_usage, time_taken) from the mobile app.
- Queries the database for user's historical puzzle performance if needed, then applies adaptive logic to reconfigure puzzle difficulty.
- Answer with new parameters of puzzle to the client (number of pieces, distribution of shapes, thresholds of hints, etc.).

3. Difficulty Adjustment Calculations:

Success Rate: Calculated as:

Success Rate = Correct Moves / (Correct Moves + Wrong Moves).

Adjustment Logic: -

-If the success rate is ≥ 80% with no hints used, increase difficulty.

- If the success rate ≤ 40%, decrease difficulty.

- If the user uses hints excessively, reduce difficulty.

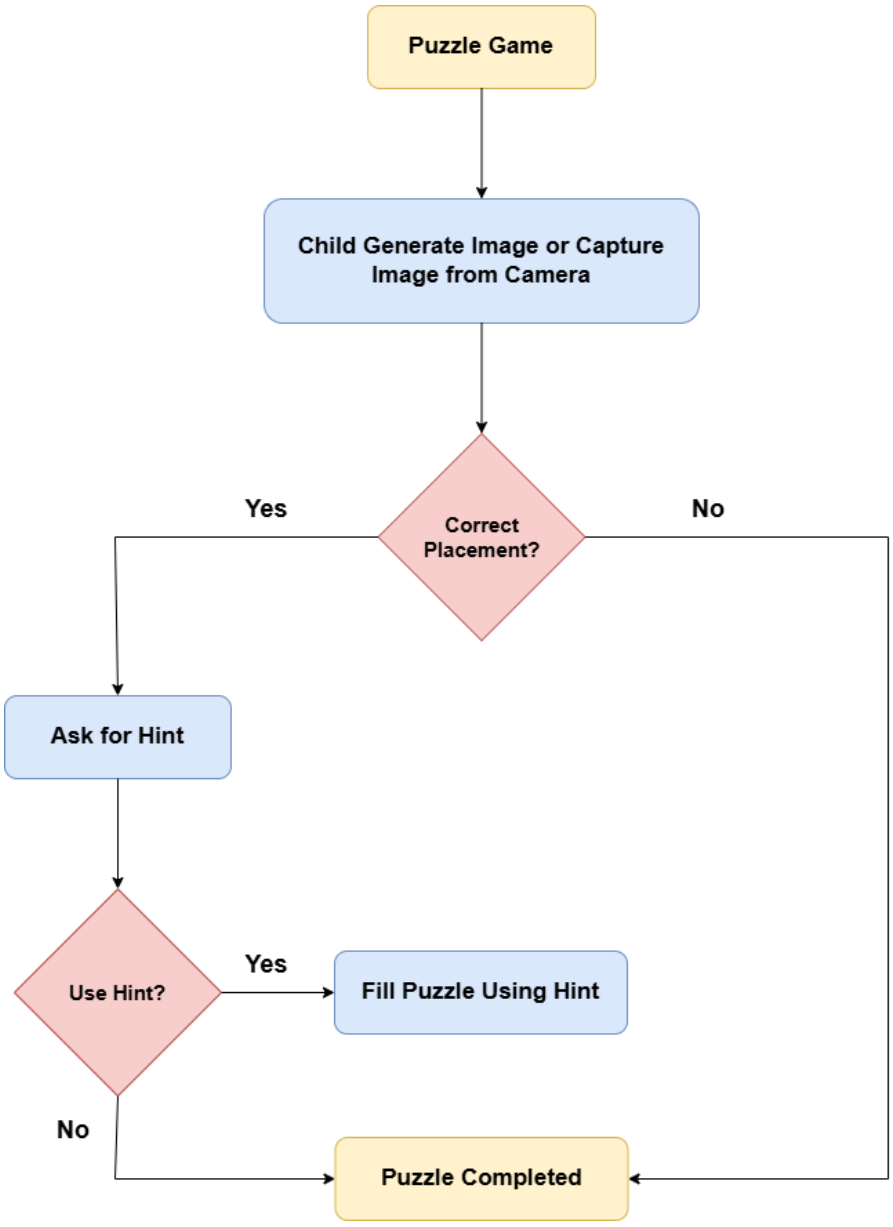
- If the user makes too many wrong moves, reduce difficulty.

The difficulty level is classified into three categories based on the number of split pieces (e.g., split count = 3 → Medium difficulty).

4. Database (MongoDB)

• Saves session logs for each puzzle in a store with unique user identifications for tracking across performance over time.

3.5.2 Detailed Puzzle Flow Diagram



Figer 3.5.2: User Flow Diagram

1. Initialization

• Child selects or generates an image for the puzzle.

• An image-splitting module divides an image into sections according to the difficulty settings currently in place.

2. Puzzle Gameplay

• The puzzle pieces come on-screen for dragging and dropping.

• Upon placing each piece, you get immediate checks (right/wrong).

• It is recorded data like continuous error or time for each piece.

3. Backend Update

• When the puzzle is solved or the user triggers for some mid-session events (e.g., more than N max moves), the app sends the uptime of that activity to the backend.

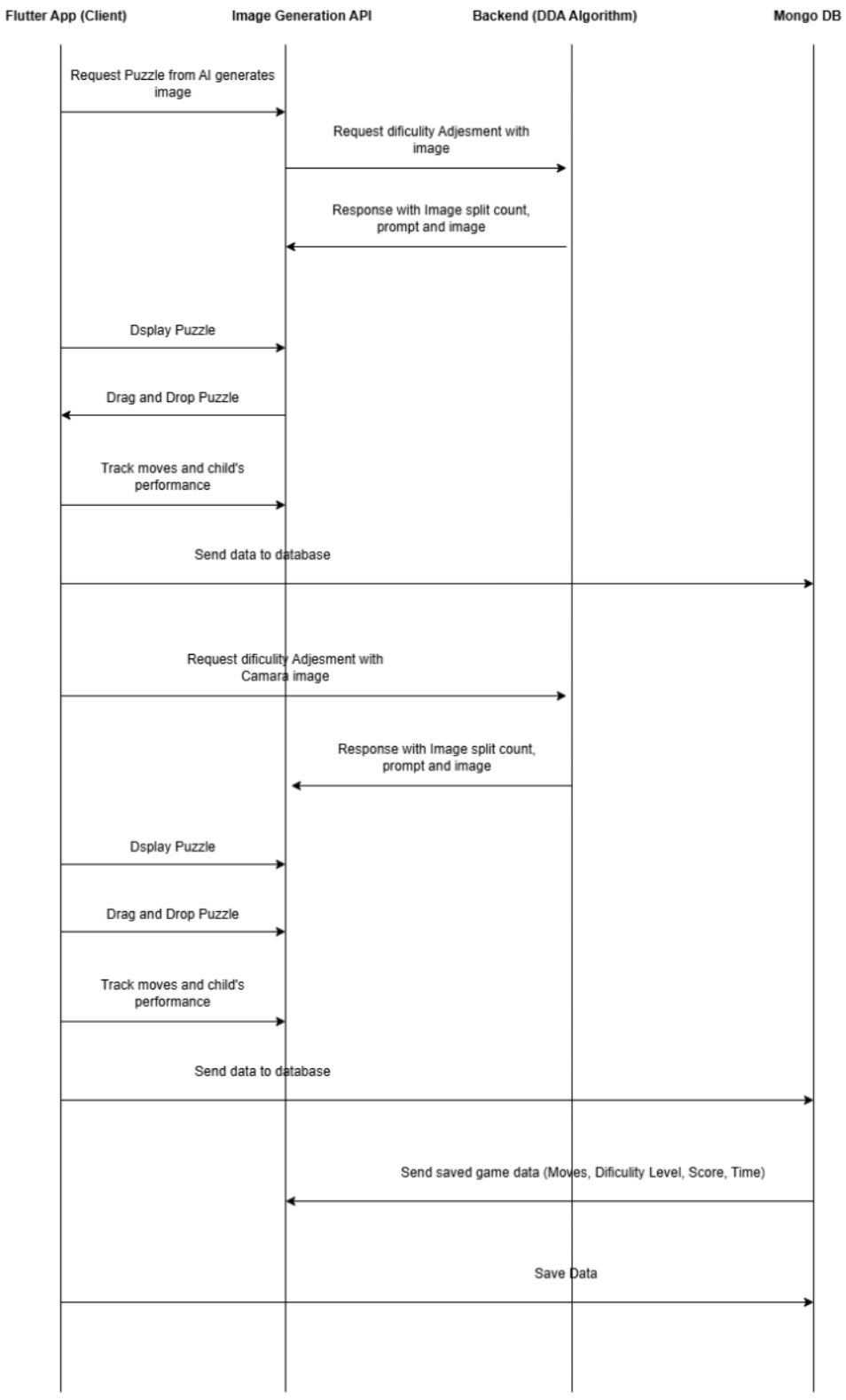
• The adaptive difficulty algorithm decides whether to escalate or reduce complexity for subsequent puzzles.

• The new configuration is stored in the user's record.

4. Feedback & Reporting

• At the end of the puzzle, users see a summary of their session (time taken to solve the puzzle, accuracy rate in terms of hints taken).

• Optional generation of a chart or PDF illustrating the child's progress trail over multiple sessions.



Figer 3.5.3: Sequence Diagram

3.6. Implementation Details

3.6.1. Image Generation and Splitting:

AI (Dezgo Cloud API- Stable Diffusion) is used to generate images, which are then broken into puzzle pieces. Based on the difficulty level, it renders a define number of pieces as well as their size. So, if we have a 3x3 grid, it divides image into 9 parts and if we have 4x4 grid then it divides it into 16 parts.

The splitting process involves:

- Determining the Grid Size: The image is divided into a grid based on the number of rows and columns (e.g., 3x3, 4x4).
- Splitting Process: The image is sliced into rectangular portions, being calculated as the row and column number divided by the shape of an image.
- Shuffling the Pieces: After the image is divided into pieces, those pieces are randomly shuffled to make it more challenging for the user.

3.6.2. Puzzle Splitting Logic:

1. Level 1 (Same Sized Pieces)

• The image is divided into a simple grid (e.g. - 2×2, 3×3).

• The puzzle pieces have similar dimensions, reducing visual complexity for initial practice.

• The system measures how quickly and accurately a child completes this level; if performance is high, it will move automatically on to more complex shapes.

```
494   List<JigsawPiece> _createJigsawPiece() {
495     return [
496       for (int i = 0; i < widget.factor; i++)
497         for (int j = 0; j < widget.factor; j++)
498           JigsawPiece(
499             key: UniqueKey(),
500             image: canvasImage,
501             imageSize: Size(300, 300),
502             points: [
503               Offset((i / widget.factor) * 300, (j / widget.factor) * 300),
504               Offset(
505                   ((i + 1) / widget.factor) * 300, (j / widget.factor) * 300), // Offset
506               Offset(((i + 1) / widget.factor) * 300,
507                   ((j + 1) / widget.factor) * 300), // Offset
508               Offset(
509                   (i / widget.factor) * 300, ((j + 1) / widget.factor) * 300), // Offset
510             ],
511           ), // JigsawPiece
512     ];
513   }
514
```

Figure 3.6.2.1 – Same sized image split function according to factor (factor means image split count that the algorithm returned.)

2. Level 2 (Different Sized Pieces)

• Each puzzle piece can have random width and height segments (e.g. - 1×2, 2×2, 2×3, etc.).

• This introduces more advanced visual-spatial challenges.

• The shape/size range is defined by a random function, so every time you play the same puzzle it will be different from the previous one.

```
2088  List<JigsawPiece> _createJigsawPiece() {
2089    int factor = widget.factor + 1;
2090
2091    List<Piece> pieces_ = [];
2092    List<Piece> all = [];
2093    List<JigsawPiece> pieces = [];
2094
2095    for (int x = 0; x < factor; x++) {
2096      for (int y = 0; y < factor; y++) {
2097        int x2 = x + ((x < (factor - 1)) ? random_length() : 1);
2098        int y2 = y + ((y < (factor - 1)) ? random_length() : 1);
2099
2100        var p = Piece(x, x2, y, y2);
2101
2102        bool issue = false;
2103        for (var b in p.sub_pieces()) {
2104          if (all.contains(b)) issue = true;
2105        }
2106        if (issue) {
2107          issue = false;
2108          x2 = x + 1;
2109          p = Piece(x, x2, y, y2);
2110          for (var b in p.sub_pieces()) {
2111            if (all.contains(b)) issue = true;
2112          }
2113        }
2114
2115        if (issue) {
2116          issue = false;
2117          y2 = y + 1;
2118          p = Piece(x, x2, y, y2);
2119          for (var b in p.sub_pieces()) {
2120            if (all.contains(b)) issue = true;
2121          }
2122        }
2123
2124        if (issue) continue;
2125
2126        pieces_.add(p);
2127        all.addAll(p.sub_pieces());
2128      }
```

Figure 3.6.2.2 – Different sized image split function.

3. Drag-and-Drop Implementation

• Each puzzle piece is generated as a Draggable widget.

• The puzzle board itself is a Drag Target, which checks whether the drop position is within the appropriate bounding region.

• When the dropped position is correct within a tolerance (e.g., 40–50 pixels), the piece “snaps” into place.

```
Expanded(
  child: Container(
    decoration: BoxDecoration(
      color: Colors.black,
      border: Border(
        top: BorderSide(
          width: 2,
          color: Colors.white,
        ), // BorderSide
      ), // Border
    ), // BoxDecoration
    child: ListView.separated(
      padding: EdgeInsets.all(32),
      scrollDirection: Axis.horizontal,
      physics: BouncingScrollPhysics(),
      itemCount: pieceOnPool.length,
      itemBuilder: (context, index) {
        final piece = pieceOnPool[index];
        return Center(
          child: Draggable(
            child: piece,
            feedback: piece,
            childWhenDragging: Opacity(
              opacity: 0.24,
              child: piece,
            ), // Opacity
            onDragEnd: (details) {
              _onPiecePlaced(piece, details.offset);
            },
          ), // Draggable
        ); // Center
      },
      separatorBuilder: (context, index) =>
          SizedBox(width: 32),
    ), // ListView.separated
  ), // Container
), // Expanded
```

Figure 3.6.2.3 – Draggable Widget Code.

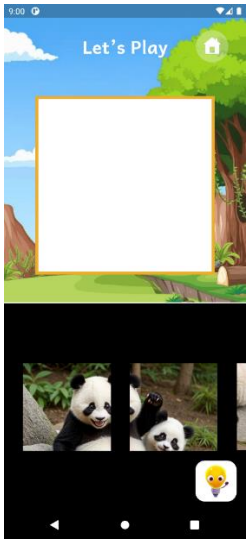


Figure 3.6.2.4 – In-Game Draggable Widget and pieces list.

3.6.3. Dynamic Difficulty Adjustment

The backend logic adjusts puzzle difficulty in real time to keep kids within their "optimal learning zone."
The difficulty of the puzzle is dynamically adjusted depending on whenever user's performance with the following criteria:

1. Correct Moves: -The solver's number of correct moves is not equal to the expected number of correct moves for the current difficulty level (which is the split count); therefore, the system marks it as an incorrectly solved puzzle and retains the difficulty.

2. Wrong Moves: -If the number of wrong moves is much larger than the number of correct moves in comparison (e.g. 1.5x) difficulty is lowered to suggest that the user is struggling

3. Hint Usage: - Excessive use of hints will lead to a decrease in difficulty. If the user uses hints frequently (e.g., more than half of the split count), the system reduces complexity.

4. Success Rate: - If the user completes a puzzle with a success rate ≥ 80% and no hints, the system increases the difficulty.

: -If the success rate is ≤ 40%, the system reduces the puzzle's complexity.

Formula Review:

Success Rate = Correct Moves / (Correct Moves + Wrong Moves).

The addition of 1e-5 (small value) in the denominator ensures that division by zero doesn't occur if there are no moves.

Difficulty Level Classification:

Low Difficulty: Split Count < 3 (Simple tasks).

Medium Difficulty: Split Count = 3-4 (Moderate complexity).

High Difficulty: Split Count > 4 (complex tasks).

Conditions for Adjusting Difficulty:

- The algorithm correctly reflects:
    - Reducing difficulty when the success rate is less than 40% or when hints are used excessively.
    - Increasing difficulty when the success rate is over 80% and hints are not used.

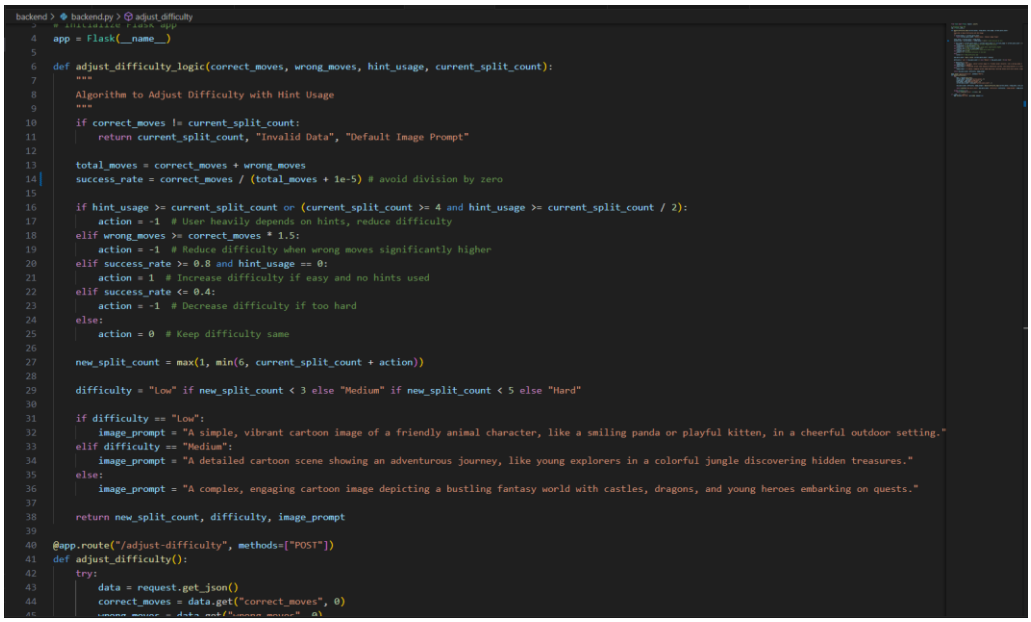


Figure 3.6.3 – Logic flow for difficulty adjustment algorithm.

3.6.4. Hint System & Auto-Fill Features:

1. Simple Hint

• Displays the complete image to get idea about puzzle placement.

• When child tap on the hint button and see the complete image at a once increment the child's hint_usage counts by 1.

2. Auto-Fill

• If child keeps placing wrong placements, meanwhile if there are not correct placements it will detect separately. Depending on image split count and pieces remaining in pool, a function decided what time I need to provide auto filling help to prevents the child from getting too frustrated but still keeps it partially challenging.

Pieces Remaining	Hint Usage	Pieces to Auto-Place
4 or fewer	+2	1
5 - 8	+4	2
9 - 12	+6	6
13 - 20	+8	8
21 - 35	+10	10

Table 3.6.4.2.1 – Auto filling puzzle logic

```
void fillPuzzleHints() {
  if (pieceOnPool.length <= 4) {
    setState(() {
      int piecesToFill = min(1, pieceOnPool.length);
      for (int i = 0; i < piecesToFill; i++) {
        var piece = pieceOnPool.removeAt(0);
        pieceOnBoard.add(piece);
      }
      hintUsed = hintUsed + 2;
    });
  } else if (pieceOnPool.length > 4 && pieceOnPool.length <= 8) {
    setState(() {
      int piecesToFill = min(2, pieceOnPool.length);
      for (int i = 0; i < piecesToFill; i++) {
        var piece = pieceOnPool.removeAt(0);
        pieceOnBoard.add(piece);
      }
      hintUsed = hintUsed + 4;
    });
  } else if (pieceOnPool.length > 8 && pieceOnPool.length <= 12) {
    setState(() {
      int piecesToFill = min(3, pieceOnPool.length);
      for (int i = 0; i < piecesToFill; i++) {
        var piece = pieceOnPool.removeAt(0);
        pieceOnBoard.add(piece);
      }
      hintUsed = hintUsed + 6;
    });
  } else if (pieceOnPool.length > 12 && pieceOnPool.length <= 20) {
    setState(() {
      int piecesToFill = min(5, pieceOnPool.length);
      for (int i = 0; i < piecesToFill; i++) {
        var piece = pieceOnPool.removeAt(0);
        pieceOnBoard.add(piece);
      }
      hintUsed = hintUsed + 8;
    });
  } else if (pieceOnPool.length > 20 && pieceOnPool.length <= 35) {
    setState(() {
      int piecesToFill = min(8, pieceOnPool.length);
```

Figure 3.6.4.2.2 – Logic for Auto filling and struggle tracking.

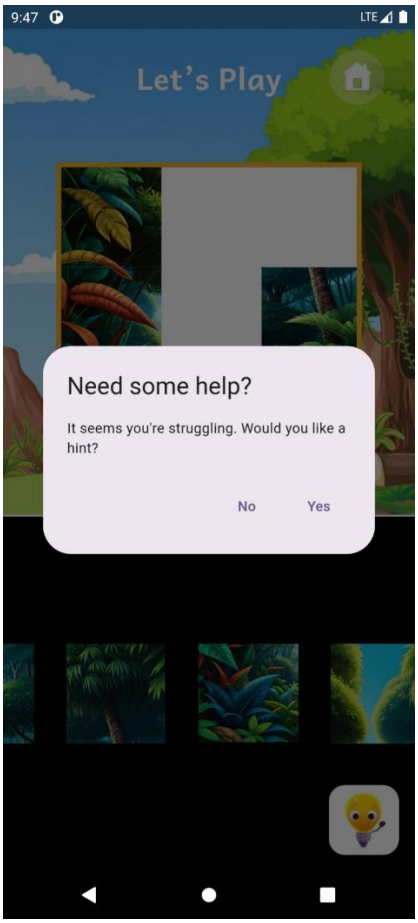


Figure 3.6.4.2.3 – In-game autofill alert asking for a child, want to get some autofill pieces.

3.6.5 Reporting & Analytics:

3.6.5.1. Session Summary

Each puzzle session concludes with a summary screen display:

- Completion time
- Wrong moves
- Hints used
- Score

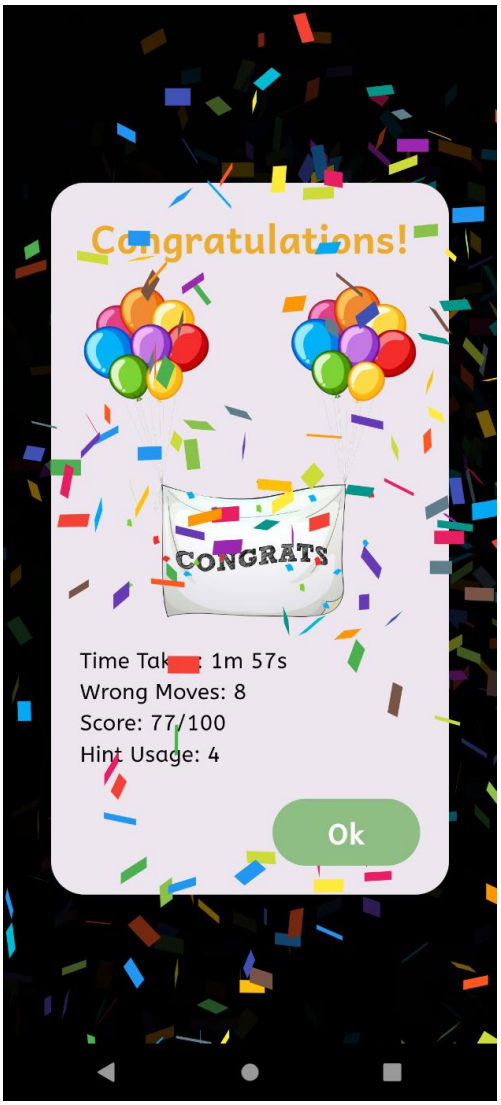


Figure 3.6.5.1 – In-game summary screen

3.6.5.2. Performance Reports

- Session-to-session progress tracking as seen by user, parent or educator.
- Status: Indicates whether the student is improving, struggling, or the performance is neutral. This status is determined based on:

1. **Score Improvement:** How much the score has improved (or declined) over the last 5 sessions.

2. **Incorrect Moves:** The number of incorrect moves and how much it has increased or decreased.

3. **Hint Usage:** Whether the student is using a lot of hints, which could indicate struggle.

Algorithm for Analyzing Performance:

If the score improvement is greater than 5, incorrect moves have decreased, and hints are used less, it indicates improvement.

If the score improvement is less than -5, incorrect moves are rising, or hints are frequently used, it indicates that the student is struggling.

Otherwise, the performance is neutral.

Puzzle Session Records:

Displays a list of the last 5 puzzle sessions, including:

* Session Number.
* Score: The score obtained in that session.
* Incorrect Moves: The number of incorrect moves made.
* Hints Used: The number of hints used during the session.
* Difficulty Level: The level of difficulty the student worked on (Low, Medium, High).

This section provides a summary of the student's performance across their recent sessions.

Performance Charts: These charts provide a visual representation of the student's progress and help to better understand the data.

Score Progression: Shows the trend of the student’s scores across multiple sessions. A rising trend indicates improvement.

Mistakes & Hint Usage: Displays how many mistakes the student made and how many hints they used. Fewer mistakes and less hint usage typically indicate better problem-solving skills.

Difficulty Level Distribution: Shows how much time was spent on each difficulty level (Low, Medium, Hard). More time spent on higher difficulty levels suggests improvement.

Downloadable PDF Report: The report is saved as a PDF file, which includes all the above information and charts. The PDF is generated using the pdf package and shared with the user via the printing package, allowing the user to download the report.

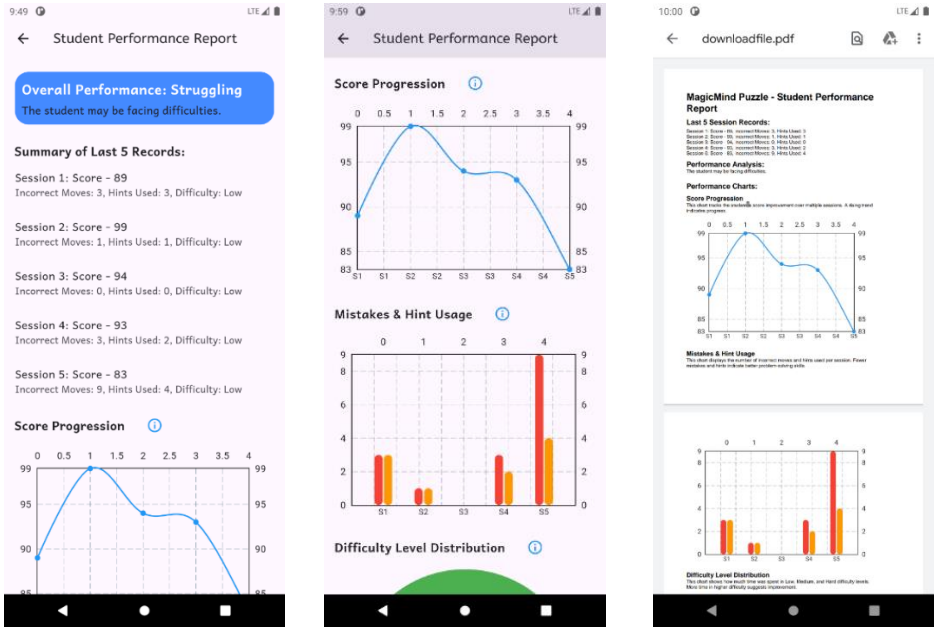


Figure 3.6.5.2 – Report summarizing puzzle statistics.

3.6.6. Puzzle Solving Mechanism

1. Puzzle Initialization:

Before solving, the puzzle is generated and split into small pieces based on a factor (grid size like 3×3, 4×4, etc.). Each piece is initially stored in pieceOnPool and shuffled using.shuffle().

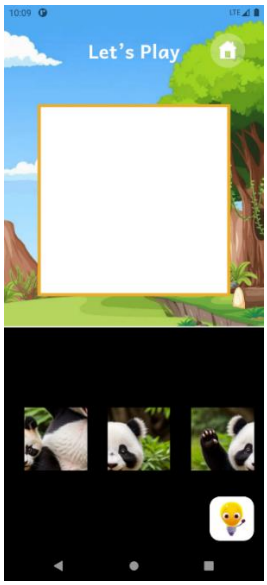


Figure 3.6.6.1 – Example screenshot of created puzzle

2. Drag & Drop:

The child picks a puzzle piece and drags and drops to the canvas.

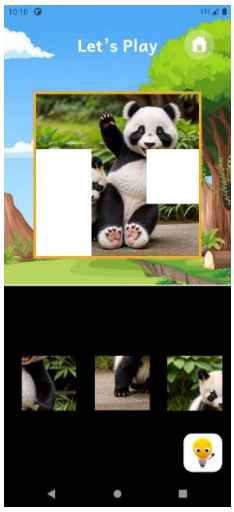


Figure 3.6.6.2 – Dragging and dropping puzzle pieces into the canvas.

3. Placement Verification:

The core logic checks whether a piece is placed correctly is in _onPiecePlaced():

```
546   void _onPiecePlaced(JigsawPiece piece, Offset pieceDropPosition) async {
547     _totalMoves++; // Increment total moves
548     final RenderBox box =
549         _boardWidgetKey.currentContext?.findRenderObject() as RenderBox;
550     final boardPosition = box.localToGlobal(Offset.zero);
551     final targetPosition =
552         boardPosition.translate(piece.boundary.left, piece.boundary.top);
553
554     const threshold = 48.0;
555     final distance = (pieceDropPosition - targetPosition).distance;
556
557     if (distance < threshold) {
558       setState(() {
559         _currentPiece = piece;
560         pieceOnPool.remove(piece);
561         _movesMade++; // Correct move made
562         _consecutiveWrongMoves = 0; // Reset wrong move counter
563       });
```

Figure 3.6.6.3 – Check the placement is correct or not.

4. Incorrect Move Handling:

If the child made a wrong move, the _consecutiveWrongMoves counter is incremented.

```
    final simulation = SpringSimulation(spring, 0, 1, -distance);
    _animController.animateWith(simulation);
  } else {
    // If move was incorrect, increment wrong move counter
    _consecutiveWrongMoves++;
    _checkWrongMoveProgress();
    await _audioPlayer.play(AssetSource('audios/wrong_move.wav'));
  }
}
```

Figure 3.6.6.4 – Increment the wrong moves count.

5. Puzzle Completion Check:

When all pieces are correctly placed:

- The game timer is stopping.
- The score is calculated based on:
  - Time taken
  - Number of correct vs. wrong moves
  - Hints used

```
572     _animController.addStatusListener((status) async {
573       if (status == AnimationStatus.completed) {
574         setState(() {
575           pieceOnBoard.add(piece);
576           _currentPiece = null;
577         });
578
579         if (pieceOnPool.isEmpty) {
580           _stopTimer();
581           _calculateScore();
582           setState(() {
583             isCorrect = true;
584           });
585
586           await _audioPlayer.play(AssetSource('audios/completed.wav'));
587
588           Future.delayed(Duration(seconds: 2), () async {
589             setState(() {
590               isCorrect = false;
591             });
592             _showCompletionDialog(context);
593           }); // Future.delayed
594         }
595       }
596     });
```

Figure 3.6.6.4.5 – Completing the puzzle.

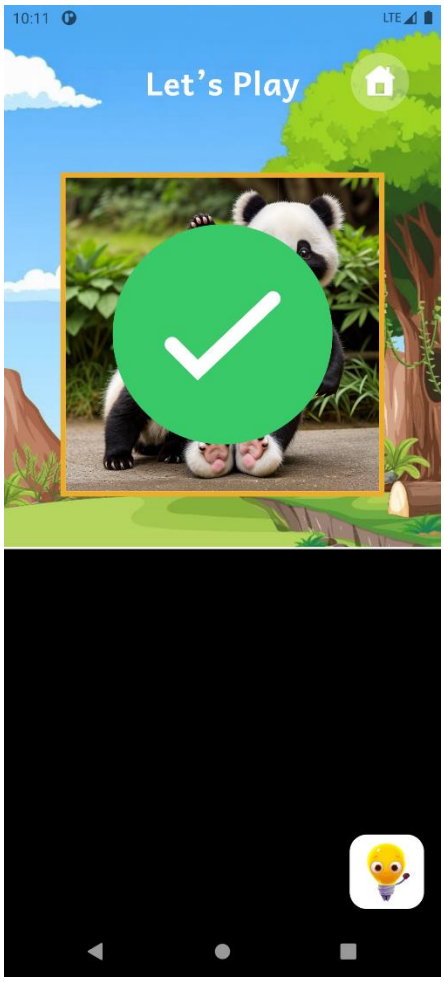


Figure 3.6.6.4.6 – In app screenshot of completing the puzzle.

Summary of Puzzle Sorting Machanism

Component	Description
Puzzle Pieces	Created based on a grid split (factor), randomized for gameplay.
Drag & Drop	Allows users to move pieces to target locations
Placement Verification	Distance-based logic to determine if piece is correctly placed.

Feedback Loop	Audio + animation feedback for success/failure
Hint System	Offers progressive hints based on the number of remaining pieces.
Scoring	Calculates score based on moves, time, and hint usage.
Adaptive Difficulty	Learns from user performance and adjusts next level accordingly.

3.7. Testing and Evaluation

This chapter details how the puzzle-based learning module was tested to ensure functionality, reliability, and educational efficacy for children aged 10–13 with Nonverbal Learning Disability (NVLD). It describes the testing environment, methods used to verify each system feature, and evaluation metrics guiding model selection and subsequent analysis.

3.7.1. Objectives of Testing

1. Functionality Validation: Confirm the puzzle generation, drag-and-drop, and hint features perform correctly under typical and edge-case scenarios.

2. Adaptive Logic Verification: Ensure that dynamic difficulty adjustments accurately reflect user performance metrics (correct moves, wrong moves, hint usage).

3. Usability & Accessibility: Evaluate whether children with NVLD can easily navigate the interface, complete puzzles, and interpret feedback or hints without confusion.

4. Performance & Reliability: Assess responsiveness (puzzle loading times, real-time difficulty updates) and confirm the system handles unexpected inputs or interruptions gracefully.

3.7.2. Evaluation Metrics:

Engagement: Measures user interaction with the puzzle (time spent on puzzles, number of correct/incorrect moves).

Success Rate: Percentage of successful puzzle completions.

Difficulty Adjustments: Tracks how often difficulty levels are adjusted based on user performance.

User Feedback: Gathered through direct testing with children to assess their engagement and satisfaction with the app.

3.7.3. Testing Strategy

1. Define the testing functions.
2. Design test cases.
3. Execute and check whether expected results are met.
4. Record results
5. Identify bugs
6. Fix bugs
7. Repeat the test case until the expected results are met.

3.7.3.1. Test case design

Test Case ID	01
Test Scenario	Verify correct placement of puzzle piece
Steps	1. Start puzzle 2. Drag a piece to its correct location 3. Drop within threshold distance
Expected Output	Piece snapped into place, removed from pool, added to board, and correct sound is played.

Actual Output	Piece snapped into place, removed from pool, added to board, and correct sound is played.
Status (Pass/Fail)	Pass

Test Case ID	02
Test Scenario	Verify incorrect placement of puzzle piece.
Steps	1. Start puzzle 2. Drag a piece far from its correct location 3. Drop outside threshold
Expected Output	Piece remains in pool, wrong move sound is played, and wrong moves is incremented.
Actual Output	Piece remains in pool, wrong move sound is played, and wrong moves is incremented.
Status (Pass/Fail)	Pass

Test Case ID	03
Test Scenario	Verify hint button auto-fills correct pieces.
Steps	1. Start puzzle 2. Make wrong moves until the helping dialog pops up. 3. Press “Yes” button on the dialog.
Expected Output	Based on pool size, 1–8 pieces are auto filled randomly.
Actual Output	Based on pool size, 1–8 pieces are auto filled randomly.
Status (Pass/Fail)	Pass

Test Case ID	04
Test Scenario	Verify timer starts with the puzzle load.
Steps	1. Start puzzle 2. Wait and observe

Expected Output	Time Elapsed increases every second
Actual Output	Time Elapsed increases every second
Status (Pass/Fail)	Pass

Test Case ID	05
Test Scenario	Verify puzzle completion triggers congratulation flow.
Steps	1. Solve all pieces correctly 2. Observe completion behavior
Expected Output	Confetti dialog with score/time/moves is shown.
Actual Output	Confetti dialog with score/time/moves is shown.
Status (Pass/Fail)	Pass

Test Case ID	06
Test Scenario	Verify wrong move triggers autofill helping dialog.
Steps	1. Start puzzle 2. Make multiple incorrect moves depending on pool size
Expected Output	A dialog asking if the user needs help is shown.
Actual Output	A dialog asking if the user needs help is shown.
Status (Pass/Fail)	Pass

Test Case ID	07
Test Scenario	Verify image prompt is correctly sent to generation.
Steps	1. Start puzzle. 2. Go ahead with some puzzles with a good score.
Expected Output	Generated image difficulty will increase.
Actual Output	Generated image difficulty will increase.
Status (Pass/Fail)	Pass

Test Case ID	08
Test Scenario	Verify adaptive difficulty adjustment after session.
Steps	1. Complete a puzzle 2. Observe transition to new puzzle
Expected Output	A new puzzle is loaded with updated difficulty and prompt based on past performance.
Actual Output	A new puzzle is loaded with updated difficulty and prompt based on past performance.
Status (Pass/Fail)	Pass

4. RESULTS & DISCUSSION

This chapter examines the outcomes of the puzzle-based learning module's testing, focusing on how adaptive difficulty and interactive puzzle mechanics influenced the progress of children with Nonverbal Learning Disability (NVLD). It synthesizes quantitative metrics (accuracy, hint usage, completion time) with qualitative feedback (user satisfaction, observed cognitive engagement) to offer insights into the system's efficacy and potential areas for refinement.

4.1. Overview of Quantitative Findings

4.1.1. Accuracy and Error Reduction

In the pilot study with a group of children aged 10–13, accuracy improved from an initial average of around 68% to approximately 85% by the final sessions. This improvement suggests that learners quickly adapted to puzzle mechanics and benefited from real-time difficulty adjustments.

• Reduction in Wrong Moves: Over successive puzzles, participants exhibited fewer incorrect placements, implying that the puzzle-based activities may have strengthened their visual-spatial recognition and reasoning skills.

4.1.2. Hint Usage and Completion Time

Hint usage declined by roughly 57% across multiple sessions, indicating children relied less on scaffolding as they became more comfortable with puzzle tasks. Meanwhile, average completion time per puzzle decreased from around 4.5 minutes to 3.2 minutes, reflecting enhanced problem-solving efficiency and familiarity with drag-and-drop interactions.

Interpretation: The combination of gradual puzzle complexity escalation and targeted hints appears to maintain an optimal level of challenge, reinforcing the concept of zone of proximal development. When tasks are neither too easy nor too difficult, learners remain motivated to improve autonomously.

4.1.3. Adaptive Difficulty Shifts

System logs show consistent adjustments to puzzle parameters (piece size, irregular shapes) for each child based on performance metrics such as time, accuracy, and hint usage. On average:

• Difficulty Escalations occurred twice per session by the final testing phase, indicating participants typically progressed to more complex puzzle configurations once they mastered simpler ones.

4.2. Qualitative Observations and User Feedback

4.2.1. Engagement and Motivation

Many children described the puzzle activities as “fun” or “challenging in a good way.” Teachers/parents observed increased focus compared to more static exercises, suggesting that gamified elements like point systems, partial auto-fill hints successfully captured and retained user interest.

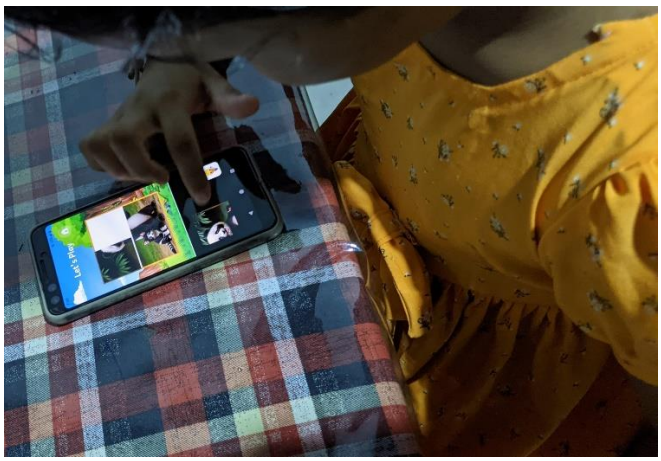


Figure 4.2.1.1

Figure 4.2.1.2

Figure 4.2.1.1 and Figure 4.2.1.2 above show actual user interactions during puzzle solving. These snapshots provide visual evidence of child engagement, fine motor interaction, and overall interest.

4.2.2. NVLD-Specific Challenges

Children with NVLD often struggle with spatial perception and nonverbal cues. The puzzle-based module's drag-and-drop approach and immediate feedback (e.g., piece snapping when correct) helped clarify spatial relationships. However:

• Some participants still found Level 2 puzzles overwhelming when the shapes were overly irregular.

• Occasional confusion arose if the puzzle board was visually cluttered, highlighting the ongoing need for visual simplicity and age-appropriate designs.

4.2.3. Importance of Customizable Parameters

Both educators and parents requested the option to manually fine-tune puzzle complexity in special cases (e.g., if a child is extremely anxious or has co-occurring

motor difficulties). This feedback underscores the utility of a semi-automatic difficulty override—allowing a teacher or therapist to step in and momentarily simplify the puzzle.

4.3. Alignment with Project Objectives

Recapping the objectives set out for the puzzle-based module:

1. Enhancing Visual-Spatial and pattern recognition Skills: The measured increases in accuracy and decreases in completion time suggest meaningful improvements in children's ability to process spatial layouts and recognize patterns.

2. Adaptive Difficulty: The successful real-time scaling of puzzle complexity based on user performance indicates the system's capacity to individualize learning and keep tasks optimally challenging.

3. Usability and Engagement: User feedback indicated generally positive experiences, affirming the interface's child-friendly features and the motivational impact of puzzle-style exercises.

Hence, the key objectives—improving visual-spatial reasoning while maintaining an engaging, adaptive environment—were substantially met in this pilot.

4.4. Limitations and Considerations

1. Sample Size: Initial testing relied on a modest cohort (~15 children), limiting the generalizability of results across more diverse NVLD populations.

2. Short Duration: Most participants completed only 5–7 puzzle sessions, which may be insufficient to gauge long-term skill retention or advanced puzzle mastery.

3. Device and Connectivity Variance: Performance differences arose between higher-end tablets vs. older mobile devices with limited RAM or weaker processors. This occasionally caused minor lag in puzzle rendering.

4. Complexity Overload: For children who struggle severely with visual overload, some advanced puzzle shapes proved too challenging. I need to consider toggling off certain shapes or providing layered tutorial modes.

4.5. Future Directions and Recommendations

1. Extended Trials: Conduct a larger study over several weeks/months to validate sustained improvements and examine potential skill transfer to other visual-spatial tasks.

2. Adaptive Overlays: Instead of a static puzzle board, incorporate optional overlays that guide children step-by-step (like “Tap here first” or partial color-coded hints) for extremely low-accuracy users.

3. Enhanced Reporting: Expand data visualization options, such as heatmaps showing where children frequently drop puzzle pieces incorrectly, offering deeper insights into specific spatial concepts needing reinforcement.

4. Expansion to Multi-User or Group Settings: Investigate how collaborative puzzle challenges might foster peer interaction and social learning, especially beneficial for children with NVLD who often face social or communicative hurdles.

The results from this pilot confirm that adaptive puzzle-based learning can significantly boost the visual-spatial abilities and confidence of children with NVLD, particularly when combined with real-time feedback and carefully tuned difficulty. While these findings are promising, continued research with larger samples and extended usage is recommended to fully leverage the potential of puzzle-based interventions for children facing nonverbal learning challenges.

5. CONCLUSION

This project set out to develop a puzzle-based learning module that adapts to the visual-spatial needs of children aged 10–13 with Nonverbal Learning Disability (NVLD). By integrating image splitting, dynamic difficulty adjustments, and hint-based support, the module demonstrated the ability to:

• Enhance Visual-Spatial Skills: Children showed measurable improvements in puzzle accuracy and reduced reliance on hints, suggesting strengthened spatial reasoning.

• Maintain Engagement: Gamified elements, including immediate feedback, partial auto-fills, and progress tracking, helped sustain user motivation.

• Provide Individualized Support: Real-time metrics (e.g., correct vs. wrong moves, hint usage) enabled the system to tailor puzzle complexity, preventing boredom or frustration.

Despite encouraging results, certain limitations (e.g., small sample sizes, short testing durations, device variability) highlight the need for wider implementation and longer studies to fully assess the solution's potential in broader educational settings.

Future Works:

1. Larger-Scale Trials: Extend testing to a more diverse group of NVLD learners over multiple weeks or months, capturing data on long-term retention and skill transfer to real-world tasks.

2. Expanded Puzzle Modes: Integrate advanced puzzle variations (e.g., 3D shapes, thematically rich puzzles) to continuously challenge higher-level visual-spatial skills.

3. Enhanced Customization: Allow educators or therapists to manually override complex puzzles, ensuring children with co-occurring conditions (e.g., severe motor impairments) receive a perfectly matched challenge.

4. Deeper Analytics: Implement in-depth performance dashboards (e.g., heatmaps of puzzle piece placements, session comparison graphs) to provide actionable feedback for teachers and caregivers.

5. Collaborative Features: Explore multiplayer or group puzzle tasks that foster social engagement and peer collaboration—beneficial for NVLD learners who often struggle with social cues.

6. REFERENCES

1. Brown, J., Smith, A., & Lewis, P. (2021). Dynamic Difficulty in Educational Games: A Comparative Study. Journal of Learning Technologies, 14(2), 45–58.

2. Lee, M., Wang, T., & Cooper, D. (2019). Age-Appropriate User Interface Design for Children: Best Practices and Case Studies. Computers in Education, 23(1), 12–25.

3. Williams, R. & Matthews, G. (2021). Adaptive UX Strategies for Children with Learning Disabilities: A Systematic Review. Learning Disability Quarterly, 44(3), 182–194.

4. Garrison, T. & Delgado, M. (2020). Personalized Learning Pathways in Cognitive Tutoring Systems. International Journal of AI in Education, 28(4), 350–365.

7. APPENDICES

Appendices contain supplementary information not included in the main body of the report but still valuable for completeness and reference. Examples might include raw

data sets, detailed code samples, expanded survey questionnaires, or user consent forms.

7.1. Sample Puzzle Screens and Code Snippets

Appendix A – Screenshots Demonstrating Puzzle Levels.

• Level 1: Sample images with uniform grid splits (2×2, 3×3).

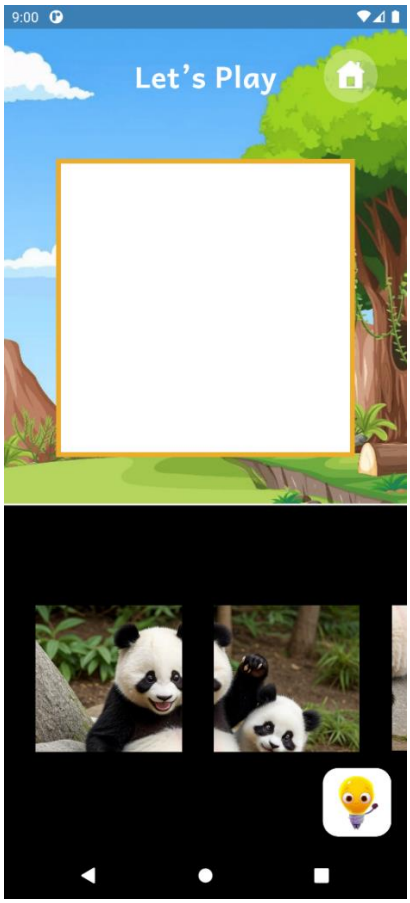


Figure 7.1.1

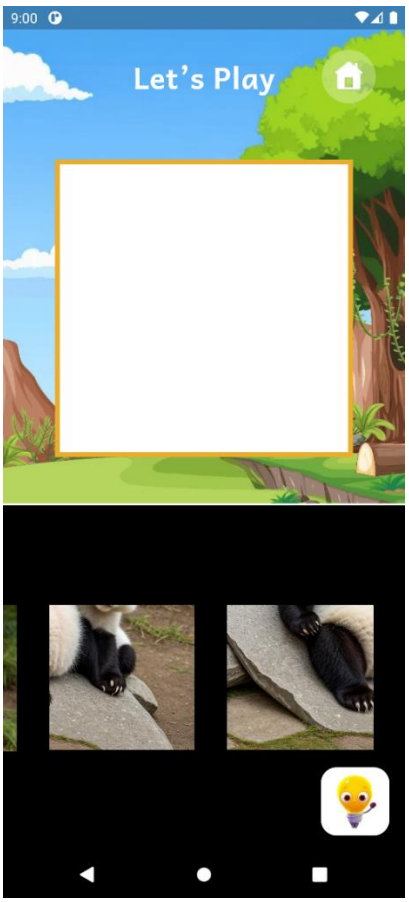
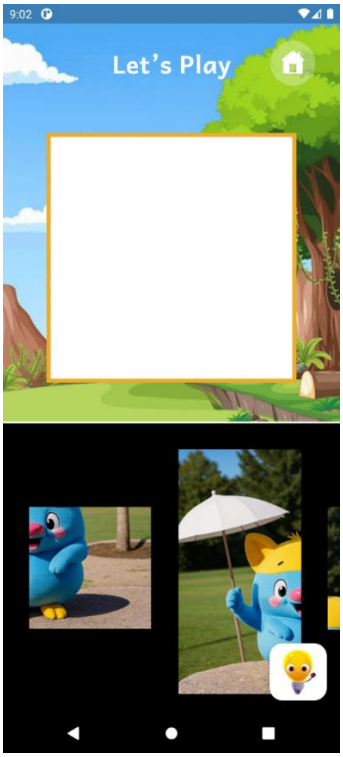


Figure 7.1.2

• Level 2: Randomly sized segments (1×2, 2×2, 2×3, etc.).



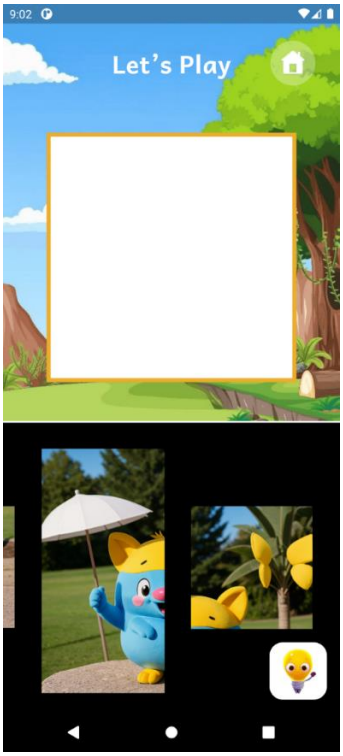


Figure 7.1.3 Figure 7.1.4

Appendix B – Implementation Snippets

• Puzzle-splitting logic in Python (backend).

```python
app = Flask(__name__)

def adjust_difficulty_logic(correct_moves, wrong_moves, hint_usage, current_split_count):
    """
    Algorithm to Adjust Difficulty with Hint Usage
    """
    if correct_moves != current_split_count:
        return current_split_count, "Invalid Data", "Default Image Prompt"

    total_moves = correct_moves + wrong_moves
    success_rate = correct_moves / (total_moves + 1e-5) # avoid division by zero

    if hint_usage >= current_split_count or (current_split_count >= 4 and hint_usage >= current_split_count / 2):
        action = -1  # User heavily depends on hints, reduce difficulty
    elif wrong_moves >= correct_moves * 1.5:
        action = -1  # Reduce difficulty when wrong moves significantly higher
    elif success_rate >= 0.8 and hint_usage == 0:
        action = 1  # Increase difficulty if easy and no hints used
    elif success_rate <= 0.4:
        action = -1  # Decrease difficulty if too hard
    else:
        action = 0  # Keep difficulty same

    new_split_count = max(1, min(6, current_split_count + action))

    difficulty = "Low" if new_split_count < 3 else "Medium" if new_split_count < 5 else "Hard"

    if difficulty == "Low":
        image_prompt = "A simple, vibrant cartoon image of a friendly animal character, like a smiling panda or playful kitten, in a cheerful outdoor setting."
    elif difficulty == "Medium":
        image_prompt = "A detailed cartoon scene showing an adventurous journey, like young explorers in a colorful jungle discovering hidden treasures."
    else:
        image_prompt = "A complex, engaging cartoon image depicting a bustling fantasy world with castles, dragons, and young heroes embarking on quests."

    return new_split_count, difficulty, image_prompt

@app.route("/adjust-difficulty", methods=["POST"])
def adjust_difficulty():
    try:
        data = request.get_json()
        correct_moves = data.get("correct_moves", 0)
```

Figure 7.1.5

• Flutter widget code for drag-and-drop.

```dart
void _onPiecePlaced(JigsawPiece piece, Offset pieceDropPosition) async {
  _totalMoves++; // Increment total moves
  final RenderBox box =
      _boardWidgetKey.currentContext?.findRenderObject() as RenderBox;
  final boardPosition = box.localToGlobal(Offset.zero);
  final targetPosition =
      boardPosition.translate(piece.boundary.left, piece.boundary.top);

  const threshold = 48.0;
  final distance = (pieceDropPosition - targetPosition).distance;

  if (distance < threshold) {
    setState(() {
      _currentPiece = piece;
      pieceOnPool.remove(piece);
      _movesMade++; // Correct move made
      _consecutiveWrongMoves = 0; // Reset wrong move counter
    });

    await _audioPlayer.play(AssetSource('audios/correct_move.wav'));

    _offsetAnimation = Tween<Offset>(
      begin: pieceDropPosition,
      end: targetPosition,
    ).animate(_animController);

    _animController.addStatusListener((status) async {
      if (status == AnimationStatus.completed) {
        setState(() {
          pieceOnBoard.add(piece);
          _currentPiece = null;
        });

        if (pieceOnPool.isEmpty) {
          _stopTimer();
          _calculateScore();
          setState(() {
            isCorrect = true;
          });

```

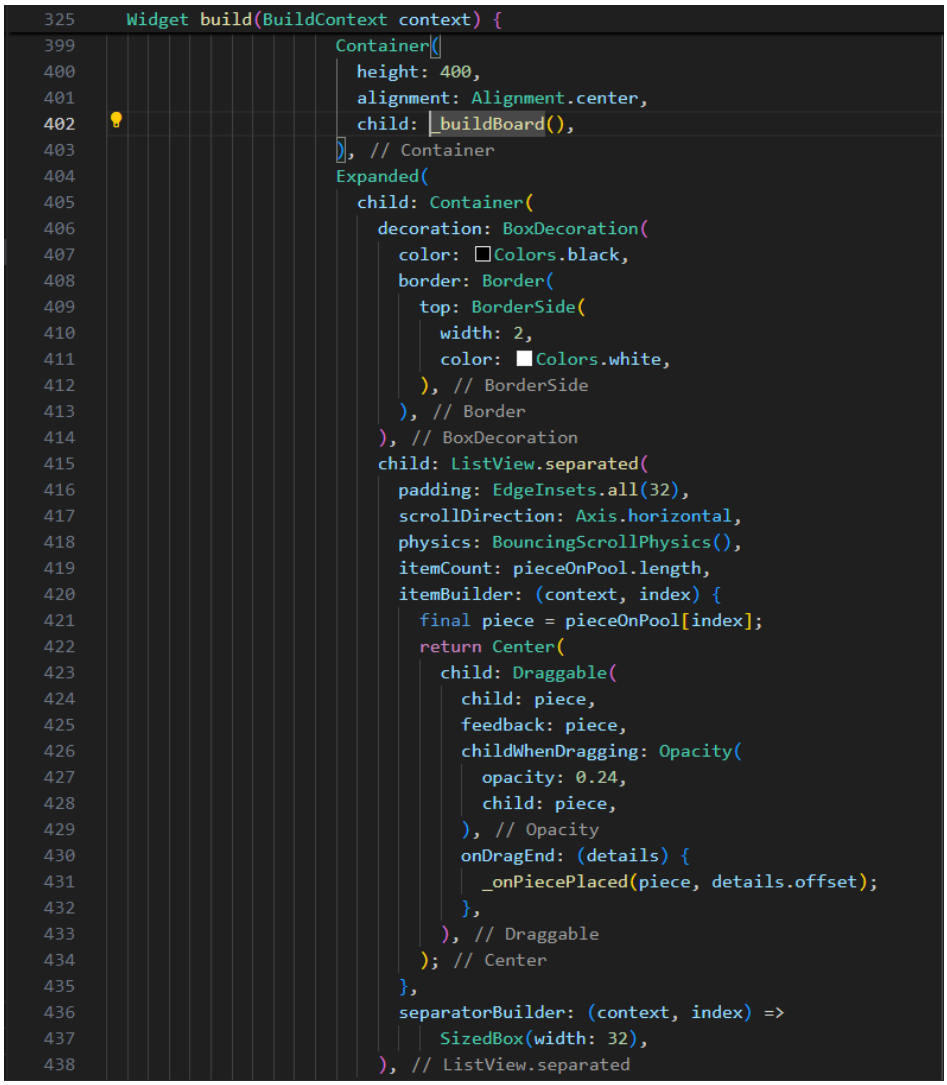
Figure 7.1.6

```
585
586                 await _audioPlayer.play(AssetSource('audios/completed.wav'));
587
588                 Future.delayed(Duration(seconds: 2), () async {
589                   setState(() {
590                     isCorrect = false;
591                   });
592                   _showCompletionDialog(context);
593                 }); // Future.delayed
594               }
595             }
596           });
597
598           const spring = SpringDescription(
599             mass: 30,
600             stiffness: 1,
601             damping: 1,
602           );
603
604           final simulation = SpringSimulation(spring, 0, 1, -distance);
605           _animController.animateWith(simulation);
606         } else {
607           // If move was incorrect, increment wrong move counter
608           _consecutiveWrongMoves++;
609           _checkWrongMoveProgress();
610           await _audioPlayer.play(AssetSource('audios/wrong_move.wav'));
611         }
612       }
613
```

Figure 7.1.7

```
911 class JigsawPainter extends CustomPainter {
924   });
925
926   @override
927   void paint(ui.Canvas canvas, ui.Size size) {
928     final paint = Paint();
929     final path = getClip(size);
930     if (elevation > 0) {
931       canvas.drawShadow(path, Colors.black, elevation, false);
932     }
933
934     canvas.clipPath(path);
935     if (image != null) {
936       canvas.drawImageRect(
937           image!,
938           Rect.fromLTRB(boundary.left * pixelScale, boundary.top * pixelScale,
939               boundary.right * pixelScale, boundary.bottom * pixelScale),
940           Rect.fromLTWH(0, 0, boundary.width, boundary.height),
941           paint);
942     }
943   }
944
945   Path getClip(Size size) {
946     final path = Path();
947     for (var point in points) {
948       if (points.indexOf(point) == 0) {
949         path.moveTo(point.dx - boundary.left, point.dy - boundary.top);
950       } else {
951         path.lineTo(point.dx - boundary.left, point.dy - boundary.top);
952       }
953     }
954     path.close();
955     return path;
956   }
957
958   @override
959   bool shouldRepaint(covariant JigsawPainter oldDelegate) {
960     return oldDelegate.image != image;
961   }
962 }
963
```

Figure 7.1.8



```
325   Widget build(BuildContext context) {
399                   Container(
400                     height: 400,
401                     alignment: Alignment.center,
402                     child: _buildBoard(),
403                   ), // Container
404                   Expanded(
405                     child: Container(
406                       decoration: BoxDecoration(
407                         color: Colors.black,
408                         border: Border(
409                           top: BorderSide(
410                             width: 2,
411                             color: Colors.white,
412                           ), // BorderSide
413                         ), // Border
414                       ), // BoxDecoration
415                       child: ListView.separated(
416                         padding: EdgeInsets.all(32),
417                         scrollDirection: Axis.horizontal,
418                         physics: BouncingScrollPhysics(),
419                         itemCount: pieceOnPool.length,
420                         itemBuilder: (context, index) {
421                           final piece = pieceOnPool[index];
422                           return Center(
423                             child: Draggable(
424                               child: piece,
425                               feedback: piece,
426                               childWhenDragging: Opacity(
427                                 opacity: 0.24,
428                                 child: piece,
429                               ), // Opacity
430                               onDragEnd: (details) {
431                                 _onPiecePlaced(piece, details.offset);
432                               },
433                             ), // Draggable
434                           ); // Center
435                         },
436                         separatorBuilder: (context, index) =>
437                             SizedBox(width: 32),
438                       ), // ListView.separated
```

Figure 7.1.8

```
518   Widget _buildBoard() {
519     return Stack(
520       children: [
521         Container(
522           decoration: BoxDecoration(
523             color: Colors.white,
524             border: Border.all(
525               width: 5,
526               color: const Color.fromARGB(255, 236, 173, 44),
527             ), // Border.all
528           ), // BoxDecoration
529           key: _boardWidgetKey,
530           width: 305,
531           height: 305,
532           child: Stack(
533             children: [
534               for (var piece in pieceOnBoard)
535                 Positioned(
536                   left: piece.boundary.left,
537                   top: piece.boundary.top,
538                   child: piece,
539                 ), // Positioned
540             ],
541           ), // Stack
542         ), // Container
543         isCorrect
544             ? SizedBox(
545                 width: 305,
546                 height: 305,
547                 child: Center(
548                   child: Lottie.asset('assets/animations/correct.json'),
549                 ), // Center
550               ) // SizedBox
551             : SizedBox(),
552       ],
553     ); // Stack
554   }
```

Figure 7.1.9

Appendix C - Survey

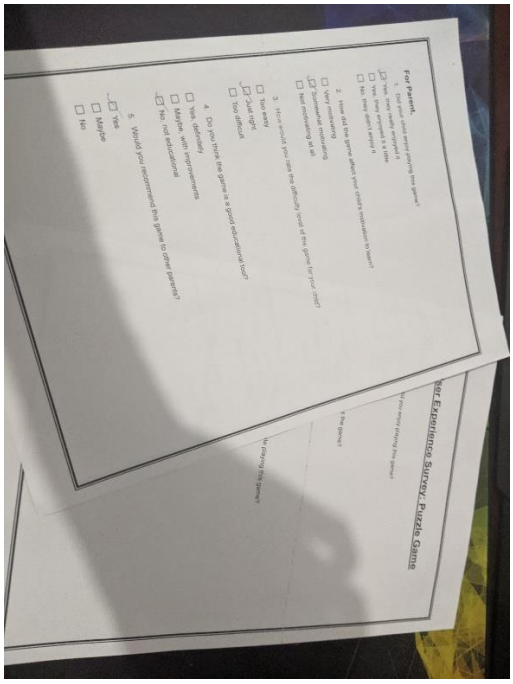


Figure 7.1.10

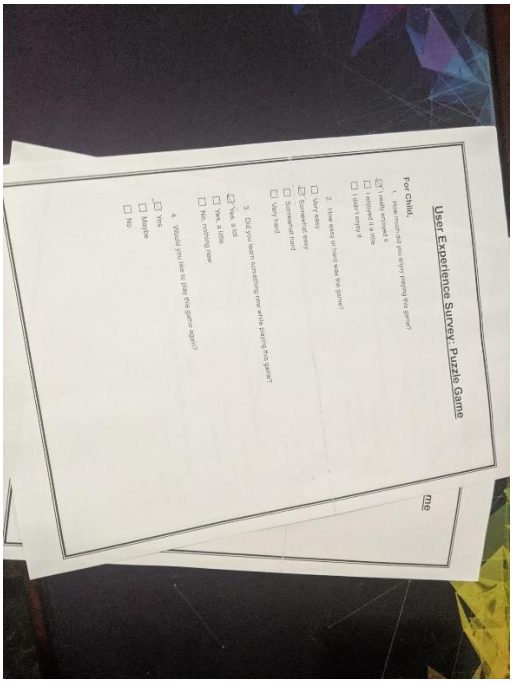


Figure 7.1.11