Game Design Document

Color Quest

Color Quest-GDD3-March 17, 2025

Version 3.0.0

Author
Ardella Malinda Sarastri
Game Product Owner / Designer
2010511021@mahasiswa.upnvj.ac.id

Universitas Pembangunan Nasional "Veteran" Jakarta
2024-2025

DOCUMENT INFORMATION

*This document is part of the academic final project for the development of the game "**Color Quest**" by **Ardella Malinda Sarastri**. The content herein is intended for educational, research, and documentation purposes only. All design decisions, system descriptions, and visual materials are based on the final implementation of the Color Quest mobile application (version 9.0). This document may be shared with supervisors, examiners, or academic reviewers as part of the evaluation process. Unauthorized commercial use or modification of the game concept or its assets without proper attribution is discouraged.*

Table of Contents

1. Introduction

1.1. Purpose of the document

This document serves as the **Game Design Specification Document** for the development of the mobile game **Color Quest**, an educational game designed to help screen color vision deficiency (color blindness) in early childhood. The document outlines the functional behavior, game mechanics, interface structure, and evaluation system used in the final version of Color Quest (v9.0). It ensures traceability between game features and design decisions, while providing a clear guideline for future enhancement, testing, and evaluation.

1.2. Project Background

Color blindness in children is often undetected due to limitations in traditional screening methods. Standard tests like the Ishihara test are not engaging for children aged 4–6 years and may not yield reliable results. To address this issue, **Color Quest** was developed as a gamified solution to engage children through playful interaction while enabling initial detection of potential color vision deficiency (Protan, Deutan, Tritan). The game uses a dynamic difficulty model via Finite State Machine (FSM), and performs diagnostic estimation using Fuzzy Logic analysis based on user input and gameplay results.

The project was carried out as part of an academic thesis at Universitas Pembangunan Nasional "Veteran" Jakarta.

1.3. Scope of the Document

This document covers:

1. The system architecture of Color Quest version 9.0.
2. Functional game design including gameplay modes, scoring logic, FSM, and Fuzzy Logic.
3. Interface structure and visual/audio assets.
4. Testing processes including alpha and beta tests with early learners.
5. Diagrams to support gameplay logic and evaluation.
6. It excludes: clinical validation, backend server deployment, monetization, or third-party integration features.

1.4. Related Document

Component	Name (with link to the document)	Description
A	SKRIPSI ARDELLA	Full documentation of Color Quest development, testing, and analysis.
B	GDD v2.0 (Obsolete)	Previous draft of game design, no longer valid for current version.
C	Unity Project Files	Source code and assets for Color Quest v9.0.

D	Test Report (Alpha-Beta)	Results and feedback from internal testing and child playtesting.

1.5. Terms and Acronyms

Term/Acronym	Definition	Description
FSM	Finite State Machine	A state-based logic system used to manage game flow and level transitions.
Fuzzy Logic	Approximate reasoning system	Used to estimate the likelihood of color blindness from gameplay scores.
Protan / Deutan / Tritan	Color vision types	Represent red-, green-, and blue-yellow-based deficiencies, respectively.
CVD	Color Vision Deficiency	General term for difficulty in distinguishing colors.

1.6. Risks and Assumptions

1. The game is intended for early screening and does not replace clinical diagnosis.
2. Children's interaction may be inconsistent due to attention span or external distractions.
3. Requires Android devices with basic multitouch and color-accurate screens.
4. No backend server or data sync is used – all evaluation is client-side.
5. Fuzzy Logic estimates are based on heuristic rules, not clinical trials.

2. System/ Solution Overview

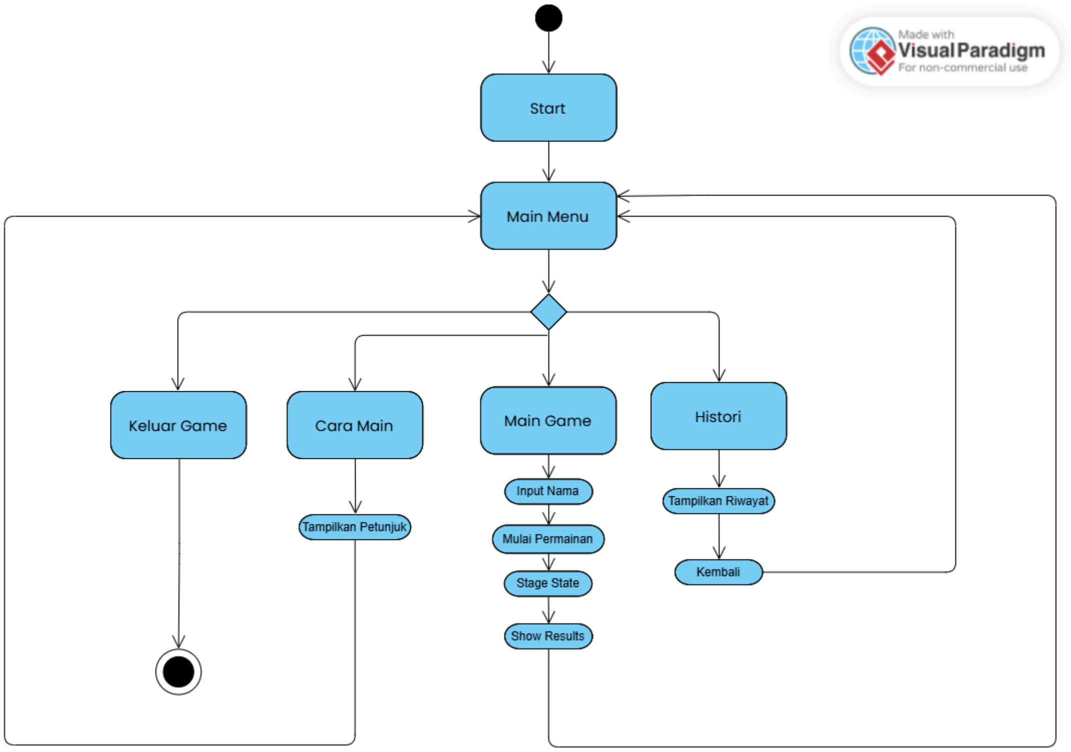
Color Quest is a mobile educational game designed to help identify potential color vision deficiencies (CVD) in children aged 4–6 years through engaging color-based gameplay. The system consists of three interactive modes (Different Color, Same Color, Sort Color) and dynamically adjusts challenge difficulty using Finite State Machine (FSM). The application evaluates gameplay responses using **Fuzzy Logic** to estimate likelihood of Protan, Deutan, or Tritan color blindness. The game provides immediate results and visual feedback without requiring internet or external servers.

Goals and Benefits:

1. Early screening of color vision issues in children.
2. More child-friendly than traditional static tests (e.g., Ishihara).
3. Encourages learning through play while collecting evaluative data.
4. Accessible on standard Android devices (offline-ready).

2.1. Context Diagram / Application Screen Flow / Process Flow

2.1.1. Application Screen Flow



This diagram illustrates the overall navigation structure within the Color Quest application. Upon launching the app (`Start`), users are directed to the `Main Menu`, where they can choose from the following:

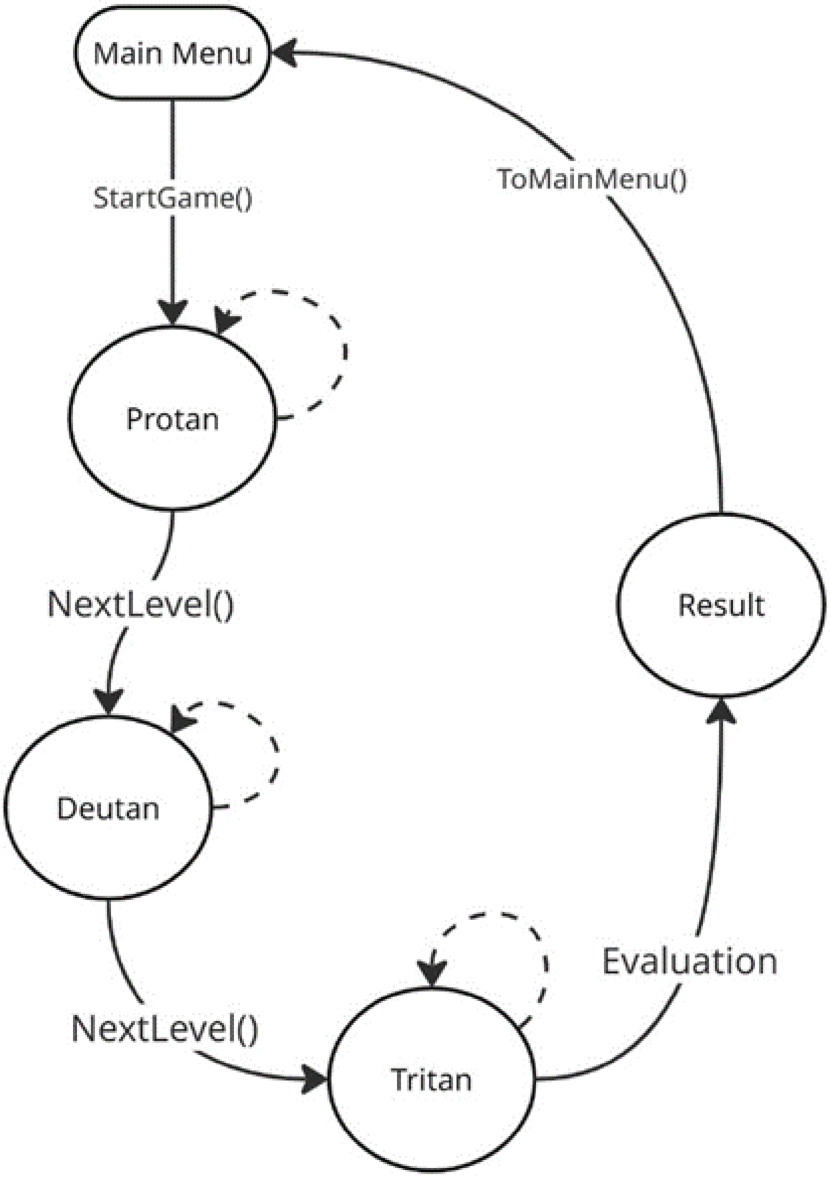
1. How to Play – Displays instructions for users and guardians.
2. Main Game – Begins the core gameplay sequence, including player name input,

level progression, and score display.

3. History – Allows users to review previous game results
4. Exit Game – Closes the application.

This flow provides a clear overview of user access points and system navigation.

2.1.2. FSM – Color Blindness Level Progression



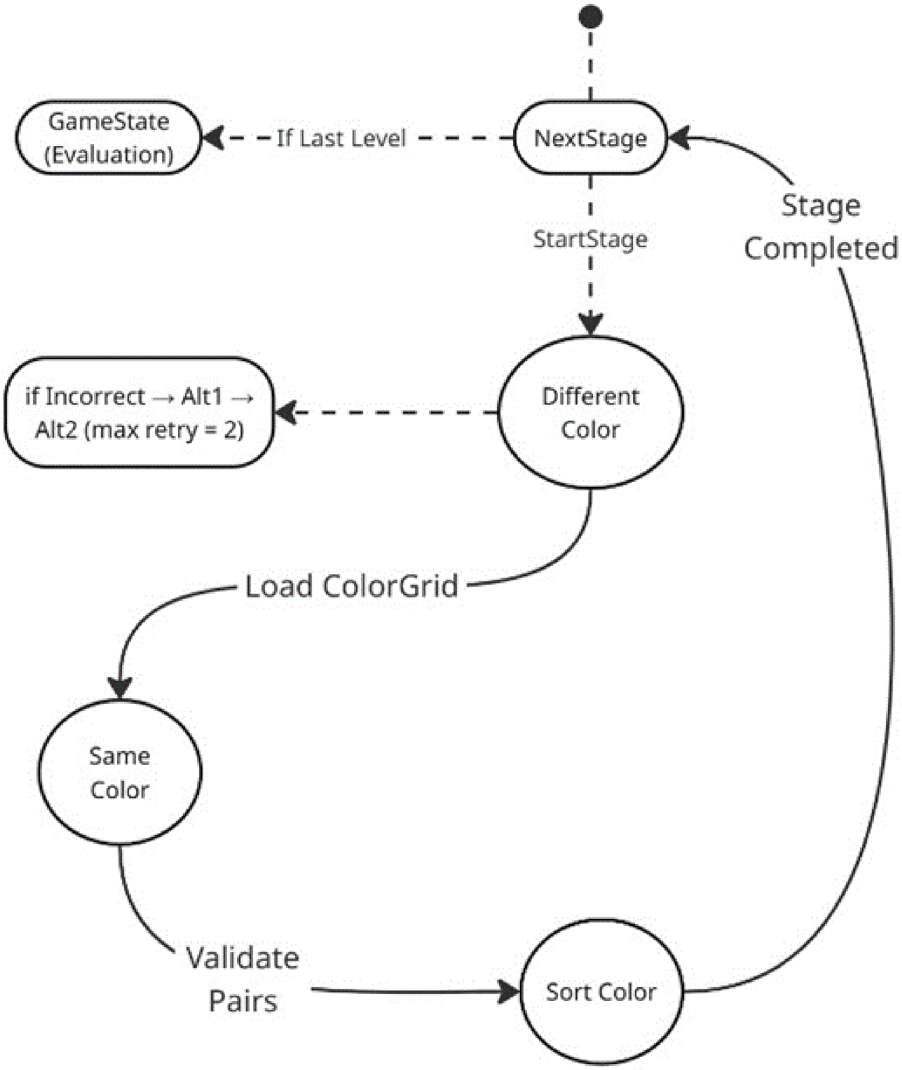
This state diagram represents the game's level-based progression system. Once the game starts, it transitions sequentially through three color vision challenge levels:

1. Protan

2. Deutan
3. Tritan

After each level is completed, the system evaluates the user's performance and proceeds using `NextLevel()` logic. Once all levels are completed, the system computes a final diagnosis and returns to the main menu.

2.1.3. FSM – Gameplay Mode Transition (StageState)

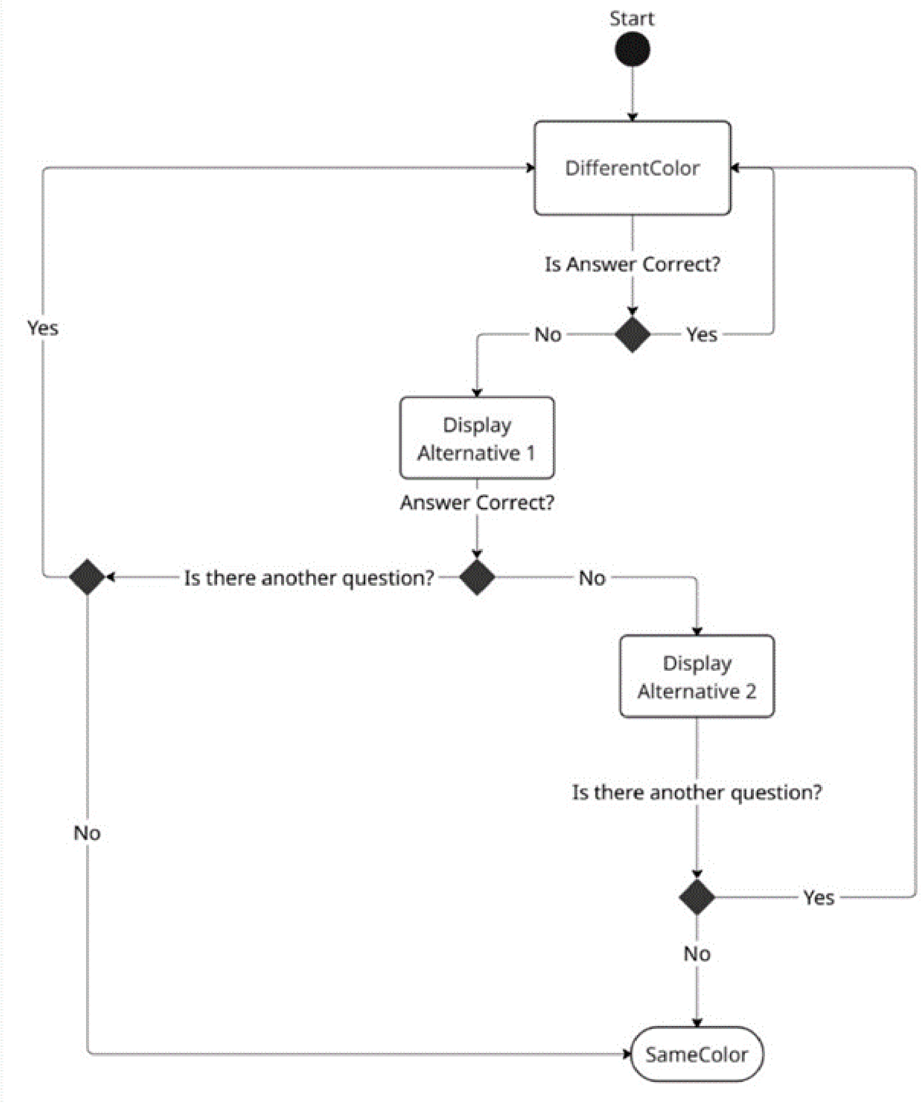


Within each color vision level, the game consists of three sequential mini-games:

1. DifferentColor
2. SameColor
3. SortColor

Each mode is initiated through `StartStage()` and transitions upon completion. Incorrect answers in the DifferentColor mode trigger an alternate question mechanism before continuing. After finishing all three modes, the system advances to the next color vision level or final result stage.

2.1.4. Alternative Question Handling – DifferentColor

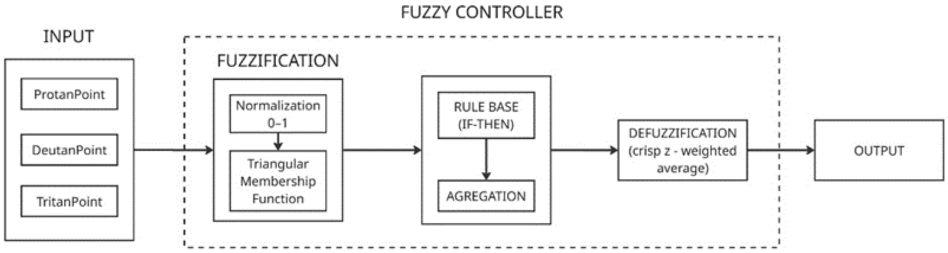


This decision-based flowchart details the internal logic for alternative questions in the DifferentColor mode:

1. If the first answer is incorrect → display Alternative 1.
2. If the second attempt is still incorrect → display Alternative 2.
3. After the third total attempt or one correct answer, the game continues to SameColor.

This mechanism ensures that each user is given up to two retries per stage to reduce random guessing effects.

2.1.5. Fuzzy Logic Controller – Diagnosis Architecture

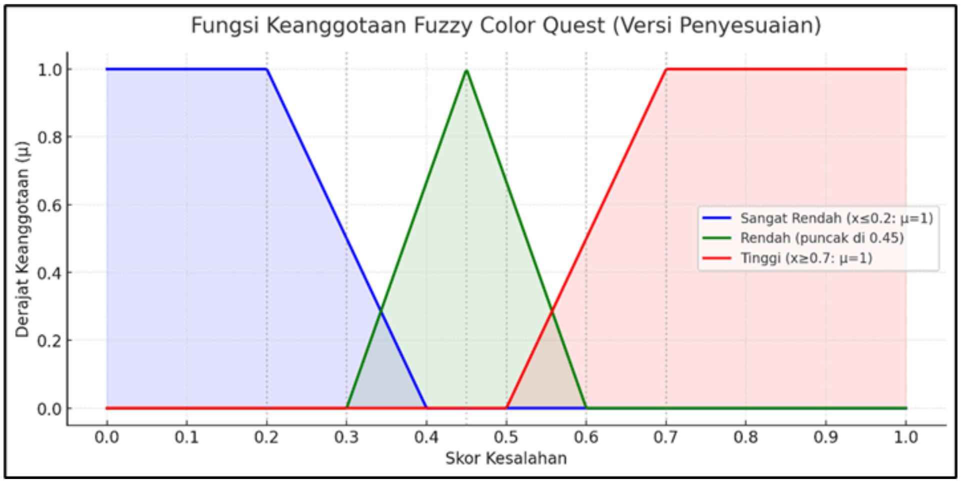


This block diagram illustrates the Fuzzy Logic architecture used for interpreting gameplay results. The process includes:

1. **Input** : Final scores from Protan, Deutan, Tritan levels.
2. **Fuzzification** : Converting raw scores to membership degrees using triangular functions.
3. **Rule Base + Aggregation** : Applying IF-THEN rules to determine fuzzy implications.
4. **Defuzzification** : Producing a final crisp score (z) via weighted average.

The output is a diagnosis label indicating the likelihood of color vision deficiency.

2.1.6. Membership Function – Error Mapping



This graph shows the fuzzy membership functions used to evaluate error ratios from each level:

1. **Very Low** (μ = 1 if score ≤ 0.2)
2. **Low** (peak μ = 1 at 0.45)
3. **High** (μ = 1 if score ≥ 0.7)

These values are essential in determining how well a player perceives each color group and support nuanced diagnosis through Fuzzy Logic inference.

2.2. System Actors

2.2.1. User Roles and Responsibilities / Authority Requirements

User/Role	Example	Frequency of Use	Security/Access, Features Used	Additional Notes
Student (Child Player)	Kindergarten students (ages 4–6) at Happy Holy Kids	Occasional (1–2 sessions per child)	Full access to gameplay and result screen; cannot access other players' history	Accompanied by a parent or teacher during gameplay
Parent / Teacher	Guardian or kindergarten teacher	Occasional	Can view the child's result summary and diagnosis; no account or settings access	Serves as a companion and supervisor during play
Developer / Evaluator	Game developer, researcher, supervisor	Frequent	Full access to all features, including optional data logs (if enabled)	For debugging, testing, and performance analysis

2.3. Dependencies and Change Impacts

2.3.1. System Dependencies

1. **Unity Engine** (version 2022.3 LTS): used to develop the entire gameplay system, UI, and visual logic.
2. **Android OS 8.0 or above:** minimum supported platform for installation.
3. **Fuzzy Logic & FSM Scripts:** implemented in local C# scripts within the Unity project.
4. **No external database or API:** the system runs entirely offline, with no dependency on network connectivity or cloud-based services

2.3.2. Change Impacts

1. **No impact on external systems:** Color Quest is a self-contained, standalone mobile application with no third-party integrations.
2. However, should the system be extended in the future to include features such as:
   a. Cloud-based history tracking
   b. Multi-device synchronization
   c. Online progress monitoring or reporting

then the following components would be required:

d. Backend database and API service
e. User login and account management module
f. Data authorization and encryption mechanisms

3. Gameplay Specifications

3.1. Game Mode: DifferentColor

3.1.1. Purpose/ Description

The "DifferentColor" mode is designed to evaluate a child's ability to distinguish color variations. The player must tap the circle with a different hue among a group of similar-colored circles. This stage is crucial in screening color vision deficiencies like Protanopia and Deuteranopia.

3.1.2. Use case

UC-01	**Identify Different Color**
Primary Actor(s)	Player (child)
Stakeholders and Interest	Parents, Teachers
Trigger	Player taps "Main Game" from the main menu and enters a name
Pre-conditions	Game is installed, player has access to a touchscreen device
Post-conditions	Player proceeds to the next stage or receives feedback on performance
Main Success Scenario	1. Player launches the game 2. Enters their name 3. Game c**urrent colorblindness level screen** a. Level 1 (Protan) b. Level 2 (Deutan) c. Level 3 (Tritan) 4. Player taps Lanjut (Continue) 5. System loads DifferentColor mode 6. System displays circle grid with 1 different color 7. Player taps a circle a. If correct → logs the score and proceeds to `SameColor`. b. If incorrect: i. Display Alternative Question 1 ii. If still incorrect → Display Alternative Question 2 iii. After max 2 retries → log error, continue to `SameColor`. 8. Save result for fuzzy evaluation 9. After answering correctly or exhausting attempts → system transitions to next mode (SameColor), continuing the gameplay loop.

Extensions	1. If the player selects the wrong answer on first attempt: → System loads **Alternative Question 1** 2. If the second attempt is also wrong: → System loads **Alternative Question 2** 3. After 2 failed retries or upon correct answer at any point: → System proceeds to `SameColor`, logging total error count.
Priority	High
Special Requirements	Touch input, visual accessibility
Open Questions	Should scoring vary per retry? (current implementation: no)

3.1.3. Mock-up

Check on Figma

3.1.4. Functional Requirements

Spec ID	Specification Description	Business Rules/ Data Dependency
DC-01	Display instruction at top of screen	Must show same text for every question
DC-02	Display color circle grid	Always include 1 distinct color (hue/value diff)
DC-03	Validate player's touch input	Record tap position and check match to answer key
DC-04	Retry handling logic	If wrong, show alternative layout (max 2 retries)
DC-05	Store result per question for diagnosis	Connects to Fuzzy Logic input (Protan/Deutan/etc)

3.1.5. Field level specifications

Field Label	UI Control	Editable	Data Type	Validation Rule	Message
Instruction Text	Label	No	String	Always visible	Static text: "Tap the different color"
Color Circles	Touch Area	No	Color + ID	One circle must be different (answer key)	Pulled from color question pool
Retry Counter	Auto Increment	No	Integer	Increase if answer is incorrect	Max 2 retries allowed
Tap Action	Click/Tap	Yes	Input Trigger	Must match correct circle to proceed	Highlighted on tap, triggers retry if wrong

3.2. Game Mode: SameColor

3.2.1. Purpose/ Description

This mode evaluates the child's ability to perceive color similarity. The system displays colored dots in pairs. The player must tap two dots with the same color in a single round. It is the second stage in each colorblindness level (Protan, Deutan, Tritan).

3.2.2. Use case

UC-02	Match Same Color
Primary Actor(s)	Player (child)
Stakeholders and Interest	Parents, Teachers
Trigger	System loads SameColor mode after finishing DifferentColor
Pre-conditions	Player has entered name and passed (or exhausted) the DifferentColor stage
Post-conditions	Player proceeds to SortColor stage
Main Success Scenario	1. System loads SameColor level for current ColorBlindLevel. 2. The player taps two circles. 3. The system checks color matches. a. If correct: proceed to the next round or next stage. b. If incorrect: no retry, error recorded.
Extensions	—
Priority	High
Special Requirements	Must record error if wrong selection; no retry allowed
Open Questions	Should the same color always appear in the same position for every child? (current implementation: no)

3.2.3. Mock-up

Check on Figma

3.2.4. Functional Requirements

Spec ID	Specification Description	Business Rules/ Data Dependency
SC-01	Load paired dots of the same color	Dot colors must be chosen based on colorblind level
SC-02	Allow only two taps per round	Validate pair after second tap
SC-03	Validate player's touch input	CountErrorsForLevel() integrated

3.2.5. Field level specifications

Field Label	UI Control	Editable	Data Type	Validation Rule	Message
Game Grid	Tap Target	Yes	Position	Max 2 taps before submit	"Select two circles"
Retry Counter	Integer View	No	Integer	Auto-incremented if answer is wrong	—

3.3. Game Mode: SortColor

3.3.1. Purpose/ Description

This final mode requires the player to sort colored dots from the darkest to the lightest shade. It reinforces color differentiation and sequencing. Appears after SameColor in every ColorBlindLevel.

3.3.2. Use case

UC-03	Sort Shades
Primary Actor(s)	Player (child)
Stakeholders and Interest	Parents, Teachers
Trigger	Transition from SameColor stage
Pre-conditions	Player completed SameColor stage
Post-conditions	Game proceeds to evaluation result (Fuzzy) or next ColorBlindLevel
Main Success Scenario	1. The system loads a set of 3–5 color variants. 2. The player drags and reorders the colors from darkest to lightest. 3. System validates order. 4. Error recorded if sequence is incorrect. 5. Proceed to the next ColorBlindLevel or show the result.
Extensions	—
Priority	High
Special Requirements	System must evaluate based on defined hue/lightness order
Open Questions	Should order validation be strict (100% match) or allow close approximation?

3.3.3. Mock-up

Check on Figma

3.3.4. Functional Requirements

Spec ID	Specification Description	Business Rules/ Data Dependency
SO-01	Load set of color shades	Shades determined based on colorblind level
SO-02	Allow drag-and-drop reorder	Use SortDot.cs functionality
SO-03	Validate order after confirmation or timeout	Must integrate with CountErrorsForLevel()

3.3.5. Field level specifications

Field Label	UI Control	Editable	Data Type	Validation Rule	Message
Instruction Text	Label	No	String	Always visible	Static text: "Sort from darkest to lightest"
Color Dots (Items)	Draggable Objects	Yes	Color[] + Index	All items must be placed in one of the drop slots	Shades are dynamically generated per level
Drop Area / Slots	Drop Target	Yes	Array[Position]	Each slot must contain one color dot	Number of slots = number of items
Submit Action	Button (Optional)	No	Action Trigger	Validate order (darkest to lightest)	May be triggered automatically after drag complete
Retry Tracker	Integer (internal)	No	Integer	Incremented if order is incorrect	Used for fuzzy logic evaluation (Tritan score)

3.4. FSM – Level Transition

3.4.1. Purpose/ Description

The Level FSM (Finite State Machine) in Color Quest controls the transition between the three diagnostic levels:

1. Protan
2. Deutan
3. Tritan

Each level represents a type of color vision deficiency test. The FSM ensures that the player completes each level in order before reaching the final result screen.

3.4.2. Transition Logic

State	Trigger / Condition	Next State	Description
`StartGame`	Player taps "Start" after name input	`Protan`	Initial entry point for gameplay
`Protan`	All 3 stages completed	`Deutan`	Transitions only after Different, Same, Sort done
`Deutan`	All 3 stages completed	`Tritan`	Same as above
`Tritan`	All 3 stages completed	`ResultScreen`	Triggers evaluation and diagnosis
`ResultScreen`	Player reviews results or exits	`MainMenu`	Final screen before return

3.4.3. FSM Diagram Reference

Refer to Figure 2 – Level FSM Diagram in Section 2.1

The diagram visualizes sequential level flow:
`Start → Protan → Deutan → Tritan → Result`

3.4.4. FSM Code Snippet – Level Transition (`ColorBlindLevel`)

```
public enum ColorBlindLevel
{
        Protan,
        Deutan,
        Tritan,
        Result
}
```

Location:
`GameManager.cs` (Top-level FSM enum used to control current diagnostic level)

Transition handled in:

```
public void NextLevel()
{
    if (colorBlindLevel == ColorBlindLevel.Protan)
        colorBlindLevel = ColorBlindLevel.Deutan;
    else if (colorBlindLevel == ColorBlindLevel.Deutan)
        colorBlindLevel = ColorBlindLevel.Tritan;
    else
        colorBlindLevel = ColorBlindLevel.Result;

    StartLevel(); // Resets stage state and triggers next
level
}
```

Location:

`GameManager.cs`

Function NextLevel() is invoked after the SortColor stage is completed.

3.5. FSM – Stage Transition

3.5.1. Purpose/ Description

The Stage FSM controls the sequence of mini-games within each level. It ensures the player plays:

1. `DifferentColor` (with retry logic),
2. followed by `SameColor`,
3. and then `SortColor`.

This FSM runs independently inside each diagnostic level (Protan/Deutan/Tritan).

3.5.2. Transition Logic

State	Trigger / Condition	Next State	Description
`DifferentColor`	Player completes or retries exhausted	`SameColor`	Supports up to 2 alternative questions before proceeding
`SameColor`	Player completes a match attempt	`SortColor`	No retry; proceeds regardless of correctness
`SortColor`	Player completes drag-and-drop sorting	`NextLevel`	Moves to next ColorBlindLevel or Result if last level

3.5.3. FSM Diagram Reference

Refer to Figure 3 – Stage FSM Diagram in Section 2.1

The diagram shows internal stage control:
`DifferentColor → SameColor → SortColor → [Loop to Next Level]`

3.5.4. FSM Code Snippet – Stage Transition (`StageState`)

```
public enum StageState
{
    DifferentColor,
    SameColor,
    SortColor
}
```

Transition handled in:

```
public void NextStage()
{
    if (stageState ==
StageState.DifferentColor)
        stageState = StageState.SameColor;
    else if (stageState ==
StageState.SameColor)
        stageState = StageState.SortColor;
    else
        GameManager.Instance.NextLevel();
}
```

Location:
`StageManager.cs`
Function `NextStage()` is called after each stage is completed, unless it's the final one (`SortColor`), which then triggers `NextLevel()`

3.6. Fuzzy Logic Evaluation

3.6.1. Purpose/ Description

Color Quest uses a **Fuzzy Logic system** to estimate the likelihood of color vision deficiency (CVD) in children based on their performance in three levels:

1. **Protan** (Red sensitivity)
2. **Deutan** (Green sensitivity)
3. **Tritan** (Blue-yellow sensitivity)

This evaluation provides a **soft diagnosis** using fuzzy inference rather than rigid pass/fail thresholds, making it more adaptable to the playful, variable nature of child responses.

3.6.2. Inputs and Fuzzification

1. Each level (Protan, Deutan, Tritan) yields a **score from 0.0 to 1.0**
2. Scores are converted into **fuzzy sets** using **triangular membership functions:**
   a. Very Low
   b. Low
   c. High
3. Example:
   If `ProtanScore` = 0.3, it belongs partially to both "Very Low" and "Low".

```
// MembershipFunction.cs
public static float VeryLow(float x) => (x <= 0.2f) ? 1f
: (x >= 0.4f) ? 0f : (0.4f - x) / 0.2f;
public static float Low(float x)     => (x <= 0.2f || x
>= 0.7f) ? 0f : (x <= 0.45f) ? (x - 0.2f) / 0.25f :
(0.7f - x) / 0.25f;
public static float High(float x)    => (x <= 0.5f) ? 0f
: (x >= 0.7f) ? 1f : (x - 0.5f) / 0.2f;
```

3.6.3. Inference Rules

A set of **243 IF-THEN** rules are defined to interpret combinations of Protan, Deutan, and Tritan scores.

Example rule:

```
IF Protan IS Low AND Deutan IS Low AND Tritan IS Low THEN
Diagnosis IS Moderate CVD
```

Rule base is implemented in:

```
// FuzzyLogic.cs
```

```
private static List<FuzzyRule> rules = new
List<FuzzyRule>()
{
    new FuzzyRule("Low", "Low", "Low", 60), // example
    ...
};
```

3.6.4. Defuzzification

After inference, the system uses the **Weighted Average** method to produce a final crisp score (`z`), calculated by:

```
// Defuzzification formula
z = Σ(w_i × z_i) / Σ(w_i)
```

This final score `z` determines the diagnosis category.

3.6.5. Output Diagnosis

After inference, the system uses the **Weighted Average** method to produce a final crisp score (`z`), calculated by:

Z-Score Range	Diagnosis Label
0–40	High Risk of Color Vision Deficiency
41–60	Moderate Risk / Potential Mixed Deficiency
61–80	Uncertain / Suggest Recheck
81–100	Likely Normal Color Vision

The result is displayed on the **Result screen**, accompanied by visual and verbal feedback.

3.6.6. Code Reference

1. `FuzzyLogic.cs` : Main fuzzy logic processing and rule base
2. `GameManager.cs` : Sends scores to `FuzzyLogic.DetermineDiagnosis()`
3. `MembershipFunction.cs` : Contains membership formulas

4. User Interface Design

This section outlines the visual and interactive elements of the Color Quest game, designed for young children aged 4–6 years. The UI emphasizes simplicity, large touch targets, bright contrasting colors, and clear visual feedback through icons and animations. Each screen is structured to guide the child intuitively through the game with minimal reading required.

4.1. UI Components & Navigation Structure

Screen Name	Description
Opening Screen	Initial splash screen with logo and loading indicator.
Main Menu	Contains 4 main options: Start Game, How to Play, History, Exit.
Name Entry	Simple input form for child's name (one text field + start button).
Level Intro	Display current level (Protan/Deutan/Tritan) before gameplay starts.
Game Screens	Three game modes: DifferentColor, SameColor, SortColor (one per screen).
Result Screen	Displays Fuzzy Logic-based diagnosis and summary of child’s score.
History Screen	List of previous results, only if stored locally.
How to Play	Illustrated explanation of gameplay using minimal text.

4.2. Visual & Audio Elements

Element	Details
Mascot	A cheerful character guides the user with gestures and expressions.
Color Palette	RGB base with variations adjusted for each level (Protan/Deutan/Tritan).
Feedback UI	Correct answers trigger next question
Music	Background music loops calmly during menu; muted during gameplay.
Sound Effects	Tap sounds, correct/wrong chimes, voice prompts.
Result Screen	Displays Fuzzy Logic-based diagnosis and summary of child’s score.
History Screen	List of previous results, only if stored locally.
How to Play	Illustrated explanation of gameplay using minimal text.

4.3. UI Flow (Sitemap)

```
Opening Screen
    ↓
Main Menu
 ├── How to Play
 ├── Start Game
 │      ↓
│   Name Entry
│     ↓
│   Level Intro (Protan)
│     ↓
│   DifferentColor
│     ↓
│   SameColor
│     ↓
│   SortColor
│     ↓
│   Level Intro (Deutan) → (repeat flow)
│     ↓
│   Level Intro (Tritan)
│     ↓
│   Result Screen
└── History
```

UI Design Guidelines Used

1. **Touch-first** layout (minimum 48x48dp targets)
2. **Text size:** Large and bold (20pt or higher)
3. **Icons:** Lucid and recognizable by age 4+
4. **Color:** Contrast-checked; no reliance on color alone for feedback
5. **Navigation:** One clear action per screen, limited decision points

5. Diagrams And Data

This section presents the key diagrams and data structures used in the Color Quest application. Each diagram provides a visual representation of system flow, gameplay control, fuzzy logic evaluation, and data processing that supports the overall functionality of the game. All referenced visuals in this section can be found in **Section 2.1 – Context Diagram, Interface Diagram, and Process Flow**.

5.1. Application Screen Flow

This diagram illustrates the primary navigation structure of Color Quest. It shows how users move through the application starting from the Opening screen to the Result screen, including optional flows like How to Play and History.

📌 *See: Figure 1 – Application Screen Flow (Section 2.1)*

5.2. FSM Diagrams

The Color Quest gameplay is governed by a dual-layer Finite State Machine (FSM) structure that ensures organized level progression and mini-game sequencing.

5.2.1. Level FSM Diagram

Controls progression between colorblind diagnostic levels in this order:

`Protan → Deutan → Tritan → Result`

Transition is triggered by `NextLevel()` once all stages in the current level are completed.

📌 *See: Figure 2 – Level FSM Diagram (Section 2.1)*

5.2.2. Stage FSM Diagram

Controls the mini-game sequence within each level:

`DifferentColor → SameColor → SortColor`

Transitions are managed via `StartStage()` and `NextStage()` methods in `StageManager.cs`.

📌 *See: Figure 3 – Stage FSM Diagram (Section 2.1)*

5.2.3. Alternative Question Handling

The `DifferentColorstage` uses a retry mechanism to reduce random guessing:

1. 1st wrong ⟶ show Alternative 1
2. 2nd wrong ⟶ show Alternative 2
3. Then continue to `SameColor`

📌 *See: Figure 4 – Alternative Flow Diagram (Section 2.1)*

5.3. Fuzzy Logic Evaluation Architecture

Color Quest interprets gameplay results using a fuzzy logic controller. The process follows these steps:

1. **Input:** Raw scores from Protan, Deutan, Tritan
2. **Fuzzification:** Translates scores into membership degrees
3. **Rule Evaluation:** Applies IF-THEN rules
4. **Defuzzification:** Converts result to crisp value (z)
5. **Output:** Diagnosis label (e.g., High Risk, Moderate Risk, Normal)

📌 *See: Figure 5 – Fuzzy Architecture Diagram (Section 2.1)*

5.4. Membership Functions

Triangular membership functions are used to evaluate score quality:

Label	Range	Description
Very Low	0.0 – 0.4	Indicates very few errors (high accuracy)
Low	0.2 – 0.7 (peak 0.45)	Intermediate error zone
High	0.5 – 1.0	Indicates many errors (low accuracy)

These functions support nuanced decision-making in the fuzzy system.

📌 *See: Figure 6 – Membership Function Graph (Section 2.1)*

6. Testing Strategy & Feedback

This section outlines the testing approach used to validate the Color Quest system, including internal logic, user interface functionality, gameplay flow, and fuzzy evaluation accuracy. The goal is to ensure the game is robust, age-appropriate, and delivers consistent diagnostic feedback.

6.1. Testing Types

Test Type	Purpose
Unit Testing	Verifies individual functions (e.g., score conversion, fuzzy membership).
Black Box Test	Ensures UI and gameplay features function correctly as expected.
Alpha Testing	Internal test among developer and academic peers to find major bugs.
Beta Testing	Real-world test with target users (children aged 4–6 in kindergarten).
Usability Testing	Observes how children interact with the game and identifies pain points.

6.2. Unit Test Coverage

Unit tests were conducted on the following critical components:

Function	Location	Test Focus
`ConvertToFuzzyScale()`	`GameManager.cs`	Converts error count to fuzzy input
`DetermineDiagnosis()`	`FuzzyLogic.cs`	Fuzzy inference and defuzzification
`CountErrorsForLevel()`	`GameManager.cs`	Calculates score per level
`MembershipFunction.*()`	`MembershipFunction.cs`	Validates fuzzy membership output

All tests passed with expected results across test cases covering edge values (0.0, 0.5, 1.0).

6.3. Level FSM Diagram

The following screens and actions were tested using black-box techniques:

1. Navigation (Main Menu → Game → Result → Back)
2. Input handling (name field, taps, drag-drop)
3. Visual and audio feedback (correct/wrong)
4. Result interpretation display

No major logic bugs found. Minor alignment issues on low-resolution screens were corrected.

6.4. Beta Testing Feedback

Test Group

Children aged 4–6 from TK Happy Holy Kids Pondok Indah, with assistance from parents and teachers.

Method

1. Individual play sessions during break time
2. Observation + guided interview
3. Rewards (e.g., snacks) given to encourage participation

Findings

Area	Observation
Game Flow	Easy to follow; most children understood tap and drag mechanics
Engagement	Mascot and color visuals improved attention span
Challenge	Some children tapped randomly, but retry logic helped balance
Feedback Clarity	Clear indicators (sound + icon) made correct answers recognizable

6.5. Diagnostic Validation

Although the system does **not provide a clinical diagnosis**, it serves as an early screening tool. In consultation with a local general practitioner (*dr. Suryadi – Bekasi*), the logic was confirmed to **align with basic indicators of color vision response**.

6.6. Known Limitations

1. Does not detect partial/mild CVD types with high precision.
2. Relies on color perception via screen; may vary by device.
3. No cloud data sync or long-term tracking.

7. Release Plan & Wrap-Up

This section summarizes the deployment strategy, versioning, and completion status of the Color Quest application. It also includes notes on post-release evaluation and next steps for potential improvements.

7.1. Version Information

Version	Status	Description	Date
1.0.0	Prototype Draft	Initial gameplay logic and FSM structure, no UI design applied	Sep 30, 2024
2.0.0	Prototype Update	Added fuzzy scoring logic, UI placeholders used	Oct 8, 2024
4.0.0	Internal Alpha	Full logic implemented, basic navigation flow, minimal UI	Oct 30, 2024
5.0.0	UI Milestone	First version with finalized child-friendly UI and mascots	Nov 4, 2024
7.0.0	Usability Test	Version tested with children, added retry mechanism refinement	Dec 9, 2024
8.0.0	Pre-release QA	Full gameplay + fuzzy evaluation tested internally	Jan 13, 2025
9.0.0	Final Release	Stable version for PlayStore internal testing	Apr 30, 2025

Versions 3.0.0 and 6.0.0 were skipped during development due to internal branching or merged improvements from previous iterations.

7.2. Post-Release Activities

1. **Playstore listing preparation** (if public release is planned)
2. **Bug fix monitoring** via observation and user feedback
3. **Future version consideration:**
    a. Add voice-over instructions
    b. Dynamic difficulty adjustment based on age
    c. Cloud-based result saving for teacher/parent tracking

8. Appendix

8.1. Author Responsibility

Name	Role(s)	Responsibility
Ardella Malinda Sarastri	Game Designer, Developer, Analyst, QA	Responsible for the full development cycle of Color Quest, including gameplay design, implementation, testing, and documentation as part of an individual academic thesis project.

8.2. Acknowledgements

This project was made possible with the voluntary participation of:

1. Children and teachers at **TK Happy Holy Kids Pondok Indah,** who assisted during playtesting.
2. **dr. Suryadi,** a general practitioner, who provided external insight on the logic used for color vision screening.
3. Academic supervisors and advisors who guided the direction of the research and documentation.

Note: All development and implementation were completed individually by the author.

8.3. Approval Record

Prepared By	Date
Ardella Malinda Sarastri	Apr 15, 2025
Verified By	**Date**
(Academic Supervisor)	(TBD)