Super Mario

Mobile Application

Submitted By:

Hassan Rajput - 22SW108

Dayaan Mughal - 22SW150

Submitted To:

Ma’am Mariam Memon

**1. Real World Problem Identification**

**Problem Statement**

Need for an interactive mobile game that:

* Provides entertainment
* Offers a simple, engaging mechanic
* Tracks player progress over time
* Runs efficiently on mobile devices
* Offers Nostalgic experience

**Key Requirements**

1. User engagement: simple controls and clear objectives
2. Progress tracking: persistent high scores across sessions
3. Performance: smooth animations and responsive controls
4. Accessibility: easy to learn, suitable for various age groups
5. Cross-platform support: works on Android, iOS, and other platforms

**Target Audience**

* Casual mobile gamers
* Players seeking quick sessions
* Ages 8+

**2. Proposed Solution**

**Solution Overview**

Developed a Super Mario-themed collection game using Flutter, where players control Mario to collect mushrooms within a 60-second timer.

**Core Features**

1. Character control

* Left/right movement with running animation
* Jumping with parabolic physics
* Visual feedback for actions

1. Game mechanics

* Collect mushrooms to increase score
* 60-second timer
* Random mushroom spawns
* Character growth with each collection

1. User interface

* On-screen directional controls
* Real-time stats (score, time, high score)
* Retro-style typography (Press Start 2P)
* Landscape orientation

1. Progress tracking

* High score with date stamp
* Persistent storage across sessions

**Technical Architecture**

* Framework: Flutter (Dart)
* Architecture: Stateful widget with State management
* Physics: Custom jump mechanics using parabolic equations
* Animation: Timer-based position updates with smooth transitions

**3. Data Storage**

**Selected Database: Hive**

Hive was chosen as the local NoSQL key-value database.

**Justification**

1. Flutter-optimized

* Works well with Flutter
* Written in Dart, no platform channels
* Fast on-device access

1. Performance

* Fast reads/writes
* Suitable for small datasets
* Minimal overhead

1. Offline-first

* No internet required
* Data stored locally
* Fits game session needs

1. Lightweight

* Small footprint
* Simple setup
* No external dependencies

1. Type safety

* Code generation with hive\_generator
* Type-safe operations
* Autocomplete support

1. Ease of use

* Simple initialization
* Straightforward save/load API
* Clear documentation

1. Learning Purposes

* Just to learn new things that make us more skilled

**Implementation Details**

**Data Model:**

A screen shot of a computer program

AI-generated content may be incorrect.

**User Interface:**

A screen shot of a video game

AI-generated content may be incorrect.

**Storage Operations:**

* Write: progressBox.put('highScore', progress)
* Read: progressBox.get('highScore')
* Persistence: Automatic with flush operations

**Alternative Considered: SharedPreferences**

* SharedPreferences: limited type support and less efficient for structured data
* Decision: Hive was chosen for better performance and type safety

**4. APIs/Packages/Plug-ins**

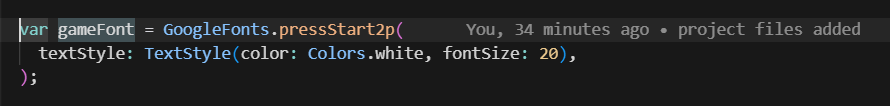
**4.1 Google Fonts (google\_fonts: ^6.3.2)**

**Purpose:** Retro-style typography

**Justification:**

* Press Start 2P font matches classic game aesthetics
* Ease of integration
* Dynamic font loading
* Good cross-platform support

**Usage:**

****

**4.2 Hive (hive: ^2.2.3)**

**Purpose:** Local database for high score persistence

**Justification:**

* Fast, lightweight NoSQL database
* Native Dart, good performance
* Type-safe with code generation
* Straightforward API

**Usage:**

* Storing and retrieving high scores
* Persisting game progress

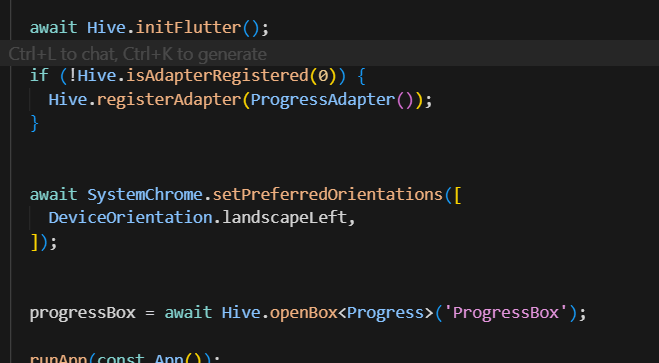
**4.3 Hive Flutter (hive\_flutter: ^1.1.0)**

**Purpose:** Flutter-specific Hive initialization

**Justification:**

* Simplifies Flutter setup
* Handles initialization paths
* Provides proper file system access

**Usage:**



**4.4 Hive Generator (hive\_generator: ^2.0.1)**

**Purpose:** Code generation for Hive adapters

**Justification:**

* Generates type adapters automatically
* Reduces boilerplate
* Ensures type safety
* Standard practice with Hive

**Usage:**

* Generates ProgressAdapter from Progress model annotations

**4.5 Build Runner (build\_runner: ^2.4.6)**

**Purpose:** Code generation tool

**Justification:**

* Required for Hive generator
* Handles code generation workflows
* Standard in Dart projects

**5. Issues and Bugs Encountered and Resolved During Development**

**Issue 1: Hive Box Initialization Errors**

**Problem:** App crashed when accessing Hive box before initialization or after the box was closed.

A screen shot of a computer code

AI-generated content may be incorrect.

**Root Cause:** Async initialization timing and missing null checks.

**Solution:**

*// Added comprehensive initialization check*

*A black screen with yellow and blue text

AI-generated content may be incorrect./*

*/ Added box open check before operations*

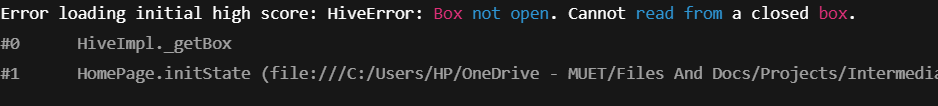
A screen shot of a computer program

AI-generated content may be incorrect.

**Result:** Stable initialization and reliable data access.

**Issue 2: High Score Not Persisting Across Sessions**

**Problem:** High score reset on app restart.



**Root Cause:**

* Missing async initialization handling
* Not loading data in initState()

**Solution:**

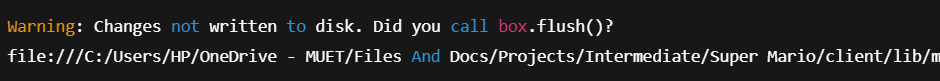
**Result:** High score persists correctly across sessions.

A screenshot of a computer program

AI-generated content may be incorrect.

**Issue 3: Score Not Saving During Gameplay**

**Problem:** High score updated in memory but not saved immediately.



**Root Cause:** Missing explicit flush operations.

**Solution:**

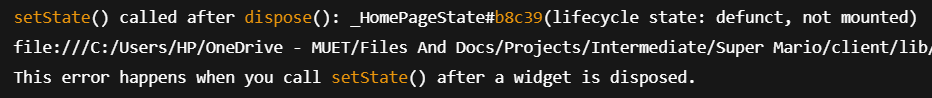
A black background with white text

AI-generated content may be incorrect.

**Result:** Scores saved reliably and persist.

**Issue 4: Timer Not Canceling on Game End**

**Problem:**Timer continued after game ended, causing UI updates after end.



**Root Cause:**Timer not canceled in cleanup.

**Solution:**

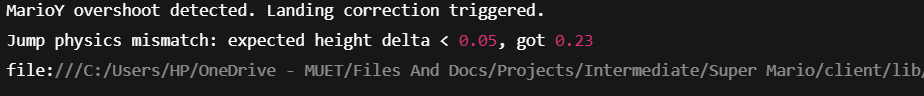
A computer screen with text

AI-generated content may be incorrect.

**Result:** Timer stops correctly; no memory leaks.

**Issue 5: Jump Physics Not Smooth**

**Problem:** Jump felt inconsistent.



**Root Cause:** Timer interval and physics calculations mismatch.

**Solution:**

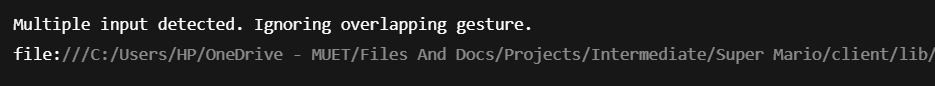
**Result:** Smooth, predictable jumps.

A computer screen shot of a program code

AI-generated content may be incorrect.

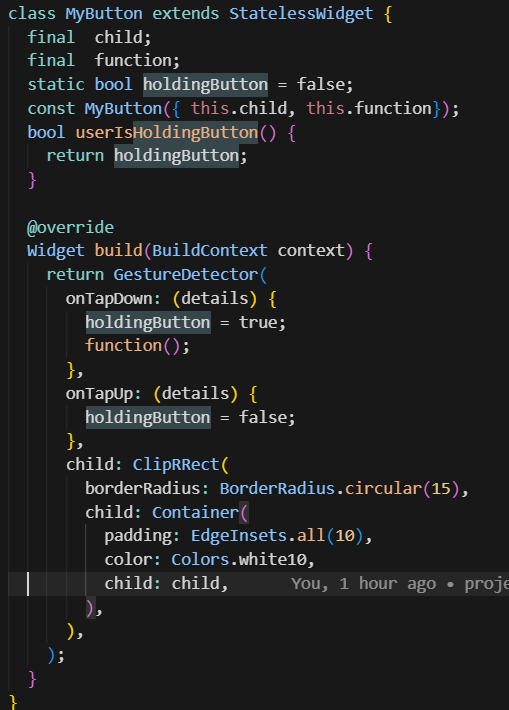
**Issue 6: Multiple Simultaneous Button Presses**

**Problem:** Holding multiple buttons caused conflicts.



**Root Cause:** No state management for simultaneous inputs.

**Solution:**



**Result:** Clean handling of simultaneous inputs.

**Issue 7: Character Base Height Calculation**

**Problem:**Character size increased, but base height didn’t update, causing visual misalignment.**Symptoms:**

* Character appeared floating after collecting mushrooms
* Ground collision incorrect

**Root Cause:**Base height not recalculated after size changes.

**Solution:**

A black background with white text and colorful letters

AI-generated content may be incorrect.

**Result:** Proper alignment regardless of character size.