GoopOnTheGo

Team Project Report

CSD3156 Mobile and Cloud Computing
Spring 2026

Team Information

Edwin Lee Zirui (2301299)
(Name) — (SIT ID)
(Name) — (SIT ID)
(Name) — (SIT ID)
(Name) — (SIT ID)

GitHub Repository (link)
App Demo Video (Link)
Presentation Video (Link)

Team Contributions

Member	Contributions
Edwin	AR camera pipeline, colour detection CV algorithm, evolution system fix, habitat map, database design, CI/CD setup, report
Jeremy	report
Kai Rui	report
Aqif	report
Ridhwan	report

Table of Contents

1. Introduction & Game Overview

GoopOnTheGo is a location-based creature collection game for Android, inspired by Pokemon Go. Players scan their physical environment using the device camera to spawn and catch creatures called "Goops." Each creature type is linked to a real-world colour signature detected via the camera in real time. Players build a collection, evolve creatures by merging duplicates, fuse two different types into hybrid creatures, and track progress through achievements and daily challenges.

Core Game Loop

Camera Scan → Colour Detected → Creature Spawns → Tap to Catch → Collect → Evolve / Fuse

Creature System

- **19 creatures** (15 + 4) across 5 base types (Water, Fire, Nature, Electric, Shadow) and 4 hybrid fusion types
- **3-stage evolution chains** — e.g. Droplet Goop → Aqua Goop → Tsunami Goop
- **4 fusion combinations** producing hybrid creatures — e.g. Water + Fire = Steam Goop

Key Features Summary

Feature	Description
AR-style Camera Scanning	Real-time colour detection triggers creature spawns
Catch Mechanic	Tap creature within radius; rarity determines success rate
Creature Collection	Filterable inventory with nicknames and favourites
Evolution	Collect 3 identical creatures and merge them to evolve via the Evolution screen or Creature Detail screen
Fusion	Combine 2 compatible types into a hybrid creature
Habitat Map	GPS-based map showing nearby creature habitat zones
Daily Challenges	Time-limited objectives refreshed each midnight
Achievements	10 persistent achievements across 5 categories
Login Streaks	Consecutive daily login tracking

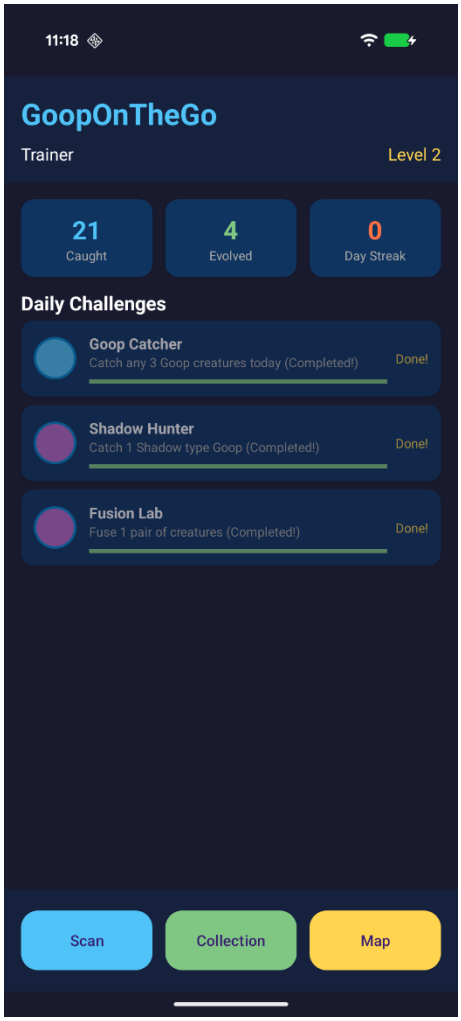
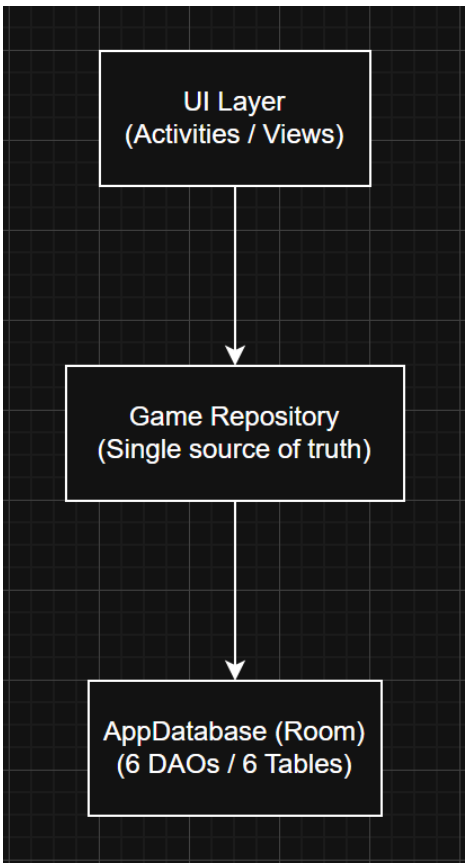


Figure 1: Home screen showing player stats and daily challenges

2. Software Architecture

The app follows an **MVVM-adjacent architecture** using the Repository pattern with Android Jetpack components.

Architecture Diagram

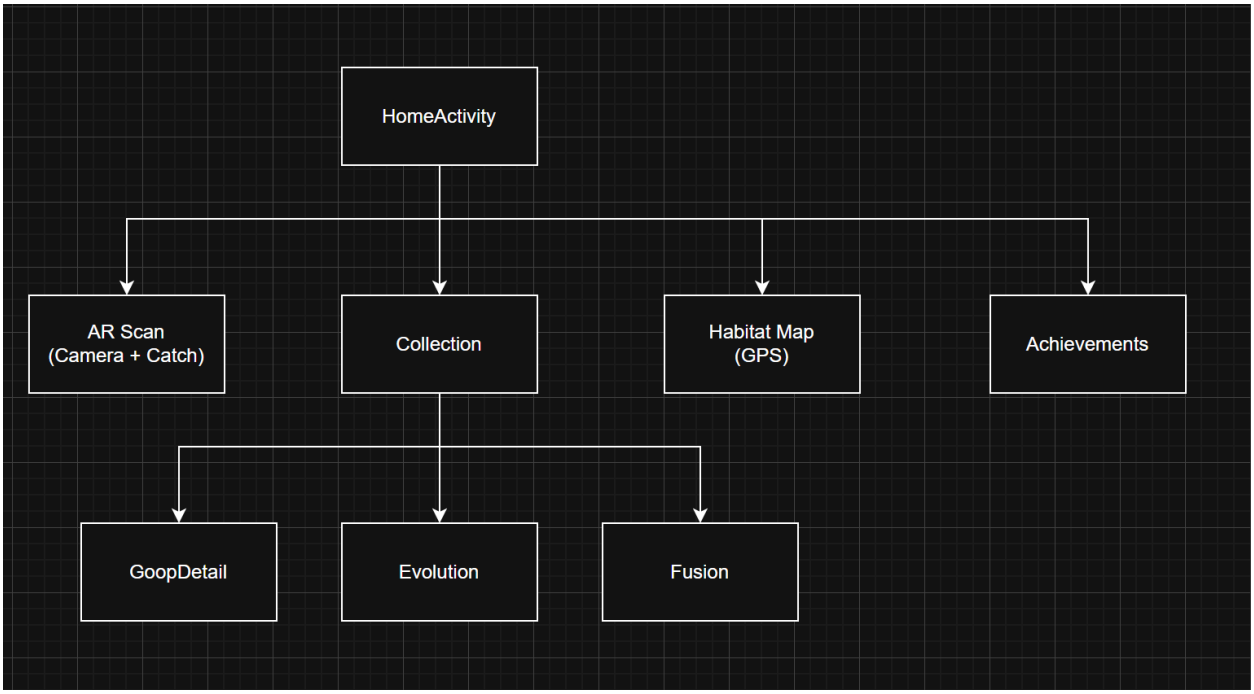


Architectural Decisions

Pattern	Implementation	Rationale
Repository Pattern	GameRepository.kt — centralises all data access and game logic	Decouples UI from data layer
Reactive Data	Kotlin Flow<> from Room DAOs	UI auto-updates on data change without polling
Coroutines	lifecycleScope.launch, Dispatchers.IO / Main	Non-blocking database and camera operations
Singleton App	GoopApplication holds database and repository lazily	Shared state across all Activities
Custom Views	AROverlayView, HabitatMapView	Specialised canvas rendering without a game engine

Application Entry: GoopApplication.kt initialises the Room database and GameRepository lazily, making them available to all Activities via (application as GoopApplication).repository.

Navigation Flow



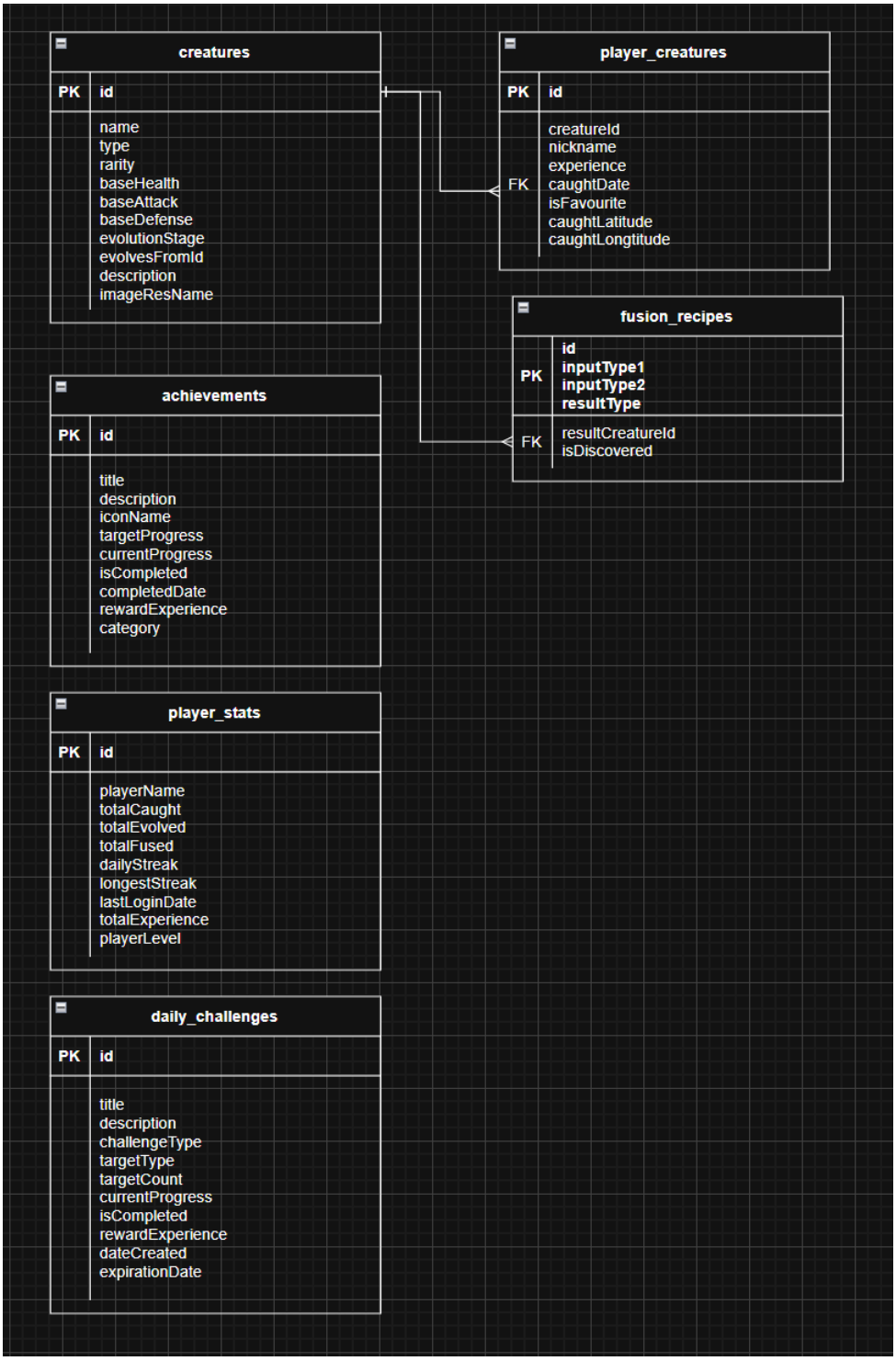
Activities Overview

Activity	Purpose
HomeActivity	Main hub — player stats, daily challenges, navigation
ARScanActivity	Camera preview, colour analysis, creature spawning, catch mechanic
CollectionActivity	RecyclerView grid (3 columns), type and favourite filtering
CreatureDetailActivity	Creature stats, nickname editing, evolve/release actions
EvolutionActivity	Lists creatures with 3+ copies ready to evolve; shows creature images in evolution preview card
FusionActivity	Two-slot selection for type fusion
HabitatMapActivity	Live GPS tracking, habitat zone detection
AchievementsActivity	Achievement list with progress bars

3. Database Design

The app uses **Room ORM** on top of SQLite with a database named gooponthego_database (schema version 2), comprising **6 entity tables**.

3.1 Entity-Relationship Overview



3.2 Table Definitions

creatures — Master creature list (19 rows seeded on first launch)

Column	Type	Constraints	Purpose
id	LONG	PK, autoGenerate	Unique creature ID
name	STRING	NOT NULL	Display name
type	STRING	NOT NULL	GoopType enum via TypeConverter
rarity	INT	NOT NULL	1–5 (1 = common, 5 = legendary)
baseHealth	INT	NOT NULL	Base HP stat
baseAttack	INT	NOT NULL	Base attack
baseDefense	INT	NOT NULL	Base defense
evolutionStage	INT	NOT NULL	1, 2, or 3
evolvesFromId	LONG	NULLABLE	Reference to parent form
evolvesToId	LONG	NULLABLE	Reference to next stage
description	STRING	NOT NULL	Dex entry text
imageResName	STRING	NULLABLE	Drawable resource name

player_creatures — Player's caught creature instances

Column	Type	Constraints	Purpose
id	LONG	PK, autoGenerate	Unique instance ID
creatureId	LONG	FK → creatures.id, CASCADE DELETE	Base creature reference
nickname	STRING	NULLABLE	Player-assigned name
experience	INT	DEFAULT 0	Accumulated XP
caughtDate	LONG	DEFAULT now	Epoch timestamp of capture
isFavorite	BOOLEAN	DEFAULT false	Favourite flag
caughtLatitude	DOUBLE	NULLABLE	GPS latitude at capture
caughtLongitude	DOUBLE	NULLABLE	GPS longitude at capture

player_stats — Singleton player progress (always id = 1)

Column	Type	Purpose
totalCaught / Evolved / Fused	INT	Cumulative event counters
dailyStreak / longestStreak	INT	Login streak tracking
lastLoginDate	LONG	For streak day-diff calculation
totalExperience	INT	Account-level XP
playerLevel	INT	Derived: (totalExperience / 1000) + 1

daily_challenges — Time-limited objectives (expire at midnight)

Column	Type	Purpose
challengeType	STRING (Enum)	CATCH_ANY, CATCH_TYPE, EVOLVE, FUSE, SCAN_LOCATIONS
targetType	STRING?	GoopType for type-specific challenges
targetCount / currentProgress	INT	Goal vs current progress
expirationDate	LONG	Next day 00:00:00 epoch
rewardExperience	INT	XP on completion

achievements — Persistent unlock system (10 seeded achievements)

Column	Type	Purpose
category	STRING (Enum)	COLLECTION, EVOLUTION, EXPLORATION, DAILY, SPECIAL
targetProgress / currentProgress	INT	Goal tracking
completedDate	LONG?	Null until completed
rewardExperience	INT	XP reward

fusion_recipes — 4 fusion combination rules

Column	Type	Purpose
inputType1 / inputType2	STRING	GoopType ingredients
resultType	STRING	Output hybrid type
resultCreatureId	LONG	FK to resulting creature
isDiscovered	BOOLEAN	Revealed to player on first use

3.3 Fusion Recipes

Type 1	Type 2	Result
WATER	FIRE	STEAM
ELECTRIC	NATURE	LIGHTNING_PLANT
FIRE	NATURE	MAGMA
WATER	ELECTRIC	ICE

3.4 Notable Queries

Bidirectional Fusion Recipe Lookup (FusionRecipeDao):

```
SELECT * FROM fusion_recipes
WHERE (inputType1 = :type1 AND inputType2 = :type2)
  OR (inputType1 = :type2 AND inputType2 = :type1)
```

This matches a recipe regardless of the order the two creature types are passed in.

Joined Collection Query (PlayerCreatureDao):

```
SELECT * FROM player_creatures
INNER JOIN creatures ON player_creatures.creatureId = creatures.id
```

Returns all player creatures with their full base creature data in a single transaction.

3.5 Type Conversion

Converters.kt provides Room with @TypeConverter functions to persist the GoopType enum as a String in SQLite and restore it on read.

4. Mobile Features Implemented

The app employs **six** advanced mobile features, exceeding the required minimum of three.

4.1 Camera / Computer Vision

Classes: ARScanActivity.kt, inner class ColorAnalyzer

CameraX ImageAnalysis with a single-thread ExecutorService processes camera frames continuously. A custom pixel-sampling and colour-classification algorithm detects the dominant environmental colour and maps it to a GoopType. A 2-second temporal hold filter prevents accidental spawns from transient colour changes. *(Full algorithm documented in Section 6 — Bonus Component.)*

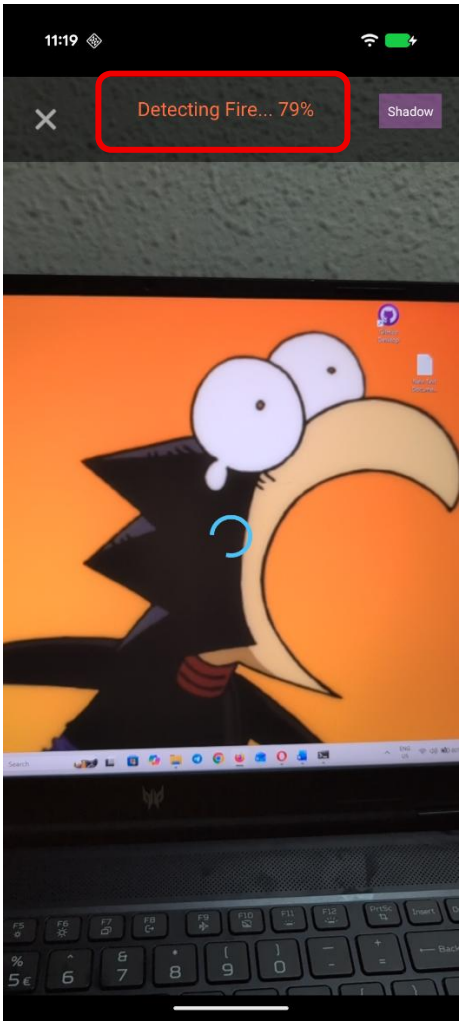


Figure 2: AR scanning screen detecting environmental colour — 2-second temporal hold timer in progress before creature spawn

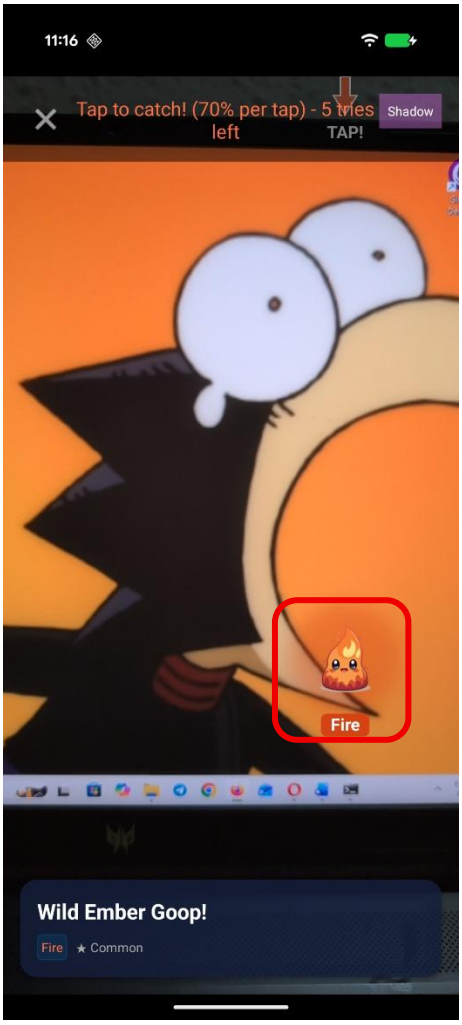


Figure 3: AR scanning screen showing a creature spawned via real-time colour detection

4.2 GPS Location Services

Class: HabitatMapActivity.kt

The **Fused Location Provider** (com.google.android.gms:play-services-location) tracks the player's position with update interval of 5 seconds and minimum update interval of 2 seconds. Permissions: ACCESS_FINE_LOCATION and ACCESS_COARSE_LOCATION. GPS coordinates feed into the geohashing algorithm to determine the local habitat type, displayed on the custom canvas map.

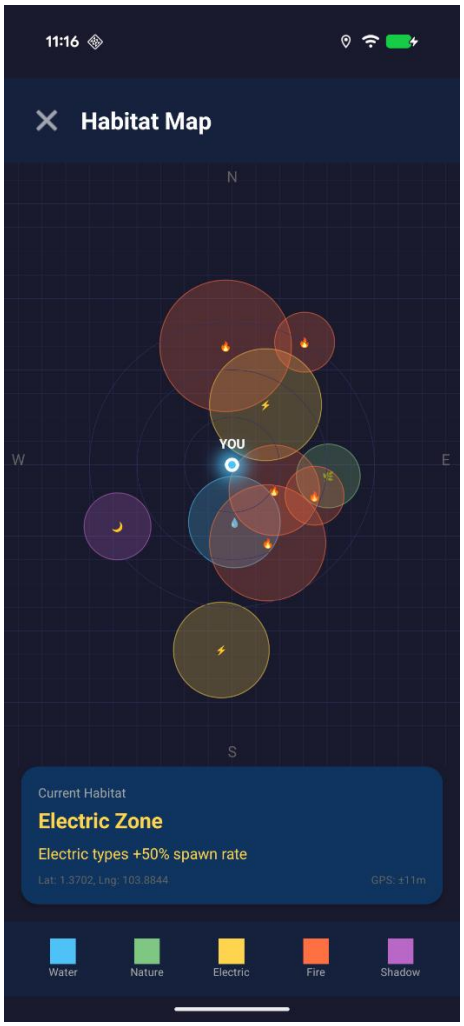


Figure 4: Habitat map showing GPS-tracked player position and habitat zones

4.3 SQLite Database (Room ORM)

Classes: AppDatabase.kt, 6 DAOs, GameRepository.kt

A full relational database with **6 tables**, **foreign key constraints**, **cascade deletion**, **type converters**, and **23 queries** across 6 DAOs. The database is auto-initialised with 19 creatures, 4 fusion recipes, and 10 achievements via RoomDatabase.Callback.

4.4 Canvas-Based 2D Graphics

Classes: AROverlayView.kt, HabitatMapView.kt

Two custom View subclasses render game graphics entirely via the Android **Canvas API** — no external game engine is used. AROverlayView renders animated creatures with procedural blob shapes, layered glow effects, and a directional catch arrow. HabitatMapView renders a live map with habitat zones, a player position marker, a grid background, and a compass.

4.5 Animations

Class: AROverlayView.kt

Two ValueAnimator instances drive continuous creature animations:

- **Bounce:** 1000 ms cycle, AccelerateDecelerateInterpolator, plus/minus 25 px vertical displacement, INFINITE / REVERSE repeat
- **Glow pulse:** 1500 ms cycle, 0–30 px BlurMaskFilter radius expansion, INFINITE / REVERSE repeat

Both animators call invalidate() on each frame to trigger canvas redraw.

4.6 Multi-Threading / Coroutines

All database operations run on Dispatchers.IO via Kotlin coroutines (lifecycleScope.launch). Camera frame analysis runs on a dedicated ExecutorService. Results are dispatched back to Dispatchers.Main for UI updates. The main thread is never blocked.

5. Technical Implementation Details

5.1 Evolution System

Evolution uses a **merge-to-evolve** mechanic. The player must own **3 identical base-stage creatures**. All 3 instances are deleted from the database and one evolved-stage creature is inserted. The evolved creature inherits the nickname of the original.

- Requirement: sameCreatures.size >= 3
- Action: delete instances[0..2], insert PlayerCreature(evolution.id)
- XP reward: 50 XP added to player_stats

Evolution can be triggered from two places in the app:

1. CreatureDetailActivity — when viewing a specific Goop, a progress bar shows X/3 copies owned. The Evolve button enables automatically once 3 copies are collected.
2. EvolutionActivity — a dedicated screen accessible from the Collection screen. It displays all creature types where the player owns 3 or more copies, showing the actual creature image alongside the evolved form in a preview card. Previously this screen used an XP-based filter which never surfaced any creatures; this was identified and fixed so the screen now correctly reflects the merge-to-evolve mechanic.

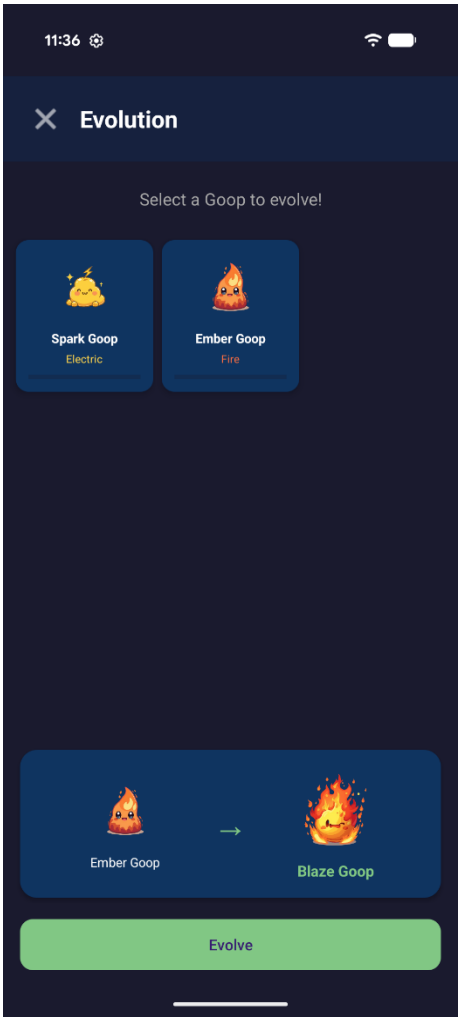


Figure 5: EvolutionActivity showing eligible creatures and evolution preview with creature images

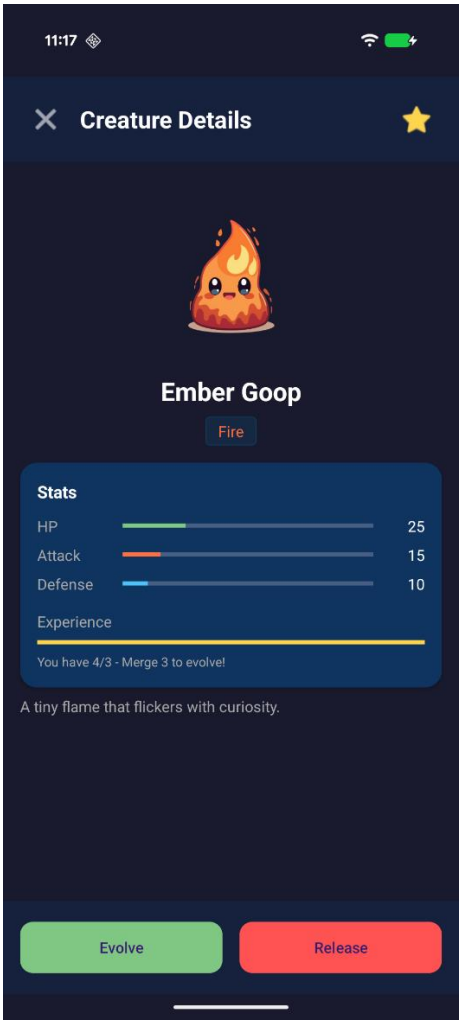


Figure 6: CreatureDetailActivity displaying X/3 evolution progress bar with Evolve button enabled

5.2 Fusion System

Two creatures of compatible types are selected by the player. FusionRecipeDao performs a bidirectional lookup for a matching recipe. Both source creatures are deleted and one hybrid creature is inserted with averaged experience. The recipe is flagged isDiscovered = true on first successful use.

- Fused XP = (experience1 + experience2) / 2
- XP reward: 75 XP added to player_stats

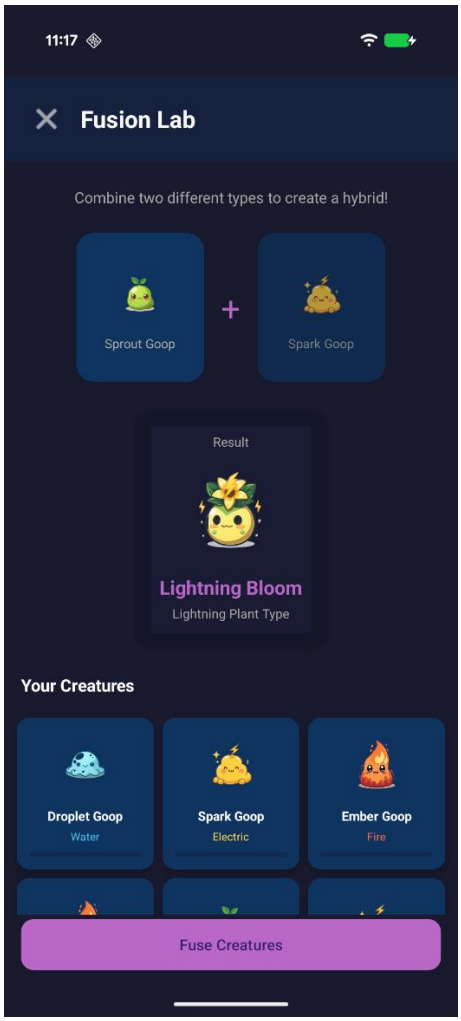


Figure 7: FusionActivity showing two compatible creature slots selected for fusion

5.3 Daily Challenge System

Challenges are generated fresh when no active (non-expired) challenges exist. Three challenges are created with expiration set to the next day at 00:00:00. Progress is updated by event hooks inside GameRepository that fire on catch / evolve / fuse actions.

Default Daily Challenges:

Title	Type	Target	Reward
Catch 3 Goops	CATCH_ANY	3	50 XP
Water Hunter	CATCH_TYPE (WATER)	2	75 XP
Evolution Time	EVOLVE	1	100 XP

5.4 Daily Login Streak

The streak system compares Calendar.DAY_OF_YEAR and Calendar.YEAR between the last login timestamp and the current time:

- Same day: streak unchanged
- Next consecutive day: streak incremented by 1
- Gap of 2+ days: streak resets to 1
- Cross-year transitions handled by comparing both year and day-of-year fields

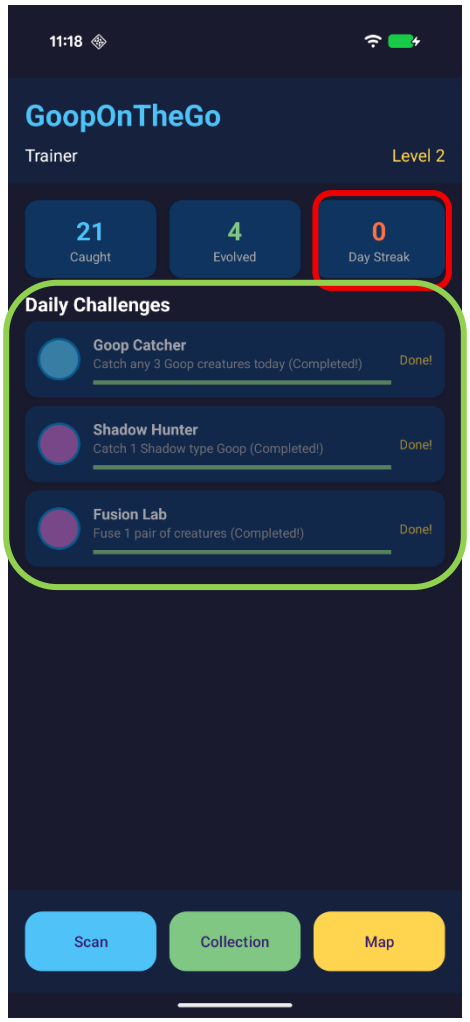


Figure 8: Home screen zoomed in on the daily challenges and daily login sections

5.5 Achievement System

10 achievements across 5 categories (COLLECTION, EVOLUTION, EXPLORATION, DAILY, SPECIAL). Progress is updated passively on game events and compared against targetProgress. Once isCompleted = true, no further updates occur. Completion awards rewardExperience XP.

Title	Category	Target	Reward
First Catch	COLLECTION	1 catch	50 XP
Collector	COLLECTION	10 catches	100 XP
Goop Master	COLLECTION	50 catches	500 XP
First Evolution	EVOLUTION	1 evolution	100 XP
Evolution Expert	EVOLUTION	10 evolutions	300 XP
Mad Scientist	EVOLUTION	1 fusion	150 XP
Explorer	EXPLORATION	5 locations	200 XP
Dedicated Trainer	DAILY	7-day streak	250 XP
Type Specialist	COLLECTION	5 types caught	300 XP
Challenge Champion	DAILY	10 challenges	200 XP

5.6 Probabilistic Catch Mechanic

Tap detection computes Euclidean distance from the touch point to the creature's current animated position (accounting for live bounce offset). A tap within 90% of the creature radius triggers a catch attempt:

Rarity	Stars	Catch Rate
Common	1 star	70%
Uncommon	2 stars	55%
Rare	3 stars	40%
Epic	4 stars	25%
Legendary	5 stars	15%

Random.nextFloat() < catchRate determines success or failure.

5.7 Creature Type System — GoopType.kt

A Kotlin enum with 10 values, each carrying displayName, primaryColor, secondaryColor, and habitat. A companion object implements getFusionResult(type1, type2) using setOf() for order-independent matching.

Type	Primary Colour	Habitat
WATER	Blue (#4FC3F7)	Near water sources
FIRE	Red-orange (#FF7043)	Warm areas
NATURE	Green (#81C784)	Parks and plants

Type	Primary Colour	Habitat
ELECTRIC	Yellow (#FFD54F)	Near electronics
SHADOW	Purple (#BA68C8)	Dark areas
STEAM	Grey (#B0BEC5)	Misty areas
LIGHTNING_PLANT	Yellow-green (#AED581)	Stormy gardens
MAGMA	Deep orange (#FF5722)	Volcanic regions
ICE	Light blue (#E1F5FE)	Cold areas
VOID	Dark indigo (#311B92)	Unknown

5.8 Custom AR Overlay Graphics — AROverlayView.kt

Procedural Blob Rendering uses a 12-point polygon with sine-wave radial distortion:

```
val angle  = (i.toFloat() / 12) * 2 * PI
val wobble = sin(angle * 3 + System.currentTimeMillis() / 200.0) * 15f
val r      = radius + wobble.toFloat()
val x      = cx + (r * cos(angle)).toFloat()
val y      = cy + (r * sin(angle)).toFloat()
```

- Wobble cycles at approximately 5 Hz (time / 200)
- 3rd-order angular frequency (angle × 3) creates 3 lobes of organic distortion
- Rendered with a drop shadow, primary filled path, and white highlight circle
- Glow effect uses BlurMaskFilter(glowRadius + 20f, BlurMaskFilter.Blur.NORMAL) drawn behind the creature

5.9 Custom Habitat Map — HabitatMapView.kt

Habitat Zone Generation uses random polar coordinates:

```
val angle    = random.nextDouble() * 2 * PI
val distance = 100f + random.nextFloat() * 300f   // 100–400 px from centre
val offsetX  = (cos(angle) * distance).toFloat()
val offsetY  = (sin(angle) * distance).toFloat()
val radius   = 60f + random.nextFloat() * 80f     // 60–140 px zone radius
```

8–12 zones are generated per session. Canvas layers drawn in order: dark grid background, habitat circles, range rings (100/200/300 px), player glow marker, compass labels.

5.10 Geohashing for Habitat Zones — HabitatMapActivity.kt

```
val latZone = (lat * 100).toInt()   // 0.01 degree resolution = approx 1.1 km
val lngZone = (lng * 100).toInt()
val hash    = abs(latZone + lngZone)

return when (hash % 10) {
   0, 1 -> GoopType.WATER
   2, 3 -> GoopType.NATURE
   4, 5 -> GoopType.ELECTRIC
```

```
    6, 7 -> GoopType.FIRE
    else -> GoopType.SHADOW
}
```

The hash is **deterministic** — the same real-world location always produces the same habitat type, creating a consistent geographic distribution of creature spawning zones.

6. Bonus: Computer Vision — Real-Time Colour-Based AR Spawning

This section documents the custom-implemented CV component submitted for the bonus rubric criterion: *"Complex implemented component with special algorithms, or other advanced features e.g. CG, ML, CV, VR, AR (mostly using a library or copying code from somewhere doesn't count). This component should be clearly documented within the report."*

Overview

ARScanActivity.kt implements a real-time computer vision pipeline using CameraX ImageAnalysis to classify the player's physical environment by dominant colour and map it to a creature type. The pixel sampling, saturation filtering, type classification rules, and temporal hold detection are all **custom-implemented logic** — no CV or ML library is used for the classification itself.

Pipeline Architecture

```
Camera Frame (CameraX ImageAnalysis, every 200 ms)
        |
        v
  Bitmap Capture
        |
        v
  Dominant Colour Extraction    <-- Custom sparse pixel sampling
        |
        v
  Saturation Filter             <-- Reject achromatic surfaces
        |
        v
  RGB Channel Classification    <-- Map colour to GoopType
        |
        v
  Temporal Hold Timer           <-- 2-second sustained colour required
        |
        v
  Creature Spawn Trigger        <-- evolutionStage = 1 creatures only
```

Step 1 — Dominant Colour Extraction

The centre region of each camera frame is sampled. Every 4th pixel in both axes is read (1/16th of the region) to balance accuracy with CPU performance on the camera thread:

```
for (x in startX..endX step 4) {
    for (y in startY..endY step 4) {
        val pixel = bitmap.getPixel(x, y)
        rSum += Color.red(pixel)
        gSum += Color.green(pixel)
        bSum += Color.blue(pixel)
        count++
    }
}
return Color.rgb(rSum / count, gSum / count, bSum / count)
```

Step 2 — Saturation Filter

Pixels with low RGB spread are rejected as achromatic. This prevents white walls, grey floors, or overcast skies from falsely triggering a spawn:

```
val max = maxOf(r, g, b)
val min = minOf(r, g, b)
if (max - min < 30) return null   // Too grey / neutral — no type
```

Step 3 — RGB Channel Type Classification

```
return when {
    b > r && b > g && b > 80       -> GoopType.WATER      // Blue dominant
    r > g && r > b && r > 80       -> GoopType.FIRE       // Red dominant
    g > r && g > b && g > 80       -> GoopType.NATURE     // Green dominant
    r > 120 && g > 120 && b < 100  -> GoopType.ELECTRIC   // Yellow: R+G high, B low
    r < 50  && g < 50  && b < 50   -> GoopType.SHADOW     // Near-black
    else                           -> null
}
```

The ELECTRIC case is the most notable: yellow is not a primary additive colour, so it is identified by requiring both red and green channels to exceed 120 with blue below 100 — a deliberate multi-channel rule rather than single-channel dominance.

Step 4 — Temporal Hold Timer

A single camera frame is unreliable due to lighting flicker and motion. A **2-second sustained colour hold** is required before any spawn is triggered. If the detected type changes mid-scan, the timer resets, preventing accidental spawns from brief colour transitions:

```
private var currentDetectedType: GoopType? = null
private var colorDetectionStartTime = 0L
private val COLOR_HOLD_TIME_MS = 2000L

if (detectedType == currentDetectedType) {
```

```
    val elapsed  = currentTimeMillis - colorDetectionStartTime
    val progress = ((elapsed / 2000.0) * 100).toInt().coerceIn(0, 100)
    // UI feedback: "Detecting Water... 47%"
    if (elapsed >= COLOR_HOLD_TIME_MS) spawnCreatureOfType(detectedType)
} else {
    currentDetectedType     = detectedType
    colorDetectionStartTime = currentTimeMillis   // Reset on type change
}
```

Step 5 — Spawn Selection

Only base-stage creatures (evolutionStage = 1) can be wild-spawned via colour detection. Evolved and hybrid creatures are exclusive rewards for gameplay investment (merging, fusing):

```
SELECT * FROM creatures WHERE type = :type AND evolutionStage = 1
```

CV Summary

Aspect	Detail
Input	CameraX ImageAnalysis bitmap frames at 200 ms intervals
Sampling	Sparse centre-region sampling (1/16 pixel density)
Saturation filter	max(R,G,B) − min(R,G,B) < 30 rejects achromatic frames
Classification	5-rule RGB channel comparison mapping to 5 GoopTypes
Temporal filter	2-second same-type hold with real-time UI progress feedback
Output	spawnCreatureOfType(GoopType) → DB query → AR overlay render
Libraries used	CameraX for frame delivery only — classification is entirely custom code

7. Software Engineering Practices

7.1 Version Control

- **17 commits** on master with descriptive messages tracking incremental feature development
- **Feature branch workflow:** feature/unit-tests and jeremy branches merged via **6 pull requests**
- Commit history demonstrates iterative, working-at-every-stage development

Commit History:

Hash	Message
a52ec83	pokemonGOOP in progress
5a162be	Add sustained colour scanning for creature spawning
479fc1b	Add tap-to-catch mechanic with success/fail chance
9fe907c	Add escape timer — creatures flee after 6 seconds
edcae87	Revamp AR scanning: random spawns, multi-tap catch, arrow indicator
0b4b890	Simplify AR scanning — remove rotation sensor complexity
2fd4989	Add colour-based AR scanning and fix daily challenge tracking
2a00f92	Add merge-to-evolve system and fix daily challenge display
c5f34be	Add custom creature images and fix database migration
588f7ba	Add GitHub Actions CI/CD workflow
d27aaa9	Unit tests
d20fbd6	Merge pull request #2 from SquigglyOwl/jeremy

7.2 CI/CD — GitHub Actions

A GitHub Actions workflow (.github/workflows/android-ci.yml) runs automatically on every push and pull request to master, main, and develop:

Step	Action
1	Checkout code — actions/checkout@v4
2	Set up JDK 17 (Temurin distribution)
3	Configure Gradle — gradle/actions/setup-gradle@v4
4	Build Debug APK — ./gradlew assembleDebug
5	Run unit tests — ./gradlew testDebugUnitTest
6	Upload APK artifact (14-day retention)
7	Upload test result reports (14-day retention)

This ensures the build is never broken on the main branch and all unit tests pass before any merge is accepted.

7.3 Unit Testing

Unit tests are present and merged into master via PR #1 (feature/unit-tests):

- CreatureTest — creature entity validation (stats, rarity bounds)
- GoopTypeTest — type system logic (fusion rules, getFusionResult correctness)
- PlayerStatsTest — player stats calculations (level formula, streak edge cases)

Tests run automatically in CI on every push.

7.4 Code Structure & Style

- Modular package structure: data/database/, data/repository/, ui/ar/, ui/map/, ui/collection/, ui/achievements/
- Separation of concerns: DAOs handle queries only; Repository handles business logic; Activities handle UI only
- Kotlin idioms: data classes, when expressions, extension functions, coroutines, Flow
- Code follows the Android Kotlin Style Guide (https://developer.android.com/kotlin/style-guide)
- KDoc and inline comments on key classes and algorithms

8. Third-Party Libraries & Assets

All third-party libraries are listed below as required. No library was used to implement the core classification logic of the CV algorithm.

Library	Purpose
androidx.room (runtime + ktx + compiler/KSP)	SQLite ORM — all data persistence
androidx.camera.camera2	Camera hardware access
androidx.camera.lifecycle	Lifecycle-aware camera binding
androidx.camera.view	PreviewView UI component
com.google.android.gms:play-services-location	Fused GPS location provider
kotlinx.coroutines.android	Async / non-blocking operations
androidx.lifecycle.viewmodel.ktx	Lifecycle-aware ViewModel scope
androidx.lifecycle.livedata.ktx	LiveData
androidx.lifecycle.runtime.ktx	lifecycleScope coroutine support
androidx.recyclerview	Scrollable list and grid displays
com.google.android.material	Material Design UI components
androidx.cardview	Card layout containers
androidx.navigation.fragment.ktx	Fragment navigation
junit	Unit testing framework
androidx.test.espresso.core	UI instrumented testing
androidx.test.ext:junit	Android JUnit test runner

Build Configuration: Android API 24 (min) — 34 (target/compile), JVM target 17, KSP for Room code generation.

Assets: All creature names, types, and game designs are original. rawable assets are AI-generated, created for this project

9. AI Usage Declaration

AI tools were used during this project as declared below:

Phase	Tool	How It Was Used
Design	Claude (Anthropic)	Brainstorming the colour-detection approach for AR-style spawning without ARCore
Implementation	Claude (Anthropic)	Assistance with Room DAO query syntax, CameraX ImageAnalysis setup, coroutine patterns
Debugging	Claude (Anthropic)	Diagnosing Room database migration version conflicts and camera thread management issues
Report writing	Claude Code (Anthropic)	Structuring and drafting this report

Prompts used:

Design phase:

- "I want to build an AR creature spawning mechanic without using ARCore. Can I use the camera to detect colours in real time and use that to trigger spawns?"
- "What's a good way to structure a merge-to-evolve system like Dragon City in Android?"

Implementation phase:

- "My Room DAO isn't returning results when I join two tables. Here is my query:…. What is wrong?"
- "How do I set up CameraX ImageAnalysis to process frames on a background thread without blocking the UI?"
- "How do I use Kotlin Flow with Room so my RecyclerView updates automatically when the database changes?"
- "What's the correct way to use lifecycleScope and Dispatchers.IO together for database calls?"

Debugging phase:

- "I'm getting a Room migration error after adding a new column. My schema version is still 1. How do I fix this?"

- "My camera analyzer is running on the main thread and causing jank. How do I move it to a dedicated ExecutorService?"

Report writing phase:

- "Can you help me structure a technical report section explaining how my colour detection CV pipeline works?"
- "How should I document my database schema in a report — as a table or ER diagram?"

All AI-generated suggestions were reviewed, tested, and adapted to fit the project's specific requirements before integration. Final implementation decisions and code verification were carried out by the team.

10. Results & Evaluation

Features Delivered

Feature	Status
Camera colour detection (CV pipeline)	Complete
Colour to GoopType classification	Complete
Temporal hold filter (2 seconds)	Complete
Tap-to-catch with rarity probability	Complete
Creature escape timer (6 seconds)	Complete
Creature collection with filtering	Complete
Merge-to-evolve (3x same creature)	Complete
Fusion system (2-type hybrid, 4 recipes)	Complete
GPS habitat mapping	Complete
Geohashing for deterministic habitat zones	Complete
Daily challenges with midnight expiration	Complete
Login streak system (cross-year aware)	Complete
Achievement system (10 achievements, 5 categories)	Complete
Room database (6 tables, seeded on first launch)	Complete
GitHub Actions CI/CD pipeline	Complete
Unit tests (3 test classes)	Complete

Mobile Features Summary (Requirement: 3+)

Feature	API / Component
Camera	CameraX ImageAnalysis
GPS / Location	Fused Location Provider

Feature	API / Component
Database	Room ORM (SQLite)
Canvas Graphics	Android Canvas API
Animation	ValueAnimator, BlurMaskFilter
Multi-threading	Kotlin Coroutines + ExecutorService

6 mobile features implemented

Known Limitations

- Colour detection is sensitive to lighting conditions — very dark or very bright environments may not trigger spawns reliably
- Habitat zone positions are randomly seeded each session — zones are not spatially persistent across app restarts
- All data is stored locally; no cloud sync or remote leaderboard is implemented

Future Work

- Persist habitat zone seeds to the database so zones remain consistent across app restarts
- Implement cloud sync via Firebase to enable cross-device progress and a global leaderboard
- Expand the fusion recipe system with more type combinations and rare hybrid creatures
- Add push notifications for daily challenge reminders and login streak alerts
- Introduce a trading system allowing players to exchange creatures with nearby players via Bluetooth or NFC
- Implement a turn-based battle system allowing players to pit their Goops against each other or against AI-controlled encounters

Reflections

What went well:

- The colour-based CV pipeline proved more effective than initially expected — the 2-second temporal hold filter significantly reduced false spawns and made the mechanic feel responsive and intentional
- Using Room with Kotlin Flow meant the UI updated automatically without manual polling, which simplified a lot of the collection and home screen logic
- The Repository pattern kept the codebase clean and made debugging easier since all data logic was centralised in one place

What was harder than expected:

- Colour detection is inherently sensitive to lighting — achieving consistent spawns across different real-world environments required significant tuning of the saturation filter and RGB thresholds
- Room database migrations caused repeated issues during development when new columns were added, requiring careful versioning to avoid data loss on reinstall
- The EvolutionActivity initially used an XP-based filter that never surfaced any creatures, identifying and fixing this bug required tracing the filter logic back through the repository layer

What we would do differently:

- We would persist habitat zone seeds to the database earlier in development, as the current session-based random seeding means zones reset on every app restart, which reduces the sense of a persistent game world.

11. Links

Resource	URL
Source Code (GitHub)	*(paste GitHub repo link)*
App Demo Video	*(paste demo video link)*
Presentation Video	*(paste presentation video link)*