Godot Interactive Assessment Application

Design Document

Prepared for Development in Godot Engine by Michael Knighten

June 7, 2025

---

1 Project Overview

This project is an interactive application developed in the Godot Engine. The purpose is to allow users to evaluate themselves and be evaluated by their decisions through a structured message system. The design simulates a psychological or personality evaluation tool delivered through a gamified email-like interface.

The user experience includes:

- Logging in with a persistent profile.
- Self-assessing personality attributes.
- Responding to contextual message prompts with weighted reply options.
- Viewing an evaluation based on choices and comparing it to self-perception.
- Optionally exporting or reviewing their results.

2 Core Data Structures

2.1 Player Profile (PlayerPrefs)

Simulates Unity's PlayerPrefs using Godot's file system ('user://').

- **username:** String
- **password:** String (store securely or hashed)
- **self_assessed_attributes:** Dictionary { attribute: int }
- **evaluated_attributes:** Dictionary { attribute: int }

2.2 Message Object

Encapsulates a single question scenario with attribute influence.

- **title:** String
- **question:** String

- **options:** Array of replies, each with:
  - **text:** String
  - **weights:** Dictionary { attribute: int }
- **answered:** Boolean
- **chosen_option:** Integer (index of chosen reply)

2.3 Message Stack Manager

Controls inbox message logic and temporary file loading.

- **Text Reader Module:** Reads raw strings from file.
- **String Parser Module:** Converts strings into Message objects.
- **Dictionaries and Arrays:**
  - **new_messages:** Array of unanswered Message objects
  - **read_messages:** Array of answered Message objects

3 Scene Breakdown

3.1 Scene 1: Login

- Input fields for username and password
- Checks if profile exists in local storage
- Creates new profile if no match is found
- Loads data into a global singleton

3.2 Scene 2: Self Assessment

- User-adjustable sliders or inputs for attribute values
- Stores results in `self_assessed_attributes`
- Button to proceed to the Inbox

3.3 Scene 3: Inbox

- Email-style interface with two panels
- **Left Panel:** Message stack with New/Read toggle
- **Right Panel:** Message content and reply options
- Choosing a reply applies weighted attribute changes
- Answered messages move to the Read folder

3.4 Scene 4: Evaluation Comparison

- Two-column view:
  - Left: `self_assessed_attributes`
  - Right: `evaluated_attributes`
- Visual markers to highlight differences
- Export or display evaluation results
- Button to restart or exit

4 System Flow Overview

Sequential Flow

1. **Login**
2. **Self Assessment**
3. **Inbox**
4. **Evaluation**

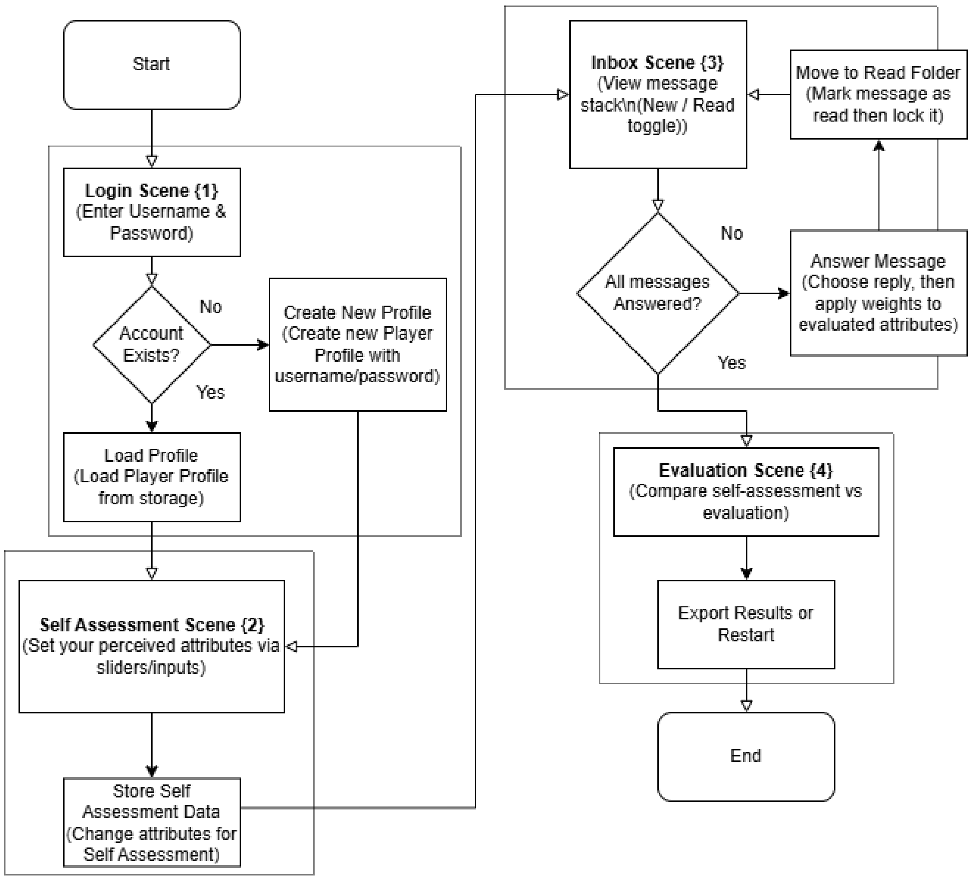
State Transitions

- Each scene transition requires prior completion.
- Global game state stores user progress and data.
- Inbox logic allows viewing old replies but locks re-editing.

5 Feature Checklist

- Persistent player profiles using file storage
- Dynamic attribute evaluation system
- Message stack with reply weighting logic
- Clear separation of logic and UI
- End evaluation with user feedback

6 Flowchart


Start
Login Scene {1} (Enter Username & Password)
Account Exists?
No
Yes
Create New Profile (Create new Player Profile with username/password)
Load Profile (Load Player Profile from storage)
Self Assessment Scene {2} (Set your perceived attributes via sliders/inputs)
Store Self Assessment Data (Change attributes for Self Assessment)
Inbox Scene {3} (View message stack\n(New / Read toggle))
Move to Read Folder (Mark message as read then lock it)
All messages Answered?
No
Yes
Answer Message (Choose reply, then apply weights to evaluated attributes)
Evaluation Scene {4} (Compare self-assessment vs evaluation)
Export Results or Restart
End

7 Implementation Notes

- Use Godot AutoLoad singletons to store global data like PlayerPrefs and MessageStack.
- Ensure attribute names are standardized (e.g., stored in an enum or constant array).
- Parse messages once at start for efficiency.
- Prepare for future backend integration by designing file readers as swappable modules.