

Technical Design Document

DILo Incremental Game

Content version history

Time	Ver	Author	Changes
10-09-2021	0.0	Abdurrahman L. H.	Initial Document
03-10-2021	0.1	Abdurrahman L. H.	Penambahan fitur: local storage
05-10-2021	0.2	Abdurrahman L. H.	Penambahan fitur: firebase cloud storage

Table of Contents

Content version history	2
Table of Contents	3
1. Introduction	4
1.1 Purpose & Objective	4
1.2 Project Scope	4
2. Technical Overview	5
2.1 Target System Requirements	5
2.2 Tools Used	5
2.3 Engines & Middleware	5
2.4 File Format	5
2.5 Technical Code	5
2.5.1 Technical specification	5
3. Technical Feature	6
3.1 <i>Tap</i> untuk mendapatkan gold	6
3.1.1 Implementation	6
3.2 <i>Passive Income</i> berupa sejumlah <i>gold</i> per detik	6
3.2.1 Implementation	6
3.3 Meng- <i>unlock</i> dan menaikkan level setiap <i>resources</i>	6
3.3.1 Implementation	6
3.4 Local storage dan Cloud storage	7
3.4.1 Implementation	7
4. Technical Design	8
4.1 Class Diagram	8
4.2 Activity Diagram	9
5. Administrative Data	10
5.1 Firebase Storage	10
6. Reusable Codes	11
6.1 Load and Save Progress module	11
6.1.1 Description	11
6.1.2 How to Use	11
7. Other Notes	12

1. Introduction

DILo Incremental Game adalah game idle casual dimana *player* dapat melakukan *tap* pada layar untuk mendapatkan *gold*, yang bisa digunakan untuk meng-*unlock* dan meng-*upgrade resource* sehingga *player* juga bisa mendapatkan *passive income* otomatis per detik. Game ini menggunakan metode *update* untuk meng-*collect income* dari *resource* yang sudah di *unlock*.

1.1 Purpose & Objective

DILo Incremental Game ditarget untuk pemain casual, yang dapat dimainkan dengan ditinggalkan saja (*idle*) karena ada sistem *passive income*.

Target utama dalam game ini adalah mengumpulkan *gold* agar dapat meng-*unlock* dan meng-*upgrade* seluruh *resources*. *Player* dapat menggunakan fitur *tap* sebagai cara cepat untuk mendapatkan *gold*, sampai *gold* tersebut cukup untuk meng-*unlock* atau meng-*upgrade* sebuah *resources* sehingga *player* juga bisa mendapatkan *gold* dari *resources* yang di-*unlock* tersebut. *Tap* dan *resources* lain dapat di-*upgrade* sehingga *gold* yang didapatkan per tap atau per detik semakin banyak.

1.2 Project Scope

Pada game ini akan diimplementasikan fitur-fitur berikut:

- Tap bagian layar untuk mendapatkan sejumlah *gold*
- Meng-*unlock* dan meng-*upgrade resources* menggunakan *gold* yang dimiliki
- Mendapatkan *passive income* sejumlah *gold* per detik
- Jumlah *gold* yang didapatkan dari tap atau *passive income* sesuai dengan banyak dan level *resources* yang dimiliki
- *Achievement* untuk setiap *resources* baru yang di-*unlock*
- Sistem *local* dan *cloud storage* yang menyimpan progress dari *player*.

2. Technical Overview

2.1 Target System Requirements

Platform/OS: Android 4.4 'Kitkat' API level 19 (minimum)

RAM: 512mb

2.2 Tools Used

Users	Tool	Use
Prog	<i>Unity</i>	Ver 2020.3.19f1
Prog	Android SDK and NDK	sdk ver 30.0.2 and ndk ver R19
Prog	OpenJDK	Ver 1.8.0_152
Prog	Firebase SDK	Ver unity sdk 8.3.0

2.3 Engines & Middleware

Game ini tidak menggunakan *engine* atau *middleware* spesifik.

2.4 File Format

Assets	.png	Image used in game
Code Script	.cs	Game script

2.5 Technical Code

2.5.1 Technical specification

Platform:	Android Mobile
Programming language:	C#

3. Technical Feature

3.1 Tap untuk mendapatkan gold

Player dapat melakukan *tap* terhadap suatu bagian layar dengan gambar koin, dan akan mendapatkan koin karena aktivitas tersebut dengan banyak sesuai dengan jumlah *resources* yang dimiliki, dan level setiap *resources*.

3.1.1 Implementation

Sebuah *script* akan digunakan untuk mendeteksi *tap* pada bagian layar, dan akan memanggil fungsi pada *game manager* untuk mendapatkan koin dengan *tap*. Dalam fungsi tersebut, akan dikalkulasikan berapa jumlah *gold* total yang didapatkan dengan cara menjumlahkan *gold* yang didapatkan dari setiap *resources* dengan memanggil metode untuk mendapatkan jumlah *gold* dari *resource* dengan iterasi semua *resources* yang dimiliki. Lalu total *gold* tersebut akan ditambahkan ke jumlah *gold* yang dimiliki *player*.

3.2 Passive Income berupa sejumlah gold per detik

Player akan mendapatkan *passive income* setiap detiknya berupa sejumlah *gold*, dengan jumlah *resources* yang dimiliki, dan level setiap *resources*.

3.2.1 Implementation

Menggunakan fungsi *update* dari metode *class MonoBehaviour* pada *game manager*, akan diperiksa apakah waktu sudah jeda satu detik. Jika ya, maka akan dipanggil fungsi untuk mendapatkan koin per detik. Cara menghitung jumlah *coin* yang didapatkan sama dengan fungsi mendapatkan koin dengan *tap*, namun kemudian dikalikan *collect percentage* berupa 0.1 (10 persen). Lalu total *gold* ditambahkan ke jumlah *gold* yang dimiliki *player*.

3.3 Meng-unlock dan menaikkan level setiap resources

Player dapat meng-unlock *resources* yang masih dikunci, dan menaikkan level *resources* yang telah di unlock dengan menggunakan *gold* yang dimiliki.

3.3.1 Implementation

Setiap *resources* yang ada dalam game akan direpresentasikan dalam bentuk objek *struct ResourceConfig* dengan variable *Name*, *UnlockCost*, *UpgradeCost*, dan *Output*. Setiap *resources* yang ada di dalam game akan disimpan dalam sebuah *array ResourceConfig*, dan pada awal game hanya *resource* pertama yang telah ter-unlock. Kemudian ada *class* yang dibuat untuk berinteraksi terhadap setiap *resources*, baik untuk meng-unlock, menaikkan level, mendapatkan data *resources*, dan juga mendapatkan jumlah *income* dari sebuah *resources*. Ketika *player* meng-unlock dan menaikkan level dari *resource*, atau mendapatkan *income*, maka akan dipanggil metode dari *class* ini.

3.4 Local storage dan Cloud storage

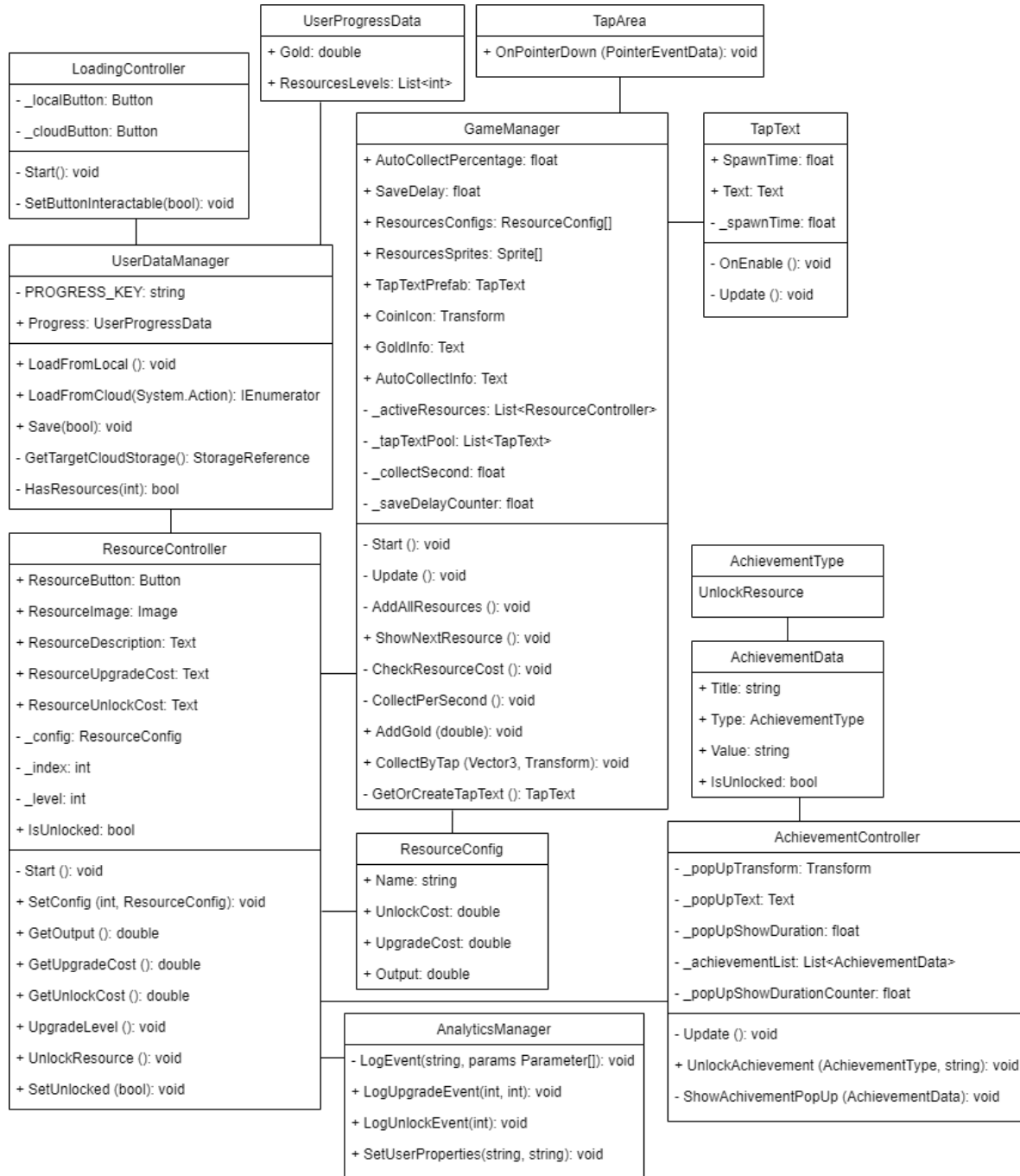
Player dapat menyimpan *progress* dari game berupa jumlah *gold* yang dimiliki, *resources* yang telah di *unlock*, dan setiap level dari *resources* yang telah dimiliki, yang disimpan pada *local* maupun *cloud storage*. Lalu *player* dapat memilih saat memulai *game* untuk melanjutkan *progress* dari *local* atau *cloud storage*.

3.4.1 Implementation

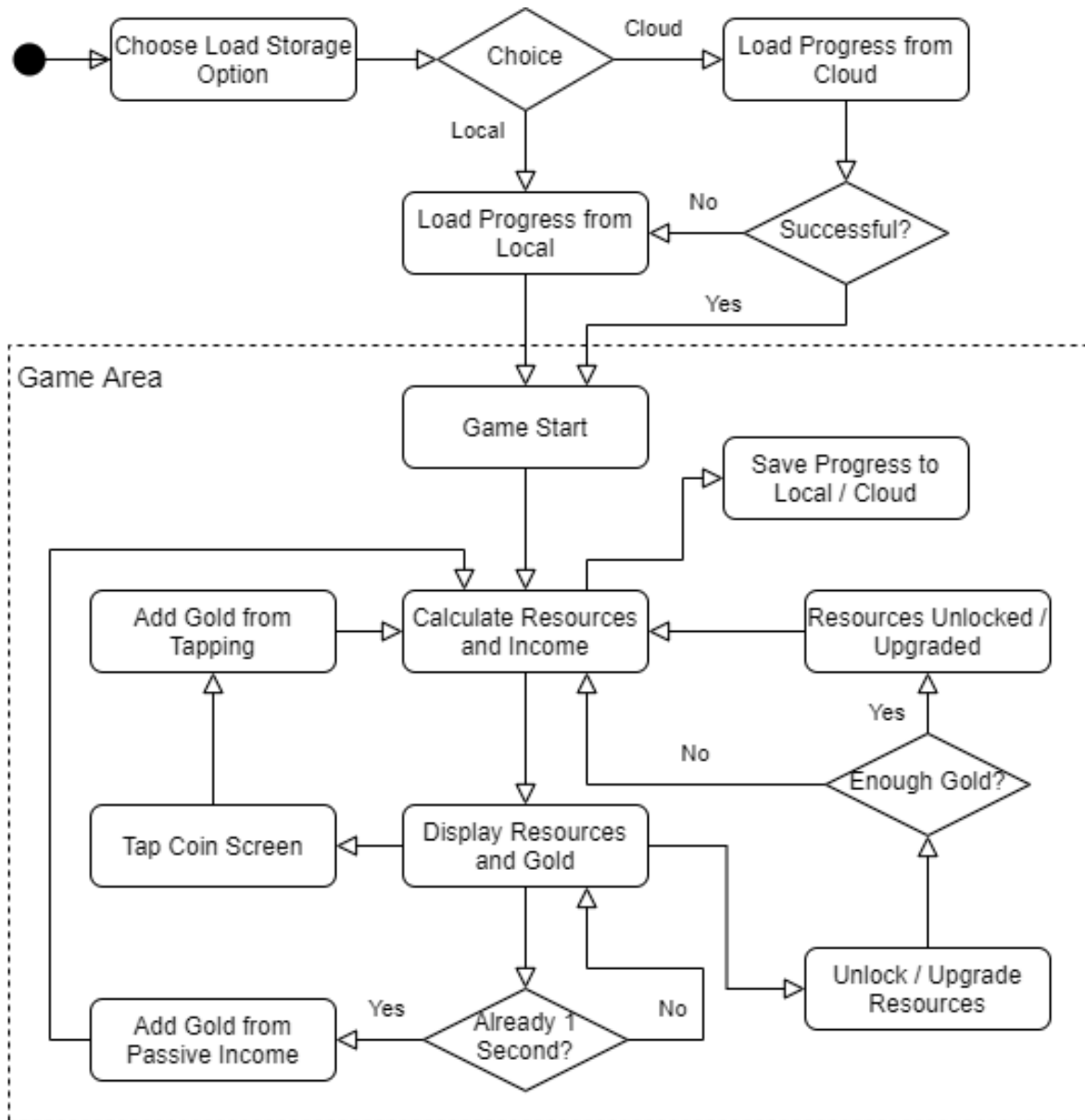
Progress dari game akan disimpan dalam *class UserProgressData*, yang menyimpan jumlah *gold player*, dan list yang menyimpan setiap level dari *resources* yang dimiliki. Lalu *class* tersebut akan dijadikan objek *string* dalam format *json* sehingga *progress* dapat disimpan. Untuk *local storage*, *progress* akan disimpan pada *PlayerPrefs* dalam Unity dalam bentuk *json*, sedangkan untuk *cloud storage*, *progress* akan disimpan menggunakan *storage* dari *Firebase* dalam bentuk *ArrayBite*. Jika pada awal permainan *player* memilih untuk me-load *progress* dari *local*, maka *progress* akan langsung diambil dari *PlayerPrefs*. Sedangkan jika dipilih *cloud*, maka akan diambil dari *storage Firebase*, dan jika gagal akan me-load *progress* dari *local*. *Progress* pada *storage local* dan *cloud* akan di-update setiap *player gold* bertambah, ada *resource* yang ter-unlock, atau ada *resource* yang dinaikkan levelnya. Namun untuk *cloud*, *progress* hanya akan di save setiap 5 detik untuk *event gold* bertambah, agar tidak terlalu berat.

4. Technical Design

4.1 Class Diagram



4.2 Activity Diagram



Notes: No termination condition in the game

5. Administrative Data

5.1 Firebase Storage

Item	Value	Note
Storage Bucket	dilo---incremental-game-14c63.appspot.com	name of the bucket for storage
Email	email@gmail.com	to access the firebase project
Password	Password12345	to access the firebase project

Notes: Email dan password yang dicantumkan palsu karena menggunakan email pribadi

6. Reusable Codes

6.1 Load and Save Progress module

6.1.1 Description

Modul ini digunakan untuk mengimplementasikan konsep *load* dan *save data progress*. Misalkan game memiliki progress seperti level karakter atau uang yang dimiliki. Modul ini memiliki metode yang dapat mengatur *progress* tersebut, baik untuk menyimpan *data progress*, melakukan *load* terhadap *progress* diawal permainan, dan juga membuat penyimpanan *progress* baru jika belum ada penyimpanan *progress* sebelumnya. Untuk *load progress* ada dua metode yang dapat digunakan, yaitu untuk *load data* dari *local* maupun *cloud*. Sedangkan ketika melakukan *save data*, ada opsi untuk meng-*upload* data ke *local* dan *cloud* ataupun *local* saja.

Modul ini terdiri dari *class* bernama *UserDataManager.cs* yang berfungsi untuk mengolah *progress data* tersebut, dan *UserProgressData.cs* yang berfungsi sebagai *class* yang menyimpan atribut-atribut yang ingin disimpan sebagai *progress*.

6.1.2 How to Use

Pertama, tentukan atribut apa saja yang ingin disimpan dan masukkan ke dalam *class* *UserProgressData.cs* sebagai atribut dari *class* tersebut. Kemudian, untuk melakukan *load* dapat memanggil fungsi *LoadFromLocal()* atau *LoadFromCloud()* dari *class* *UserDataManager.cs*. Dan untuk melakukan *save*, *assign* semua nilai atribut yang ingin disimpan pada objek *UserProgressData.cs* yang disimpan pada *class* *UserDataManager* dengan nama variabel *Progress*, dan setelah semua atribut yang ingin disimpan telah di *assign* nilainya, panggil fungsi dari *class* *UserDataManager* yaitu *Save()*, atau *Save(true)* jika *progress* juga ingin disimpan pada *cloud*. Namun, jika tidak menggunakan *cloud storage* atau *cloud storage* yang digunakan bukan menggunakan *Firebase*, maka fungsi *LoadFromCloud()* dan *Save()* harus diubah kembali.

7. Other Notes

Untuk sekarang tidak ada