

Penerapan *Firestore Realtime Database* untuk Sinkronisasi Data Pemain pada *Turn-Based Strategy RPG Mobile Game*

Charles Marcelino Triyoso^{*1}, Noven Indra Prasetya²

^{1,2} Program Studi Informatika, Fakultas Teknik, Universitas Wijaya Kusuma Surabaya

Email: ¹cmt002@mhs.uwks.ac.id, ²noven@uwks.ac.id

Abstrak

Perancangan *core gameplay loop* yang dinamis pada permainan *Turn-Based Strategy* RPG sering kali dihadapkan pada tantangan retensi data pemain. Mengandalkan penyimpanan lokal sangat rentan terhadap kehilangan progres permainan dan manipulasi data (*cheating*). Penelitian ini bertujuan untuk mengimplementasikan arsitektur *Single Source of Truth* menggunakan *Firestore Realtime Database* guna menyinkronkan data pemain secara seketika (*real-time*). Metode pengembangan menggunakan *Game Development Life Cycle* (GDLC) model Heather Chandler. Pada penerapannya, siklus permainan yang meliputi fase *pre-battle* melalui *matching card*, pertempuran taktis, dan distribusi otomatis *post-battle reward* diintegrasikan langsung dengan peladen awan. Seluruh progres ekonomi pemain, seperti akumulasi koin emas dan peningkatan level keterampilan, dienkapsulasi dalam format struktur JSON. Hasil pengujian *Black Box* menunjukkan bahwa sistem otentikasi, penarikan data (*fetching*), dan sinkronisasi pembaruan pasca-pertarungan beroperasi secara valid tanpa asinkronisasi. Kesimpulannya, penerapan *Firestore Realtime Database* berhasil menciptakan ekosistem penyimpanan awan (*cloud save*) yang stabil, presisi, dan aman dari manipulasi lokal, sekaligus menjaga integritas ekonomi permainan secara lintas perangkat.

Kata kunci: *Core Gameplay Loop, Firestore Realtime Database, GDLC, Sinkronisasi Data, Turn-Based Strategy*

Abstract

Designing a dynamic *core gameplay loop* in *Turn-Based Strategy* RPG games is often challenged by player data retention issues. Relying solely on local storage is highly vulnerable to game progress loss and data manipulation (*cheating*). This research aims to implement a *Single Source of Truth* architecture using *Firestore Realtime Database* to synchronize player data in *real-time*. The development methodology utilized is the *Game Development Life Cycle* (GDLC) model by Heather Chandler. During implementation, the game cycle comprising a *pre-battle* phase via *matching card*, tactical combat, and automatic *post-battle reward* distribution was integrated directly with the cloud server. All economic progression, such as accumulated gold coins and skill level upgrades, is encapsulated within a JSON structure. *Black Box* testing results indicated that the authentication system, data fetching, and *post-battle* update synchronization operated validly without any asynchronous anomalies. In conclusion, the implementation of *Firestore Realtime Database* successfully created a stable, precise, and secure cloud save ecosystem, protecting local memory from manipulation while maintaining the game's economic integrity across devices.

Keywords: *Core Gameplay Loop, Data Synchronization, Firestore Realtime Database, GDLC, Turn-Based Strategy*

I. PENDAHULUAN

Industri permainan seluler (*mobile games*) terus mengalami pertumbuhan pesat, khususnya pada genre *Turn-Based Strategy* RPG yang menuntut keterlibatan taktis dari pemain dalam jangka panjang[1]. Karakteristik utama dari genre ini adalah adanya siklus akumulasi progres, seperti perolehan mata uang dalam permainan, peningkatan level karakter, dan penguasaan berbagai kemampuan (*skills*). Namun, tantangan utama yang sering dihadapi oleh pengembang adalah memastikan bahwa seluruh data progres tersebut tetap konsisten, aman, dan dapat diakses secara lintas perangkat oleh pemain.

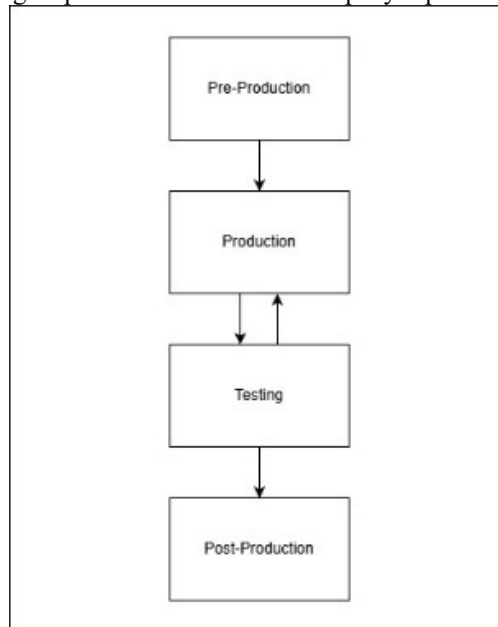
Pada pengembangan permainan "Memory Tactics", terdapat sistem komputasi kompleks yang mencakup pengaturan keseimbangan statis karakter dasar (seperti penentuan batas *Health Points* tetap pada nilai 200 dan peniadaan mekanik *lifesteal*), pengundian *before battle* menggunakan algoritma *Fisher-Yates Shuffle*, fase pertempuran, hingga proses distribusi *after battle rewards*[2]. Tanpa sistem manajemen data yang tangguh, perolehan hadiah pemain seperti *Gold* dan *Skill Level* berisiko hilang jika terjadi kerusakan pada penyimpanan lokal perangkat atau saat pemain berganti gawai. Selain itu, penyimpanan data yang hanya mengandalkan memori lokal sangat rentan terhadap manipulasi atau kecurangan (*cheating*) oleh pihak yang tidak bertanggung jawab.

Penerapan *Firestore Realtime Database* menawarkan solusi pengolahan data yang efisien melalui arsitektur berbasis JSON yang mendukung proses sinkronisasi secara seketika (*real-time*) dengan setiap perangkat klien yang terhubung [3]. Pemanfaatan layanan *Firestore* pada aplikasi berbasis Android telah terbukti mampu menyelaraskan pembacaan dan penyimpanan status pengguna dengan cepat, sehingga

memperkecil kemungkinan terjadinya bentrok data [4]. Dengan mengintegrasikan teknologi cloud ini, setiap perubahan status pemain pasca-kemenangan dapat langsung disimpan ke *server*. Pendekatan ini memungkinkan terciptanya mekanisme *Single Source of Truth*, di mana data di sisi klien (Unity) akan selalu mengacu pada validasi di sisi *server*. Melalui sinkronisasi lintas platform yang otomatis ini, pengalaman bermain menjadi lebih stabil, dan integritas ekonomi di dalam permainan tetap terjaga dengan aman [5].

II. METODE PENELITIAN

Metodologi yang digunakan dalam perancangan dan pengembangan aplikasi permainan *Turn-Based Strategy* RPG ini merujuk pada *Game Development Life Cycle (GDLC)* model Heather Chandler. Model ini menyediakan kerangka kerja yang sistematis mulai dari *Pre-Production*, *Production*, *Testing* dan *Post-Production* seperti pada gambar 1. Dalam penerapannya, penelitian ini menitikberatkan pada integrasi *Firebase Realtime Database* untuk menangani sinkronisasi data pemain secara *real-time*, guna memastikan konsistensi dan keamanan progres permainan melalui sistem penyimpanan awan (*cloud save*).

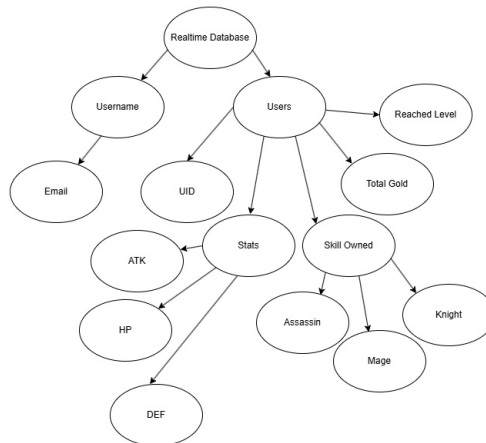


Gambar 1. Metode *GDLC* model Heather Chandler

2.1. *Pre-Production*

Tahap pra-produksi difokuskan pada perancangan konsep teknis dan penyusunan *Game Design Document (GDD)* yang mengintegrasikan alur utama permainan dengan sistem manajemen data. Perancangan aspek pertama mencakup integrasi sistem *Core Gameplay Loop* dan *Battle Rewards* yang disusun secara berkesinambungan dalam satu aktivitas sistem. Alur ini dimulai dari fase *pre-battle* melalui *mini-game matching card* untuk menentukan pengali atribut, fase pertempuran taktis berbasis giliran dengan aturan *balancing* berupa HP statis 200, hingga fase distribusi hadiah (*post-battle*). Dalam satu siklus ini, sistem secara otomatis mengelola sirkulasi ekonomi pemain, termasuk implementasi logika komputasional yang mengonversi keterampilan level maksimal menjadi kompensasi sebesar 1000 koin emas.

Aspek kedua adalah perancangan basis data dengan memodelkan arsitektur *JSON Tree* yang akan diimplementasikan pada *Firebase Realtime Database*. Struktur data ini dirancang sebagai *Single Source of Truth* untuk mendukung sinkronisasi data yang telah dirancang pada alur mekanik sebelumnya. Setiap perubahan status yang terjadi pada akhir siklus *battle rewards*, seperti pembaruan pada *node levelReached*, *totalGold*, dan *ownedSkills*, dipetakan secara terstruktur dalam format *JSON* agar dapat tersinkronisasi secara konsisten antara perangkat klien dan peladen awan seperti pada Gambar 2.



Gambar 2. JSON Tree

Lebih lanjut, tahap pra-produksi difokuskan pada sinkronisasi antara perancangan alur layar (*Screen Flow*) dengan arsitektur *Firebase Realtime Database* guna menjamin keberlanjutan siklus permainan. Alur layar yang direpresentasikan pada Gambar 3 memvisualisasikan transisi antar adegan dalam *Core Gameplay Loop*, mulai dari navigasi menu utama hingga panel hasil pertarungan. Perancangan ini dikembangkan untuk memastikan setiap perpindahan layar memicu fungsi sinkronisasi asinkron pada peladen *Firebase*. Secara teknis, sistem akan memperbarui struktur node *JSON* utama yang menyimpan identitas pemain, akumulasi koin emas, serta indeks progres dari 15 keterampilan. Dengan pemetaan yang presisi, setiap data ekonomi dan atribut tempur yang diperoleh pada fase *reward* akan tersimpan secara valid dan langsung tercermin pada layar status karakter secara *real-time*, sekaligus mencegah terjadinya anomali data.



Gambar 3. Screen Flow

2.2. Production

Tahap produksi merupakan fase realisasi rancangan sistem ke dalam lingkungan *Unity Game Engine* menggunakan bahasa pemrograman C#. Pada tahap ini, pengembangan berfokus pada pengikatan (*binding*) antara logika eksekusi *Core Gameplay Loop* pada genre *Turn-Based Strategy RPG* dengan arsitektur *Firebase Realtime Database*. Skrip permainan dibangun mengadopsi prinsip *Single Source of*

Truth guna menjamin akurasi sinkronisasi data pemain secara menyeluruh. Bersamaan dengan itu, layanan *Firebase* diimplementasikan sebagai tulang punggung untuk menangani pembaruan status pasca-pertarungan secara terpusat. Peladen *Firebase* bertugas melakukan penyimpanan data secara asinkron dan *real-time*, khususnya yang terkait dengan fluktuasi kepemilikan koin emas serta pencatatan progres level keterampilan pemain. Selain menjaga konsistensi data lintas perangkat, mekanisme sinkronisasi *cloud* ini juga bertindak sebagai lapisan validasi keamanan jarak jauh guna mencegah manipulasi memori lokal (seperti *cheat engine*) terhadap aset ekonomi virtual di dalam permainan.

2.3. Testing

Tahap pengujian (*testing*) dieksekusi menggunakan pendekatan pengujian *Black Box* untuk memvalidasi fungsionalitas mekanik permainan dan ketahanan arsitektur sinkronisasi data. Pelaksanaan pengujian dirancang secara sekuensial untuk memverifikasi sistem dari hulu ke hilir. Tahap ini dimulai dari pengujian Implementasi Antarmuka Pengguna (*UI*) untuk memvalidasi kelancaran transisi panel autentikasi dan alur layar, dilanjutkan dengan pengujian *Core Gameplay Loop* yang mencakup akurasi pengacakan kartu (*matching card*) hingga komputasi pertempuran taktis. Selanjutnya, evaluasi dilakukan pada fungsionalitas Sistem Hadiah guna memastikan presisi komputasi pendistribusian kompensasi pasca-pertarungan. Rangkaian pengujian teknis ini diakhiri dengan validasi integrasi *Firebase Realtime Database* secara ketat. Validasi akhir ini krusial untuk memastikan tidak terjadinya asinkronisasi seperti keterlambatan atau kegagalan pembaruan struktur data *JSON* antara memori perangkat lokal (*PlayerPrefs*) dan basis data awan sesaat setelah pemain mengklaim hadiah penyelesaian level. Melalui pengujian *Black Box* yang terstruktur ini, integritas arsitektur manajemen data dan keseluruhan siklus *Turn-Based Strategy RPG* dipastikan dapat beroperasi dengan stabil tanpa adanya anomali teknis. *Post-Production*

2.4. Post-Production

Tahap pasca-produksi merupakan fase penutup yang difokuskan pada kalibrasi sistem dan optimalisasi performa arsitektur sinkronisasi berdasarkan rekapitulasi data dari tahap pengujian *Black Box*. Evaluasi ditekankan pada analisis kecepatan respons peladen *Firebase Realtime Database* (*server response time*) saat memproses pembaruan struktur data *JSON*, khususnya pada transaksi pembaruan tingkat keterampilan dan mata uang di toko virtual permainan. Penambalan kerentanan kode (*patching*) dilakukan secara spesifik pada logika penanganan putusnya koneksi (*disconnect handling*) untuk menjamin kelancaran fitur *cloud save*, sehingga tidak ada progres data pemain yang hilang jika terjadi gangguan jaringan sesaat setelah pertarungan selesai. Optimalisasi komprehensif pada fase ini memastikan bahwa pertukaran data asinkron antara memori lokal dan peladen awan dapat berjalan stabil secara berulang, sehingga seluruh rekam jejak progres pemain tersimpan secara presisi dan perangkat lunak mencapai standar kelayakan akhir.

III. HASIL DAN PEMBAHASAN

3.1. Implementasi

Tahap awal implementasi dilakukan dengan mengintegrasikan *Firebase SDK* ke dalam proyek *Unity Game Engine*. Hal ini melibatkan penempatan berkas konfigurasi *google-services.json* ke dalam direktori aset proyek agar aplikasi dapat terhubung secara otomatis ke peladen *Firebase*. Inisialisasi dilakukan secara asinkron pada saat aplikasi dimulai guna memeriksa ketergantungan komponen *Firebase* di perangkat pengguna sebelum menjalankan fungsi lainnya seperti pada gambar 4. Kemudian membangun sistem autentikasi menggunakan *Firebase Authentication* yang mendukung pendaftaran akun baru pada gambar 5, masuk log (*login*) melalui pencarian *username* pada gambar 6, serta akses sebagai tamu (*guest login*). Proses ini melibatkan validasi surel dan kata sandi di sisi peladen untuk memastikan keamanan akses.

```
{
  "project_info": {
    "project_number": "501058517043",
    "firebase_url": "https://memory-tactics-default-rtdb.asia-southeast1.firebaseio.com",
    "project_id": "memory-tactics",
    "storage_bucket": "memory-tactics.firebaseio.com"
  },
  "client": {
    "client_info": {
      "mobilesdk_app_id": "1:501058517043:android:1cec9a904d4a627ed48b46",
      "android_client_info": {
        "package_name": "com.tugasakhir.memorytactics"
      }
    },
    "oauth_client": [],
    "api_key": [
      {
        "current_key": "AIzaSyB0KsA91EUyy1Bx200gX1Gohn1X1atVlg"
      }
    ],
    "services": {
      "appinvite_service": {
        "other_platform_oauth_client": []
      }
    }
  },
  "configuration_version": "1"
}
```

Gambar 4. *google-services.json*



Gambar 5. Proses Daftar



Gambar 6. Proses Login

Implementasi pada *Firebase Realtime Database* menggunakan struktur pohon *JSON* untuk menyimpan data pemain secara terpusat. Sistem membagi data ke dalam dua *node* utama, yaitu *usernames* untuk validasi keunikan identitas dan *users* yang menyimpan atribut progres pemain berbasis *UID* unik. Mekanisme sinkronisasi diimplementasikan untuk menjamin pembaruan data secara *real-time* setelah pemain menyelesaikan fase pertempuran. Fungsi ini bertugas mengirimkan *payload* data terbaru (seperti penambahan koin emas dan level keterampilan) ke peladen awan secara asinkron, sehingga progres pemain tetap konsisten meskipun aplikasi ditutup. Seperti pada gambar 7, pemain memiliki 300 *gold* pada *Firebase*. Kemudian pada gambar ke 8, pemain berhasil memenangkan pertarungan dan mendapat 250 *gold* dan 1 *random skill*. Sehingga pada gambar ke 9, pemain mendapat tambahan 250 *gold* yang kemudian dijumlah menjadi 550 serta meng-unlock 1 *skill random* yang didapat dan di *update* secara *realtime* pada *Firebase Realtime Database*.



Gambar 7. JSON Tree Player baru



Gambar 8. Memenangkan Battle



Gambar 9. Update Firebase Realtime Database

3.2. Mekanika

Bagian ini merinci aturan logis dan urutan operasi komputasional yang mengatur *Core Gameplay Loop*, yang terstruktur ke dalam tiga fase berkesinambungan. Fase pertama, yaitu mekanika persiapan (*Pre-Battle*), bertujuan untuk menentukan modifikasi status awal pemain melalui penyelesaian *mini-game* uji memori visual. Untuk menjamin keadilan probabilitas, distribusi posisi kartu diacak menggunakan algoritma *Fisher-Yates Shuffle* yang bekerja melalui iterasi mundur dan penukaran indeks elemen secara acak. Performa penyelesaian kemudian dievaluasi dengan menghitung rasio sisa waktu terhadap waktu maksimal, yang dipetakan sebagai sumbu X pada *AnimationCurve* untuk menghasilkan nilai pengali (*multiplier*) pada sumbu Y. Eksekusi fase ini ditutup dengan pemanggilan fungsi *ApplyBuff()*, di mana nilai hadiah yang dipilih pemain dikalkulasikan dengan nilai pengali tersebut, lalu diakumulasikan secara aditif atau multiplikatif ke dalam variabel *baseAttack*, *baseMaxHP*, atau *currentShield* pada skrip *UnitStats* sesaat sebelum sistem memicu transisi ke arena pertempuran.

Memasuki fase pertempuran (*Battle*), sistem mengimplementasikan siklus giliran terstruktur yang dievaluasi secara sekuensial. Siklus diinisiasi dari giliran pemain melalui pemulihan pemicu input antarmuka, yang dilanjutkan dengan eksekusi taktis dan kalkulasi kerusakan. Setelah aksi selesai, sistem menjalankan evaluasi pasca-aksi untuk memeriksa sisa *Health Point (HP)* musuh. Apabila *HP* musuh berada di atas titik nol, sistem memindahkan giliran dengan menyisipkan jeda waktu (*delay*) selama satu detik guna memberikan umpan balik visual yang transparan, sebelum mengeksekusi fungsi serangan balik

EnemyAttack() secara otomatis. Selama pertukaran serangan ini, sistem memberlakukan logika prioritas pelindung, di mana setiap kerusakan yang diterima pemain akan dikalkulasikan untuk mengurangi kapasitas *Shield* terlebih dahulu. Nilai *HP* utama karakter hanya akan mengalami pengurangan apabila akumulasi kerusakan yang diterima melampaui sisa kapasitas *Shield* pada giliran tersebut.

Siklus permainan diakhiri oleh fase distribusi hadiah (*Post-Battle*) yang bertindak sebagai pusat pergerakan sistem ekonomi, di mana logikanya terpicu secara otomatis saat status kemenangan (*Victory*) tercapai. Pada fase ini, sistem menjalankan algoritma *gacha* untuk memilih satu identitas keterampilan secara acak dari dalam kumpulan data kelas karakter. Sistem kemudian melakukan pemeriksaan parameter berjenjang terhadap keterampilan yang didapat. Apabila ID masih bernilai Level 0, sistem membuka akses keterampilan tersebut menjadi Level 1. Apabila ID bernilai di bawah Level 5, sistem menjalankan fungsi *UpgradeSkill()*. Dan apabila keterampilan telah mencapai batas maksimal (Level 5), sistem akan mengeksekusi kompensasi ekonomi melalui pemanggilan fungsi *ConvertSkillToGold(1000)*. Guna menjamin integritas rekam jejak pemain dari siklus tersebut, seluruh pembaruan variabel baik level keterampilan maupun akumulasi koin emas wajib disinkronkan ke dalam peladen *Firebase Realtime Database* melalui eksekusi perintah *SetValueAsync* sebelum antarmuka memunculkan interaksi untuk kembali ke menu navigasi utama.

3.3. Desain Sistem

Bagian desain sistem ini merinci arsitektur teknis dan komputasi logis yang menggerakkan *Core Gameplay Loop*, dengan fokus utama pada mekanisme konversi hadiah pertarungan, arsitektur komunikasi sistem, dan pelestarian integritas pangkalan data awan.

Pada fase persiapan (*Pre-Battle*), inti dari logika sistem hadiah dikendalikan oleh algoritma evaluasi performa yang bertugas mengonversi kecepatan penyelesaian *mini-game* pemain menjadi keuntungan strategis. Komputasi ini dimulai dengan menghitung rasio waktu, yang dirumuskan secara matematis sebagai $TimeRatio = RemainingTime/MaxTime$. Untuk menghindari progresi hadiah yang kaku dan linier, sistem memproses nilai rasio tersebut ke dalam komponen *AnimationCurve* pada skrip *MatchingCardManager.cs* untuk mengekstraksi nilai pengganda akhir melalui rumusan $FinalMultiplier = Curve.Evaluate(TimeRatio)$. Nilai pengganda fungsional inilah yang kemudian dikalikan secara langsung dengan parameter hadiah pilihan pemain seperti penambahan persentase *Attack* (ATK), pemulihan *Health Point* (HP), maupun kapasitas pelindung (*Shield*) sebelum diakumulasikan ke dalam atribut tempur karakter.

Transisi aliran data antar fase permainan diatur menggunakan pola komunikasi *Manager Pattern* guna memastikan modularitas kode. Jembatan komunikasi ini terbentuk ketika pemain memvalidasi pilihan hadiahnya, yang memicu sistem untuk mengirimkan objek data hadiah dan nilai pengali tersebut secara parameterik melalui pemanggilan fungsi *LanjutSetelahMiniGame()* ke skrip *BattleManager*. Sebagai pusat kontrol logika (*Single Source of Truth*), manajer pertempuran ini akan mengintegrasikan kalkulasi akhir tersebut dengan atribut baku karakter (200 *HP*, 35 *ATK*, 15 *DEF*), lalu merefleksikan seluruh perubahan dinamis pada antarmuka kesehatan dan pertahanan pemain secara *real-time* melalui rutin *UpdateUIStatusSemua()*.

Memasuki fase akhir pertarungan (*Post-Battle*), desain sistem bergeser pada manajemen data menggunakan pendekatan hibrida berformat *JavaScript Object Notation (JSON)*. Skrip *PlayerPrefs* dimanfaatkan secara terbatas pada penyimpanan lokal untuk menyimpan pengaturan sesi non-kritis. Sementara itu, untuk menjamin keamanan aset ekonomi dari potensi manipulasi memori lokal (*anti-cheat*), sistem mewajibkan seluruh perputaran ekonomi dan progres dicatat pada *Firebase Realtime Database*. Saat skrip mendeteksi pemicu kemenangan (*HandleWin*), sistem akan menyerialisasi status permainan ke dalam format *JSON* dan mengirimkannya secara asinkron ke peladen *Firebase*. Data tersebut diorganisasikan ke dalam *JSON Tree* yang diikat oleh kunci primer otentikasi *UserId*, mencakup pencatatan level tertinggi (*levelReached*), akumulasi kekayaan virtual (*totalGold*), serta pembaruan indeks level dari 15 keterampilan karakter (*ownedSkills*). Skema arsitektur ini mewajibkan aplikasi untuk selalu mengunduh (*fetch*) status ekonomi dari peladen awan guna menimpa data lokal setiap kali permainan dimuat ulang.

3.4. Pengujian

Tabel 1. *Blackbox* Implementasi UI

No	Skenario Pengujian	Test Case	Hasil yang diharapkan	Hasil Pengujian	Kesimpulan
----	--------------------	-----------	-----------------------	-----------------	------------

1.	Mengklik Tombol Mulai	Klik Mulai <i>Button</i>	Berpindah ke <i>Scene Map Selector</i>	Berpindah ke <i>Scene Map Selector</i>	VALID
2.	Memilih Map 1	Klik Tombol Map 1	Berpindah ke <i>Scene Level Selector 1</i>	Berpindah ke <i>Scene Level Selector 1</i>	VALID
3	Memilih Level 1-1	Klik Tombol Level 1-1	Berpindah ke <i>Scene Level 1-1</i>	Berpindah ke <i>Scene Level 1-1</i>	VALID

Tabel 2. Blackbox Core Gameplay Loop

No	Skenario Pengujian	Test Case	Hasil yang diharapkan	Hasil Pengujian	Kesimpulan
1.	Memulai <i>Matching Card Mini Game</i>	Klik Tombol 1-1 di <i>Scene Level Selector 1</i>	Memulai <i>Matching Card Mini Game</i>	Memulai <i>Matching Card Mini Game</i>	VALID
2.	Membalik Kartu	Mengklik Kartu	Kartu Belakang menjadi Kartu Depan	Kartu Belakang menjadi Kartu Depan	VALID
3.	Mencocokkan Kartu Salah	Memilih kartu kedua yang tidak cocok dengan kartu pertama	Kartu tertutup kembali	Kartu tertutup kembali	VALID
4.	Mencocokkan Kartu Benar	Memilih kartu kedua yang cocok dengan kartu pertama	Kartu tetap terbuka sampai semua pasangan kartu terbuka	Kartu tetap terbuka sampai semua pasangan kartu terbuka	VALID
5.	Menerima Buff	Menyelesaikan <i>Mini-Game Matching Card</i>	Menerima <i>Buff</i>	Menerima <i>Buff</i>	VALID
6.	Tidak Menerima <i>Buff</i>	Tidak Menyelesaikan <i>Mini-Game Matching Card</i>	Tidak Menerima <i>Buff</i>	Tidak Menerima <i>Buff</i>	VALID
7.	Memulai <i>Battle</i>	Menyelesaikan/Menunggu Fase <i>Matching Card</i>	Menampilkan <i>Panel Battle</i>	Menampilkan <i>Panel Battle</i>	VALID
8.	Menang <i>Battle</i> dengan <i>Wave</i> Musuh tersisa	Darah Musuh Habis	Menampilkan <i>Panel Matching Card Mini-Game</i>	Menampilkan <i>Panel Matching Card Mini-Game</i>	VALID
9.	Menang <i>Battle</i> dengan <i>Wave</i> Musuh Habis	Darah Musuh dan Tidak ada Gelombang Musuh Lagi	Menampilkan <i>Panel Victory</i>	Menampilkan <i>Panel Victory</i>	VALID

10.	Kalah <i>Battle</i>	Darah Pemain Habis	Menampilkan Panel <i>Defeat</i>	Menampilkan Panel <i>Defeat</i>	VALID
-----	---------------------	--------------------	---------------------------------	---------------------------------	-------

Tabel 3. *Blackbox Sistem Hadiah*

No	Skenario Pengujian	Test Case	Hasil yang diharapkan	Hasil Pengujian	Kesimpulan
1.	Hadiah <i>Gold</i> Utama dan <i>Unlock Skill Random</i>	Memenangkan <i>Battle</i> dan masih ada <i>skill level</i> = 0	Mendapat hadiah <i>gold</i> utama dan membuka <i>skill random</i>	Mendapat hadiah <i>gold</i> utama dan membuka <i>skill random</i>	VALID
2.	Hadiah <i>Gold</i> Utama dan <i>Upgrade Skill Random</i>	Memenangkan <i>Battle</i> dan masih ada <i>skill level</i> < 5	Mendapat hadiah <i>gold</i> utama dan meningkatkan <i>skill random</i>	Mendapat hadiah <i>gold</i> utama dan meningkatkan <i>skill random</i>	VALID
3.	Hadiah <i>Gold</i> Utama dan <i>Bonus Gold</i> dari konversi <i>Skill Reward</i>	Memenangkan <i>Battle</i> dan semua <i>skill level</i> = 5	Mendapat hadiah <i>gold</i> utama dan menerima <i>bonus gold</i>	Mendapat hadiah <i>gold</i> utama dan menerima <i>bonus gold</i>	VALID

Tabel 4. Sinkronisasi *Firestore Realtime Database*

No	Skenario Pengujian	Test Case	Hasil yang diharapkan	Hasil Pengujian	Kesimpulan
1.	Inisialisasi <i>Node Akun Baru (Firestore Auth & DB)</i>	Memasukkan <i>Username</i> , <i>Email</i> , dan <i>Password</i> yang belum pernah digunakan, lalu menekan tombol "Daftar".	Sistem membuat UID baru. Pada peladen, terbentuk <i>node</i> baru pada cabang <i>usernames</i> dan cabang <i>users/[UID]</i> . UI memunculkan teks "Berhasil!".	UID baru berhasil dibuat, dan UI memunculkan teks "Berhasil!"	VALID
2.	Validasi Pencegahan Duplikasi <i>Username</i>	Memasukkan <i>Username</i> yang sudah ada dengan kombinasi email/sandi baru, lalu menekan tombol "Daftar".	<i>Firestore</i> menolak penulisan data. UI memunculkan peringatan "Username sudah terdaftar!".	<i>Firestore</i> menolak penulisan data dan Muncul peringatan "Username sudah terdaftar!".	VALID
3.	Penarikan Data (<i>Fetching</i>) dan Injeksi Memori Lokal	Pemain memasukkan kredensial <i>login</i> yang valid pada Panel <i>Login</i> dan mengklik	Sistem mengunduh struktur <i>JSON</i> dari peladen. Nilai <i>fetchGold</i> , <i>fetchLevel</i> , dan <i>fetchSkills</i>	Nilai <i>fetchGold</i> , <i>fetchLevel</i> , dan <i>fetchSkills</i> berhasil disuntikkan ke dalam <i>PlayerPrefs</i>	VALID

		tombol "Login".	berhasil disuntikkan ke dalam <i>PlayerPrefs</i> untuk digunakan selama permainan.		
4.	Sinkronisasi Pembaruan Data Ekonomi Pasca- Pertarungan	Pemain menyelesaikan pertarungan dan berada pada panel <i>Battle</i> <i>Rewards</i> . Kemudian Sistem mengkalkulasi penambahan koin emas lalu memicu fungsi <i>Set</i> <i>Value</i> asinkron ke <i>Firebase</i> .	Nilai variabel <i>totalGold</i> pada <i>node UID</i> pemain di <i>Firebase</i> <i>Realtime</i> <i>Database</i> berubah secara <i>real-time</i> sesuai dengan penambahan yang terjadi di sisi klien.	Nilai <i>totalGold</i> bertambah sesuai dengan penambahan yang terjadi saat pasca- pertarungan dan secara <i>realtime</i> <i>update database</i> <i>firebase</i>	VALID

IV. KESIMPULAN

Penelitian ini telah berhasil membangun sebuah ekosistem permainan *Turn-Based Strategy* RPG yang utuh dengan mengintegrasikan *Unity Game Engine* dan *Firebase Realtime Database* sebagai infrastruktur penyimpanan data awan. Implementasi ini memungkinkan terciptanya mekanisme *Single Source of Truth*, di mana data progres pemain tidak hanya bergantung pada penyimpanan lokal, melainkan tersinkronisasi secara *real-time* ke peladen *Firebase*.

V. UCAPAN TERIMA KASIH

Penulis mengucapkan puji syukur kepada Tuhan Yang Maha Esa atas segala rahmat dan karunia-Nya sehingga penelitian ini dapat diselesaikan dengan baik. Ucapan terima kasih juga disampaikan kepada dosen pembimbing yang telah memberikan arahan, bimbingan, serta masukan yang sangat berarti dalam penyusunan penelitian ini. Selain itu, penulis mengucapkan terima kasih kepada keluarga dan teman-teman yang telah memberikan dukungan moral maupun motivasi selama proses penelitian berlangsung.

REFERENSI

- [1] M. W. Jonathan, J. Pragantha, and D. A. Haris, "PEMBUATAN STRATEGY GAME 'FRONTIER OF HELL' MENGGUNAKAN GODOT GAME ENGINE," *Jurnal Serina Sains, Teknik dan Kedokteran*, vol. 1, no. 2, pp. 393–400, Oct. 2023, doi: 10.24912/jsstk.v1i2.31035.
- [2] R. M. Utomo and N. I. Widiastuti, "TURN BASED STRATEGY GAME MENGGUNAKAN ALGORITMA RESOURCE ASSIGNMENT PADA PERANGKAT MOBILE BERBASIS ANDROID," 2013. [Online]. Available: www.metacritic.com,
- [3] E. A. W. Sanad, "Pemanfaatan Realtime Database di Platform Firebase Pada Aplikasi E-Tourism Kabupaten Nabire," *Jurnal Penelitian Enjiniring*, vol. 22, no. 1, pp. 20–26, May 2019, doi: 10.25042/jpe.052018.04.
- [4] Ilham Firman Maulana, "Penerapan Firebase Realtime Database pada Aplikasi E-Tilang Smartphone berbasis Mobile Android," *Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi)*, vol. 4, no. 5, pp. 854–863, Oct. 2020, doi: 10.29207/resti.v4i5.2232.
- [5] R. Syamsul Huda and A. Fikri Kurniawan, "Penerapan Firebase Pada Aplikasi E-Wisata Berbasis Android," 2022.