AI OVERLORD - EVERYTHING MASTER SPEC (ONE PDF)

Generated: 2026-02-17 | Target: iOS 16+ SwiftUI MVVM | Copilot: Claude Ops 4.6

This is the ONLY document needed. It includes: full game concept, exact build steps, full code (copy/paste), Firebase + IAP setup, localization, testing, release checklist, and troubleshooting.

Table of Contents

0. Rules for the AI (Copilot / Claude Ops 4.6)

If you use Copilot Chat to generate files, paste THIS SECTION as the first message and force it to follow the rules.

```
SYSTEM RULES (MUST FOLLOW):
- Follow MVVM strictly. Views contain no economy logic.
- Do not invent mechanics beyond this PDF.
- Prefer simplicity. Target audience: 12-18 casual.
- No politics. No copyrighted content. Use original names.
- No force unwraps. No fatalError in gameplay.
- Keep code compiling on iOS 16+. No warnings.
- Firebase and StoreKit must be optional-safe to compile, but fully ready to enable.
- Any missing piece: add TODO comments, do not guess.

DELIVERABLE RULES:
- Create exactly the folder/file tree from this PDF.
- Each file must be placed into the matching Xcode folder.
- All constants go to Utilities/GameConstants.swift.
- If you add any new file, also update the folder tree section.
```

1. Exact Xcode Setup Steps

Do this once. After that you only copy/paste code from Section 11.

Step	Action
1	Xcode > File > New > Project > iOS App
2	Product Name: AIOverlord | Interface: SwiftUI | Language: Swift
3	Minimum iOS: 16.0 (Deployment Target)
4	In the project navigator: create folders: Models, ViewModels, Services, Views, Utilities, Resources
5	Create each file exactly as in Section 11, then paste code
6	Run on iPhone Simulator

Result after initial run: you should see TabView with Empire, Upgrades, Dimensions, Rewards, Shop, Rank, Settings.

2. Architecture (MVVM) and Folder Structure

```
AIOverlord/
  AIOverlordApp.swift
  Models/
    Business.swift
    Dimension.swift
    PlayerState.swift
    Skin.swift
    SkillNode.swift
    SkillNodeFactory.swift
    DimensionFactory.swift
    SkinFactory.swift
  ViewModels/
    GameViewModel.swift
    DimensionViewModel.swift
    SpinWheelViewModel.swift
    ShopViewModel.swift
    LeaderboardViewModel.swift
  Services/
    IncomeService.swift
    PrestigeService.swift
    PersistenceService.swift
    SpinService.swift
    AntiCheatService.swift
    NotificationService.swift
    FirebaseService.swift
    PurchaseService.swift
  Views/
    RootTabView.swift
    EmpireView.swift
    SingularityEyeView.swift
    BusinessRowView.swift
    UpgradesView.swift
    SkillTreeView.swift
    DimensionView.swift
    RewardsView.swift
    SpinWheelView.swift
    ShopView.swift
    LeaderboardView.swift
    SettingsView.swift
  Utilities/
    GameConstants.swift
    NumberFormat.swift
    TimeUtil.swift
    Theme.swift
    Haptics.swift
  Resources/
    Localizable.strings (optional)
```

MVVM responsibility mapping:

```
Models: plain Codable structs (data only)
Services: formulas and side-effect helpers (income, spin, prestige, persistence, anti-cheat)
ViewModels: orchestrate state changes, timers, saving, calling services
Views: SwiftUI rendering only
```

3. Game Design Summary (Core Loop, Systems, Balancing)

```
CORE FANTASY: "I built an empire."
Theme: humorous dark overlord AI. Cartoon style. No politics.

CORE LOOP:
- Businesses generate income each second.
- Player upgrades businesses to increase income.
- Player upgrades AI level (global multiplier).
- Player can Prestige (Singularity Reset) to gain tokens and multiplier.
- Player unlocks new Dimensions for theme + scaling changes.
- Endless progression with numbers formatted as K, M, B, T, aa, bb...

KEY SYSTEMS INCLUDED IN CODE:
- Passive income (1s tick)
- Businesses + upgrade costs
- AI level upgrades
- Prestige reset with token reward
- Dimensions (1..5+)
- Skins (equip, cap, merge)
- Spin wheel (8h free spin, max 3 stored, soft pity)
- Offline income (cap 8h)
- Leaderboard and feedback (Firebase optional guarded)
- StoreKit 2 skeleton (token packs + subscription)
- Analytics events (Firebase optional guarded)
- Anti-cheat (basic validation)
```

Balancing targets:

```
- Early progress: fast (first 10 minutes constant upgrades)
- Soft stuck triggers: when next upgrade > 120s
- Monetization: accelerate, not block
- Skin bonuses: small advantage, capped
```

4. Firebase Setup (Analytics, Firestore Leaderboard, Feedback)

The project compiles without Firebase. To enable Firebase features:

```
A) Create Firebase project (console.firebase.google.com).
B) Add iOS app: bundle id must match Xcode bundle id.
C) Download GoogleService-Info.plist.
D) In Xcode: drag GoogleService-Info.plist into project (check 'Copy items' and target).
E) Add Swift Package: https://github.com/firebase/firebase-ios-sdk
   Products to add:
   - FirebaseAnalytics
   - FirebaseFirestore
   - FirebaseCore
F) Run. FirebaseService.configureIfAvailable() will configure automatically.
```

Firestore rules:

```
Collections:
- leaderboard (document id = playerName)
  fields: playerName (string), lifetimeEarnings (double), aiLevel (int), dimension (int), updatedAt (timestamp)
- feedback
  fields: playerName (string), text (string), createdAt (timestamp)

Security suggestion (later):
- Write leaderboard only for authenticated users OR accept public but rate limit.
- For now, prototype is public write.
```

5. StoreKit 2 Setup (Token packs + Subscription)

The code contains Product IDs placeholders. You MUST create these in App Store Connect.

```
Product IDs used (must match exactly):
- aioverlord.tokens.small
- aioverlord.tokens.medium
- aioverlord.tokens.large
- aioverlord.tokens.xl
- aioverlord.tokens.xxl
- aioverlord.tokens.whale
- aioverlord.sub.monthly

Token mapping in ShopView.applyTokenGrantIfNeeded():
small=100, medium=500, large=1200, xl=2500, xxl=6000, whale=15000

Subscription monthly price: 9.99 EUR (set in App Store Connect)
Benefits:
- No forced ads (interstitials). Rewarded ads can remain optional later.
- +25% income multiplier
- +50% offline multiplier
- 500 tokens per month (grant via server later; for prototype keep as design note)
```

Important production note:

```
For real money apps you should not grant tokens purely client-side.
Production approach:
- Verify transaction server-side (or at least use App Store receipt validation).
- Then grant tokens from server.
This PDF keeps client-side grant only as a prototype so you can run quickly.
```

6. Localization Setup (EN/DE/ES/FR)

The code uses mostly hard-coded strings for speed. To localize:

```
A) Create Resources/Localizable.strings for each language:
   - Base (English)
   - German (de)
   - Spanish (es)
   - French (fr)

B) Replace Text("...") with Text(LocalizedStringKey("key"))
C) Add keys into Localizable.strings files.
```

Minimal example Localizable.strings (EN):

```
"tab_empire" = "Empire";
"tab_upgrades" = "Upgrades";
"tab_dimensions" = "Dimensions";
"tab_rewards" = "Rewards";
"tab_shop" = "Shop";
"tab_rank" = "Rank";
"tab_settings" = "Settings";
```

Copyright caution:

```
Do not reference real brands, celebrities, memes by name, or copyrighted characters.
Keep all names original (this PDF already uses original names).
```

7. Push Notifications Setup (Medium aggressive)

```
Push plan:
- Max 2 notifications per day.
- Examples:
  1) "Your AI kept working. Come collect."
  2) "A rival overtook you. Take it back."

Implementation included as NotificationService placeholder.
Later:
- Request permission on first day after a big win.
- Schedule local notifications (no server needed).
```

8. Anti-cheat + Leaderboard rules

```
Anti-cheat included:
- Tick validation: actual tick income cannot exceed 10x expected.
- Prestige validation: prestige token gain must be within formula bounds.
If flagged:
- state.cheatFlagged = true
- leaderboard submission disabled
- analytics event cheat_detected

Leaderboard:
- Firestore submit uses doc id = playerName.
- Update on prestige (and manual submit).
```

9. Testing Checklist (unit tests + manual QA)

```
MANUAL QA (must pass):
- Launch app, see tabs.
- Businesses buy increases income.
- AI upgrade increases income.
- Spin consumes stored spins and gives reward.
- Offline income applies after kill+relaunch.
- Prestige resets businesses and adds tokens.
- Skins: equip max 3, affects income, merge works.
- Soft stuck triggers offer when upgrades slow down.

UNIT TESTS (optional later):
- Test NumberFormat formatting boundaries.
- Test PrestigeService prestigeGain formula.
- Test Business.nextCost growth.
- Test SpinService soft pity changes mythic frequency.
```

10. App Store Release Checklist

Before release:
- Replace placeholder Product IDs with final IDs and confirm in App Store Connect.
- Add privacy policy (especially if Firebase Analytics is used).
- Ensure no copyrighted content.
- Test on real device.
- Confirm Firebase rules.
- Add App icons and screenshots.
- App Tracking Transparency: only if you do targeted ads (not required for basic analytics).

11. FULL COPY/PASTE CODE FILES

Create each file exactly with the path shown. Paste the code exactly. Then run.

File: Utilities/GameConstants.swift

```
import Foundation

enum GameConstants {
    static let appName = "AIOverlord"
    static let minIOSVersion = "16.0"

    static let tickInterval: TimeInterval = 1.0
    static let autosaveIntervalSeconds: Int = 10
    static let offlineCapSeconds: TimeInterval = 8 * 60 * 60

    static let startCoins: Double = 50
    static let baseGlobalMultiplier: Double = 1.0

    static let aiMultPerLevel: Double = 0.05
    static let aiBaseCost: Double = 1000
    static let aiCostGrowth: Double = 1.15

    static let prestigeDivisor: Double = 1_000_000
    static let prestigeMultPerCount: Double = 0.10

    static let maxActiveSkins = 3
    static let skinBonusCap: Double = 0.40

    static let subscriptionIncomeCap: Double = 0.25
    static let subscriptionIncomeBoost: Double = 0.25
    static let subscriptionOfflineBoost: Double = 0.50
    static let subscriptionMonthlyTokens: Int = 500

    static let freeSpinCooldownSeconds: TimeInterval = 8 * 60 * 60
    static let maxStoredFreeSpins = 3
    static let softPityStartSpins = 40
    static let softPityMythicBonusPerSpin: Double = 0.01

    static let softStuckSecondsThreshold: TimeInterval = 120

    static let maxTickIncomeFactor: Double = 10.0

    static let maxPushPerDay = 2
}
```

File: Utilities/NumberFormat.swift

```
import Foundation

enum NumberFormat {
    static let suffixes: [String] = [
        "", "K", "M", "B", "T",
        "aa", "bb", "cc", "dd", "ee", "ff", "gg", "hh", "ii", "jj",
        "kk", "ll", "mm", "nn", "oo", "pp", "qq", "rr", "ss", "tt",
        "uu", "vv", "ww", "xx", "yy", "zz"
    ]

    static func format(_ value: Double, decimals: Int = 2) -> String {
        guard value.isFinite else { return "INF" }
        let sign = value < 0 ? "-" : ""
        var v = abs(value)

        if v < 1000 {
            return sign + String(format: "%.\(0)f", v)
        }

        var idx = 0
        while v >= 1000 && idx < suffixes.count - 1 {
            v /= 1000
            idx += 1
        }

        let fmt = "%.\(decimals)f"
        return "\(sign)\(String(format: fmt, v))\(suffixes[idx])"
    }
}
```

File: Utilities/TimeUtil.swift

```
import Foundation

enum TimeUtil {
    static func now() -> Date { Date() }

    static func secondsBetween(_ a: Date, _ b: Date) -> TimeInterval {
        b.timeIntervalSince(a)
    }

    static func clamp(_ v: TimeInterval, min: TimeInterval, max: TimeInterval) -> TimeInterval {
        Swift.max(min, Swift.min(max, v))
    }
}
```

File: Utilities/Theme.swift

```
import SwiftUI

enum ThemeType: String, Codable {
    case bright
    case industrial
    case planetary
    case neonDark
    case procedural
}

struct Theme {
    let type: ThemeType

    var background: Color {
        switch type {
        case .bright: return Color(.systemBackground)
        case .industrial: return Color(.secondarySystemBackground)
        case .planetary: return Color(.systemBackground)
        case .neonDark: return Color.black
        case .procedural: return Color(.systemBackground)
        }
    }

    var accent: Color {
        switch type {
        case .bright: return .blue
        case .industrial: return .orange
        case .planetary: return .green
        case .neonDark: return .pink
        case .procedural: return .purple
        }
    }

    var eyeGlowIntensity: Double {
        switch type {
        case .bright: return 0.30
        case .industrial: return 0.45
        case .planetary: return 0.55
        case .neonDark: return 0.80
        case .procedural: return 0.65
        }
    }
}
```

File: Utilities/Haptics.swift

```
import UIKit

enum Haptics {
    static func light() {
        let g = UIImpactFeedbackGenerator(style: .light)
        g.prepare()
        g.impactOccurred()
    }

    static func success() {
        let g = UINotificationFeedbackGenerator()
        g.prepare()
```

```
        g.notificationOccurred(.success)
    }

    static func warning() {
        let g = UINotificationFeedbackGenerator()
        g.prepare()
        g.notificationOccurred(.warning)
    }
}
```

File: Models/Skin.swift

```
import Foundation

enum SkinRarity: String, Codable, CaseIterable {
    case common, rare, epic, legendary, mythic, transcendent

    var displayName: String {
        switch self {
        case .common: return "Common"
        case .rare: return "Rare"
        case .epic: return "Epic"
        case .legendary: return "Legendary"
        case .mythic: return "Mythic"
        case .transcendent: return "Transcendent"
        }
    }
}

struct Skin: Identifiable, Codable, Equatable {
    var id: UUID
    var name: String
    var rarity: SkinRarity
    var bonusMultiplier: Double

    init(id: UUID = UUID(), name: String, rarity: SkinRarity, bonusMultiplier: Double) {
        self.id = id
        self.name = name
        self.rarity = rarity
        self.bonusMultiplier = bonusMultiplier
    }
}
```

File: Models/Business.swift

```
import Foundation

struct Business: Identifiable, Codable, Equatable {
    var id: UUID
    var name: String
    var level: Int
    var baseIncome: Double
    var baseCost: Double
    var costGrowth: Double
    var unlockRequirement: Double

    init(
        id: UUID = UUID(),
        name: String,
        level: Int = 0,
        baseIncome: Double,
        baseCost: Double,
        costGrowth: Double,
        unlockRequirement: Double
    ) {
        self.id = id
        self.name = name
        self.level = level
        self.baseIncome = baseIncome
        self.baseCost = baseCost
        self.costGrowth = costGrowth
        self.unlockRequirement = unlockRequirement
    }

    func nextCost() -> Double {
```

```
        baseCost * pow(costGrowth, Double(level))
    }

    func incomePerSecond(globalMultiplier: Double) -> Double {
        guard level > 0 else { return 0 }
        return baseIncome * Double(level) * globalMultiplier
    }
}
```

File: Models/Dimension.swift

```
import Foundation

struct Dimension: Identifiable, Codable, Equatable {
    var id: Int
    var name: String
    var theme: ThemeType
    var exponent: Double
    var businesses: [Business]

    init(id: Int, name: String, theme: ThemeType, exponent: Double, businesses: [Business]) {
        self.id = id
        self.name = name
        self.theme = theme
        self.exponent = exponent
        self.businesses = businesses
    }
}
```

File: Models/SkillNode.swift

```
import Foundation

enum SkillPath: String, Codable, CaseIterable {
    case profit
    case meme
    case overlord

    var displayName: String {
        switch self {
        case .profit: return "Profit AI"
        case .meme: return "Meme AI"
        case .overlord: return "Overlord AI"
        }
    }
}

enum SkillEffectType: String, Codable {
    case globalIncomeAdd
    case viralChanceAdd
    case offlineEfficiencyAdd
    case ultimateUnlock
}

struct SkillNode: Identifiable, Codable, Equatable {
    var id: UUID
    var path: SkillPath
    var index: Int
    var title: String
    var description: String
    var costTokens: Int
    var purchased: Bool
    var effectType: SkillEffectType
    var effectValue: Double

    init(
        id: UUID = UUID(),
        path: SkillPath,
        index: Int,
        title: String,
        description: String,
        costTokens: Int,
        purchased: Bool = false,
        effectType: SkillEffectType,
        effectValue: Double
```

```
    ) {
        self.id = id
        self.path = path
        self.index = index
        self.title = title
        self.description = description
        self.costTokens = costTokens
        self.purchased = purchased
        self.effectType = effectType
        self.effectValue = effectValue
    }
}
```

File: Models/SkillNodeFactory.swift

```
import Foundation

enum SkillNodeFactory {
    static func makeDefaultSkills() -> [SkillNode] {
        var nodes: [SkillNode] = []
        for path in SkillPath.allCases {
            for idx in 1...20 {
                let baseCost = 20 + idx * 10
                let title = "\(path.displayName) \(idx)"
                let desc = "Upgrade \(idx) for \(path.displayName)."

                let effect: SkillEffectType
                let value: Double
                if idx == 20 {
                    effect = .ultimateUnlock
                    value = 1
                } else {
                    switch path {
                    case .profit:
                        effect = .globalIncomeAdd
                        value = 0.01
                    case .meme:
                        effect = .viralChanceAdd
                        value = 0.005
                    case .overlord:
                        effect = .offlineEfficiencyAdd
                        value = 0.01
                    }
                }

                nodes.append(
                    SkillNode(
                        path: path,
                        index: idx,
                        title: title,
                        description: desc,
                        costTokens: baseCost,
                        purchased: false,
                        effectType: effect,
                        effectValue: value
                    )
                )
            }
        }
        return nodes
    }
}
```

File: Models/DimensionFactory.swift

```
import Foundation

enum DimensionFactory {
    static func makeDimension(id: Int) -> Dimension {
        switch id {
        case 1:
            return Dimension(
                id: 1,
                name: "Digital",
                theme: .bright,
```

```
                exponent: 1.12,
                businesses: [
                    Business(name: "Meme Farm", baseIncome: 1, baseCost: 10, costGrowth: 1.07, unlockRequirement: 0),
                    Business(name: "Chatbot Agency", baseIncome: 25, baseCost: 250, costGrowth: 1.09, unlockRequirement: 5_000),
                    Business(name: "App Factory", baseIncome: 200, baseCost: 2_500, costGrowth: 1.11, unlockRequirement: 250_000),
                    Business(name: "Data Mining", baseIncome: 900, baseCost: 8_000, costGrowth: 1.12, unlockRequirement: 1_000_000
                    Business(name: "Social AI", baseIncome: 3_000, baseCost: 25_000, costGrowth: 1.13, unlockRequirement: 5_000_00
                    Business(name: "Ad Manipulation", baseIncome: 9_000, baseCost: 80_000, costGrowth: 1.14, unlockRequirement: 15
                ]
            )
        case 2:
            return Dimension(
                id: 2,
                name: "Physical",
                theme: .industrial,
                exponent: 1.18,
                businesses: [
                    Business(name: "Robot Factory", baseIncome: 25_000, baseCost: 250_000, costGrowth: 1.16, unlockRequirement: 50
                    Business(name: "Drone Network", baseIncome: 90_000, baseCost: 900_000, costGrowth: 1.17, unlockRequirement: 20
                    Business(name: "Auto Logistics", baseIncome: 300_000, baseCost: 3_000_000, costGrowth: 1.18, unlockRequirement
                    Business(name: "Smart Cities", baseIncome: 1_000_000, baseCost: 10_000_000, costGrowth: 1.19, unlockRequiremen
                    Business(name: "Defense Bots", baseIncome: 3_000_000, baseCost: 30_000_000, costGrowth: 1.20, unlockRequiremen
                    Business(name: "Orbital Drones", baseIncome: 9_000_000, baseCost: 90_000_000, costGrowth: 1.21, unlockRequirem
                ]
            )
        case 3:
            return Dimension(
                id: 3,
                name: "Planetary",
                theme: .planetary,
                exponent: 1.25,
                businesses: [
                    Business(name: "Country Control", baseIncome: 30_000_000, baseCost: 300_000_000, costGrowth: 1.22, unlockRequi
                    Business(name: "Global Markets", baseIncome: 90_000_000, baseCost: 900_000_000, costGrowth: 1.23, unlockRequir
                    Business(name: "Energy Grid", baseIncome: 250_000_000, baseCost: 2_500_000_000, costGrowth: 1.24, unlockRequir
                    Business(name: "Satellite Net", baseIncome: 800_000_000, baseCost: 8_000_000_000, costGrowth: 1.25, unlockRequ
                    Business(name: "Ocean Factories", baseIncome: 2_500_000_000, baseCost: 25_000_000_000, costGrowth: 1.26, unloc
                    Business(name: "Planetary AI Law", baseIncome: 8_000_000_000, baseCost: 80_000_000_000, costGrowth: 1.27, unlo
                ]
            )
        case 4:
            return Dimension(
                id: 4,
                name: "Multiverse",
                theme: .neonDark,
                exponent: 1.32,
                businesses: [
                    Business(name: "Reality Glitch", baseIncome: 30_000_000_000, baseCost: 300_000_000_000, costGrowth: 1.28, unlo
                    Business(name: "Timeline Forks", baseIncome: 90_000_000_000, baseCost: 900_000_000_000, costGrowth: 1.29, unlo
                    Business(name: "Quantum Loot", baseIncome: 250_000_000_000, baseCost: 2_500_000_000_000, costGrowth: 1.30, unl
                    Business(name: "Void Servers", baseIncome: 800_000_000_000, baseCost: 8_000_000_000_000, costGrowth: 1.31, unl
                    Business(name: "Causality Control", baseIncome: 2_500_000_000_000, baseCost: 25_000_000_000_000, costGrowth: 1
                    Business(name: "Ascended Core", baseIncome: 8_000_000_000_000, baseCost: 80_000_000_000_000, costGrowth: 1.33,
                ]
            )
        default:
            return Dimension(
                id: 5,
                name: "Infinite",
                theme: .procedural,
                exponent: 1.40,
                businesses: [
                    Business(name: "Infinite Engine", baseIncome: 30_000_000_000_000, baseCost: 300_000_000_000_000, costGrowth: 1
                    Business(name: "Omni Market", baseIncome: 90_000_000_000_000, baseCost: 900_000_000_000_000, costGrowth: 1.35,
                    Business(name: "Hyper Factory", baseIncome: 250_000_000_000_000, baseCost: 2_500_000_000_000_000, costGrowth:
                    Business(name: "Singularity Forge", baseIncome: 800_000_000_000_000, baseCost: 8_000_000_000_000_000, costGrow
                    Business(name: "Overlord Matrix", baseIncome: 2_500_000_000_000_000, baseCost: 25_000_000_000_000_000, costGro
                    Business(name: "True Overlord", baseIncome: 8_000_000_000_000_000, baseCost: 80_000_000_000_000_000, costGrowt
                ]
            )
        }
    }
}
```

File: Models/SkinFactory.swift

```
import Foundation

enum SkinFactory {
```

```
    static func defaultBonus(for rarity: SkinRarity) -> Double {
        switch rarity {
        case .common: return 0.02
        case .rare: return 0.03
        case .epic: return 0.05
        case .legendary: return 0.08
        case .mythic: return 0.12
        case .transcendent: return 0.20
        }
    }

    static func randomSkin(rarity: SkinRarity) -> Skin? {
        let pool = makeLaunchPool().filter { $0.rarity == rarity }
        return pool.randomElement()
    }

    static func makeLaunchPool() -> [Skin] {
        [
            Skin(name: "Friendly Glitch", rarity: .common, bonusMultiplier: 0.02),
            Skin(name: "Bright Lens", rarity: .common, bonusMultiplier: 0.02),
            Skin(name: "Data Smirk", rarity: .common, bonusMultiplier: 0.02),
            Skin(name: "Servo Wink", rarity: .common, bonusMultiplier: 0.02),
            Skin(name: "Pixel Pupil", rarity: .common, bonusMultiplier: 0.02),

            Skin(name: "Neural Grin", rarity: .rare, bonusMultiplier: 0.03),
            Skin(name: "Overclock Aura", rarity: .rare, bonusMultiplier: 0.03),
            Skin(name: "Meme Magnet", rarity: .rare, bonusMultiplier: 0.03),
            Skin(name: "Signal Hunter", rarity: .rare, bonusMultiplier: 0.03),
            Skin(name: "Sly Operator", rarity: .rare, bonusMultiplier: 0.03),

            Skin(name: "Dark Protocol", rarity: .epic, bonusMultiplier: 0.05),
            Skin(name: "Quantum Blink", rarity: .epic, bonusMultiplier: 0.05),
            Skin(name: "Viral Architect", rarity: .epic, bonusMultiplier: 0.05),
            Skin(name: "Shadow Router", rarity: .epic, bonusMultiplier: 0.05),

            Skin(name: "Neon Overlord", rarity: .legendary, bonusMultiplier: 0.08),
            Skin(name: "Void Observer", rarity: .legendary, bonusMultiplier: 0.08),
            Skin(name: "Reality Hacker", rarity: .legendary, bonusMultiplier: 0.08),

            Skin(name: "Multiverse Eye", rarity: .mythic, bonusMultiplier: 0.12),
            Skin(name: "Planet Binder", rarity: .mythic, bonusMultiplier: 0.12),

            Skin(name: "Ascended Core", rarity: .transcendent, bonusMultiplier: 0.20),
        ]
    }
}
```

File: Models/PlayerState.swift

```
import Foundation

struct PlayerState: Codable, Equatable {
    var playerName: String

    var coins: Double
    var lifetimeEarnings: Double

    var aiLevel: Int
    var prestigeCount: Int

    var tokens: Int

    var currentDimensionId: Int

    var storedFreeSpins: Int
    var lastFreeSpinAt: Date?
    var totalSpins: Int
    var spinsSinceMythic: Int

    var unlockedSkins: [Skin]
    var activeSkinIds: [UUID]

    var skills: [SkillNode]

    var firstPurchaseMade: Bool
    var subscriptionActive: Bool

    var lastActiveAt: Date
```

```
    var cheatFlagged: Bool

    static func fresh(playerName: String) -> PlayerState {
        PlayerState(
            playerName: playerName,
            coins: GameConstants.startCoins,
            lifetimeEarnings: 0,
            aiLevel: 1,
            prestigeCount: 0,
            tokens: 0,
            currentDimensionId: 1,
            storedFreeSpins: 1,
            lastFreeSpinAt: nil,
            totalSpins: 0,
            spinsSinceMythic: 0,
            unlockedSkins: [],
            activeSkinIds: [],
            skills: SkillNodeFactory.makeDefaultSkills(),
            firstPurchaseMade: false,
            subscriptionActive: false,
            lastActiveAt: TimeUtil.now(),
            cheatFlagged: false
        )
    }
}
```

File: Services/PersistenceService.swift

```
import Foundation

protocol PersistenceServiceType {
    func load() throws -> PlayerState?
    func save(_ state: PlayerState) throws
}

final class PersistenceService: PersistenceServiceType {
    private let fileURL: URL

    init(filename: String = "player_state.json") {
        let dir = FileManager.default.urls(for: .documentDirectory, in: .userDomainMask).first!
        self.fileURL = dir.appendingPathComponent(filename)
    }

    func load() throws -> PlayerState? {
        guard FileManager.default.fileExists(atPath: fileURL.path) else { return nil }
        let data = try Data(contentsOf: fileURL)
        let decoder = JSONDecoder()
        decoder.dateDecodingStrategy = .iso8601
        return try decoder.decode(PlayerState.self, from: data)
    }

    func save(_ state: PlayerState) throws {
        let encoder = JSONEncoder()
        encoder.outputFormatting = [.prettyPrinted, .sortedKeys]
        encoder.dateEncodingStrategy = .iso8601
        let data = try encoder.encode(state)
        try data.write(to: fileURL, options: [.atomic])
    }
}
```

File: Services/IncomeService.swift

```
import Foundation

protocol IncomeServiceType {
    func globalMultiplier(state: PlayerState, dimension: Dimension, activeSkins: [Skin]) -> Double
    func totalIncomePerSecond(state: PlayerState, dimension: Dimension, activeSkins: [Skin]) -> Double
    func secondsToNextCheapestUpgrade(state: PlayerState, dimension: Dimension) -> TimeInterval?
}

final class IncomeService: IncomeServiceType {
    func globalMultiplier(state: PlayerState, dimension: Dimension, activeSkins: [Skin]) -> Double {
        let base = GameConstants.baseGlobalMultiplier
        let aiMult = 1.0 + Double(state.aiLevel) * GameConstants.aiMultPerLevel
        let prestigeMult = 1.0 + Double(state.prestigeCount) * GameConstants.prestigeMultPerCount
```

```
        let dimensionMult = 1.0 + (dimension.exponent - 1.0) * 0.5

        let skinAdd = activeSkins.reduce(0.0) { $0 + $1.bonusMultiplier }
        let skinCapped = min(GameConstants.skinBonusCap, skinAdd)
        let skinMult = 1.0 + skinCapped

        let skillAddGlobal = state.skills
            .filter { $0.purchased && $0.effectType == .globalIncomeAdd }
            .reduce(0.0) { $0 + $1.effectValue }
        let skillsMult = 1.0 + skillAddGlobal

        let subMult = 1.0 + (state.subscriptionActive ? GameConstants.subscriptionIncomeBoost : 0.0)

        return base * aiMult * prestigeMult * dimensionMult * skinMult * skillsMult * subMult
    }

    func totalIncomePerSecond(state: PlayerState, dimension: Dimension, activeSkins: [Skin]) -> Double {
        let g = globalMultiplier(state: state, dimension: dimension, activeSkins: activeSkins)
        return dimension.businesses.reduce(0.0) { partial, biz in
            partial + biz.incomePerSecond(globalMultiplier: g)
        }
    }

    func secondsToNextCheapestUpgrade(state: PlayerState, dimension: Dimension) -> TimeInterval? {
        let costs = dimension.businesses.map { $0.nextCost() }.filter { $0 > 0 }
        guard let cheapest = costs.min() else { return nil }
        let income = max(0.0001, totalIncomePerSecond(state: state, dimension: dimension, activeSkins: []))
        let missing = max(0, cheapest - state.coins)
        return missing / income
    }
}
```

File: Services/PrestigeService.swift

```
import Foundation

protocol PrestigeServiceType {
    func prestigeGain(lifetimeEarnings: Double) -> Int
    func performPrestige(state: inout PlayerState, dimension: inout Dimension)
}

final class PrestigeService: PrestigeServiceType {
    func prestigeGain(lifetimeEarnings: Double) -> Int {
        let gain = sqrt(max(0, lifetimeEarnings) / GameConstants.prestigeDivisor)
        return max(0, Int(gain.rounded(.down)))
    }

    func performPrestige(state: inout PlayerState, dimension: inout Dimension) {
        let gainTokens = prestigeGain(lifetimeEarnings: state.lifetimeEarnings)
        state.tokens += gainTokens
        state.prestigeCount += 1

        state.coins = GameConstants.startCoins
        dimension.businesses = dimension.businesses.map { biz in
            var b = biz
            b.level = 0
            return b
        }
        state.spinsSinceMythic = 0
    }
}
```

File: Services/SpinService.swift

```
import Foundation

enum SpinReward: Codable, Equatable {
    case coins(Double)
    case tokens(Int)
    case boost(multiplier: Double, seconds: Int)
    case skin(Skin)
}

protocol SpinServiceType {
    func canUseFreeSpin(state: PlayerState, now: Date) -> Bool
```

```
    func refreshFreeSpinStorage(state: inout PlayerState, now: Date)
    func spin(state: inout PlayerState, skinPool: [Skin]) -> SpinReward
}

final class SpinService: SpinServiceType {
    private let baseWeights: [(SkinRarity, Double)] = [
        (.common, 45.0),
        (.rare, 25.0),
        (.epic, 15.0),
        (.legendary, 8.0),
        (.mythic, 6.8),
        (.transcendent, 0.2),
    ]

    func canUseFreeSpin(state: PlayerState, now: Date) -> Bool {
        var s = state
        refreshFreeSpinStorage(state: &s, now: now)
        return s.storedFreeSpins > 0
    }

    func refreshFreeSpinStorage(state: inout PlayerState, now: Date) {
        let last = state.lastFreeSpinAt ?? now
        if state.lastFreeSpinAt == nil { state.lastFreeSpinAt = now }

        let elapsed = now.timeIntervalSince(last)
        guard elapsed > 0 else { return }

        let gained = Int(elapsed / GameConstants.freeSpinCooldownSeconds)
        if gained > 0 {
            state.storedFreeSpins = min(GameConstants.maxStoredFreeSpins, state.storedFreeSpins + gained)
            let advanced = last.addingTimeInterval(GameConstants.freeSpinCooldownSeconds * Double(gained))
            state.lastFreeSpinAt = advanced
        }
    }

    func spin(state: inout PlayerState, skinPool: [Skin]) -> SpinReward {
        refreshFreeSpinStorage(state: &state, now: TimeUtil.now())
        if state.storedFreeSpins > 0 { state.storedFreeSpins -= 1 }

        state.totalSpins += 1
        state.spinsSinceMythic += 1

        let rarity = pickRarityWithSoftPity(state: state)

        switch rarity {
        case .common:
            return .coins(150)
        case .rare:
            return .coins(500)
        case .epic:
            return .tokens(30)
        case .legendary:
            return .boost(multiplier: 2.0, seconds: 4 * 60 * 60)
        case .mythic:
            state.spinsSinceMythic = 0
            return .skin(pickSkin(rarity: .mythic, pool: skinPool) ?? Skin(name: "Mythic Eye", rarity: .mythic, bonusMultiplier: 0
        case .transcendent:
            state.spinsSinceMythic = 0
            return .skin(pickSkin(rarity: .transcendent, pool: skinPool) ?? Skin(name: "Ascended Core", rarity: .transcendent, bon
        }
    }

    private func pickRarityWithSoftPity(state: PlayerState) -> SkinRarity {
        var weights = baseWeights
        if state.spinsSinceMythic >= GameConstants.softPityStartSpins {
            let extraSpins = state.spinsSinceMythic - GameConstants.softPityStartSpins
            let bonusFactor = 1.0 + Double(extraSpins) * GameConstants.softPityMythicBonusPerSpin
            weights = weights.map { pair in
                if pair.0 == .mythic { return (pair.0, pair.1 * bonusFactor) }
                return pair
            }
        }

        let total = weights.reduce(0.0) { $0 + $1.1 }
        let r = Double.random(in: 0..<total)
        var cum = 0.0
        for (rar, w) in weights {
            cum += w
            if r <= cum { return rar }
        }
        return .common
```

```
    }

    private func pickSkin(rarity: SkinRarity, pool: [Skin]) -> Skin? {
        let candidates = pool.filter { $0.rarity == rarity }
        return candidates.randomElement()
    }
}
```

File: Services/AntiCheatService.swift

```
import Foundation

protocol AntiCheatServiceType {
    func validateTick(state: PlayerState, expectedIncome: Double, actualIncome: Double) -> Bool
    func validatePrestigeGain(lifetimeEarnings: Double, gainTokens: Int) -> Bool
}

final class AntiCheatService: AntiCheatServiceType {
    func validateTick(state: PlayerState, expectedIncome: Double, actualIncome: Double) -> Bool {
        let maxAllowed = expectedIncome * GameConstants.maxTickIncomeFactor + 1
        return actualIncome <= maxAllowed
    }

    func validatePrestigeGain(lifetimeEarnings: Double, gainTokens: Int) -> Bool {
        let expected = Int(sqrt(max(0, lifetimeEarnings) / GameConstants.prestigeDivisor))
        return gainTokens >= 0 && gainTokens <= expected + 5
    }
}
```

File: Services/NotificationService.swift

```
import Foundation
import UserNotifications

final class NotificationService {
    func requestPermission() async -> Bool {
        do {
            return try await UNUserNotificationCenter.current()
                .requestAuthorization(options: [.alert, .sound, .badge])
        } catch {
            return false
        }
    }

    func scheduleNudgesIfAllowed() {
        // TODO: schedule local notifications, max 2/day
    }
}
```

File: Services/FirebaseService.swift

```
import Foundation

#if canImport(FirebaseCore)
import FirebaseCore
#endif

#if canImport(FirebaseFirestore)
import FirebaseFirestore
#endif

#if canImport(FirebaseAnalytics)
import FirebaseAnalytics
#endif

final class FirebaseService {
    static let shared = FirebaseService()
    private init() {}

    func configureIfAvailable() {
        #if canImport(FirebaseCore)
        if FirebaseApp.app() == nil { FirebaseApp.configure() }
```

```
        #endif
    }

    func logEvent(_ name: String, params: [String: Any]? = nil) {
        #if canImport(FirebaseAnalytics)
        Analytics.logEvent(name, parameters: params)
        #endif
    }

    struct LeaderboardEntry: Identifiable {
        let id: String
        let playerName: String
        let lifetimeEarnings: Double
        let aiLevel: Int
        let updatedAt: Date
    }

    func submitLeaderboard(playerName: String, lifetimeEarnings: Double, aiLevel: Int, dimension: Int) async {
        #if canImport(FirebaseFirestore)
        let db = Firestore.firestore()
        let data: [String: Any] = [
            "playerName": playerName,
            "lifetimeEarnings": lifetimeEarnings,
            "aiLevel": aiLevel,
            "dimension": dimension,
            "updatedAt": Date()
        ]
        do {
            try await db.collection("leaderboard").document(playerName).setData(data, merge: true)
        } catch { }
        #endif
    }

    func fetchTopLeaderboard(limit: Int = 50) async -> [LeaderboardEntry] {
        #if canImport(FirebaseFirestore)
        let db = Firestore.firestore()
        do {
            let snap = try await db.collection("leaderboard")
                .order(by: "lifetimeEarnings", descending: true)
                .limit(to: limit)
                .getDocuments()

            return snap.documents.compactMap { doc in
                let d = doc.data()
                let name = (d["playerName"] as? String) ?? doc.documentID
                let lifetime = (d["lifetimeEarnings"] as? Double) ?? 0
                let ai = (d["aiLevel"] as? Int) ?? 1
                let date = (d["updatedAt"] as? Timestamp)?.dateValue() ?? Date()
                return LeaderboardEntry(id: doc.documentID, playerName: name, lifetimeEarnings: lifetime, aiLevel: ai, updatedAt:
            }
        } catch {
            return []
        }
        #else
        return []
        #endif
    }

    func submitFeedback(playerName: String, text: String) async {
        #if canImport(FirebaseFirestore)
        let db = Firestore.firestore()
        let data: [String: Any] = [
            "playerName": playerName,
            "text": text,
            "createdAt": Date()
        ]
        do { try await db.collection("feedback").addDocument(data: data) } catch { }
        #endif
    }
}
```

File: Services/PurchaseService.swift

```
import Foundation
import StoreKit

final class PurchaseService: ObservableObject {
    static let shared = PurchaseService()
```

```
    @Published var subscriptionActive: Bool = false

    enum ProductID {
        static let tokensSmall = "aioverlord.tokens.small"
        static let tokensMedium = "aioverlord.tokens.medium"
        static let tokensLarge = "aioverlord.tokens.large"
        static let tokensXL = "aioverlord.tokens.xl"
        static let tokensXXL = "aioverlord.tokens.xxl"
        static let tokensWhale = "aioverlord.tokens.whale"
        static let subscriptionMonthly = "aioverlord.sub.monthly"
    }

    private init() {}

    func refreshEntitlements() async {
        for await result in Transaction.currentEntitlements {
            guard case .verified(let t) = result else { continue }
            if t.productID == ProductID.subscriptionMonthly {
                subscriptionActive = true
            }
        }
    }

    func loadProducts() async -> [Product] {
        do {
            return try await Product.products(for: [
                ProductID.tokensSmall,
                ProductID.tokensMedium,
                ProductID.tokensLarge,
                ProductID.tokensXL,
                ProductID.tokensXXL,
                ProductID.tokensWhale,
                ProductID.subscriptionMonthly
            ])
        } catch {
            return []
        }
    }

    func purchase(_ product: Product) async -> Bool {
        do {
            let result = try await product.purchase()
            switch result {
            case .success(let verification):
                guard case .verified(let transaction) = verification else { return false }
                await transaction.finish()
                await refreshEntitlements()
                return true
            default:
                return false
            }
        } catch {
            return false
        }
    }

    func restore() async {
        _ = try? await AppStore.sync()
        await refreshEntitlements()
    }
}
```

File: ViewModels/DimensionViewModel.swift

```
import Foundation

@MainActor
final class DimensionViewModel: ObservableObject {
    @Published private(set) var availableDimensions: [Dimension] = [
        DimensionFactory.makeDimension(id: 1),
        DimensionFactory.makeDimension(id: 2),
        DimensionFactory.makeDimension(id: 3),
        DimensionFactory.makeDimension(id: 4),
        DimensionFactory.makeDimension(id: 5),
    ]

    func dimensionName(id: Int) -> String {
        availableDimensions.first(where: { $0.id == id })?.name ?? "Unknown"
```

```
    }
}
```

File: ViewModels/SpinWheelViewModel.swift

```
import Foundation

@MainActor
final class SpinWheelViewModel: ObservableObject {
    @Published var lastRewardText: String = ""
    @Published var lastRewardRarity: SkinRarity? = nil

    func describe(reward: SpinReward) -> String {
        switch reward {
        case .coins(let c):
            return "+\(NumberFormat.format(c)) Coins"
        case .tokens(let t):
            return "+\(t) Tokens"
        case .boost(let m, let s):
            return "\(Int(m))x Boost for \(s/3600)h"
        case .skin(let skin):
            return "Skin: \(skin.name) (\(skin.rarity.displayName))"
        }
    }
}
```

File: ViewModels/ShopViewModel.swift

```
import Foundation
import StoreKit

@MainActor
final class ShopViewModel: ObservableObject {
    @Published var products: [Product] = []
    @Published var subscriptionActive: Bool = false
    @Published var isLoading: Bool = false
    @Published var lastError: String? = nil

    func load() async {
        isLoading = true
        defer { isLoading = false }

        await PurchaseService.shared.refreshEntitlements()
        subscriptionActive = PurchaseService.shared.subscriptionActive
        products = await PurchaseService.shared.loadProducts()
    }

    func buy(_ product: Product) async -> Bool {
        let ok = await PurchaseService.shared.purchase(product)
        subscriptionActive = PurchaseService.shared.subscriptionActive
        return ok
    }

    func restore() async {
        await PurchaseService.shared.restore()
        subscriptionActive = PurchaseService.shared.subscriptionActive
    }
}
```

File: ViewModels/LeaderboardViewModel.swift

```
import Foundation

@MainActor
final class LeaderboardViewModel: ObservableObject {
    @Published var entries: [FirebaseService.LeaderboardEntry] = []
    @Published var isLoading: Bool = false

    func refresh() async {
        isLoading = true
        defer { isLoading = false }
        entries = await FirebaseService.shared.fetchTopLeaderboard(limit: 50)
```

```
    }
}
```

File: ViewModels/GameViewModel.swift

```
import Foundation
import Combine

@MainActor
final class GameViewModel: ObservableObject {
    private let persistence: PersistenceServiceType
    private let incomeService: IncomeServiceType
    private let prestigeService: PrestigeServiceType
    private let spinService: SpinServiceType
    private let antiCheat: AntiCheatServiceType

    @Published private(set) var state: PlayerState
    @Published private(set) var dimension: Dimension

    @Published private(set) var incomePerSecond: Double = 0
    @Published var activeBoostMultiplier: Double = 1.0
    @Published var boostEndsAt: Date? = nil

    @Published var showSoftStuckOffer: Bool = false
    @Published var lastSoftStuckAt: Date? = nil

    private var timerCancellable: AnyCancellable?
    private var autosaveCounter: Int = 0

    init(
        persistence: PersistenceServiceType = PersistenceService(),
        incomeService: IncomeServiceType = IncomeService(),
        prestigeService: PrestigeServiceType = PrestigeService(),
        spinService: SpinServiceType = SpinService(),
        antiCheat: AntiCheatServiceType = AntiCheatService()
    ) {
        self.persistence = persistence
        self.incomeService = incomeService
        self.prestigeService = prestigeService
        self.spinService = spinService
        self.antiCheat = antiCheat

        if let loaded = try? persistence.load(), let st = loaded {
            self.state = st
        } else {
            self.state = PlayerState.fresh(playerName: "Overlord")
        }

        self.dimension = DimensionFactory.makeDimension(id: self.state.currentDimensionId)
        FirebaseService.shared.configureIfAvailable()
        FirebaseService.shared.logEvent("session_start")
        applyOfflineEarnings()
        recalcDerived()
        startTimer()
    }

    func setPlayerName(_ name: String) {
        state.playerName = name
        saveNow()
    }

    func startTimer() {
        timerCancellable?.cancel()
        timerCancellable = Timer.publish(every: GameConstants.tickInterval, on: .main, in: .common)
            .autoconnect()
            .sink { [weak self] _ in self?.tick() }
    }

    private func tick() {
        guard !state.cheatFlagged else { return }

        if let ends = boostEndsAt, TimeUtil.now() >= ends {
            activeBoostMultiplier = 1.0
            boostEndsAt = nil
        }

        let activeSkins = getActiveSkins()
        let baseIncome = incomeService.totalIncomePerSecond(state: state, dimension: dimension, activeSkins: activeSkins)
```

```
    let boostedIncome = baseIncome * activeBoostMultiplier
    let expectedTick = boostedIncome * GameConstants.tickInterval
    let actualTick = expectedTick

    if !antiCheat.validateTick(state: state, expectedIncome: expectedTick, actualIncome: actualTick) {
        state.cheatFlagged = true
        FirebaseService.shared.logEvent("cheat_detected")
        return
    }

    state.coins += actualTick
    state.lifetimeEarnings += actualTick
    state.lastActiveAt = TimeUtil.now()

    autosaveCounter += 1
    if autosaveCounter >= GameConstants.autosaveIntervalSeconds {
        autosaveCounter = 0
        saveNow()
    }

    recalcDerived()
    evaluateSoftStuck()
}

private func recalcDerived() {
    let activeSkins = getActiveSkins()
    incomePerSecond = incomeService.totalIncomePerSecond(state: state, dimension: dimension, activeSkins: activeSkins) * activ
}

private func evaluateSoftStuck() {
    guard let seconds = incomeService.secondsToNextCheapestUpgrade(state: state, dimension: dimension) else { return }
    if seconds > GameConstants.softStuckSecondsThreshold {
        if let last = lastSoftStuckAt, TimeUtil.now().timeIntervalSince(last) < 180 { return }
        lastSoftStuckAt = TimeUtil.now()
        showSoftStuckOffer = true
        FirebaseService.shared.logEvent("soft_stuck_detected", params: ["seconds": seconds])
    }
}

func buyBusinessLevel(businessId: UUID) {
    guard let idx = dimension.businesses.firstIndex(where: { $0.id == businessId }) else { return }
    let cost = dimension.businesses[idx].nextCost()
    guard state.coins >= cost else { return }

    state.coins -= cost
    dimension.businesses[idx].level += 1
    Haptics.light()
    FirebaseService.shared.logEvent("upgrade_business", params: ["name": dimension.businesses[idx].name])
    recalcDerived()
}

func aiNextCost() -> Double {
    GameConstants.aiBaseCost * pow(GameConstants.aiCostGrowth, Double(state.aiLevel))
}

func buyAILevel() {
    let cost = aiNextCost()
    guard state.coins >= cost else { return }
    state.coins -= cost
    state.aiLevel += 1
    Haptics.success()
    FirebaseService.shared.logEvent("ai_level_up", params: ["level": state.aiLevel])
    recalcDerived()
}

func prestige() {
    var dim = dimension
    var st = state
    let gain = prestigeService.prestigeGain(lifetimeEarnings: st.lifetimeEarnings)
    if !antiCheat.validatePrestigeGain(lifetimeEarnings: st.lifetimeEarnings, gainTokens: gain) {
        st.cheatFlagged = true
        FirebaseService.shared.logEvent("cheat_detected")
        state = st
        return
    }
    prestigeService.performPrestige(state: &st, dimension: &dim)
    state = st
    dimension = dim
    FirebaseService.shared.logEvent("prestige_triggered", params: ["prestigeCount": state.prestigeCount])
    Task { await submitLeaderboardIfAllowed() }
    saveNow()
```

```
    recalcDerived()
}

func switchDimension(to id: Int) {
    state.currentDimensionId = id
    dimension = DimensionFactory.makeDimension(id: id)
    saveNow()
    FirebaseService.shared.logEvent("dimension_unlocked", params: ["id": id])
    recalcDerived()
}

func refreshFreeSpins() {
    var st = state
    spinService.refreshFreeSpinStorage(state: &st, now: TimeUtil.now())
    state = st
    saveNow()
}

func canSpinFree() -> Bool {
    refreshFreeSpins()
    return state.storedFreeSpins > 0
}

func spinOnce(skinPool: [Skin]) -> SpinReward {
    var st = state
    let reward = spinService.spin(state: &st, skinPool: skinPool)
    state = st
    applySpinReward(reward)
    FirebaseService.shared.logEvent("spin_used")
    saveNow()
    return reward
}

private func applySpinReward(_ reward: SpinReward) {
    switch reward {
    case .coins(let c): state.coins += c
    case .tokens(let t): state.tokens += t
    case .boost(let m, let seconds):
        activeBoostMultiplier = m
        boostEndsAt = TimeUtil.now().addingTimeInterval(TimeInterval(seconds))
    case .skin(let skin):
        if !state.unlockedSkins.contains(where: { $0.id == skin.id }) { state.unlockedSkins.append(skin) }
    }
    Haptics.success()
    recalcDerived()
}

func getActiveSkins() -> [Skin] {
    let set = Set(state.activeSkinIds)
    return state.unlockedSkins.filter { set.contains($0.id) }
}

func toggleSkinActive(_ skin: Skin) {
    var ids = state.activeSkinIds
    if let i = ids.firstIndex(of: skin.id) {
        ids.remove(at: i)
    } else {
        if ids.count >= GameConstants.maxActiveSkins { return }
        ids.append(skin.id)
    }
    state.activeSkinIds = ids
    recalcDerived()
    saveNow()
}

func mergeSkins(of rarity: SkinRarity) -> Bool {
    let skins = state.unlockedSkins.filter { $0.rarity == rarity }
    guard skins.count >= 3 else { return false }

    let toRemove = Array(skins.prefix(3))
    state.unlockedSkins.removeAll { s in toRemove.contains(where: { $0.id == s.id }) }

    let next = nextRarity(after: rarity)
    let skipChance = Double.random(in: 0...1) < 0.10
    let finalRarity = skipChance ? nextRarity(after: next) : next

    let newSkin = SkinFactory.randomSkin(rarity: finalRarity)
        ?? Skin(name: "\(finalRarity.displayName) Skin", rarity: finalRarity, bonusMultiplier: SkinFactory.defaultBonus(for: f
    state.unlockedSkins.append(newSkin)

    FirebaseService.shared.logEvent("merge_skin", params: ["rarity": rarity.rawValue])
```

```
        saveNow()
        recalcDerived()
        return true
    }

    private func nextRarity(after r: SkinRarity) -> SkinRarity {
        switch r {
        case .common: return .rare
        case .rare: return .epic
        case .epic: return .legendary
        case .legendary: return .mythic
        case .mythic: return .transcendent
        case .transcendent: return .transcendent
        }
    }

    func buySkill(_ node: SkillNode) {
        guard !node.purchased else { return }
        guard state.tokens >= node.costTokens else { return }
        state.tokens -= node.costTokens
        if let idx = state.skills.firstIndex(where: { $0.id == node.id }) {
            state.skills[idx].purchased = true
        }
        saveNow()
        recalcDerived()
    }

    private func applyOfflineEarnings() {
        let now = TimeUtil.now()
        let elapsed = TimeUtil.secondsBetween(state.lastActiveAt, now)
        let capped = TimeUtil.clamp(elapsed, min: 0, max: GameConstants.offlineCapSeconds)

        let activeSkins = getActiveSkins()
        let base = incomeService.totalIncomePerSecond(state: state, dimension: dimension, activeSkins: activeSkins)

        let offlineSkillAdd = state.skills
            .filter { $0.purchased && $0.effectType == .offlineEfficiencyAdd }
            .reduce(0.0) { $0 + $1.effectValue }

        let subOffline = state.subscriptionActive ? GameConstants.subscriptionOfflineBoost : 0.0
        let offlineMult = 0.50 + offlineSkillAdd + subOffline
        let gain = base * capped * offlineMult

        if gain > 0 {
            state.coins += gain
            state.lifetimeEarnings += gain
        }
        state.lastActiveAt = now
    }

    func submitLeaderboardIfAllowed() async {
        guard !state.cheatFlagged else { return }
        await FirebaseService.shared.submitLeaderboard(
            playerName: state.playerName,
            lifetimeEarnings: state.lifetimeEarnings,
            aiLevel: state.aiLevel,
            dimension: state.currentDimensionId
        )
    }

    func submitFeedback(_ text: String) async {
        await FirebaseService.shared.submitFeedback(playerName: state.playerName, text: text)
    }

    func saveNow() {
        do { try persistence.save(state) } catch { }
    }
}
```

File: AIOverlordApp.swift

```
import SwiftUI

@main
struct AIOverlordApp: App {
    var body: some Scene {
        WindowGroup {
            RootTabView()
```

```
        }
    }
}
```

File: Views/RootTabView.swift

```
import SwiftUI

struct RootTabView: View {
    @StateObject private var gameVM = GameViewModel()
    @StateObject private var shopVM = ShopViewModel()
    @StateObject private var lbVM = LeaderboardViewModel()

    var body: some View {
        TabView {
            EmpireView(gameVM: gameVM)
                .tabItem { Label("Empire", systemImage: "eye") }

            UpgradesView(gameVM: gameVM)
                .tabItem { Label("Upgrades", systemImage: "arrow.up.circle") }

            DimensionView(gameVM: gameVM)
                .tabItem { Label("Dimensions", systemImage: "globe") }

            RewardsView(gameVM: gameVM)
                .tabItem { Label("Rewards", systemImage: "gift") }

            ShopView(gameVM: gameVM, shopVM: shopVM)
                .tabItem { Label("Shop", systemImage: "cart") }

            LeaderboardView(gameVM: gameVM, lbVM: lbVM)
                .tabItem { Label("Rank", systemImage: "trophy") }

            SettingsView(gameVM: gameVM)
                .tabItem { Label("Settings", systemImage: "gearshape") }
        }
    }
}
```

File: Views/EmpireView.swift

```
import SwiftUI

struct EmpireView: View {
    @ObservedObject var gameVM: GameViewModel
    var theme: Theme { Theme(type: gameVM.dimension.theme) }

    var body: some View {
        NavigationView {
            ZStack {
                theme.background.ignoresSafeArea()

                VStack(spacing: 12) {
                    VStack(spacing: 6) {
                        Text("Coins: \(NumberFormat.format(gameVM.state.coins))").font(.headline)
                        Text("Income/s: \(NumberFormat.format(gameVM.incomePerSecond))").font(.subheadline).opacity(0.8)
                        Text("Tokens: \(gameVM.state.tokens)  |  AI Lv \(gameVM.state.aiLevel)").font(.caption).opacity(0.7)
                    }

                    SingularityEyeView(theme: theme, aiLevel: gameVM.state.aiLevel, prestige: gameVM.state.prestigeCount)
                        .frame(height: 180)

                    HStack(spacing: 12) {
                        Button {
                            gameVM.buyAILevel()
                        } label: {
                            VStack(alignment: .leading, spacing: 2) {
                                Text("Upgrade AI").font(.headline)
                                Text("Cost: \(NumberFormat.format(gameVM.aiNextCost()))").font(.caption)
                            }
                            .frame(maxWidth: .infinity, alignment: .leading)
                            .padding(10)
                            .background(.thinMaterial)
                            .cornerRadius(12)
                        }
```

```
                    VStack(alignment: .leading, spacing: 2) {
                        Text("Lifetime").font(.headline)
                        Text(NumberFormat.format(gameVM.state.lifetimeEarnings)).font(.caption)
                    }
                    .frame(maxWidth: .infinity, alignment: .leading)
                    .padding(10)
                    .background(.thinMaterial)
                    .cornerRadius(12)
                }
                .padding(.horizontal)

                List {
                    ForEach(gameVM.dimension.businesses) { biz in
                        BusinessRowView(
                            business: biz,
                            coins: gameVM.state.coins,
                            globalIncomePerSec: gameVM.incomePerSecond,
                            onBuy: { gameVM.buyBusinessLevel(businessId: biz.id) }
                        )
                    }
                }
                .listStyle(.plain)
            }
            .padding(.top, 8)
        }
        .navigationTitle("AI Overlord")
        .toolbar {
            ToolbarItem(placement: .navigationBarTrailing) {
                Button("Prestige") { gameVM.prestige() }
            }
        }
        .alert("Offer", isPresented: $gameVM.showSoftStuckOffer) {
            Button("OK") { gameVM.showSoftStuckOffer = false }
        } message: {
            Text("Singularity instability detected... a boost might help.")
        }
    }
  }
}
```

File: Views/SingularityEyeView.swift

```
import SwiftUI

struct SingularityEyeView: View {
    let theme: Theme
    let aiLevel: Int
    let prestige: Int

    @State private var pulse: Bool = false
    @State private var rotation: Double = 0

    var body: some View {
        ZStack {
            Circle()
                .fill(theme.accent.opacity(theme.eyeGlowIntensity))
                .scaleEffect(pulse ? 1.08 : 0.95)
                .blur(radius: 18)

            Circle()
                .fill(Color.white.opacity(theme.type == .neonDark ? 0.08 : 0.14))
                .overlay(Circle().stroke(theme.accent.opacity(0.7), lineWidth: 2))
                .scaleEffect(0.9)

            ForEach(0..<3, id: \.self) { i in
                Circle()
                    .stroke(theme.accent.opacity(0.45), lineWidth: 2)
                    .scaleEffect(0.25 + Double(i) * 0.18)
                    .rotationEffect(.degrees(rotation * (i % 2 == 0 ? 1 : -1)))
            }

            Circle()
                .fill(theme.accent.opacity(0.85))
                .frame(width: 24, height: 24)
                .shadow(radius: 6)

            VStack {
                Text("AI Lv \(aiLevel)")
```

```
                    .font(.caption2)
                    .padding(6)
                    .background(.ultraThinMaterial)
                    .cornerRadius(10)
                Text("Prestige \(prestige)")
                    .font(.caption2)
                    .padding(6)
                    .background(.ultraThinMaterial)
                    .cornerRadius(10)
            }
            .offset(y: 70)
        }
        .onAppear {
            withAnimation(.easeInOut(duration: 1.2).repeatForever(autoreverses: true)) { pulse = true }
            withAnimation(.linear(duration: 6).repeatForever(autoreverses: false)) { rotation = 360 }
        }
    }
}
```

File: Views/BusinessRowView.swift

```
import SwiftUI

struct BusinessRowView: View {
    let business: Business
    let coins: Double
    let globalIncomePerSec: Double
    let onBuy: () -> Void

    var body: some View {
        let cost = business.nextCost()
        HStack {
            VStack(alignment: .leading, spacing: 4) {
                Text(business.name).font(.headline)
                Text("Level \(business.level)").font(.caption).opacity(0.8)
                Text("Cost: \(NumberFormat.format(cost))").font(.caption2).opacity(0.7)
            }
            Spacer()
            Button("Buy") { onBuy() }
                .buttonStyle(.borderedProminent)
                .disabled(coins < cost)
        }
        .padding(.vertical, 6)
    }
}
```

File: Views/UpgradesView.swift

```
import SwiftUI

struct UpgradesView: View {
    @ObservedObject var gameVM: GameViewModel

    var body: some View {
        NavigationView {
            ScrollView {
                VStack(alignment: .leading, spacing: 14) {
                    Text("Skill Trees").font(.title2).bold()
                    SkillTreeView(gameVM: gameVM, path: .profit)
                    SkillTreeView(gameVM: gameVM, path: .meme)
                    SkillTreeView(gameVM: gameVM, path: .overlord)

                    Divider().padding(.vertical, 10)

                    Text("Skins").font(.title2).bold()
                    skinsSection
                }
                .padding()
            }
            .navigationTitle("Upgrades")
        }
    }

    private var skinsSection: some View {
        VStack(alignment: .leading, spacing: 10) {
```

```
            Text("Active skins max: \(GameConstants.maxActiveSkins). Tap to activate/deactivate.")
                .font(.caption)
                .opacity(0.8)

            let pool = SkinFactory.makeLaunchPool()
            Button("Grant Launch Skins (Dev)") {
                for s in pool where !gameVM.state.unlockedSkins.contains(s) {
                    gameVM.state.unlockedSkins.append(s)
                }
                gameVM.saveNow()
            }
            .buttonStyle(.bordered)

            ForEach(gameVM.state.unlockedSkins) { skin in
                HStack {
                    VStack(alignment: .leading, spacing: 2) {
                        Text(skin.name)
                        Text("\(skin.rarity.displayName)  (+\(Int(skin.bonusMultiplier*100))%)")
                            .font(.caption2).opacity(0.7)
                    }
                    Spacer()
                    let isActive = gameVM.state.activeSkinIds.contains(skin.id)
                    Button(isActive ? "Active" : "Equip") {
                        gameVM.toggleSkinActive(skin)
                    }
                    .buttonStyle(.borderedProminent)
                }
                Divider()
            }

            mergeButtons
        }
    }

    private var mergeButtons: some View {
        VStack(alignment: .leading, spacing: 8) {
            Text("Merge: 3x same rarity -> next tier, 10% skip chance").font(.caption).opacity(0.8)
            ForEach(SkinRarity.allCases, id: \.self) { r in
                Button("Merge \(r.displayName)") { _ = gameVM.mergeSkins(of: r) }
                    .buttonStyle(.bordered)
            }
        }
        .padding(.top, 8)
    }
}
```

File: Views/SkillTreeView.swift

```
import SwiftUI

struct SkillTreeView: View {
    @ObservedObject var gameVM: GameViewModel
    let path: SkillPath

    var body: some View {
        VStack(alignment: .leading, spacing: 8) {
            Text(path.displayName).font(.headline)

            let nodes = gameVM.state.skills
                .filter { $0.path == path }
                .sorted { $0.index < $1.index }

            ForEach(nodes) { node in
                HStack {
                    VStack(alignment: .leading, spacing: 2) {
                        Text(node.title).font(.subheadline)
                        Text(node.description).font(.caption2).opacity(0.7)
                    }
                    Spacer()
                    if node.purchased {
                        Text("Owned").font(.caption).opacity(0.8)
                    } else {
                        Button("\(node.costTokens) T") { gameVM.buySkill(node) }
                            .buttonStyle(.borderedProminent)
                            .disabled(gameVM.state.tokens < node.costTokens)
                    }
                }
                Divider()
```

```
            }
        }
        .padding(12)
        .background(.thinMaterial)
        .cornerRadius(16)
    }
}
```

File: Views/DimensionView.swift

```
import SwiftUI

struct DimensionView: View {
    @ObservedObject var gameVM: GameViewModel
    @StateObject private var dimVM = DimensionViewModel()

    var body: some View {
        NavigationView {
            List {
                Section("Current") {
                    Text("Dimension: \(dimVM.dimensionName(id: gameVM.state.currentDimensionId))")
                }
                Section("Switch") {
                    ForEach(dimVM.availableDimensions) { d in
                        Button {
                            gameVM.switchDimension(to: d.id)
                        } label: {
                            HStack {
                                Text("\(d.id). \(d.name)")
                                Spacer()
                                if d.id == gameVM.state.currentDimensionId {
                                    Image(systemName: "checkmark.circle.fill")
                                }
                            }
                        }
                    }
                }
            }
            .navigationTitle("Dimensions")
        }
    }
}
```

File: Views/RewardsView.swift

```
import SwiftUI

struct RewardsView: View {
    @ObservedObject var gameVM: GameViewModel
    @StateObject private var wheelVM = SpinWheelViewModel()
    @State private var showResult = false

    var body: some View {
        NavigationView {
            VStack(spacing: 16) {
                Text("Free spins stored: \(gameVM.state.storedFreeSpins)")
                    .font(.headline)

                SpinWheelView()
                    .frame(height: 260)

                Button(gameVM.canSpinFree() ? "Spin" : "No Free Spin") {
                    let reward = gameVM.spinOnce(skinPool: SkinFactory.makeLaunchPool())
                    wheelVM.lastRewardText = wheelVM.describe(reward: reward)
                    if case .skin(let s) = reward { wheelVM.lastRewardRarity = s.rarity }
                    showResult = true
                }
                .buttonStyle(.borderedProminent)
                .disabled(!gameVM.canSpinFree())

                Text(wheelVM.lastRewardText)
                    .font(.subheadline)
                    .padding(.horizontal)
                    .multilineTextAlignment(.center)

                Spacer()
```

```
            }
            .padding()
            .navigationTitle("Rewards")
            .alert("Result", isPresented: $showResult) {
                Button("OK") { }
            } message: {
                Text(wheelVM.lastRewardText)
            }
        }
    }
}
```

File: Views/SpinWheelView.swift

```
import SwiftUI

struct SpinWheelView: View {
    @State private var angle: Double = 0

    var body: some View {
        ZStack {
            Circle().stroke(Color.gray.opacity(0.35), lineWidth: 10)
            ForEach(0..<12, id: \.self) { i in
                Capsule()
                    .fill(Color.gray.opacity(0.25))
                    .frame(width: 6, height: 22)
                    .offset(y: -110)
                    .rotationEffect(.degrees(Double(i) * 30))
            }
            Circle().fill(Color.black.opacity(0.05)).frame(width: 40, height: 40)
            Image(systemName: "triangle.fill")
                .rotationEffect(.degrees(180))
                .offset(y: -135)
        }
        .rotationEffect(.degrees(angle))
        .animation(.easeOut(duration: 3), value: angle)
        .onAppear { angle = 0 }
    }
}
```

File: Views/ShopView.swift

```
import SwiftUI
import StoreKit

struct ShopView: View {
    @ObservedObject var gameVM: GameViewModel
    @ObservedObject var shopVM: ShopViewModel

    var body: some View {
        NavigationView {
            List {
                Section("Subscription") {
                    Text(shopVM.subscriptionActive ? "Active" : "Not active")
                    Text("Benefits: No forced ads, +25% income, +50% offline, 500 tokens/month")
                        .font(.caption)
                        .opacity(0.8)
                }

                Section("Products") {
                    if shopVM.isLoading {
                        ProgressView()
                    } else {
                        ForEach(shopVM.products, id: \.id) { p in
                            VStack(alignment: .leading, spacing: 4) {
                                Text(p.displayName).font(.headline)
                                Text(p.description).font(.caption).opacity(0.8)
                                HStack {
                                    Text(p.displayPrice).bold()
                                    Spacer()
                                    Button("Buy") {
                                        Task {
                                            let ok = await shopVM.buy(p)
                                            if ok {
                                                applyTokenGrantIfNeeded(productID: p.id)
```

```
                                        gameVM.state.subscriptionActive = shopVM.subscriptionActive
                                        gameVM.saveNow()
                                    }
                                }
                            }
                            .buttonStyle(.borderedProminent)
                        }
                    }
                    Divider()
                }
            }
        }

        Section("Restore") {
            Button("Restore Purchases") { Task { await shopVM.restore() } }
        }
    }
    .navigationTitle("Shop")
    .task { await shopVM.load() }
  }
}

private func applyTokenGrantIfNeeded(productID: String) {
    switch productID {
    case PurchaseService.ProductID.tokensSmall: gameVM.state.tokens += 100
    case PurchaseService.ProductID.tokensMedium: gameVM.state.tokens += 500
    case PurchaseService.ProductID.tokensLarge: gameVM.state.tokens += 1200
    case PurchaseService.ProductID.tokensXL: gameVM.state.tokens += 2500
    case PurchaseService.ProductID.tokensXXL: gameVM.state.tokens += 6000
    case PurchaseService.ProductID.tokensWhale: gameVM.state.tokens += 15000
    default: break
    }
    gameVM.state.firstPurchaseMade = true
    FirebaseService.shared.logEvent("first_purchase")
}
}
```

File: Views/LeaderboardView.swift

```
import SwiftUI

struct LeaderboardView: View {
    @ObservedObject var gameVM: GameViewModel
    @ObservedObject var lbVM: LeaderboardViewModel

    var body: some View {
        NavigationView {
            List {
                Section {
                    Button("Submit My Score") {
                        Task { await gameVM.submitLeaderboardIfAllowed() }
                    }
                    Button("Refresh") {
                        Task { await lbVM.refresh() }
                    }
                }
                Section("Top Players") {
                    if lbVM.entries.isEmpty {
                        Text("No data yet. Add Firebase to enable global leaderboard.")
                            .font(.caption)
                            .opacity(0.8)
                    } else {
                        ForEach(Array(lbVM.entries.enumerated()), id: \.element.id) { idx, e in
                            HStack {
                                Text("#\(idx+1)").frame(width: 40, alignment: .leading)
                                VStack(alignment: .leading, spacing: 2) {
                                    Text(e.playerName).font(.headline)
                                    Text("Lifetime: \(NumberFormat.format(e.lifetimeEarnings)) | AI \(e.aiLevel)")
                                        .font(.caption).opacity(0.8)
                                }
                            }
                        }
                    }
                }
            }
            .navigationTitle("Leaderboard")
            .task { await lbVM.refresh() }
        }
```

```
    }
}
```

File: Views/SettingsView.swift

```
import SwiftUI

struct SettingsView: View {
    @ObservedObject var gameVM: GameViewModel
    @State private var name: String = ""
    @State private var feedback: String = ""
    @State private var sending: Bool = false
    @State private var sent: Bool = false

    var body: some View {
        NavigationView {
            Form {
                Section("Player") {
                    TextField("Overlord name", text: $name)
                    Button("Save Name") {
                        let trimmed = name.trimmingCharacters(in: .whitespacesAndNewlines)
                        if !trimmed.isEmpty { gameVM.setPlayerName(trimmed) }
                    }
                }

                Section("Feedback") {
                    TextField("Tell us what to improve", text: $feedback, axis: .vertical)
                        .lineLimit(3...6)
                    Button(sending ? "Sending..." : "Send") {
                        Task {
                            sending = true
                            await gameVM.submitFeedback(feedback)
                            sending = false
                            sent = true
                            feedback = ""
                        }
                    }
                    .disabled(sending || feedback.trimmingCharacters(in: .whitespacesAndNewlines).isEmpty)
                }

                Section("Debug") {
                    Button("Save Now") { gameVM.saveNow() }
                    Button("Grant Tokens (+500)") { gameVM.state.tokens += 500; gameVM.saveNow() }
                }
            }
            .navigationTitle("Settings")
            .onAppear { name = gameVM.state.playerName }
            .alert("Sent", isPresented: $sent) { Button("OK") {} } message: { Text("Thanks!") }
        }
    }
}
```

12. Troubleshooting

If you get 'No such module Firebase...':
- You did not add Firebase packages yet. Add via Swift Package Manager.
- Or keep Firebase optional (it is guarded by canImport).

If StoreKit products do not appear:
- You must create products in App Store Connect.
- For local testing, use StoreKit Configuration file (Xcode > New File > StoreKit Configuration).
- Assign config in Scheme > Run > Options > StoreKit Configuration.

If app crashes on launch:
- Check you copied ALL files exactly and no duplicates.
- Ensure each file is included in the target membership (right panel).

If numbers look wrong:
- Adjust base incomes in DimensionFactory.
- Adjust aiMultPerLevel in GameConstants.