# Zusatz.md — KI‑Dev‑Tycoon (Steam‑MVP, Solo‑Dev, ≤ €500)

*Stand: 06.10.2025 (Europe/Vienna)*

Dieses Dokument fasst **alle bislang getroffenen Entscheidungen, Architekturdetails, Schemas, Algorithmen, UI‑Richtlinien, Implementierungsschritte, Milestones und Zielmetriken** für das Projekt **KI‑Dev‑Tycoon** zusammen. Ziel ist eine **sofort umsetzbare** Solo‑MVP‑Umsetzung auf **Steam (Windows, offline)** mit **Budget ≤ €500** und **einmaligem Kaufpreis €5.99**.

## 1) Entscheidungen & Projektleitplanken (Summary)

* **Plattform (MVP):** **Steam (Windows x64, Desktop)**
* **Team & Budget:** **Solo‑Dev**, Gesamtbarbudget **≤ €500**
* **Online/Cloud:** **Kein Backend, keine Cloud**, reine **Offline‑Simulation**
* **Monetarisierung:** **Paid up‑front €5.99**, **keine Ads**, **keine IAPs**
* **Region (Launch‑Pflicht):** **Österreich** (Steam erlaubt global; Fokus AT okay)
* **Engine:** **Unity 6 LTS (6000.x)**
* **Architektur:** Strikte Trennung **Sim‑Kernel (deterministisch)** ↔ **Client/UI**
* **Tick‑Rate:** **0,5 s** Fixed‑Tick (Accumulator‑Loop)
* **Achievements:** **8–12** einfache Steam‑Achievements (kein Cloud‑Save, keine Leaderboards)
* **Lokalisierung & A11y:** **EN/DE**, skalierbare Schrift, farbenblind‑sichere Paletten, reduzierte Animationen optional
* **Art‑Direction:** **„Tycoon bunt“**, **isometrisch/2.5D**, kontrastreich, verspielte UI
* **Nicht‑Ziele (MVP):** Keine Ads/IAPs/Push/Telemetry‑Cloud/PvP/UGC/Always‑Online

## 2) Background

**Produktidee & Zielsetzung.** KI‑Dev‑Tycoon ist ein Desktop‑Tycoon, der den Aufbau eines KI‑Unternehmens simuliert. Fokus: zugängliche, glaubwürdige KI‑Anspielungen; Sessions kurz bis mittel (5–20 Min) mit langfristiger Meta‑Progression; faire Monetarisierung (einmaliger Kauf).

**Plattform & Audience.** Windows (Steam, 16:9, Maus/Keyboard), Zielgruppe 16–45, tech‑affin.

**Erfolgsmaße (Post‑Launch‑Gate).** Wunschlisten‑→‑Kauf CR ≥ 10 % am Launch‑Tag, Refund‑Rate ≤ 8 %, ≥ 75 % positive Reviews (n≥20), Crash‑Rate ≤ 0,3 %.

**Architektur‑Leitplanken.** Simulations‑Kernel deterministisch (Seed/Time‑Injection), voll getestet (Coverage ≈ ≥ 90 %), klare API/Versionierung. CI/CD mit Lint/Typing/Tests. Keine Unity‑APIs im Kernel.

**Prozess.** Source of Truth: diese Spezifikation; Änderungen an Balancing‑Assets via PR‑Review; reproducible Seeds/Snapshots.

## 3) Requirements (MoSCoW)

**Must** - **Budget/Team:** Solo‑Dev, **≤ €500** (Gebühren/Assets/Store) - **Plattform:** **Windows (Steam)**, 16:9, Ziel 60 FPS; IL2CPP im Release - **Offline‑Only:** deterministische Tick‑Simulation, **lokale Save‑Slots** + Migration; **kein** Backend - **Kern‑Loops:** Hiring, Forschung (Modelle/Datasets), Training (Kosten/Qualität), Produkt‑Launches, Umsatz/Skalierung, Events - **Monetarisierung:** **€5.99**, **ohne** Ads/IAPs - **Distribution:** Veröffentlichung mind. in **Österreich** - **Compliance:** GDPR‑freundlich; minimale Berechtigungen; keine externen Tracker - **L10n/A11y:** EN/DE, skalierbare Schrift, farbenblind‑freundliche Paletten - **Build/CI:** Unity 6 LTS, deterministischer Kernel (Tests ≥ 90 %), Lint/Static‑Analysis, Nightly Local Builds

**Should** - **Store‑Assets:** 30–45 s Trailer (In‑Engine Captures), 6 Screenshots, Key‑Art - **Steam‑Deck‑Check:** Basiskompatibilität prüfen - **Optionale Demo:** (itch.io) für Wishlist/Feedback

**Could** - **Cosmetics/Skins**, **Challenges**, **Achievements** (erweitert) - **Season‑Mini‑Events** (rein kosmetisch)

**Won’t (MVP)** - Ads, IAPs, Remote‑Config, Push‑Notifications, Telemetrie‑Cloud, Always‑Online, PvP, UGC

## 4) Method (Technische Methode)

### 4.1 Architektur (Steam · Offline · Solo‑Dev)

* **Engine & Versioning.** Unity **6.0 LTS (6000.x)**; Exporte: Windows x64; IL2CPP Release, Mono Editor
* **Steam‑Integration.** **Steamworks.NET** (MIT) für Achievements/Overlay; **ohne** Steam Cloud; UPM‑Git Integration
* **Schichten.**
  + Core.Sim — deterministischer Sim‑Kernel (reines C#; **keine Unity‑APIs**)
  + Core.Data — statische Balancing‑Daten (ScriptableObjects) + Loader
  + Game.UI — UGUI (Canvas/TMP), Presenter/MVP
  + Game.App — Bootstrap, Tick‑Loop, DI‑Wire‑up, Save/Load
  + Platform.Steam — dünner Adapter (Achievements/Rich Presence)
* **Determinismus.** Fester **Tick = 0,5 s**; ITimeProvider/IRng injiziert; kein DateTime.Now/System.Random im Kernel
* **Persistenz.** **JSON + GZip**, 1–3 Rotations‑Slots, Migrations‑Layer ISaveMigrator (vN → vN+1)

@startuml  
package "Game Client (Unity)" {  
 [Bootstrap] --> [GameLoop]  
 [UI Layer] --> [Presenters]  
}  
package "Core" {  
 [Sim Kernel] --> [RNG]  
 [Sim Kernel] --> [Time Provider]  
 [Sim Kernel] --> [Data Catalog]  
}  
package "Platform" {  
 [Steam Adapter]  
}  
[Bootstrap] --> [Steam Adapter]  
[GameLoop] --> [Sim Kernel]  
[GameLoop] --> [Save System]  
[Presenters] --> [Sim Kernel]  
[Save System] --> [File (JSON.GZ)]  
@enduml

### 4.2 Tick‑Loop & Offline‑Berechnung

* **Fixed‑Tick‑Accumulator.** acc += delta; while(acc>=Tick){ Sim.Tick(); acc-=Tick; }
* **Offline‑Progress.** Beim Laden: dt = clamp(now-lastSave, 0, capHours\*3600); n = floor(dt/Tick); für große dt nutzt der Kernel **aggregierte Formeln** (z. B. analytische S‑Kurven) statt per‑Tick Iteration
* **Seed & RNG.** **PCG32** (eigene kleine Implementierung) mit worldSeed, companyId, dayIndex als Stream‑Keys (replayable)

@startuml  
actor Player  
Player -> App: Start  
App -> SaveSystem: Load(ActiveSlot)  
SaveSystem --> App: SaveState(lastTime, seed, ...)  
App -> Sim: ApplyOfflineProgress(dt)  
App -> UI: ShowMainMenu  
== Runtime ==  
loop Every Frame  
 App -> GameLoop: Update(delta)  
 GameLoop -> Sim: Tick() (0.5s)  
 Sim --> UI: DomainEvents (pub/sub)  
end  
== Quit ==  
UI -> SaveSystem: Save()  
@enduml

### 4.3 Datenmodelle & Schemas (MVP)

**IDs.** Alle Referenzen per string id (kebab‑case); keine direkten Objekt‑Refs im Save.

**ScriptableObjects (statisch):** - RoleDef { id, name, baseSalary, skills{ml,data,eng,prod}, hireWeight } - ResearchNodeDef { id, name, tier, cost, timeHrs, prereqIds[], grants{modelQuality+, infraCap+, featureFlags[]} } - ProductDef { id, name, segmentId, baseK, baseChurn, priceRange, costPerUser } - MarketSegmentDef { id, name, TAM, priceElasticity, techAffinity } - EventDef { id, name, weight, triggerRules, effects{multiplier,target,duration} }

**Save (JSON v1 Beispiel).**

{  
 "version": 1,  
 "created\_at": "2025-10-06T08:00:00Z",  
 "last\_time": "2025-10-06T10:00:00Z",  
 "seed": 9201531,  
 "company": {  
 "name": "My AI Co",  
 "reputation": 0.0,  
 "cash": 20000,  
 "burn\_rate": 0,  
 "employees": ["emp-001","emp-002"],  
 "research\_queue": ["res-basic-nlp"],  
 "active\_products": ["prod-chatbot-a"]  
 },  
 "employees": {  
 "emp-001": {"role":"ml-engineer","level":1,"skills":{"ml":3,"data":2,"eng":2,"prod":1},"salary":2200},  
 "emp-002": {"role":"data-scientist","level":1,"skills":{"ml":2,"data":3,"eng":1,"prod":1},"salary":2100}  
 },  
 "research": {"unlocked": ["res-basic-nlp"], "in\_progress": []},  
 "products": {  
 "prod-chatbot-a": {"def":"chatbot","launch\_day": 10, "users": 0, "price": 6.99, "quality": 0.42 }  
 },  
 "economy": {"day": 12, "inflation": 0.02},  
 "rng\_stream": {"hiring": 12345, "events": 777, "market": 444}  
}

### 4.4 Kern‑Formeln (einfach, erweiterbar)

* **Hiring (pro Tag):** N ~ Poisson(λ) Kandidat:innen; Qualität q ~ Normal(μ(role)+β\*reputation, σ); Gehalt salary = baseSalary(role) \* (1 + q\*kq + marketAdj)
* **Forschung/Training (pro Tick):** dp = (Σ staffSkill\*roleWeight)\*labMultiplier\*(1‑burnout); fertig bei Σdp ≥ cost → grants anwenden
* **Produkt‑Qualität:** quality = tanh(a\*modelQuality + b\*ln(dataQty+1) + c\*infraCap)
* **Adoption (S‑Kurve):** users(t) = K / (1 + e^(‑r\*(t‑t0))), K = TAM\*fit(quality,price), r skaliert mit Reputation/Marketing
* **Umsatz pro Tick:** rev = users \* price / ticksPerDay \* ARPUmult - costPerUser \* users / ticksPerDay
* **Events:** gewichtete Auswahl; Effekte als Multiplikatoren mit Dauerfenster

### 4.5 Dateistruktur & Pakete

Assets/  
 \_Project/  
 Scripts/  
 Core.Sim/  
 Core.Data/  
 Game.UI/  
 Game.App/  
 Platform.Steam/  
 Data/  
 RoleDefs/\*.asset  
 Research/\*.asset  
 Products/\*.asset  
 Markets/\*.asset  
 Events/\*.asset  
 Resources/  
 Addressables/  
Build/

**Unity‑Pakete:** TextMeshPro, Input System, Addressables (nur lokal), Localization (DE/EN)

### 4.6 Steam‑Schnittstellen (nur MVP‑Umfang)

* **Init & Overlay:** SteamAPI.Init() / SteamAPI.Shutdown(); Overlay Toggle; steam\_appid.txt im Dev
* **Achievements:** 8–12 klare Ziele (z. B. Erstes Produkt, 10k User, erste Forschung Tier 2)
* **Rich Presence:** kurzes Status‑String (z. B. „Day 12 • 1 Product • 5 Staff“)

### 4.7 Vergleichende Referenzen (Design‑Ableitungen)

* *Game Dev Tycoon* (Projekt‑Phasen → Produkt‑Score/Qualität)
* *Software Inc.* (Personal‑Management → Rollen/Skills stark vereinfacht)
* *Startup Company* (Feature‑Roadmaps → Research‑Nodes/Feature‑Flags)

### 4.8 Testbarkeit & Determinismus

* **Golden‑Seeds:** fixe Seeds + Save‑Snapshots für Regression
* **Property‑Tests:** Invarianten (z. B. Adoption ≤ TAM, Cash driftet nicht unphysikalisch)
* **Headless‑Runner:** CLI‑Simulation N‑Tage für Balancing‑Batches (CSV‑Dump lokal)

### 4.9 Visual Style Guide — **Tycoon bunt** (isometrisch/2.5D)

* **Palette (hell, kontrastreich):** Primär Indigo #4F46E5; Akzente Amber #F59E0B, Emerald #10B981, Pink #EC4899; Neutrals #F5F7FB/#FFFFFF; Text #0F172A/#475569; Status Success #16A34A, Warning #F59E0B, Danger #EF4444; Kontrast: Body ≥ 4.5:1
* **Typografie (frei/OFL):** Headline **Rubik** (Semibold), Body **Inter** (Regular)
* **Komponenten:** Buttons Radius 14–16 px, Shadow (0,4,12,0.2), Hover +4 %, Press‑Scale 0.96; Karten Radius 12 px, Verlauf‑Header; Badges/Chips gesättigt; Charts ohne 3D
* **Icon/Illustration:** **isometrisch/2.5D**, weicher Schatten (y 6–10, blur 12–18), Licht von rechts‑oben
* **Motion:** LeanTween easeOutBack (0.38–0.48 s); Achievement‑Toast 250 ms → 1.5 s → 200 ms; Confetti 25–35 Partikel
* **Audio:** sanfte UI‑Foleys; Münz‑SFX bei Umsatz‑Milestones; Pitch‑Varianz ±5 %
* **A11y:** Farbblind‑Check (Deuter/Protan), Fokus‑Outline 2 px, Toggle „reduzierte Animationen“
* **Asset‑Pfad:** Vektor‑SVG → SDF in Unity; CC0‑Packs oder Eigenproduktion (Inkscape/Krita)

## 5) Implementation (Schritte)

### 5.1 Projektaufsetzung (Tag 0–1)

1. **Unity 6 LTS** via Hub, neues 2D‑Core‑Projekt ai-dev-tycoon
2. **Pakete:** TextMeshPro, Input System, Localization, Addressables (lokal)
3. **Ordnerstruktur** gemäß §4.5; Namespaces Core.Sim, Game.App, Game.UI, Platform.Steam
4. **Steamworks.NET** via UPM‑Git; steam\_appid.txt im Projektroot; SteamManager in Bootstrap‑Scene
5. **Build‑Settings:** StandaloneWindows64, IL2CPP (Release), DebugSymbols off, Stripping Low

### 5.2 Kern‑Code‑Skeleton (Tag 1–5)

**Zeit & RNG**

public interface ITimeProvider { double UtcNow(); }  
public sealed class SystemTimeProvider : ITimeProvider {  
 public double UtcNow() => DateTimeOffset.UtcNow.ToUnixTimeSeconds();  
}  
public interface IRng { uint NextUInt(); float Next01(); }  
public sealed class Pcg32 : IRng {  
 private ulong \_state, \_inc;  
 public Pcg32(ulong seed, ulong seq){ \_state=0; \_inc=(seq<<1)|1; NextUInt(); \_state+=seed; NextUInt(); }  
 public uint NextUInt(){  
 ulong old=\_state; \_state = old\*6364136223846793005UL + \_inc;  
 uint xorshift=(uint)(((old>>18)^old)>>27); int rot=(int)(old>>59);  
 return (xorshift>>rot) | (xorshift<<((-rot)&31));  
 }  
 public float Next01()=> (NextUInt()>>8) \* (1.0f/16777216f);  
}

**Tick‑Loop**

public sealed class GameLoop : MonoBehaviour {  
 public float tickSeconds = 0.5f; float acc; Sim \_sim; ITimeProvider \_time;  
 void Awake(){ \_time=new SystemTimeProvider(); \_sim=new Sim(/\*deps\*/); }  
 void Update(){ acc += Time.deltaTime; while(acc>=tickSeconds){ \_sim.Tick(); acc-=tickSeconds; } }  
}

**Save/Load (JSON+GZip)**

public static class SaveIO {  
 static string PathFor(string slot)=> System.IO.Path.Combine(Application.persistentDataPath,$"save\_{slot}.json.gz");  
 public static void Write<T>(string slot, T obj){  
 var json=System.Text.Json.JsonSerializer.Serialize(obj);  
 using var fs=File.Create(PathFor(slot));  
 using var gz=new System.IO.Compression.GZipStream(fs, System.IO.Compression.CompressionLevel.SmallestSize);  
 using var sw=new StreamWriter(gz); sw.Write(json);  
 }  
 public static T Read<T>(string slot){  
 using var fs=File.OpenRead(PathFor(slot));  
 using var gz=new System.IO.Compression.GZipStream(fs, System.IO.Compression.CompressionMode.Decompress);  
 using var sr=new StreamReader(gz);  
 var json=sr.ReadToEnd(); return System.Text.Json.JsonSerializer.Deserialize<T>(json);  
 }  
}

**Achievements (Steam)**

public static class Ach {  
 public static void Unlock(string apiName){  
 if(!SteamManager.Initialized) return;  
 SteamUserStats.SetAchievement(apiName);  
 SteamUserStats.StoreStats();  
 }  
}

### 5.3 Daten & Balancing (Tag 3–10)

* **ScriptableObjects:** 5 Rollen, 12–18 Research‑Nodes (Tier 1–3), 3 Produkt‑Segmente
* **Start‑Ökonomie:** Startcash €20k, einfache Mieten/Gehälter, 1 Produkt‑Blueprint
* **Events:** 6–8 Einsteiger‑Events (positiv/negativ) mit Multiplikatoren

### 5.4 UI/UX (Tag 6–14)

* **Hauptscreens:** HQ‑Dashboard (Cash, Burn, Reputation), Team, Forschung, Produkt(e), Markt, Events/Log
* **Navigation:** Bottom‑Bar (5 Tabs), kontextsensitive Panels, Tooltips
* **A11y:** Schriftgrößen 90–120 %, farbenblind‑freundlich

### 5.5 Steam‑Setup & Builds (Tag 8–16)

1. **Steamworks‑Partner:** App anlegen; **Achievements** definieren (8–12)
2. **Depots:** 1× Windows x64 Depot
3. **ContentBuilder:** app\_build\_<appid>.vdf; Batch für Headless‑Build & Upload
4. **Store‑Seite:** Beschreibung (DE/EN), 6 Screenshots, Header Capsule, Preis **€5.99**, optional „Early Access“

### 5.6 QA & Tests (ab Tag 10, laufend)

* **Unit/Property‑Tests** für Kernel (Determinismus, Invarianten)
* **Headless‑Runner** für 30 Ingame‑Tage; KPIs CSV lokal
* **Manual QA:** Save‑Rotation, Achievement‑Trigger, Offline‑Cap, Performance (FrameTime ≤ 16 ms)

### 5.7 Lokalisierung (Tag 12–15)

* **EN zuerst**, danach **DE**; Umschalter im Settings‑Menü; Localization‑Package‑Keys

### 5.8 Release‑Checkliste (Tag 15–18)

* Tag v0.1.0, Build hochladen, Achievements live testen, Pricing/Tax, Altersfreigaben, QA‑Pass, Presskit‑Readme

### 5.9 Budget (≤ €500; Ziel ≤ €200)

* **Fix:** Steam Direct Fee ≈ $100 (~€95)
* **Tools (0 €):** Unity Personal, GIMP/Krita, Inkscape, Audacity
* **Assets:** bevorzugt freie/CC0‑Packs (UI‑Icons/SFX/Musik); optional €50–€100 für 1–2 Packs
* **Puffer:** Store‑Grafiken Mini‑Budget €50–€150

## 6) Milestones

* **W1 — Projekt & Kernel:** Unity‑Projekt, Steamworks.NET läuft, Tick‑Loop & RNG implementiert, ~30 Unit‑Tests grün
* **W2 — Daten & Ökonomie:** ScriptableObjects, 1 Produkt, Hiring/Forschung/Training, Offline‑Cap
* **W3 — UI‑First‑Pass:** Greybox aller Screens, Navigation & Tooltips, Save/Load Rotation
* **W4 — Steam & Content:** 8–12 Achievements, 6 Screenshots, Capsule‑Art V1, Store‑Page im Review
* **W5 — Polish & Beta:** Balancing‑Pass, Profiling, DE‑Lokalisierung, Bugfix‑Sprint
* **W6 — Release:** v0.1.0 live (AT inkl. weltweit), Preis €5.99, Kommunikations‑Post, Hotfix‑Plan

## 7) Gathering Results (Messung ohne Cloud)

* **Ziele (Launch + 2 Wochen):** Wishlist→Sale CR ≥ 10 %, Refund ≤ 8 %, ≥ 75 % positive Reviews (n≥20), Crash ≤ 0,3 %
* **Messung:** Steam Backoffice (Sales/Refunds/Wishlists/Reviews); **optional** lokaler CSV‑Export (Dev‑Mode)
* **Gates:**
  + **Gate A:** ≥ €1.000 Umsatz → Fee kompensiert; Roadmap für Early‑Access
  + **Gate B:** CR < 5 % **oder** Reviews < 80 % → Balancing/UX‑Pass vor Feature‑Erweiterung

## 8) Anhang

### 8.1 Achievement‑Startliste (Beispiele)

* *First Launch:* erstes Produkt veröffentlicht
* *First 1k:* 1.000 Nutzer:innen
* *First 10k:* 10.000 Nutzer:innen
* *Tier‑2 Unlocked:* erste Forschung Tier 2 abgeschlossen
* *Profitable Month:* 30 Tage mit positivem Cashflow
* *Stable Release:* 7 Tage ohne Crash

### 8.2 Test‑Invarianten (Property‑Tests)

* Adoption **≤ TAM** (pro Segment)
* Cash kann nicht negativ werden ohne Kredit‑System
* Produkt‑Qualität in [0,1]
* RNG‑Streams deterministisch pro Seed/StreamKey

### 8.3 UI‑Screens (Wireframe‑Plan)

* **HQ‑Dashboard:** Cash, Burn, Reputation, laufende Projekte, Events‑Ticker
* **Team:** Rollenliste, Bewerbungen (Tagesbatch), Hiring‑Dialog
* **Forschung:** Tech‑Tree (Tier 1–3), Queue, Fortschrittsbalken
* **Produkt:** Blueprint → Launch → KPIs (Users/Churn/Revenue)
* **Markt:** Segmente, TAM, Preis‑/Qualitäts‑Fit, Wettbewerbs‑Noise (einfach)
* **Events/Log:** Zeitlich sortiert, Tooltip‑Effekte, Dauer

## Need Professional Help in Developing Your Architecture?

Bitte kontaktiere mich auf [**sammuti.com**](https://sammuti.com) 🙂