ChatGPT

Step 3 – Impostazione del backlog centralizzato

1. Attività svolte

- **Analisi delle checklist e delle milestone**: ho esplorato i file `docs/checklist/milestones.md` e `docs/checklist/project-setup-todo.md` per individuare le attività ancora da completare e i riferimenti alle sezioni del repository (dataset, CLI, webapp, QA, ecc.). Le milestone aperte includono lo sviluppo dell'overlay HUD telemetrico e il riequilibrio dell'XP nel profilo Cipher [1], l'espansione degli smoke test per coprire asset statici e la creazione dei tag di release [2] 【574174961222444†L78-L80】.
- **Raccolta delle attività future**: dalla checklist generale ho estratto le attività di manutenzione continua, come l'aggiornamento della roadmap e delle checklist, l'automazione dei test, l'archiviazione dei log playtest e la programmazione delle sessioni di review [3].
- **Classificazione delle categorie**: ho organizzato le attività in categorie tematiche (dataset, CLI, backend, webapp, QA/testing, sicurezza/DevOps, documentazione, release), preparandole per essere inserite nel backlog.

2. Struttura proposta del backlog

Per ottenere un backlog centralizzato, propongo di creare un **progetto GitHub (Projects)** con colonne **Backlog → In Progress → Review QA → Done**. Ogni task va inserito come *issue* con etichette per categoria e fase. Di seguito, le categorie principali con esempi di attività da tracciare:

A. Dataset & dati

- Verificare che i file YAML dei dataset (biomi, packs, telemetria, specie, tratti) siano coerenti con il GDD; eseguire regolarmente gli script di validazione (`validate_datasets.py`, `report_trait_coverage.py`).
- Espandere i set di esempi di encounter per ciascun bioma e aggiornarli in `docs/examples/`, come indicato nei seed demo [4].
- Allineare la tabella `compat_forme` con tutte le 16 forme MBTI e assicurarsi che il glossario tratti sia aggiornato (workflow sinergie e baseline [5]).
- Integrare le nuove specie e biomi (es. planar ruins) nelle tabelle `species.yaml` e `biomes.yaml`, e aggiornare la matrice specie→tratti.

B. CLI & tools (Python/TypeScript)

- Garantire parità funzionale tra `tools/ts` e `tools/py` (roll pack, generate encounter) e documentare eventuali difformità [6].
- Aggiungere nuovi comandi per la manipolazione dei dataset (e.g. tuning telemetria, generation snapshots) e integrarne la documentazione.
- Scriptare la generazione automatica di report (es. `scripts/daily_pr_report.py`, `scripts/analytics/etl_squadsync.py`) nel flusso CI.

C. Backend & API

- Consolidare l'API GraphQL/REST esistente (squad sync, telemetria) e documentare gli endpoint.

- Integrare il meccanismo Tri-Sorgente come servizio: definire un endpoint che riceva MBTI/Enneagram, contesto e telemetria e restituisca le carte selezionate secondo la formula [7].
- Aggiornare le pipeline ETL (scripts/analytics) per includere i nuovi segnali (HUD Smart Alerts, risk tuning) e alimentare i dashboard.

D. Webapp & UI

- **Smart HUD & SquadSync**: implementare l'overlay HUD telemetrico per mostrare in tempo reale risk/cohesion e i badge SquadSync, come richiesto nella milestone [4].
- Implementare una UI per la scelta delle carte Tri-Sorgente e per i feedback dei giocatori (skip, conferma)
- Aggiornare la mission console con quick actions e timeline filtrabile, come indicato nel Canvas feature updates [8].
- Migliorare accessibilità, localizzazione e responsive design del webapp; integrare i grafici e i report generati dai moduli analytics.

E. QA & Testing

- Estendere gli smoke test per includere asset statici e validare le performance del bundle web [2].
- Aggiungere test Playwright e unit test per la UI, i generatori Python/TS e l'engine Tri-Sorgente.
- Automatizzare la generazione di report giornalieri per PR fuse e test di regressione (workflow `daily-pr-summary`).
- Preparare scenari di playtest critici (BAL-xx, PROG-xx, EVT-xx) seguendo la checklist QA e registrare i risultati [9].

F. Sicurezza & DevOps

- Integrare l'analisi statica e i controlli di sicurezza (Secure SDLC) nei workflow CI/CD, includendo la validazione YAML e l'audit dei tratti.
- Monitorare le performance delle pipeline (es. deploy test interface) e ottimizzare i tempi di build; mantenere la dipendenza dal `playwright-chromium-bundle` e altri artefatti come da checklist [10].
- Documentare e automatizzare il processo di tagging delle release (`v0.6.0-rc1`, `v0.6.0`), definendo i criteri di go/no-go [2].

G. Documentazione & Comunicazione

- Consolidare il GDD in `docs/GDD.md` (step 2) e aggiornarlo a ogni modifica di feature o dataset.
- Aggiornare README, roadmap e canvas con le nuove procedure (CLI, data sync, Playwright, Slack/Drive integration).
- Pubblicare ADR per le decisioni architetturali e creare FAQ interne. [11]
- Pianificare sessioni di onboarding e condividere presentazioni e guide nelle cartelle `docs/presentations/` [11].

H. Rilascio & marketing

- Seguire la checklist di rilascio (creazione tag, preparazione changelog, materiali di comunicazione) [12].
- Coordinarsi con marketing/product per annunci e raccolta feedback; registrare le date importanti nel backlog.

- Aggiornare le dashboard e il canvas con screenshot, metriche risk/cohesion e note VC in preparazione delle release successive [13] .

3. Creazione del backlog

1. **Creare un progetto GitHub** (“Evo Tactics Backlog”) con colonne *Backlog*, *In Progress*, *Review QA*, *Done*.
2. **Importare le attività** elencate sopra come issue, assegnando categorie tramite etichette (e.g. `dataset`, `cli`, `webapp`, `QA`, `security`, `docs`, `release`). Usare milestone dedicate (es. “VC 0.6.0 RC1”, “Smart HUD”, “Tri-Sorgente Beta”).
3. **Collegare issues e pull request**: quando un PR chiude un’issue, lo stato deve aggiornarsi automaticamente; sfruttare il workflow `daily-pr-summary` per monitorare le fusioni.
4. **Aggiungere date di scadenza** e owner per ogni issue, in linea con le milestone della roadmap (ad esempio, completare HUD telemetrico entro 2025-11-09 [4]).
5. **Mantenere aggiornato il backlog**: durante le riunioni sprint/review, spostare le carte tra colonne e aggiungere nuove attività dal playtest, dalle retrospettive e dal daily PR summary.

4. Prossimi passi

- Dopo aver creato il progetto backlog, importare progressivamente le attività correnti e pianificate; priorità alla milestone “Smart HUD & SquadSync” e alla finalizzazione della release `v0.6.0-rc1`.
- Coordinare con i responsabili di ciascuna area (dataset, tools, webapp, QA) per assegnare le issue e definire stime e dipendenze.
- Integrare la visibilità del backlog nel canvas/roadmap e assicurare che le nuove voci del GDD risultino riflettute nei ticket.

Questa impostazione consentirà al team di gestire in modo centralizzato tutte le attività aperte, seguendo il ciclo di vita dello sviluppo (backlog → in progress → review → done), migliorando la trasparenza e l’allineamento tra sviluppo, design, QA e marketing.

---

[1] [2] [4] [10] [13] milestones.md
https://github.com/MasterDD-L34D/Game/blob/HEAD/docs/checklist/milestones.md

[3] [6] [9] [11] [12] project-setup-todo.md
https://github.com/MasterDD-L34D/Game/blob/HEAD/docs/checklist/project-setup-todo.md

[5] DesignDoc-Overview.md
https://github.com/MasterDD-L34D/Game/blob/HEAD/docs/DesignDoc-Overview.md

[7] overview.md
https://github.com/MasterDD-L34D/Game/blob/HEAD/docs/tri-sorgente/overview.md

[8] feature-updates.md
https://github.com/MasterDD-L34D/Game/blob/HEAD/docs/Canvas/feature-updates.md