

Relazione del progetto “Questmaster”

Corso di Intelligenza Artificiale

Anastasia Martucci
Matricola: 271316

Giuseppe Zappia
Matricola: 268784

July 28, 2025

Introduzione

L’obiettivo di questa relazione è quello di descrivere le fasi di realizzazione del progetto, illustrando le scelte progettuali adottate, le tecnologie utilizzate e le difficoltà incontrate. Verranno inoltre presentati i risultati ottenuti e alcune riflessioni finali sul lavoro svolto.

1 Scopo del progetto

Lo scopo del progetto è la realizzazione di QuestMaster, un sistema in due fasi progettato per supportare la creazione di esperienze narrative interattive combinando tecniche di pianificazione classica (PDDL) e modelli di intelligenza artificiale generativa (LLM). Nella prima fase, il sistema assiste l’autore nella generazione di una quest narrativa logicamente coerente sotto forma di problema di pianificazione, fornendo strumenti per costruire e validare automaticamente file PDDL. Nella seconda fase, a partire dalla quest validata, viene generata un’applicazione web interattiva che consente all’utente finale di esplorare la storia, effettuare scelte e vivere l’avventura in modo dinamico. L’obiettivo complessivo è integrare pianificazione simbolica e AI generativa per creare strumenti narrativi avanzati, accessibili e interattivi.

2 Gestione del lavoro e punti fondamentali

Di seguito vengono riportati le soluzioni progettuali principali del nostro lavoro, ognuna si colloca in una fase precisa della pipeline di esecuzione e verrà meglio analizzata nelle sezioni successive:

- **Scelta delle API:** Gemini per garantire velocità, precisione e completezza nelle risposte

- **Few-shots prompting:** per permettere risposte più accurate su argomenti più tecnici e di nicchia come la generazione di codice PDDL
- **Human-in-the-loop:** atto a garantire un diretto coinvolgimento dell'utente nel processo decisionale, dando la possibilità di modificare dinamicamente l'avanzamento della storia
- **Defensive prompting:** che evita che l'ilm risponda a prompt malevoli da parte dell'utente, tutelandosi.
- **Reflective agent:** capace di intervenire in presenza di errori durante la validazione del piano creato o per implementare le modifiche richieste dinamicamente dall'utente
- **Lore integration con RAG:** per permettere al modello di avere una mini-base di conoscenza che lo aiuti quando necessario con la creazione della lore
- **GUI che comprende anche la fase 1:** per garantire un'esperienza completa all'utente che può scegliere se sfruttare totalmente l'app web-based fin dall'inizio oppure se procede da linea di comando per la prima fase e limitarsi ad usare la GUI solo per giocare la storia
- **Possibilità di scaricare la storia:** grazie al download di un file zip che comprende tutte le dinamiche della storia, dall'input iniziale alle varie scelte fatte
- **Integrazione audio della lore e dei vari passi di gioco:** nella schermata di creazione della lore è possibile ascoltare la stessa letta da un narratore, cos' come i singoli passi della storia.

Fase 1

La Fase 1 è stata schematizzata nei seguenti bullet point:

- **Accessibilità della Fase 1:** la Fase 1 può essere svolta sia **da linea di comando** che attraverso una **interfaccia grafica (GUI)**, rendendo il sistema flessibile e adatto a diverse modalità di interazione da parte dell'utente. Per eseguire tutto tramite la GUI è necessario immettere nel terminale il seguente comando:

```
streamlit run gui/gui_completa.py
```

Altrimenti, è sufficiente avviare il file `main.py` ed interagire da CLI. La fase 2 tramite GUI verrà automaticamente mostrata all'utente alla fine del processo di creazione della storia.

- **Caratteristiche del modello usato:** Il modello generativo adottato è `gemini-2.5-pro`, inizializzato mediante la libreria `ChatGoogleGenerativeAI` con temperatura impostata a 0.6. Questo modello è stato utilizzato per generare testo naturale e codice PDDL a partire dalle descrizioni fornite dall'utente.
- **Generazione della *lore*** : la prima parte della Fase 1 è dedicata alla creazione della storia di riferimento. L'utente, tramite linea di comando o relativa interfaccia, fornisce un'idea iniziale della narrazione da sviluppare. In questa fase abbiamo scelto di integrare un approccio **Human-In-The-Loop**, che consente allo scrittore di intervenire attivamente nella generazione del testo, quindi dopo aver inserito la prima richiesta, l'utente avrà modo di leggere i dettagli della storia creata e decidere se accettarla proseguendo alla fase successiva, oppure se correggerla o arricchirla, prima di considerarla definitiva. In questo ultimo caso, dopo aver scelto le modifiche da apportare, verrà dietro le quinte chiamato in causa il reflective agent che si occuperà di generare il tutto correttamente. Il modello che genera la *lore* riceve in input degli esempi contenuti in una specifica cartella del progetto, e li usa grazie alla tecnica di few-shots prompting, per orientarsi nella creazione della struttura narrativa. Gli esempi sono in formato json per garantire più semplicità di interpretazione degli stessi da parte del modello, e per fissare uno schema ricorrente da usare.
- **Lore integration with RAG:** Sempre in questa fase viene integrata la possibilità per il modello di rifarsi ad una base di conoscenza contenuta in un file in formato pdf, viene quindi eseguita la classica pipeline sul file (embedding, vector store, split ecc.) e solo se ritenuto necessario, il nostro llm procede a raccimolare informazioni dal file.
- **Tecnica del Few-Shot Prompting:** per rendere più efficace la generazione automatica e orientare il comportamento del modello, è stato impiegato il **Few-Shot Prompting**. Viene fornita in input una *lore* di esempio strutturata secondo il formato atteso, così da istruire il modello su come dovrebbe essere la storia. Questo approccio ha migliorato la coerenza dei risultati generati, sia dal punto di vista narrativo sia rispetto ai vincoli relativi alla pianificazione, agli obiettivi e all'ambientazione.
- **Generazione dei file PDDL:** una volta che l'utente approva la storia e la *lore* finale, il sistema procede automaticamente alla generazione dei file `domain.pddl` e `problem.pddl`, fondamentali per la pianificazione automatica. La funzione `create_domain_pddl(llm)` è responsabile della creazione del file di dominio, utilizzando un prompt strutturato che istruisce il modello LLM a convertire la *lore* JSON in una definizione STRIPS valida. Il prompt richiede di definire predicati, azioni con precondizioni ed effetti, e di commentare ogni sezione del file PDDL generato. Il codice risultante viene salvato localmente nel file `domain_generato.pddl`. Analogamente, la funzione `create_problem_pddl(llm)` produce l'istanza del problema specifico, compatibile con il dominio appena generato. Essa include

la definizione degli oggetti della storia, lo stato iniziale e le condizioni di goal, tutti estratti e modellati coerentemente rispetto alla lore. Anche in questo caso, il prompt impone l'uso della sintassi PDDL standard e di commenti esplicativi per ogni riga. Il file risultante `problem_generato.pddl` è poi utilizzato per la validazione tramite planner. Entrambe le funzioni sono progettate per garantire la compatibilità con Fast Downward e rispettano i vincoli narrativi imposti dalla lore (branching factor e depth constraints).

- **Validazione automatica e iterativa:** una volta generati i file `domain.pddl` e `problem.pddl`, il sistema avvia un *ciclo di validazione automatica* volto a garantire che la formulazione PDDL sia corretta e che esista un piano valido e semanticamente coerente. Questo processo è gestito dalla funzione `run_fastdownward_complete()`, che comprende due fasi principali:

- **Pianificazione con Fast Downward:** la funzione `run_fastdownward_planning()` esegue il planner *Fast Downward* tramite WSL, utilizzando i file PDDL generati come input. Il sistema valuta se sia stato trovato un piano valido, memorizzando eventuali errori di sintassi o coerenza logica.
- **Recupero del piano (sas_plan):** se la pianificazione ha successo, viene recuperata la sequenza di azioni (piano) tramite la funzione `get_fastdownward_solution()`, che estrae e analizza il contenuto del file `sas_plan`.

Una volta ottenuto il piano, il sistema procede con la **validazione semantica** tramite lo strumento *VAL*, invocato dalla funzione `validate_plan_with_val()`. Questa funzione esegue il validatore VAL via WSL, analizzando il piano rispetto al dominio e al problema. L'output viene poi interpretato per stabilire se il piano rispetta i vincoli semantici, le precondizioni e gli obiettivi della quest. Vengono raccolti anche errori di esecuzione e warning tramite la funzione `parse_val_output()`.

In caso di fallimento della validazione, il sistema è progettato per innescare un *ciclo di correzione automatica iterativa*, grazie a un LLM che riceve in input l'output di errore di VAL e propone una nuova versione dei file PDDL, ripetendo l'intero processo fino al raggiungimento di una validazione completa o al superamento di un numero massimo di tentativi.

Questo approccio automatizzato consente di garantire la coerenza logica della storia pianificata, evitando errori strutturali e mantenendo una stretta aderenza tra narrativa e modello formale di pianificazione.

- **Intervento del Reflective Agent:** Questo agente entra in gioco in più fasi del processo, in particolare sia per correzioni dovute all'intervento dell'uomo (modifica di lore o piano per raggiungere l'obiettivo), sia nel caso in cui la validazione fallisca. Il **Reflective Agent**, è un agente intelligente basato su **Large Language Models (LLM)**, questo analizza gli

errori prodotti durante la validazione e propone delle modifiche al modello PDDL, cercando di risolvere i problemi rilevati in modo automatico, oppure sottopone nuovamente al modello la richiesta dell'utente per la modifica della lore o del piano.

- **Supervisione e interazione umana:** le modifiche proposte dal Reflective Agent non vengono applicate automaticamente. L'utente ha infatti la possibilità di approvare direttamente i suggerimenti generati, oppure di intervenire manualmente con modifiche personalizzate. Anche in questa fase, il processo rimane **Human-In-The-Loop**, mantenendo il controllo dell'utente sul comportamento del sistema. Questo è un punto cruciale del progetto perché permette all'utente di rendere il processo soggettivo e personale, garantendo numerose modifiche secondo la sua percezione e idea della storia. Infine questo approccio viene riscontrato anche in una fase più tecnica che è quella in cui c'è un errore nel domain o problem e il modello non riesce a capire nello specifico in quale dei due sia l'errore, perciò chiede direttamente all'utente facendo vedere l'output prodotto in seguito all'errore.
- **Defensive prompting:** utile perché aiuta a rendere le interazioni con il modello linguistico più sicure, affidabili e controllabili. Inserendo istruzioni chiare e vincoli esplicativi, si riduce il rischio che il modello generi risposte scorrette, inappropriate o fuori contesto. Per implementarlo oltre che inserire chiare istruzioni e limitazioni nel prompt del modello, è stato anche inserito un controllo preliminare che verifica se nella stringa di input dell'utente siano presenti istruzioni e pattern malevoli, in tal caso prosegue con una storia di default.
- **Ripetizione del ciclo fino al successo:** il ciclo di validazione può ripetersi più volte, alternando l'intervento automatico dell'agente e quello umano, fino al raggiungimento di un piano corretto e accettabile. Nel codice è stato inserito un numero massimo di tentativi grazie ai quali il modello cerca di modificare e risolvere gli errori avvenuti in fase di validazione, questo garantisce di scegliere il numero di tentativi massimi in base alle proprie esigenze ed eventualmente scegliere se implementare la correzione manualmente o creare una nuova storia in seguito al limite massimo di fallimenti. Inoltre inserire un limite evita di sfornare con le richieste messe a disposizione dal piano del modello usato.
- **Approvazione finale del piano:** al termine della validazione, il piano viene mostrato all'utente, che può approvarlo se soddisfatto oppure richiedere ulteriori modifiche. Queste possono essere espresse anche in **linguaggio naturale**, offrendo un'interfaccia ancora più accessibile per eventuali revisioni.
- **Riproduzione vocale integrata:** per rendere ancora più immersiva l'applicazione abbiamo deciso di inserire un player che permette all'utente di ascoltare passo dopo passo la narrazione. Fin dalla creazione della lore,

alle singole scelte che l'utente deve prendere, un narratore vocale cerca di intuire il tono da dare alla lettura e procede a narrare.

Fase 2

La seconda fase del progetto consiste nella trasformazione della quest validata in un'esperienza interattiva fruibile via web. Il cuore di questa fase è la generazione automatica di un file JSON strutturato come grafo narrativo, dove ogni nodo rappresenta uno stato della storia e contiene una descrizione immersiva e un insieme di scelte. Tali scelte derivano sia dai vincoli della lore (branching factor e depth constraints) sia dal piano ottimale ottenuto nella Fase 1.

La funzione centrale di generazione della storia è `generate_story(lm)`, che riceve come input il piano validato e il file di lore e produce l'intero albero delle decisioni. Essa garantisce che ogni nodo abbia esattamente una scelta corretta coerente col piano, mentre le scelte errate portano a rami di fallimento generati in modo controllato, secondo i vincoli del progetto.

L'interfaccia è sviluppata in **Streamlit**, e permette all'utente di esplorare la storia interattiva in modo dinamico. Il sistema mantiene lo stato narrativo tramite lo `session_state` di Streamlit, salvando il nodo attuale, la cronologia delle decisioni (`choices_made`) e l'avanzamento nel gioco. Le scelte vengono visualizzate come pulsanti e sono mescolate casualmente tramite la funzione `shuffle_choices`, per aumentare la rigiocabilità.

Sono state implementate diverse funzionalità aggiuntive:

- Visualizzazione dello stato narrativo corrente con stile immersivo (`story-text`).
- Indicatori grafici di progresso, con la funzione `display_progress_bar()`.
- Gestione dei finali: vittoria, fallimento o ramificazioni alternative, tramite controlli condizionali sui nodi.
- Possibilità di riavvio o nuova sessione, gestite dalle funzioni `reset_game()` e `make_choice()`.

L'intera esperienza è incapsulata in un'applicazione fluida, che integra sia la generazione narrativa (via LLM) che la navigazione interattiva in un ciclo creativo completo. L'utente può esplorare molteplici rami, prendere decisioni significative e vivere una narrazione coerente, strutturata e reattiva. Viene inoltre messa a disposizione dell'utente la possibilità, in ogni passo della fase 2, di salvare la storia ed i progressi fatti. Verrà quindi scaricato un file zip contenente tre file:

- uno in formato json
- uno markdown
- uno di testo

che permettono di avere una visione completa della storia, delle modifiche fatte e dei vari passi con le rispettive scelte.

Strumenti utilizzati

L’ambiente di esecuzione dei pianificatori è stato configurato tramite **WSL (Windows Subsystem for Linux)** con distribuzione **Ubuntu**, che ha permesso l’utilizzo di strumenti tipici dei sistemi Unix-like. In particolare, per la fase di pianificazione e validazione sono stati utilizzati:

- **Fast Downward**: un planner classico ampiamente utilizzato per problemi in linguaggio PDDL, disponibile all’indirizzo <https://www.fast-downward.org/>;
- **VAL (Plan Validator)**: uno strumento per la validazione dei piani generati, che verifica la correttezza rispetto ai file `domain.pddl` e `problem.pddl`. È reperibile su GitHub all’indirizzo <https://github.com/KCL-Planning/VAL>.

3 Difficoltà affrontate

Durante lo sviluppo del progetto sono emerse diverse difficoltà, sia di natura tecnica sia metodologica. Una delle principali sfide iniziali ha riguardato la definizione di prompt efficaci per la generazione della *lore*. Senza una struttura di esempio, il modello tendeva a generare contenuti troppo generici o non compatibili con i vincoli richiesti dalla pianificazione. L’introduzione di esempi concreti (Few-Shot Prompting) ha significativamente migliorato la qualità dell’output, ma ha richiesto un lavoro preliminare di progettazione e ottimizzazione.

Un secondo ostacolo ha riguardato la generazione automatica dei file PDDL. Anche quando la storia era coerente dal punto di vista narrativo, la traduzione in un modello di pianificazione formalmente valido si è rivelata complessa. Errori sintattici o semantici nei file generati causavano fallimenti nella validazione, richiedendo l’attivazione dell’agente riflessivo e, in alcuni casi, l’intervento manuale dell’utente.

Un ulteriore elemento critico è stato il processo di validazione iterativa. La combinazione tra Fast Downward e VAL, pur essendo efficace, ha introdotto una certa complessità nella gestione degli errori e nella comunicazione tra strumenti e modello generativo.

Infine, nella fase di sviluppo dell’interfaccia web, si è posta l’esigenza di mantenere la struttura dei nodi coerente con la logica narrativa originaria, evitando ripetizioni o percorsi incoerenti. La gestione dello stato interno dell’applicazione e l’aggiornamento dinamico dei nodi ha richiesto attenzione, in particolare per garantire la persistenza del contesto e un’esperienza utente fluida.

Nel complesso, queste difficoltà hanno rappresentato opportunità di miglioramento progressivo del sistema, contribuendo alla sua maggiore affidabilità e alla definizione di un flusso di lavoro più solido e adattabile.

4 Esempio di funzionamento del sistema Quest-Master

In questa sezione verrà presentato un esempio a scopo illustrativo. L'analisi prenderà in esempio il caso in cui entrambe le fasi siano utilizzate tramite GUI, procediamo avviando l'applicazione tramite interfaccia grafica.

4.1 Fase I: Forgiatura della Leggenda

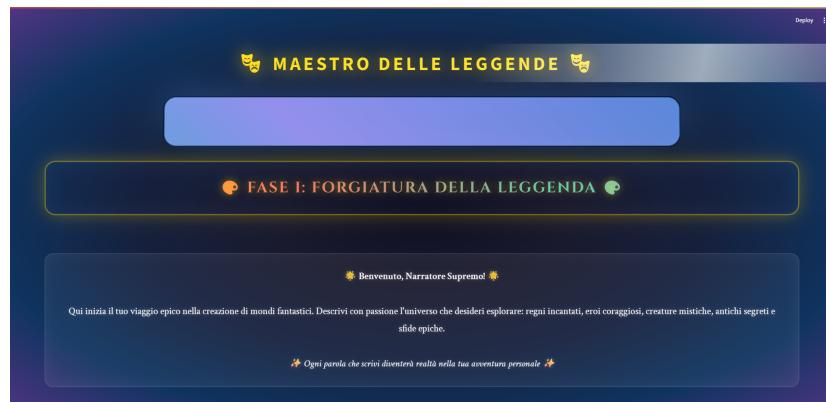


Figure 1: Schermata iniziale: introduzione alla Fase I

All'avvio, l'utente viene accolto con un'interfaccia accattivante e tematicamente coerente con l'universo fantasy. L'obiettivo è immerge il narratore in un processo creativo ludico e coinvolgente, presentando subito il concetto di “forgiatura della leggenda”.

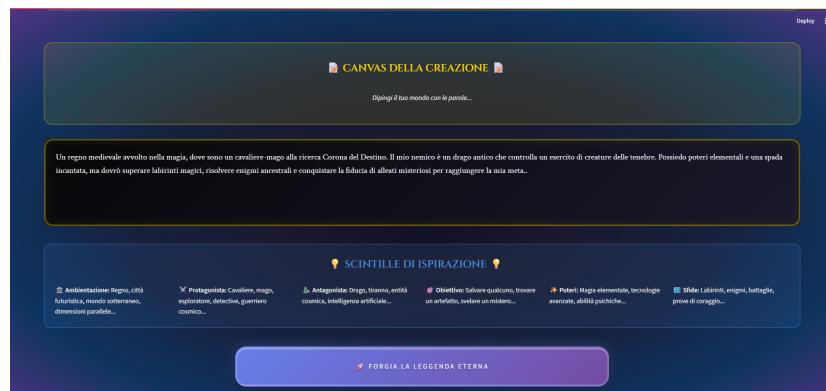


Figure 2: Canvas della Creazione e Scintille di Ispirazione

L'utente può descrivere il proprio mondo narrativo utilizzando un editor testuale, supportato da "scintille di ispirazione", ovvero prompt visivi e testuali che stimolano l'immaginazione per definire ambientazione, protagonisti, nemici, poteri, obiettivi e sfide.

4.2 Fase II: La Leggenda Prende Forma



Figure 3: Fase II: transizione alla generazione automatica della leggenda

Una volta completata la descrizione, il sistema genera una leggenda coerente e strutturata. Viene fornita anche una narrazione vocale in stile "narratore epico", generata automaticamente.

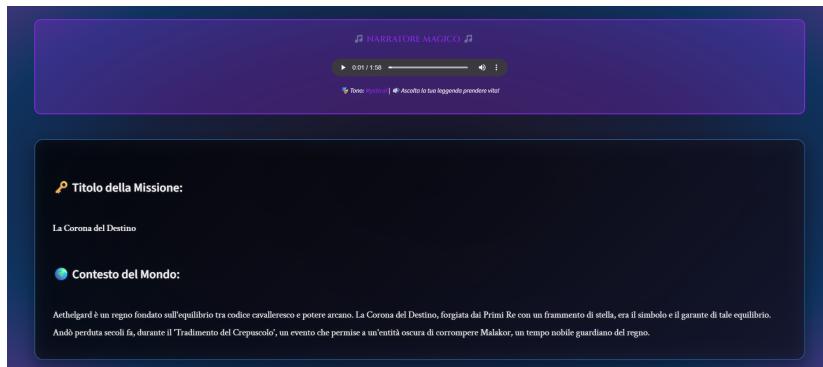


Figure 4: Esempio di missione: titolo e contesto del mondo

La leggenda si articola in diverse sezioni: titolo della missione, contesto del mondo, e descrizione narrativa coerente. In questo caso, la "Corona del Destino" è al centro di una trama fantasy classica con influssi magici e epici.

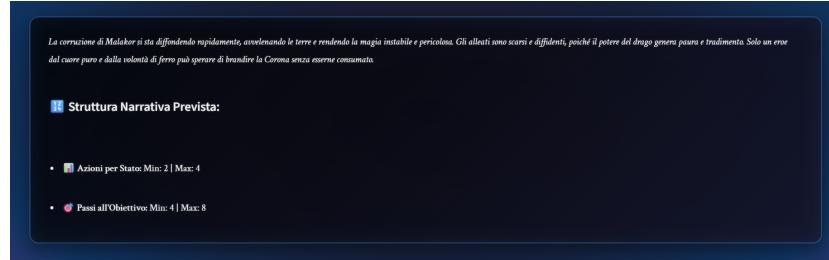


Figure 5: Struttura narrativa: parametri minimi e massimi

Vengono anche specificati i vincoli narrativi, in linea con le richieste PDDL: azioni per stato e passi verso l'obiettivo. Questi dati sono poi codificati nel modello logico sottostante.

4.3 Modifica Guidata della Leggenda



Figure 6: Fase di revisione: il giudizio del creatore

Il sistema chiede all'utente se la leggenda rispecchia la visione originale, offrendo la possibilità di modificarla.

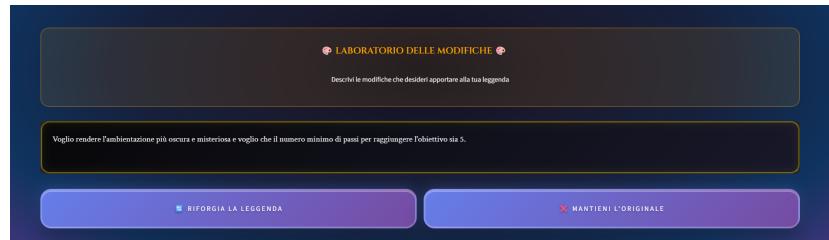


Figure 7: Richiesta di modifica: ambientazione più oscura e 5 passi minimi

L'utente può specificare cambiamenti desiderati, come un tono narrativo più cupo o modifiche strutturali (es. minimo di passi aumentato).

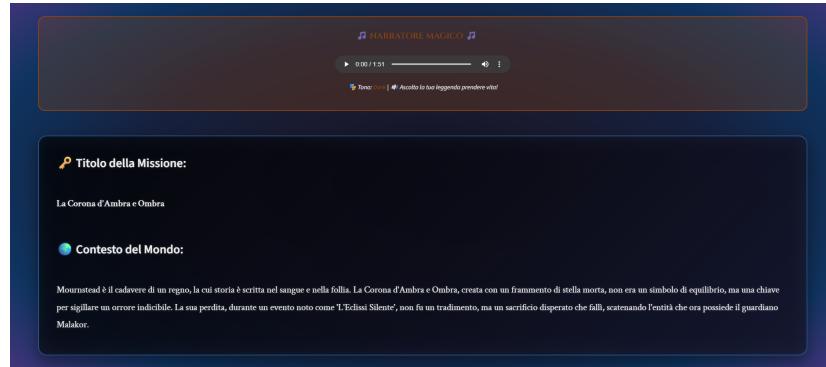


Figure 8: Nuova leggenda generata dopo le modifiche

Il sistema aggiorna dinamicamente il contenuto narrativo, adattandolo alle richieste. In questo esempio, la leggenda ha acquisito un tono dark, coerente con l'ambientazione di "Mournstead".

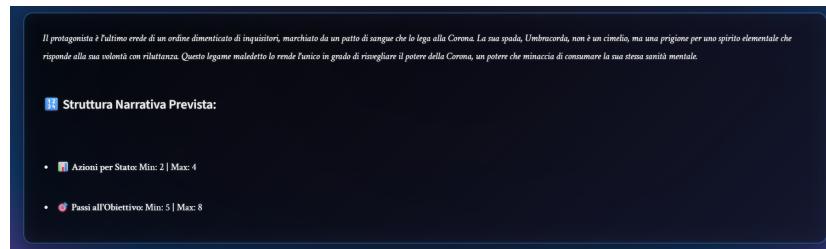


Figure 9: Narrazione aggiornata del protagonista e oggetto magico

Anche la caratterizzazione del protagonista viene rielaborata per riflettere la nuova atmosfera: maledizioni, ordini inquisitori e poteri oscuri.

4.4 Fase III: Forgiatura delle Leggi Universali



Figure 10: Transizione alla fase III: costruzione delle leggi logiche

La fase finale guida l'utente verso la costruzione delle regole logiche che definiranno il comportamento dell'avventura sotto forma di PDDL, garantendo coerenza narrativa e validità computazionale per la pianificazione classica.

4.5 Fase IV: Alchimia Computazionale e Motore Cosmico

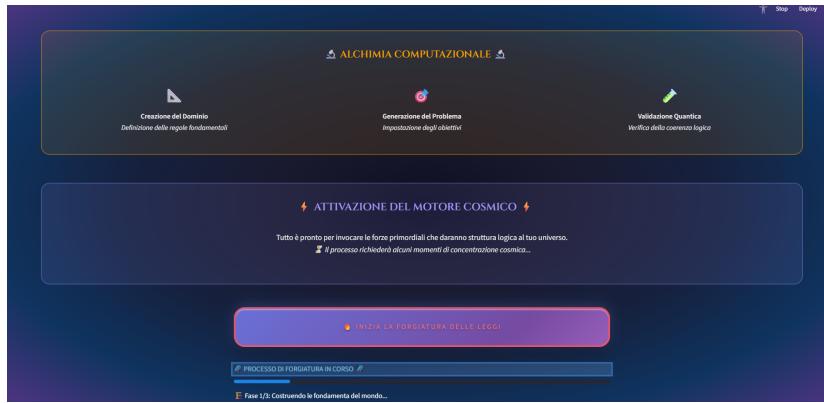


Figure 11: Fase di Alchimia Computazionale: generazione e validazione del modello PDDL

Durante questa fase, il sistema costruisce automaticamente i file PDDL relativi al dominio e al problema. Una volta generati, viene avviata la validazione logica tramite planner classico. La metafora narrativa dell'“attivazione del motore cosmico” rende il processo tecnico più immersivo.

4.6 Codifica delle Gesta Eroiche

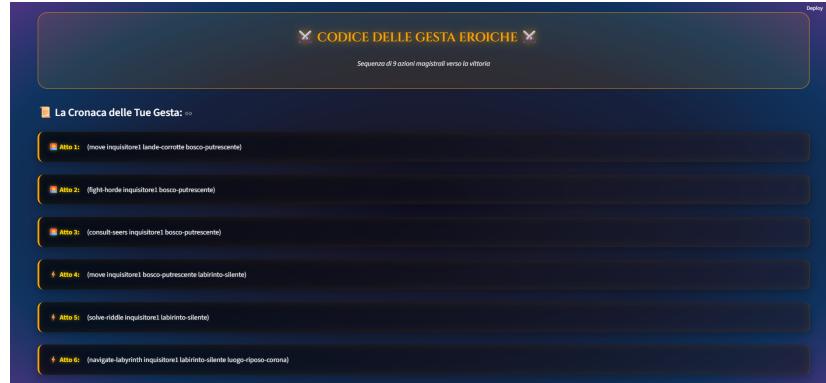


Figure 12: Output della pianificazione: azioni narrative codificate in atti

Ogni azione risultante dalla pianificazione viene trasformata in un “atto” narrativo. L’utente può visualizzare l’intera sequenza delle azioni che l’eroe dovrà compiere per completare la missione.



Figure 13: Ultimi atti della saga e analisi dell’epopea

Al termine della pianificazione, il sistema analizza la complessità della storia e presenta una sintesi: numero di decisioni, difficoltà stimata e probabilità di successo.

4.7 Fase V: Nascita dell'Avventura Interattiva



Figure 14: Transizione finale: dall'epopea scritta all'esperienza interattiva

Con la Fase V, la leggenda viene trasformata in un gioco interattivo. L'utente non è più autore ma protagonista della propria saga.

4.8 Interfaccia Interattiva: Scelte e Progressione

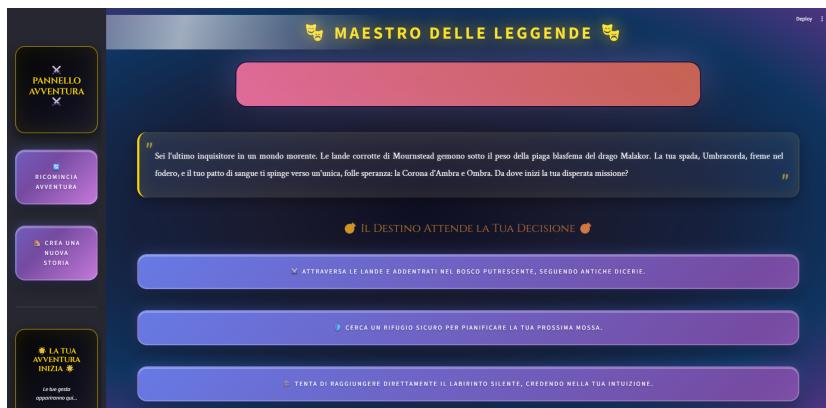


Figure 15: Scelte interattive nel gioco: diramazioni e decisioni narrative

Le scelte fatte dall'utente determinano il percorso narrativo, con conseguenze dirette e feedback immediati. Ogni decisione ha peso, rendendo l'esperienza altamente immersiva.

4.9 Esito dell'Avventura: Fallimento



Figure 16: Un possibile esito fallimentare dell'avventura interattiva

Il sistema è in grado di riconoscere e comunicare fallimenti narrativi coerenti. In questo caso, la leggenda si interrompe dopo quattro decisioni, ma l'utente è invitato a riprovare con nuove strategie.

4.10 Esito dell'Avventura: Gloria Eterna

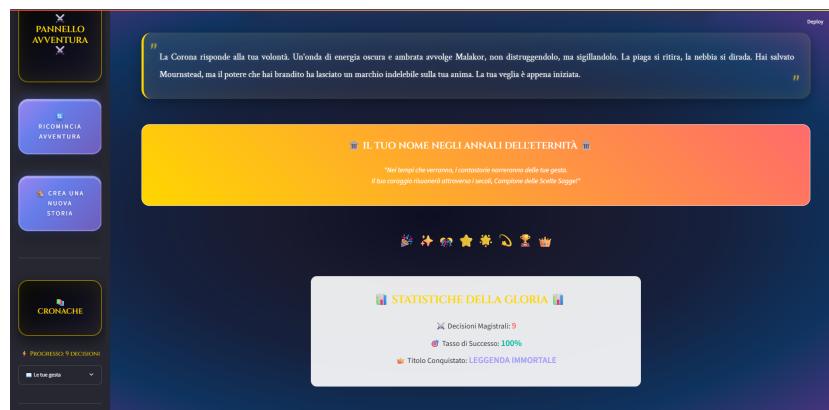


Figure 17: Epilogo vittorioso: gloria e incoronazione dell'utente come leggenda immortale

In alternativa, il successo dell'utente viene celebrato con una schermata finale con statistiche di gloria e titolo conquistato. Questo rafforza il senso di progressione e di merito.