QuestMaster AI - Relazione di Progetto

Umberto Domenico Ciccia, Salvatore Chiricosta, Giovanni Ferraro

Corso: Intelligenza Artificiale
Anno Accademico: 2024/2025

Indice

1 Introduzione

QuestMaster AI è un sistema innovativo che combina l'intelligenza artificiale generativa con tecniche di pianificazione classica per creare esperienze narrative interattive. Il progetto rappresenta un'implementazione avanzata di un sistema multi-agente che utilizza PDDL (Planning Domain Definition Language) e Large Language Models (LLM) per generare automaticamente quest narrative logicamente consistenti e coinvolgenti.

Il sistema è stato progettato come un'applicazione a due fasi che assiste gli autori nella creazione di avventure interattive attraverso un processo automatizzato ma controllabile, garantendo sia la creatività narrativa che la coerenza logica dell'esperienza di gioco.

2 Obiettivi del Progetto

2.1 Obiettivi Primari

- **Generazione Automatica di Narrative**: Creare storie coinvolgenti e coerenti utilizzando AI generativa.
- **Validazione Logica**: Garantire che le quest generate siano logicamente risolvibili attraverso PDDL.
- **Interfaccia Interattiva**: Fornire un'esperienza utente intuitiva e immersiva.
- **Architettura Modulare**: Implementare un sistema facilmente estensibile e manutenibile.

2.2 Obiettivi Secondari

- **Containerizzazione**: Deployment semplificato attraverso Docker.
- **CLI Completa**: Strumenti da riga di comando per sviluppatori.
- **Logging Avanzato**: Sistema di monitoraggio e debugging completo.
- **Validazione Continua**: Ciclo di raffinamento automatico per migliorare la qualità.

3 Architettura del Sistema

3.1 Panoramica Architetturale

QuestMaster AI segue un'architettura multi-layered basata su agenti specializzati:

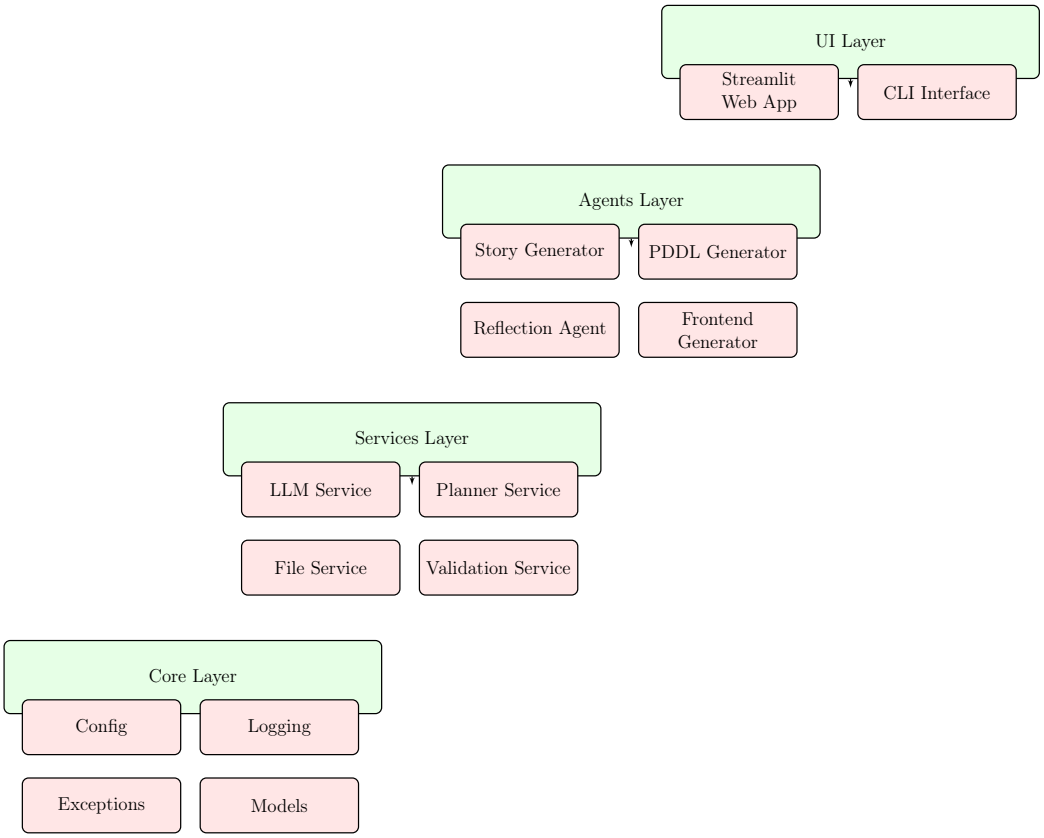

Figura 1: Architettura del Sistema QuestMaster AI

3.2 Componenti Principali

3.2.1 Core Components

- **Configuration Management**: Gestione centralizzata delle configurazioni.
- **Logging System**: Sistema di logging strutturato con Rich e Structlog.
- **Exception Handling**: Gestione degli errori specializzata per dominio.

- **Data Models**: Modelli Pydantic per validazione e serializzazione.

3.2.2 Services Layer

- **LLM Service**: Interfaccia con OpenAI GPT per generazione di contenuti.
- **Planner Service**: Integrazione con Fast Downward planner.
- **File Service**: Gestione I/O di file e persistenza.
- **Validation Service**: Validazione di PDDL e narrative.

4 Tecnologie Utilizzate

4.1 Linguaggi e Framework Core

- **Python 3.9+**: Linguaggio principale del progetto.
- **Pydantic 2.0+**: Validazione e serializzazione dati.
- **Streamlit 1.28+**: Framework per interfaccia web.
- **Click 8.0+**: Framework per CLI.

4.2 AI e Machine Learning

- **OpenAI API 1.0+**: Large Language Models (GPT-4o-mini).
- **Fast Downward**: Classical planner per PDDL.
- **PDDL**: Planning Domain Definition Language.

4.3 Containerizzazione e Deployment

- **Docker**: Containerizzazione dell'applicazione.
- **Docker Compose**: Orchestrazione multi-container.
- **Shell Scripting**: Automazione setup e deployment.

5 Agenti AI Implementati

5.1 Story Generator Agent

Responsabilità: Generazione narrativa principale.
Caratteristiche:

- Creazione di lore dettagliato con personaggi, ambientazioni e trama.
- Integrazione con template JSON per struttura consistente.
- Generazione di elementi narrativi ricchi e coinvolgenti.
- Adattamento a vincoli di branching factor e profondità.

5.2 PDDL Generator Agent

Responsabilità: Conversione narrative in modelli di pianificazione.
Caratteristiche:

- Traduzione di storie in domini e problemi PDDL.
- Generazione di azioni, predicati e condizioni goal.
- Commenti esplicativi per ogni elemento PDDL.
- Ottimizzazione per solvibilità.

Elementi PDDL Generati:

- **Domini**: Definizione azioni, predicati, tipi.
- **Problemi**: Stato iniziale, oggetti, goal.
- **Azioni**: `move`, `collect_item`, `overcome_obstacle`, `complete_quest`.

5.3 Reflection Agent

Responsabilità: Validazione e raffinamento iterativo.
Caratteristiche:

- Analisi di inconsistenze logiche in PDDL.
- Suggerimenti di miglioramento automatici.
- Ciclo di raffinamento iterativo.

- Interazione con utente per approvazione modifiche.

Processo di Riflessione:

1. Analisi del PDDL generato.
2. Identificazione problemi di solvibilità.
3. Generazione suggerimenti specifici.
4. Implementazione modifiche approvate.

5.4 Frontend Generator Agent

Responsabilità: Creazione interfacce interattive.
Caratteristiche:

- Generazione di interfacce Streamlit dinamiche.
- Adattamento UI ai requisiti della storia.
- Integrazione con sistema di navigazione.
- Componenti interattivi personalizzati.

6 Flusso di Funzionamento

6.1 Phase 1: Story Generation e PDDL Validation

Dettaglio del Processo:

1. **Input Processing**: Lettura del file lore iniziale.
2. **Story Generation**: StoryGeneratorAgent crea narrative dettagliate.
3. **PDDL Translation**: PDDLGeneratorAgent converte in formato pianificazione.
4. **Validation**: Fast Downward verifica solvibilità.
5. **Refinement Loop**: ReflectionAgent migliora iterativamente se necessario.
6. **Output**: PDDL validato e storia finalizzata.

6.2 Phase 2: Interactive Story Game

Componenti Generati:

- Interfaccia web Streamlit personalizzata.
- Sistema di navigazione basato su stati.
- Gestione delle scelte utente.
- Visualizzazione progressiva della storia.

6.3 Ciclo di Raffinamento

Il sistema implementa un ciclo di raffinamento automatico per garantire qualità:

1. **Detection**: Identificazione automatica di problemi.
2. **Analysis**: Analisi approfondita delle cause.
3. **Suggestion**: Generazione di soluzioni specifiche.
4. **Validation**: Test delle modifiche proposte.
5. **Integration**: Applicazione delle modifiche approvate.

7 Installazione e Utilizzo

7.1 Prerequisiti

- **Python 3.9+**
- **Docker** (opzionale ma raccomandato)
- **OpenAI API Key**
- **Sistema Unix-like** (macOS/Linux)

7.2 Installazione con Docker (Raccomandato)

```
# Clone del repository
git clone https://github.com/umbertocicciaa/QuestMasterAI.git
cd QuestMasterAI

# Setup automatico con Docker
./docker-setup.sh

# Oppure manualmente
docker-compose up --build
```

Listing 1: Installazione con Docker

7.3 Installazione Locale

```
# Setup ambiente virtuale
python -m venv venv
source venv/bin/activate  # su Windows: venv\Scripts\activate

# Installazione dipendenze
pip install -e .

# Setup Fast Downward
./start.sh

# Configurazione API Key
export OPENAI_API_KEY="your-api-key-here"
```

Listing 2: Installazione Locale

7.4 Utilizzo CLI

7.4.1 Esecuzione Completa

```
python -m questmaster.cli run
```

Listing 3: Esecuzione Completa

7.4.2 Esecuzione per Fasi

```
# Phase 1: Story Generation
python -m questmaster.cli phase1 --lore-path data/lore.
    json

# Phase 2: Interactive Frontend
python -m questmaster.cli phase2
```

Listing 4: Esecuzione per Fasi

7.4.3 Opzioni Avanzate

```
# Debug mode
python -m questmaster.cli --debug --log-level DEBUG
    phase1

# Custom API key
python -m questmaster.cli --api-key "sk-custom-key" run

# Help
python -m questmaster.cli --help
```

Listing 5: Opzioni Avanzate CLI

7.5 Configurazione Ambiente

7.5.1 Variabili Ambiente

```
export OPENAI_API_KEY="sk-your-openai-key"
export CHATGPT_MODEL="gpt-4o-mini-2024-07-18"
export LOG_LEVEL="INFO"
export DEBUG="False"
export FAST_DOWNWARD_TIMEOUT="300"
```

Listing 6: Variabili Ambiente

8 Esempi Pratici

8.1 PDDL Generato

8.1.1 Domain File

```
(define (domain legacy-quest)
  (:requirements :strips :typing)
  (:types character location item obstacle)

  (:predicates
    (at ?c - character ?loc - location) ; Character is at location
    (has_item ?c - character ?item - item) ; Character has item
    (quest_completed) ; Quest completion state
  )

  (:action move
    :parameters (?c - character ?from - location ?to - location)
    :precondition (at ?c ?from)
    :effect (and (not (at ?c ?from)) (at ?c ?to))
  )
)
```

Listing 7: PDDL Domain File

8.1.2 Problem File

```
(define (problem legacy-quest-scenario)
  (:domain legacy-quest)

  (:objects
    hero - character
    village forest cave - location
    sword shield quest_item - item
    dragon - obstacle
  )

  (:init
    (at hero village)
  )

  (:goal (quest_completed))
)
```

Listing 8: PDDL Problem File

8.2 Output del Sistema

8.2.1 Phase 1 Output

```
Starting Phase 1: Story Generation
Phase 1 completed successfully!
Execution time: 45.23s
Plan length: 8 steps
```

Listing 9: Output Fase 1

8.3 Interfaccia Web Generata

Il sistema genera automaticamente un'interfaccia Streamlit che include:

- **Navigation Panel**: Controlli per muoversi nella storia.
- **Story Display**: Visualizzazione testuale dell'avventura.
- **Action Buttons**: Bottoni per le scelte disponibili.
- **State Tracking**: Monitoraggio del progresso.
- **Rich Formatting**: Stile e formattazione accattivanti.

9 Conclusioni e Sviluppi Futuri

9.1 Risultati Raggiunti

Il progetto QuestMasterAI ha raggiunto con successo gli obiettivi prefissati:

9.1.1 Obiettivi Tecnici

- ✓ **Architettura Multi-Agente**: Sistema modulare e estensibile.
- ✓ **Integrazione PDDL**: Validazione logica attraverso classical planning.
- ✓ **Pipeline Automatizzata**: Processo end-to-end completamente automatico.
- ✓ **Interfaccia Moderna**: UI web intuitiva e responsive.
- ✓ **Containerizzazione**: Deployment semplificato con Docker.

9.1.2 Obiettivi di Qualità

✓ **Robustezza**: Gestione errori e recovery automatico.

✓ **Scalabilità**: Architettura predisposta per estensioni.

✓ **Manutenibilità**: Codice ben strutturato e documentato.

✓ **Testing**: Suite di test automatizzati.

✓ **Logging**: Sistema di monitoraggio completo.

9.2 Innovazioni Introdotte

9.2.1 Integrazione AI Generativa + Classical Planning

L'aspetto più innovativo del progetto è l'integrazione seamless tra:

- **LLM per creatività**: Generazione di narrative coinvolgenti.
- **PDDL per logica**: Garanzia di coerenza e solvibilità.
- **Ciclo di raffinamento**: Miglioramento iterativo automatico.

9.2.2 Sistema Multi-Agente Specializzato

Ogni agente ha responsabilità specifiche:

- **Separazione delle responsabilità**: Maggiore manutenibilità.
- **Specializzazione**: Ottimizzazione per task specifici.
- **Coordinazione**: Collaborazione efficace tra agenti.

9.2.3 Validation-Driven Development

- **Feedback immediato**: Validazione continua della qualità.
- **Auto-correzione**: Capacità di auto-miglioramento.
- **Garanzie formali**: Uso di classical planning per verifica.

9.3 Limitazioni Attuali

9.3.1 Limitazioni Tecniche

- **Dipendenza da API esterne**: Richiede connessione internet e API key.
- **Tempo di elaborazione**: Generazione può richiedere diversi minuti.
- **Complessità PDDL**: Limitato a domini relativamente semplici.
- **Lingua**: Attualmente ottimizzato principalmente per italiano.

9.3.2 Limitazioni Funzionali

- **Personalizzazione limitata**: Template predefiniti per la struttura.
- **Interazione utente**: Interfaccia principalmente read-only in Phase 2.
- **Multimedia**: Supporto limitato per immagini e audio.
- **Persistenza**: Stato di gioco non persiste tra sessioni.

10 Bibliografia e Riferimenti

- **OpenAI API Documentation**: Utilizzo di Large Language Models.
- **PDDL Documentation**: Planning Domain Definition Language.
- **Fast Downward**: Classical Planning System.
- **Streamlit Documentation**: Framework per interfacce web.
- **Pydantic**: Data validation e settings management.

11 Repository e Risorse

- **GitHub Repository**: QuestMasterAI.
- **Documentation**: Documentazione completa nel README.md.
- **Issues**: Sistema di tracking per bug e feature requests.
- **Examples**: Esempi e template nella cartella `resources/`.