Software Architecture decision-making card game con assistente AI

`Github project: https://github.com/LorenzoTucceri/DecidArchV2`

Contents

1 Introduzione

Il progetto **DecidArch** è un gioco di simulazione in cui i partecipanti interpretano il ruolo di architetti software e devono collaborare per progettare un sistema software che soddisfi i requisiti degli stakeholder. In questo progetto, un assistente basato su **AI (Intelligenza Artificiale)** guida il gruppo nelle decisioni di design, fornendo suggerimenti basati sulle preoccupazioni (concern) degli stakeholder e sugli eventi in corso nel progetto. Questo approccio permette ai giocatori di imparare in modo autonomo, senza bisogno di un facilitatore umano (come un professore).

1.1 Regole del gioco

- **Obiettivo**: I giocatori assumono il ruolo di architetti software e devono prendere decisioni progettuali che soddisfino le priorità dei vari stakeholder, tenendo conto degli eventi di progetto e delle preoccupazioni emerse.
- **Preparazione**:
  - Si sceglie il numero di giocatori (da 2 a 4).
  - Si definisce un progetto, gli stakeholder e i loro attributi di qualità prioritari (ad esempio, sicurezza, prestazioni, usabilità).
  - Si creano le carte *Preoccupazione* (concern) e *Evento* con problemi e sfide che i giocatori dovranno risolvere durante il gioco.
- **Turni di gioco**:
  - Ogni giocatore affronta una carta *Preoccupazione*, che rappresenta un problema progettuale.
  - Il *DecidArchAssistant* suggerisce l'opzione progettuale migliore, basata sullo stato attuale del progetto, le priorità degli stakeholder e gli eventi in corso.
  - Il giocatore può accettare il suggerimento o prendere una decisione diversa.
- **Punteggio**:
  - Le decisioni progettuali impattano sugli attributi di qualità prioritari degli stakeholder.
  - Alla fine del gioco, viene calcolata una *soddisfazione finale* basata su quanto le decisioni abbiano rispettato le priorità.

  - Se un punteggio di qualità scende sotto zero, il gioco è perso.

- **Durata**: Il gioco dura 30 minuti o finché non vengono affrontate tutte le carte *Preoccupazione*.

Lo scopo è mantenere soddisfatti gli stakeholder e completare il progetto senza compromettere troppo le qualità richieste.

1.2 Problema affrontato

Il problema principale affrontato nel gioco **DecidArch** è la difficoltà nel prendere decisioni di design bilanciate, che soddisfino le priorità di qualità degli stakeholder e si adattino agli eventi imprevisti. Il processo di decisione richiede di bilanciare vari attributi di qualità (come sicurezza, usabilità, manutenibilità) e gestire eventuali compromessi. Nel contesto di insegnamento, questo può diventare complesso senza un supporto continuo. Pertanto, è stato introdotto un assistente AI per fornire feedback in tempo reale e aiutare il gruppo a esplorare diverse soluzioni.

2 Struttura del Codice e Soluzione

Il codice di **DecidArch** è organizzato in modo da simulare il gioco e utilizzare un modello di linguaggio (**LLM**) per fornire suggerimenti nelle scelte di design. Di seguito è spiegato il funzionamento delle varie componenti del codice.

2.1 Classi Principali

Listing 1: Definizione delle classi principali per il gioco

```
class Player:
    def __init__(self, first_name, last_name):
        self.first_name = first_name
        self.last_name = last_name

class ProjectCard:
    def __init__(self, name, purpose):
        self.name = name
        self.purpose = purpose

class StakeholderCard:
    def __init__(self, role, goal, quality_attributes):
```

```
        self.role = role
        self.goal = goal
        self.quality_attributes = quality_attributes

class ConcernCard:
    def __init__(self, card_id, concern,
       design_decisions):
        self.card_id = card_id
        self.concern = concern
        self.design_decisions = design_decisions

class EventCard:
    def __init__(self, title, description, consequence):
        self.title = title
        self.description = description
        self.consequence = consequence
```

Descrizione del codice:

- `Player`: La classe `Player` rappresenta un giocatore all'interno del gioco. Ogni giocatore è identificato da un *nome* e un *cognome*. Durante il gioco, i giocatori assumono il ruolo di architetti e prendono decisioni progettuali per soddisfare le esigenze del progetto e degli stakeholder. Ogni giocatore partecipa attivamente nei turni, prendendo decisioni critiche in base alle carte `ConcernCard` e agli eventi che si presentano.

- `ProjectCard`: La `ProjectCard` definisce il progetto che i giocatori devono sviluppare. Essa contiene informazioni chiave sul *nome del progetto* e il suo *scopo*. Il progetto è il contesto principale attorno al quale ruotano tutte le decisioni progettuali. Le scelte fatte dai giocatori devono sempre tenere conto degli obiettivi globali del progetto, come definiti dalla carta `ProjectCard`.

- `StakeholderCard`: La `StakeholderCard` descrive un singolo stakeholder coinvolto nel progetto. Ogni stakeholder ha un *ruolo* specifico all'interno del progetto e un insieme di *obiettivi*. La parte più importante della carta riguarda gli *attributi di qualità*, ovvero le proprietà del sistema che l'architettura deve soddisfare, come prestazioni, sicurezza, manutenibilità, ecc. Ogni attributo è associato a una *priorità* che influenza il livello di attenzione che i giocatori devono dedicare a tale attributo durante il processo decisionale.

- `ConcernCard`: La `ConcernCard` rappresenta una preoccupazione o un problema specifico che il progetto deve affrontare. Ogni carta presenta

una descrizione della preoccupazione e include una serie di *opzioni di design* che i giocatori possono scegliere per risolvere il problema. Le opzioni di design indicano il loro *impatto* sugli attributi di qualità degli stakeholder, con punteggi che possono essere positivi o negativi. I giocatori devono valutare con attenzione queste opzioni per trovare la soluzione che meglio soddisfa gli stakeholder.

- `EventCard`: La `EventCard` descrive un evento imprevisto che può influenzare lo sviluppo del progetto. Gli eventi possono essere sia positivi che negativi, e solitamente richiedono un adattamento nelle decisioni di design. Ogni carta include un *titolo*, una *descrizione dettagliata* dell'evento e le sue *conseguenze* sulle decisioni progettuali già prese o future. Gli eventi rappresentano una sfida aggiuntiva che può complicare la risoluzione dei problemi già presenti, rendendo più difficile bilanciare le esigenze del progetto.

Queste classi costituiscono la base del gioco e sono dinamicamente utilizzate per costruire l'ambiente di simulazione.

2.2 DecidArchAssistant: L'Assistente AI

L'elemento centrale del gioco è l'uso di un **modello AI** (Llama2), che funge da assistente virtuale. Questo assistente guida il gruppo fornendo suggerimenti per affrontare le carte *Concern* basate sugli attributi di qualità degli stakeholder e sugli eventi in corso.

Listing 2: Definizione dell'Assistente AI con il Prompt Dinamico

```
from langchain.prompts import ChatPromptTemplate,
   HumanMessagePromptTemplate
from langchain import chains
from langchain.llms import Ollama

class DecidArchAssistant:
    """
    Virtual assistant specialized in assisting with
       architectural design decisions
    based on stakeholder concerns, project state, and
       ongoing events.
    """

    def __init__(self, configuration, system_template=
       None):
```

```
        """
        Initializes the DecidArchAssistant object with
           the provided configuration
        and an optional system template.
        Args:
            configuration (object): Configuration
               containing settings for the assistant,
                                    including URI and
                                       model name for
                                       the LLM.
            system_template (str, optional): Custom
               template for the system message.
                                             Defaults to
                                                 a
                                                predefined
                                                 message
                                                 that
                                             describes
                                                the
                                                assistant
                                                's role.
        """
        self.template = system_template or (
            "You operate as {self.assistant_name}, a
               virtual assistant specialized in "
            "assisting with design decisions based on
               concerns provided by stakeholders."
        )
        self.configuration = configuration

        # initializing an llm using the Ollama API with
           the provided configuration.
        self.llm = Ollama(
            base_url=self.configuration.uri_ollama,
            model=self.configuration.model_name,
            system=self.template,
        )

    def create_chain(self, template):
        """
        Creates a language model chain using the
           provided template and the initialized LLM.
        Args:
```

```
            template (str): The prompt template to be
                used for generating responses.
        Returns:
            chains.LLMChain: A chain object that links
                the prompt template with the language
                model.
        """
        prompt_template = ChatPromptTemplate(
            messages=[HumanMessagePromptTemplate.
                from_template(template)]
        )
        chain = chains.LLMChain(
            prompt=prompt_template,
            llm=self.llm,
            verbose=1,
        )
        return chain

    def extract_suggestion(self):
        """
        Creates a chain with a predefined prompt for
            suggesting the best design option based
        on stakeholder concerns, project state, and
            ongoing events.
        """
        template = """
        You are an AI assistant helping a team of
            software architects playing the game
        DecidArch. Your task is to suggest the best
            design option for a given concern card,
        taking into account the stakeholders' quality
            attribute priorities and any ongoing
        project events. Here's the information you'll
            need:

        Project Description:
        - {project_description}

        Stakeholders and their Quality Attribute
            Priorities:
        {stakeholders_info}

        Current Game State:
```

```
        - Current Design Decisions: {
            current_design_decisions}
        - Current QA-Scores: {current_qa_scores}
        - Ongoing Events: {ongoing_events}

        Concern Card Description:
        - {concern_card_description}

        Possible Design Options:
        - {design_options}

        Based on this information, suggest the best
            design option that satisfies the stakeholders
            '
        priorities and considers any ongoing events.
            Provide a rationale for your suggestion.
        """
        return self.create_chain(template)

    def extract_suggestion_chain(self,
        project_description, stakeholders,
                                 current_design_decisions
                                     , current_qa_scores,
                                 ongoing_events,
                                     concern_card_description
                                     , design_options):
        """
        Generates a design suggestion based on project
            description, stakeholder information,
        design decisions, QA scores, ongoing events, and
            concern card.
        Args:
            project_description (str): A description of
                the project.
            stakeholders (list): A list of stakeholders,
                where each stakeholder contains a role
                                 and quality attributes.
            current_design_decisions (str): A
                description of the current design
                decisions in place.
            current_qa_scores (str): The current quality
                attribute scores.
```

```
            ongoing_events (str): A list of ongoing
                events in the project.
            concern_card_description (str): The
                description of the concern card.
            design_options (str): A list of possible
                design options to choose from.
        Returns:
            str: The assistant's suggestion for the best
                design option, including a rationale.
        """
        stakeholders_info = "\n".join([f"- {stakeholder.
            role}: {stakeholder.quality_attributes}"
                                       for stakeholder
                                           in
                                           stakeholders])
        suggestion = self.extract_suggestion().run(
            project_description=project_description,
            stakeholders_info=stakeholders_info,
            current_design_decisions=
                current_design_decisions,
            current_qa_scores=current_qa_scores,
            ongoing_events=ongoing_events,
            concern_card_description=
                concern_card_description,
            design_options=design_options
        )
        return suggestion

    def extract_review_suggestion(self):
        """
        Creates a chain with a predefined prompt for
            reviewing and suggesting changes
        to previous design decisions based on an event.
        """
        template = """
        You are an AI assistant helping a team of
            software architects playing the game
        DecidArch. An unexpected event has occurred, and
            it might require changes to
        the previous design decisions. Your task is to
            suggest which past design decision(s)
        should be revised based on the new event. Here's
            the information you'll need:
```

```
        Project Description:
        - {project_description}

        Stakeholders and their Quality Attribute
           Priorities:
        {stakeholders_info}

        Current Game State:
        - Current Design Decisions: {
           current_design_decisions}
        - Current QA-Scores: {current_qa_scores}

        Ongoing Events:
        {ongoing_events}

        Based on this event, review the previous
           decisions and suggest necessary revisions.
        Provide a rationale for your suggestion.
        """
        return self.create_chain(template)

    def extract_review_suggestion_chain(self,
       project_description, stakeholders,
                                        current_design_decisions
                                           ,
                                           current_qa_scores
                                           ,
                                        ongoing_events):
        """
        Generates a suggestion for revising past design
           decisions based on a new event.
        Args:
            project_description (str): A description of
               the project.
            stakeholders (list): A list of stakeholders,
                where each stakeholder contains a role
                                 and quality attributes.
            current_design_decisions (str): A
               description of the current design
               decisions in place.
            current_qa_scores (str): The current quality
                attribute scores.
```

```
        ongoing_events (str): A list of ongoing
            events in the project.
    Returns:
        str: The assistant's suggestion for revising
             the design decision, including a
            rationale.
    """
    stakeholders_info = "\n".join([f"- {stakeholder.
        role}: {stakeholder.quality_attributes}"
                                   for stakeholder
                                       in
                                       stakeholders])
    suggestion = self.extract_review_suggestion().
        run(
        project_description=project_description,
        stakeholders_info=stakeholders_info,
        current_design_decisions=
            current_design_decisions,
        current_qa_scores=current_qa_scores,
        ongoing_events=ongoing_events
    )
    return suggestion
```

Descrizione del codice:

- `DecidArchAssistant`: Questa classe implementa un assistente virtuale basato su `Llama2` tramite l'API `Ollama`. L'assistente fornisce suggerimenti per decisioni di design architetturale, utilizzando informazioni dinamiche dal gioco come dettagli del progetto, priorità degli stakeholder, decisioni attuali e eventi in corso.
- `extract_suggestion`: Crea una catena di elaborazione (`LLMChain`) utilizzando un template di prompt predefinito. Questo template include:
  - **Descrizione del progetto**: Nome e scopo del progetto.
  - **Priorità degli stakeholder**: Preoccupazioni riguardanti gli attributi di qualità.
  - **Decisioni attuali**: Scelte di design finora effettuate.
  - **Punteggi QA**: Risultati correnti degli attributi di qualità.
  - **Eventi in corso**: Eventi che potrebbero influenzare le decisioni.

- **Concern Card**: Problema attuale e opzioni di design disponibili.

Il template fornisce al modello di linguaggio un contesto dettagliato per suggerire la miglior opzione di design con una spiegazione motivata.

- `extract_suggestion_chain`: Utilizza la catena creata da `extract_suggestion` per generare un suggerimento specifico. Accetta parametri che includono:
  - **Descrizione del progetto**: Sintesi del progetto.
  - **Stakeholders**: Lista degli stakeholder con ruoli e priorità.
  - **Decisioni attuali**: Scelte di design implementate.
  - **Punteggi QA**: Punteggi attuali degli attributi di qualità.
  - **Eventi in corso**: Eventi attuali.
  - **Concern Card**: Descrizione della preoccupazione e opzioni di design.

  Questo metodo prepara un prompt dettagliato e lo invia al modello di linguaggio per ottenere un suggerimento informato, con una raccomandazione e una spiegazione basata sulle informazioni fornite.

- `extract_review_suggestion`: Crea una catena di elaborazione (`LLMChain`) per gestire eventi imprevisti. Questo template di prompt invita l'AI a suggerire modifiche alle decisioni progettuali basate sugli eventi emersi.

- `extract_review_suggestion_chain`: Utilizza la catena creata da `extract_review_suggestion` per generare un suggerimento specifico. Accetta parametri che includono:
  - **Descrizione del progetto**: Sintesi del progetto.
  - **Stakeholders**: Lista degli stakeholder con ruoli e priorità.
  - **Decisioni attuali**: Scelte di design implementate.
  - **Punteggi QA**: Punteggi attuali degli attributi di qualità.
  - **Eventi in corso**: Descrizione dell'evento attuale che richiede una revisione.

  Questo metodo raccoglie tutte le informazioni necessarie e restituisce una raccomandazione per modificare le decisioni precedenti, in base agli eventi che si sono verificati.

2.3 Struttura del Gioco e Logica di Gioco

Il gioco simula una sessione collaborativa in cui i giocatori affrontano una "concern card" e prendono decisioni di design guidati dai suggerimenti dell'AI.

Listing 3: Struttura del Gioco e Implementazione

```
import time

import configuration
from decidarch_assistant import DecidArchAssistant
from models import Player, ProjectCard, StakeholderCard,
    ConcernCard, EventCard


class DecidArchGame:
    def __init__(self):
        self.configuration = configuration.Configuration
            ()
        self.assistant = DecidArchAssistant(self.
            configuration)
        self.players = []
        self.project_card = None
        self.stakeholder_cards = []
        self.concern_cards = []
        self.event_cards = []
        self.decision_template = []
        self.current_concern_index = 0
        self.start_time = None
        self.end_time = None

    def setup_game(self):
        # giocatori
        num_players = int(input(f"Enter number of
            players (max {self.configuration.MAX_PLAYERS
            }): "))
        if num_players > self.configuration.MAX_PLAYERS:
            print(f"Number of players cannot exceed {
                self.configuration.MAX_PLAYERS}. Setting
                to {self.configuration.MAX_PLAYERS}.")
            num_players = self.configuration.MAX_PLAYERS
        elif num_players < self.configuration.
            MIN_PLAYERS:
            print(f"Number of players cannot exceed {
                self.configuration.MIN_PLAYERS}.")
```

```
        num_players = self.configuration.MIN_PLAYERS

    for _ in range(num_players):
        first_name = input("Enter player's first
            name: ")
        last_name = input("Enter player's last name:
            ")
        self.players.append(Player(first_name,
            last_name))

    # progetto
    project_name = input("Enter project name: ")
    project_purpose = input("Enter project purpose:
        ")
    self.project_card = ProjectCard(project_name,
        project_purpose)

    # stakeholder
    num_stakeholders = int(input(f"Enter number of
        stakeholders (max {self.configuration.
        MAX_STAKEHOLDERS}): "))
    if num_stakeholders > self.configuration.
        MAX_STAKEHOLDERS:
        print(f"Number of stakeholders cannot exceed
            {self.configuration.MAX_STAKEHOLDERS}.
            Setting to {self.configuration.
            MAX_STAKEHOLDERS}.")
        num_stakeholders = self.configuration.
            MAX_STAKEHOLDERS
    for _ in range(num_stakeholders):
        role = input("Enter stakeholder role: ")
        goal = input("Enter stakeholder goal: ")
        num_attributes = int(input(f"Enter number of
            quality attributes for {role}: "))
        quality_attributes = {}
        for _ in range(num_attributes):
            attr = input("Enter quality attribute: "
                )
            priority = int(input(f"Enter priority
                for {attr} (1-5): "))
            quality_attributes[attr] = priority
        self.stakeholder_cards.append(
            StakeholderCard(role, goal,
```

```
        quality_attributes))

# concern cards
num_concerns = int(input(f"Enter number of
    concern cards (max {self.configuration.
    MAX_CONCERNS}): "))
if num_concerns > self.configuration.
    MAX_CONCERNS:
     print(f"Number of concern cards cannot
        exceed {self.configuration.MAX_CONCERNS}.
         Setting to {self.configuration.
        MAX_CONCERNS}.")
     num_concerns = self.configuration.
        MAX_CONCERNS
for i in range(num_concerns):
     concern = input(f"Enter concern for card {i
        +1}: ")
     num_decisions = int(input(f"Enter number of
        design decisions for concern {concern}: "
        ))
     design_decisions = {}
     for _ in range(num_decisions):
          attr = input("Enter design decision
             attribute: ")
          impact = int(input(f"Enter impact for {
             attr} (+/- value): "))
          design_decisions[attr] = impact
     self.concern_cards.append(ConcernCard(i+1,
        concern, design_decisions))

# event cards
num_events = int(input(f"Enter number of event
    cards (max {self.configuration.MAX_EVENTS}):
    "))
if num_events > self.configuration.MAX_EVENTS:
     print(f"Number of event cards cannot exceed
        {self.configuration.MAX_EVENTS}. Setting
        to {self.configuration.MAX_EVENTS}.")
     num_events = self.configuration.MAX_EVENTS
for _ in range(num_events):
     title = input("Enter event title: ")
     description = input("Enter event description
        : ")
```

```
            consequence = input("Enter event consequence
                : ")
            self.event_cards.append(EventCard(title,
                description, consequence))

    def calculate_score(self):
        qa_scores = {attr: 0 for stakeholder in self.
            stakeholder_cards for attr in stakeholder.
            quality_attributes.keys()}
        for decision in self.decision_template:
            for attr, impact in decision.items():
                qa_scores[attr] += impact

        if any(score < 0 for score in qa_scores.values()
            ):
            return -1  # Immediate loss

        stakeholder_satisfaction = []
        for stakeholder in self.stakeholder_cards:
            satisfaction = sum(max(0, qa_scores[attr] -
                priority) for attr, priority in
                stakeholder.quality_attributes.items())
            stakeholder_satisfaction.append(satisfaction
                )

        final_score = sum(stakeholder_satisfaction)
        return final_score

    def play_game(self):
        self.setup_game()
        self.start_time = time.time()
        self.end_time = self.start_time + 30 * 60  # 30
            minuti

        while time.time() < self.end_time and self.
            current_concern_index < len(self.
            concern_cards):
            for player in self.players:
                if time.time() >= self.end_time:
                    break

                concern_card = self.concern_cards[self.
                    current_concern_index]
```

```
print(f"{player.first_name} {player.
    last_name}'s turn:")
print(f"Concern: {concern_card.concern}"
    )

# stakeholders_info = "\n".join([f"- {
    stakeholder.role}: {stakeholder.
    quality_attributes}" for stakeholder
    in self.stakeholder_cards])
current_design_decisions = ", ".join([f"
    {k}: {v}" for decision in self.
    decision_template for k, v in
    decision.items()])
current_qa_scores = ", ".join([f"{k}: {v
    }" for k, v in self.
    calculate_qa_scores().items()])
ongoing_events = ", ".join([f"{event.
    title}: {event.description}" for
    event in self.event_cards])

suggestion = self.assistant.
    extract_info_trip(
     project_description=f"{self.
        project_card.name} - {self.
        project_card.purpose}",
     stakeholders=self.stakeholder_cards,
     current_design_decisions=
        current_design_decisions,
     current_qa_scores=current_qa_scores,
     ongoing_events=ongoing_events,
     concern_card_description=
        concern_card.concern,
     design_options=concern_card.
        design_decisions
)
print(f"Suggestion: {suggestion}")

# Simulazione della decisione presa e
    registrazione
self.decision_template.append(
    concern_card.design_decisions)
self.current_concern_index += 1
```

```
                if self.current_concern_index >= len(
                    self.concern_cards):
                    break

        # punteggio finale
        final_score = self.calculate_score()
        print(f"Final Score: {final_score}")

    def calculate_qa_scores(self):
        qa_scores = {attr: 0 for stakeholder in self.
            stakeholder_cards for attr in stakeholder.
            quality_attributes.keys()}
        for decision in self.decision_template:
            for attr, impact in decision.items():
                qa_scores[attr] += impact
        return qa_scores


if __name__ == "__main__":
    game = DecidArchGame()
    game.play_game()
```

Descrizione del codice:

- `DecidArchGame`: Classe principale che gestisce la logica del gioco. Il metodo `setup_game()` raccoglie informazioni su giocatori, carte progetto, stakeholder, concern card e eventi tramite input da console, configurando il gioco secondo le impostazioni dell'utente.
- `play_game`: Gestisce il ciclo del gioco. Ogni turno di gioco prevede l'affronto di una carta di preoccupazione. L'assistente AI, tramite `extract_suggestion_chain()`, fornisce suggerimenti basati sullo stato corrente del gioco e le decisioni precedenti, aiutando i giocatori a prendere decisioni informate.
- `calculate_score`: Calcola il punteggio finale del gioco. Il punteggio è determinato dall'impatto delle decisioni di design sugli attributi di qualità. La partita è persa se un attributo di qualità raggiunge un punteggio negativo; altrimenti, il punteggio finale è la somma della soddisfazione degli stakeholder, riflettendo quanto bene le decisioni hanno soddisfatto le loro priorità.

2.4 Descrizione del Problema Risolto

Il gioco affronta un problema comune nelle fasi iniziali di progettazione software: *come bilanciare le priorità degli stakeholder e prendere decisioni di design che non compromettano l'integrità del sistema.* Il gioco simula scenari reali in cui eventi imprevisti e priorità contrastanti richiedono compromessi tra attributi di qualità.

Attraverso l'uso di un assistente AI, il gruppo riceve feedback immediato sulle decisioni, guidandoli verso la miglior scelta possibile. L'AI permette di analizzare le opzioni di design senza il bisogno di un facilitatore umano, offrendo un'esperienza educativa più autonoma e immersiva.

3 Analisi del Prompt AI

Il prompt usato per il modello AI è cruciale per ottenere risposte accurate e utili. I punti chiave che permettono all'AI di operare efficacemente includono:

- *Contestualizzazione del problema*: Il prompt è progettato per fornire all'AI tutte le informazioni pertinenti sullo stato del gioco, assicurando che il suggerimento sia sempre contestualizzato.
- *Bilanciamento degli attributi di qualità*: L'AI considera le priorità degli stakeholder e cerca di massimizzare i punteggi QA, trovando il miglior compromesso tra le diverse opzioni di design.
- *Adattamento agli eventi*: Ogni evento imprevisto viene integrato nella logica di suggerimento, rendendo il modello dinamico e reattivo a cambiamenti nel contesto del gioco.

4 Esempio di uso e analisi dei suggerimenti dell'AI

Per comprendere meglio come il DecidArchAssistant supporti le decisioni architetturali, analizziamo due casi pratici di utilizzo dell'AI. Questi esempi illustrano come l'assistente offra suggerimenti basati sulle *Concern Card* e su eventi imprevisti, aiutando i giocatori a bilanciare le priorità degli stakeholder e le necessità del progetto.

4.1 Caso 1: Violazione della sicurezza e interruzione del server

Il primo caso coinvolge una carta *Concern* relativa a una violazione di sicurezza (*Security Breach*) e un evento imprevisto che provoca un'interruzione del server (*Server Outage*).

Suggerimento per la *Concern Card*: Il giocatore si trova ad affrontare una situazione in cui la sicurezza del sistema è compromessa a causa di una violazione. La descrizione del progetto indica che si tratta di una web app scalabile per il commercio elettronico, con due stakeholder chiave: il proprietario e l'utente finale. Il proprietario assegna una priorità elevata alla disponibilità e alla sicurezza, mentre l'utente dà maggiore importanza all'usabilità e alle prestazioni.

L'AI analizza lo stato attuale del gioco, che include decisioni progettuali precedenti orientate a bilanciare sicurezza e disponibilità, sebbene i punteggi di qualità per sicurezza e prestazioni siano subottimali. In risposta alla *Concern Card* sulla violazione di sicurezza, l'AI suggerisce di affrontare immediatamente il problema, consigliando un'opzione di design che, pur riducendo ulteriormente sicurezza e prestazioni, minimizza i rischi futuri di danni reputazionali e finanziari. L'opzione suggerita riflette la capacità dell'AI di prendere in considerazione il quadro generale e le priorità di lungo termine degli stakeholder, evidenziando come un mancato intervento sulla sicurezza possa avere conseguenze peggiori.

Suggerimento per l'*Event Card*: Successivamente, l'evento di *Server Outage* impone una rivalutazione delle scelte progettuali precedenti. L'AI ricalibra il suggerimento, raccomandando l'aggiunta di ridondanza ai server per migliorare la disponibilità del sistema, in linea con le priorità espresse dal proprietario. Questo esempio mostra come l'AI sia in grado di adattarsi a eventi dinamici e proporre soluzioni che tengano conto dell'intero contesto di progetto.

4.2 Caso 2: Degrado delle prestazioni e sforamento del budget

Il secondo scenario esplora un degrado delle prestazioni (*Performance Degradation*) e un evento che segnala un superamento del budget (*Cost Overrun*).

Suggerimento per la *Concern Card*: In questo caso, l'AI si trova a gestire un problema di prestazioni, con l'applicazione che non riesce a mantenere la velocità necessaria per gestire un traffico crescente. Dopo aver analizzato le priorità degli stakeholder — con l'utente finale che privilegia usabilità e prestazioni, e il proprietario che punta su disponibilità e sicurezza — l'AI consiglia di privilegiare le prestazioni rispetto all'usabilità. Il suggerimento riflette la necessità di garantire che l'applicazione resti performante e in grado di scalare in base al traffico, bilanciando le esigenze dei diversi stakeholder.

Suggerimento per l'*Event Card*: A seguito dell'evento *Cost Overrun*, l'AI riconosce la necessità di rivedere le scelte progettuali in modo da ridurre i costi, pur mantenendo alti livelli di sicurezza e disponibilità. Propone di rivedere le decisioni sull'infrastruttura, ottimizzando i costi senza compromettere la qualità complessiva del sistema. Questo dimostra come l'AI riesca a bilanciare vincoli economici e priorità tecniche, fornendo un supporto dinamico e flessibile per prendere decisioni strategiche.

Conclusioni dagli esempi: Questi casi pratici mostrano chiaramente come il DecidArchAssistant sia in grado di fornire suggerimenti ben calibrati e motivati, tenendo conto sia delle priorità degli stakeholder sia delle dinamiche del progetto. In entrambi gli esempi, l'AI ha aiutato i giocatori a navigare tra compromessi complessi, come sicurezza vs. prestazioni o costi vs. qualità, dimostrando la sua utilità nell'assistere decisioni architetturali in ambienti dinamici e incerti.

5 Conclusione

Il progetto DecidArch dimostra come un assistente AI possa essere integrato in un contesto di gioco educativo per simulare decisioni architetturali complesse. Gli esempi discussi illustrano chiaramente come l'AI sia in grado di gestire scenari dinamici, fornendo suggerimenti ben ponderati che tengono conto delle priorità degli stakeholder, delle scelte progettuali precedenti e degli eventi imprevisti.

L'approccio di utilizzare un assistente virtuale offre numerosi vantaggi, tra cui:

- **Apprendimento autonomo**: I giocatori possono esplorare diverse soluzioni architetturali senza la necessità di un facilitatore umano,

sviluppando una comprensione più profonda delle dinamiche di progettazione software.

- **Supporto decisionale immediato**: L'AI fornisce feedback in tempo reale, aiutando i giocatori a prendere decisioni informate e a valutare i compromessi tra diversi attributi di qualità.
- **Adattabilità a eventi imprevisti**: L'AI è in grado di rivedere le decisioni passate in risposta a eventi che alterano le priorità del progetto, garantendo un approccio dinamico e flessibile.

In conclusione, DecidArch rappresenta un esempio innovativo di come l'AI possa supportare l'apprendimento collaborativo e il processo decisionale in contesti complessi, promuovendo una riflessione critica e una comprensione approfondita delle sfide architetturali. La combinazione tra gioco e AI crea un ambiente immersivo in cui i partecipanti possono sviluppare competenze chiave nel design architetturale, rendendo l'esperienza educativa coinvolgente ed efficace.