Titolo avventura

The Mystery Castle

Nome del team

Guardiani del Chill

Membri del team di sviluppo

Alessandro Palmitessa

Abdulkarim Zahidi

Alessandro Moré

Esame di

Metodi Avanzati di Programmazione

(Track M-Z)

A.A. 2024/2025

Descrizione dell’avventura

L’avventura che si è decisa di implementare è un avventura grafica ambientata nel Castello Carlo V di Monopoli. Riprende un’antica leggenda della città dove la moglie del Re, quest’ultimo allontanato in mare per una missione e non facendo più ritorno, ogni notte lo chiama con un lamento per cercare di fargli trovare la via giusta del ritorno.
La nostra avventura ha principalmente questi personaggi e, come protagonista, un turista sudamericano venuto in vacanza, dopo aver ascoltato numerosa pubblicità di questo famoso castello lo vuole a tutti i modi visitare. Una sera nei suoi giorni di permanenza a Monopoli decide di andare a visitarlo, dove ovviamente trova chiuso non essendo aperto la sera.
Il giorno dopo, con pensiero fisso solo a quello, ritorna nuovamente al mattino presto.
Trova chiuso, ma ad un tratto gli viene incontro la portineria Maria che gli concede una possibilità di entrare nel castello fuori orario offrendogli anche un incentivo visto il suo spirito vivace: se risponde ai quesiti posti in modo corretto poteva avere un agevolazione sul prezzo del biglietto d’ingresso. Quindi, entrato in qualche modo nel castello ha due modi di proseguire, dove anche scegliendo una delle due porte, poi la stanza scelta avrà il collegamento diretto con la successiva non scelta: una è la stanza del falegname, l’altra è la stanza del fantasma. Finite di visitare queste due stanze con i loro indovinelli e quiz di cui si lascia al giocatore la possibilità di capire il funzionamento senza spoilerare nulla, si prosegue salendo sul terrazzo. Qui, con una probabilità calcolata scivolerà 1 / 2 volte o proprio nessuna andando direttamente al terrazzo. Se ci troviamo nel caso che scivola la prima volta e ha eseguito la stanza del falegname per ultima esecuzione, allora tornerà lì (in quel livello) e risponderà a domande ovviamente diverse da quelle di prima. Se ci troviamo nel caso che scivola la prima volta e ha eseguito la stanza del fantasma per ultima esecuzione, allora andrà nella cantina tutta buia e sporca.
Superata una delle due stanze di prima, ritornerà al pianerottolo della salita in terrazzo; dove potrebbe farcela a salire le scale oppure ricadere nuovamente a qualche livello più basso.
A questo punto, se prima aveva fatto il livello 2 con risposte diverse (2-bis) allora farà il livello 4 della cantina: se invece, aveva fatto il livello 4 della cantina farà il livello 2-bis.
Nel caso ci troviamo a fare il livello 4 della cantina, lui è costretto a trovare una torcia per fare luce e raccogliere qualche oggetto (il pastorale) da portarsi a casa come “souvenir” dell’avventura. Risale al pianerottolo delle scale e questa volta ce la fà! (Se non è già salito in tutte queste situazioni precedenti, ora arriva sul terrazzo!) Vede in lontananza il Re in ritorno dall’avventura in mare che gli porta la bandiera da incastrare in cima al castello per terminare l’avventura. Qui, brevi dialoghi tra re e turista finendo l’avventura dopo aver soddisfatto il criterio: Inserire la bandiera sul tetto del Castello. La gestione dell’avventura avviene totalmente attraverso un database che si trova nel package Database, che esegue ad ogni chiusura della finestra del gioco il salvataggio di livello.

Progettazione

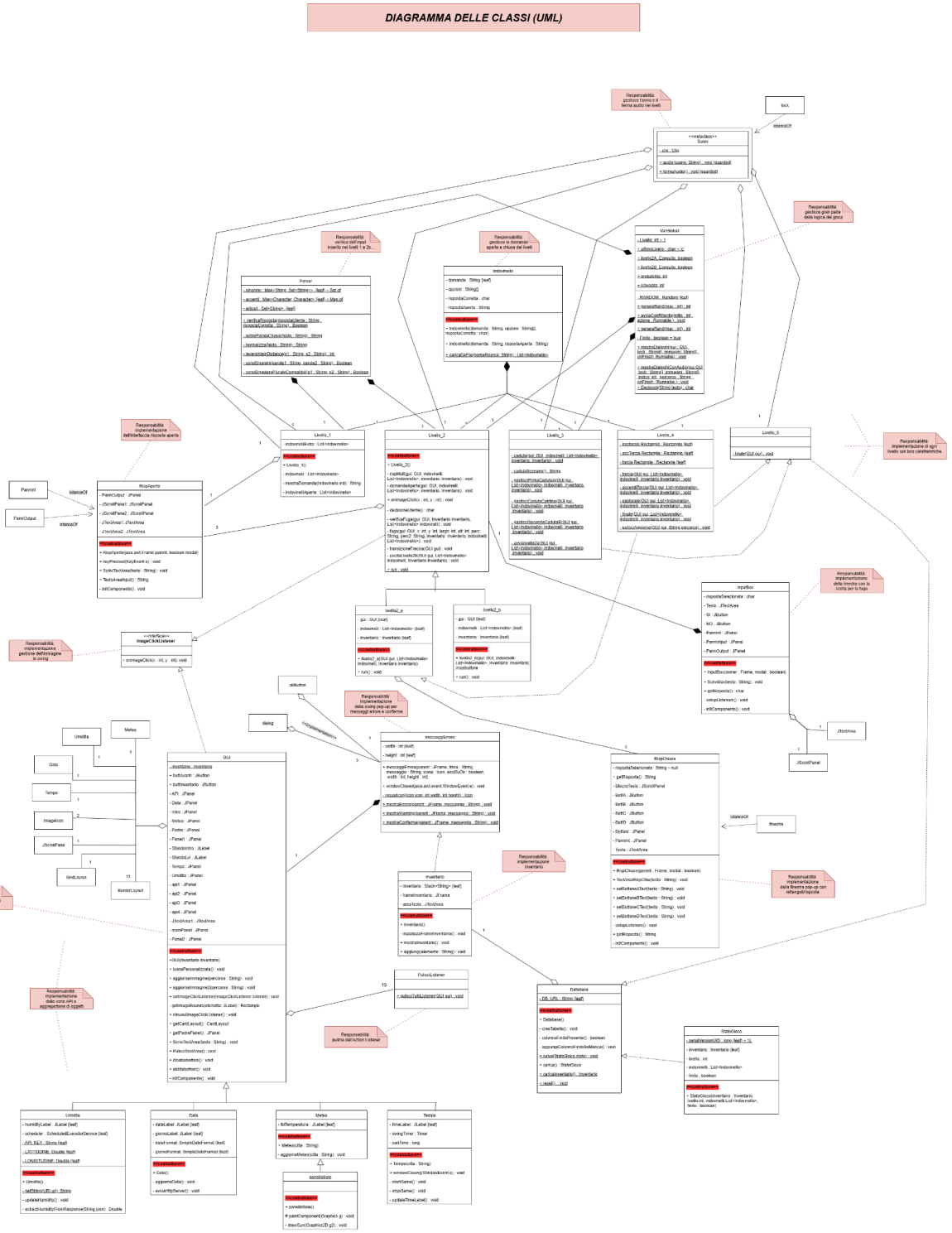
La progettazione è stata implementata in riferimento alla storia che abbiamo stipulato assieme, basandosi su dettagli tecnici e implementativi per far rimanere il gioco realistico e divertente. I package presenti nel progetto sono quattro:

- *Interfacce;*
- *Livelli;*
- *API;*
- *Database.*

Questi quattro package sono a loro volta composti da classi utili all'evoluzione del codice.
Nel primo package "Interfacce" abbiamo una classe riguardante la gui di tutti i livelli, una classe per indovinelli a risposte aperte (Livello 1 e livello 3), una classe per risposte chiuse (Livello 2/2-bis), una classe per messaggi d'errore causa sbaglio risposte agli indovinelli o errori di server o errori di click in posti sbagliati dell'immagine, una classe inputBox() che riguarda il "lancio" dalla finestra dei livelli 2 e 3, una classe VarGlobali() per avere variabili utilizzabili in più file già inizializzate.
Nel secondo package "Livelli" abbiamo una classe per ogni livello che implementa la logica del suddetto gioco, una classe per il parser che ci servirà per controllare l'input dell'utente nelle risposte aperte, una classe per i suoni implementati in quasi ogni livello, una classe che contiene tutti gli indovinelli e le risposte del gioco letti da file, una classe Inventario che servirà come database per gli oggetti raccogli durante l'avventura.
Nel terzo package "Api" abbiamo una classe per ogni api implementata, quindi una classe per l'api meteo, una classe per l'api umidità, una classe per l'api data e una classe per l'api tempo. Ogni API ha una sua chiave logica che si collega con il server dedicato a ciascuna.
Le classi sono state individuate dandole un significato completo e composto della situazione che dovevano svolgere. Per esempio: nel package Interfaccia avevamo una classe gui che era uguale per ogni livello dove ci implementava come doveva essere visibile ai nostri occhi il gioco tra cambi immagine, tra cambi livelli e azioni varie; l'interfaccia rimane comune e fissa, cambiano solo i testi nel JText e le immagini nel JLabel.
Nel quarto package "Database" abbiamo due file: uno ci servirà nella creazione e aggiornamento continuo di un db in base alle giocate sull'avventura, l'altro ci servirà a capire quali elementi devono essere salvati nel db in modo tale da trovarli già pronti in maniera efficiente all'avvio della stessa una prossima volta (si ipotizza il salvataggio di livelli già superati).

Diagramma delle classi

Vedere qui: [Link per visualizzazione online del diagramma]



Specifica algebrica

Specifica algebrica della struttura dati utilizzata nel progetto: il Set

Specifica Algebrica per Set[T] dove con T intendiamo il tipo degli elementi presenti all'interno del Set ed il set assume lo stesso valore che ha T;
ad esempio se T è stringa allora Set[T] sarà un Set di stringhe. Riassumendo i tipi usati all'interno del Set sono:

- Set<T> -> tipo del set;
- T -> tipo degli elementi;
- boolean -> tipo booleano per operazioni di verifica.

Specifica sintattica:

Nella specifica sintattica ogni insieme viene detto Sort.
Qui abbiamo come sort:

-> T che indica il tipo di parametri,
-> Set il set che assume lo stesso tipo di T,
-> Boolean che serve solo a dirci se è vero o falso.

Sorts: Set, T, boolean

Operazioni ammesse

empty() → Set<T>

insert(Set<T>, T) → Set<T>

contains(Set<T>, T) → boolean

remove(Set<T>, T) → Set<T>

isEmpty(Set<T>) → boolean

size(Set<T>) → int

union(Set<T>, Set<T>) → Set<T>

intersection(Set<T>, Set<T>) → Set<T>

difference(Set<T>, Set<T>) → Set<T>

Specifica semantica:

Nella specifica semantica, specifico i parametri su cui gli operatori devono lavorare:

contains(empty(), x) = false
contains(insert(s, x), x) = true
x ≠ y ⇒ contains(insert(s, x), y) = contains(s, y)

insert(insert(s, x), x) = insert(s, x)
x ≠ y ⇒ insert(insert(s, x), y) = insert(insert(s, y), x)

remove(empty(), x) = empty()
remove(insert(s, x), x) = remove(s, x)
x ≠ y ⇒ remove(insert(s, x), y) = insert(remove(s, y), x)

isEmpty(empty()) = true
isEmpty(insert(s, x)) = false
size(empty()) = 0
contains(s, x) ⇒ size(insert(s, x)) = size(s)
¬contains(s, x) ⇒ size(insert(s, x)) = size(s) + 1

union(s1, s2) = { x | x ∈ s1 ∨ x ∈ s2 }
intersection(s1, s2) = { x | x ∈ s1 ∧ x ∈ s2 }
difference(s1, s2) = { x | x ∈ s1 ∧ x ∉ s2 }

****Specifica di restrizione****

Indica le restrizioni che ci devono essere all'interno delle varie operazioni, ovvero quel tipo di errori che serve a limitare una tale operazione
(circuito chiuso).

restrictions
remove(empty(), x) = error
contains(insert(s, x), y) = contains(s, y) se x ≠ y = error

Dettagli implementativi

Programmazione generica

Nella programmazione sono stati usati: i metodi dove servivano, le classi opportune (come già detto nel punto precedente), gli oggetti nelle classi, le strutture algebriche necessarie per il raccoglimento di informazioni (liste e dizionari), variabili globali istanziate in classe varGlobali visti i comportamenti molte volte dannosi per tutto il codice, costruttori istanziati dove servono (esempio: public Indovinelli in classe Indovinelli; metodo speciale public GUI vista l'istanziazione di Componenti) e distruttori simili.
Utilizzati ancora: classi astratte (esempio: classe Suoni), le metaclassi (es: livello 4 viene ripreso come istanza nel livello 3), buon utilizzo dell' ereditarietà (esempio di ereditarietà multipla: classe Indovinello che generalizza in classe RispAperte e classe Risp Chiuse per accedere all'interfaccia con i suoi indovinelli presi da file), buona modularità sui metodi

implementati, ottimo uso del polimorfismo (esempio: scriviTextArea con diverse realizzazioni), uso di metodi statiche (es: caduta implementata nel livello 3), uso di metodi dinamica (esempio: fuga implementata nel livello 2a e 2b per la caduta dalla finestra in caso di risposte sbagliate). Viene utilizzata anche l'istruzione "this" per richiamare gli attributi domanda, opzione, rispostaCorretta da classe Indovinello. Abbiamo anche dei metodi getter e setter come livello2a_eseguito e livello2b_eseguito dichiarati in classe VarGlobali; abbiamo richiamato la libreria java.util.Random che serve per calcolare la probabilità che il turista cada dalle scale nel livello 3. Questa probabilità che và decrementando come era scritto nella descrizione dell'avventura, con decremento fisso del 20%.
Passando a parlare un pò di espressioni regolari e verifica degli input da utente abbiamo implementato la classe Parser, dove utilizziamo dei range di caratteri su cui il programma analizza e si sofferma capendo se approvare o meno la stringa passata e inserita.
Inserendo delle Collection nel codice, si dà subito occhio ad estraiParolaChiave() che è a sua volta una Collection di paroleDaIgnorare appartenente alla classe Parser. Nelle Collection sono state utilizzate le Set, le Map, lo Stack e la List come strutture. Si è fatto grande uso, oltre al calcolo della probabilità, della libreria Math implementando ed integrando il metodo levensteinDistance nella classe Parser che serve a determinare di quanto due stringhe siano simili o per determinare di quanto cambino come distanza di caratteri tra di loro, caso non lo sono simili. Abbiamo implementato anche numerosi I/O stream, tra cui un esempio tattico è quello dell'AudioInputStream nella classe Audio.
L'interruzione dell'esecuzione è stata usata con il metodo fermaAudio() nel livello 2a, nel livello 2b e in altri. Usato anche uno dei tanti metodi implementati con tipologia identificazione di tipo a run-time per pulire la textArea ogniqualvolta che si passa da un'immagine all'altra nel JPanel con pulisciTextArea(). Ulteriore metodo di quest'ultima tipologia di dichiarazione di metodo è scriviTextArea(), che ci inserisce del testo nella JText usata come prompt di dialogo per far vedere all'utente tutta l'esecuzione verbale del gioco.

File

I file sono stati utilizzati per contenere gli indovinelli con domande e risposte precise oltre che dettagliate cui vengono controllate:
- se con risposta dell'utente da tastiera, l'input viene controllato dal metodo Parser appositamente implementata;
- se le risposte dell'utente sono quelle implementate con Interfaccia SWING e risponde con Event click sul rettangolo corretto, viene controllata la correttezza del click attraverso la posizione del quadrante eseguito con risposta esatta istanziata con un attributo rispostaCorretta.

Database (JDBC)

Il Java DataBase Connectivity lo abbiamo implementato inserendo nella logica del gioco un database per salvataggio e caricamento successivo del gioco da dove si è voluto fermare.
Quindi, ogni partita una volta chiusa la finestra viene salvata nel db e, se non è stata ancora conclusa, alla prossima apertura viene fatto ripartire dal livello in cui ci si era fermati.
Salva tutti i dati che ci sono nell'interfaccia, dagli oggetti presi fino a quell'istante nell'inventario + il livello che si è fermati.

Lambda Expression (compresi stream e pipeline)

L'utilizzo nel codice di questa sezione è implementata attraverso una libreria SWING chiamata .addActionListener. Ancora, l'utilizzo delle Lambda Expression è stato voluto per i timer che ci servono semplicemente per il cambio immagine in automatico dopo aver passato un tot-tempo fermi in una determinata situazione particolare (non è abilitato a tutti, ma solo alle immagini di tipologia scontata scelta da noi o di risposta a veloce interpretazione). Nel codice c'è ne sono parecchie commentabili. C'è un esempio anche di Lambda Expression di tipo stream & pipeline nel Livello 1 per filtrare le risposte agli indovinelli.

SWING

L'avventura implementata è totalmente SWING. Si è voluto inserire le immagini, anche sotto approvazione richiesta, perché la realizzazione di un'avventura totalmente Swing richiederebbe tantissimo tempo. Si è cercato comunque il compromesso giusto abilitando l'interazione con l'immagine attraverso eventi di ActionListener, iterazioni con pulsanti Button, rappresentazione di una JTextArea e un JPanel per l'immagine racchiusi tutti da un JPanel Padre che li assimila e raccoglie. Si è deciso di implementare a livello grafico anche alcuni indovinelli (si veda i livello 2.1) con risposte multiple e anche una mini grafica rappresentativa per quanto riguarda quiz a risposta aperta (si veda livello 1 e livello 2b). Oltretutto, come già detto prima gli eventi di ActionListener nel livello 1 (l'apertura della porta), nel livello 2 (le frecce per la decisione della stanza), nel livello 3 (la bomboletta), nel livello 4 (la torcia e il pastorale reale), nel livello 5 (la bandiera). I suoni, fanno parte sempre di un implementazione grafica (testuale è impossibile inserirli) che sono:

- ☐ suono vittorioso ad ogni cambio livello;
- ☐ rincorsa dei cani;
- ☐ chiusura porta;
- ☐ versi del fantasma;
- ☐ spruzzo della bomboletta;
- ☐ suono della salita delle scale;
- ☐ fuga del turista dalla finestra;
- ☐ caduta dalle scale;
- ☐ suono finale di fine avventura.

Thread e programmazione concorrente

Nell'implementazione, gran parte del progetto è suddiviso in thread ed è concorrente in azioni. Facciamo solamente esempio al livello 2, cui si suddivide in due thread (livello 2a e livello 2b) che vanno parallelamente come esecuzione: come finisce uno con la porta d'emergenza passa all'altra stanza… ma questo è solo uno degli esempi presenti in quest'avventura. Come esempio di programmazione concorrente si fà esempio anche al parallelo nel livello 3 quando cadendo torna a fare un livello tra il 2a o livello 4 (cantina) già programmato. E' tutto stipulato e programmato al minimo dettaglio.
Addirittura, si è voluto fortemente implementare (come forse già citato nella parte sopra) un metodo fermaAudio() per la sospensione dell'esecuzione del suono una volta passato da un'immagine all'altra in maniera frettolosa (quando ancora non è finito neppure l'audio-esecuzione-automatica). La sincronizzazione tra thread (processo leggero) e

programma principale del livello (processo pesante) è stata vista e rivista, anche ottimizzata in modo tale che la logica e la storia andassero di pari passo (si veda la gestione tra livello 3 caduta e livelli intercambiabili 2a e livello 4).

Socket e/o REST

Per quanto riguarda il parametro Socket e Rest si sono implementate quattro tipologie di API che stanno sulla parte bassa dell'interfaccia per tutta la durata del gioco. Queste API hanno metodi propri del protocollo HTTP che sono: HTTP Request e HTTP Response, che a loro volta riportano la key di accesso al database per "catturare" o "afferrare" le informazioni utili che ci devono mostrare. A loro volta, ad ogni avvio o anche all'interno del gioco vengono aggiornate con dati realistici. Le API implementate sono state già citate prima.

Dettagli sull'organizzazione lavoro collettivo

Si era d'accordo fin da subito sull'implementazione di un avventura reale ma allo stesso tempo empatica. Quindi si è puntato a valorizzare una città che nell'ultimo periodo si parla molto: MONOPOLI e le sue leggende. Il lavoro senza nessuna polemica è stato subito concordato, avendo tanto da fare e da svolgere, ci si è messi subito a lavoro.
Parte della gestione e coordinamento del codice è stata di Alessandro Palmitessa (capogruppo), cui ci ha diviso i ruoli e ha organizzato gli incontri settimanali.
La direzione grafica e correttiva di alcuni bug è stata di Alessandro Moré che ha stipulato anche, insieme ad AbdulKarim Zahidi, la storia e la realizzazione dei sottolivelli più complessi. Sempre il capogruppo, ci ha diviso i livelli da implementare e, come già detto, ci si incontrava una volta a settimana per capire a che punto stavamo e fare una review totale del codice con una possibile integrazione casomai era terminata l'implementazione e sviluppo di un determinato livello.
Dopo circa 8 settimane: tra modifiche della storia aggiunte nel durante, verifica del codice funzionante, implementazione documentazione, riformulazione della grafica errata inizialmente in alcuni livelli, si è concluso facendo un merge di tutto il codice e un testing finale. Come analisi retrospettiva del progetto direi che ci siamo trovati tutti e tre abbastanza bene, siamo stati tutti e tre uniti e collaborativi in ogni momento oltre che ben preparati (più o meno) in quasi tutte le evenienze di cambiamento e, tempestivamente, siamo riusciti sempre a correggere.

Soluzione del gioco

Vista la descrizione dell'avventura, non può mancare una sezione dove si spiega l'obiettivo finale dell'avventura. La nostra avventura (come forse più volte detto) serviva per far conoscere leggenda, storia e cultura della città di Monopoli… ma anche per divertire un pochino! L'obiettivo principale era far conoscere, era giocare su più fronti e con immaginazione (tipo la vicenda del tutto casuale che c'erano dei materassi proprio sotto al castello lasciati da un passante avendone in più in casa). La soluzione del gioco è quella di terminare l'avventura portando in cima al castello la "bandiera vittoriosa" che il Re porta nella

sua barca e cede al turista sudamericano per tutto quello che lui ha fatto. Ha fatto grandi cose: è stato curioso! Curioso di cultura, di amore per la patria; aveva “fame di storia”. Questo gioco è per chi lo avrà potuto scaricare e giocarci, un modo per spronare i ragazzi a leggere e acculturarsi sulle cose belle che ogni paese ha da offrire sulla sua storia.