ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Meta-relazione per “Programmazione ad Oggetti”: Progetto “Indovina Chi?”

Petreti Alex
Pojaghi Gaia
Ventaloro Mirco
Volpini Francesco

Anno Accademico 2023-24

Indice

Sommario

Lo scopo di questa relazione è illustrare e spiegare le diverse fasi attraversate durante la progettazione dell'elaborato in oggetto, fungendo come documentazione del programma e fornendo sufficienti dettagli tecnici affinché il lettore sia in grado di comprendere la struttura del software realizzato.

Il progetto in esame è stato realizzato nell'ambito del corso di Programmazione ad Oggetti, per l'anno accademico 2023-2024. Esso consiste nell'implementazione del gioco da tavolo "Indovina Chi" in modalità giocatore singolo. Il giocatore dovrà indovinare il personaggio scelto dal computer attraverso una serie di domande che gli permetteranno di escludere alcuni nomi fino ad arrivare alla soluzione e quindi al termine della partita. Le modalità di gioco saranno settate all'inizio della partita.

La relazione andrà quindi definendo le metodologie usate nella modellazione, nello sviluppo e nell'implementazione del software, esponendo anche requisiti iniziali, scelte di design e dettagli tecnici inerenti all'implementazione.

Capitolo 1

Analisi

Il progetto consiste nella creazione di un'applicazione informatica che riproduce il gioco da tavolo "Indovina Chi", da giocare in modalità singolo. L'obiettivo del giocatore sarà quello di indovinare il personaggio selezionato casualmente dal computer. Il giocatore potrà porre una domanda circa una caratteristica dell'aspetto del personaggio da indovinare: in base alla risposta, potrà eliminare tutti i personaggi che non si adattano alla caratteristica. Ad esempio, un giocatore può domandare se il personaggio porta gli occhiali: in caso di risposta affermativa possono essere eliminati tutti i personaggi che non li portano. Scopo del gioco è ovviamente indovinare il personaggio, facendo domande fino ad eliminare progressivamente tutte le figure tranne una, giungendo alla soluzione finale e concludendo così la partita.

1.1 Requisiti

Requisiti funzionali

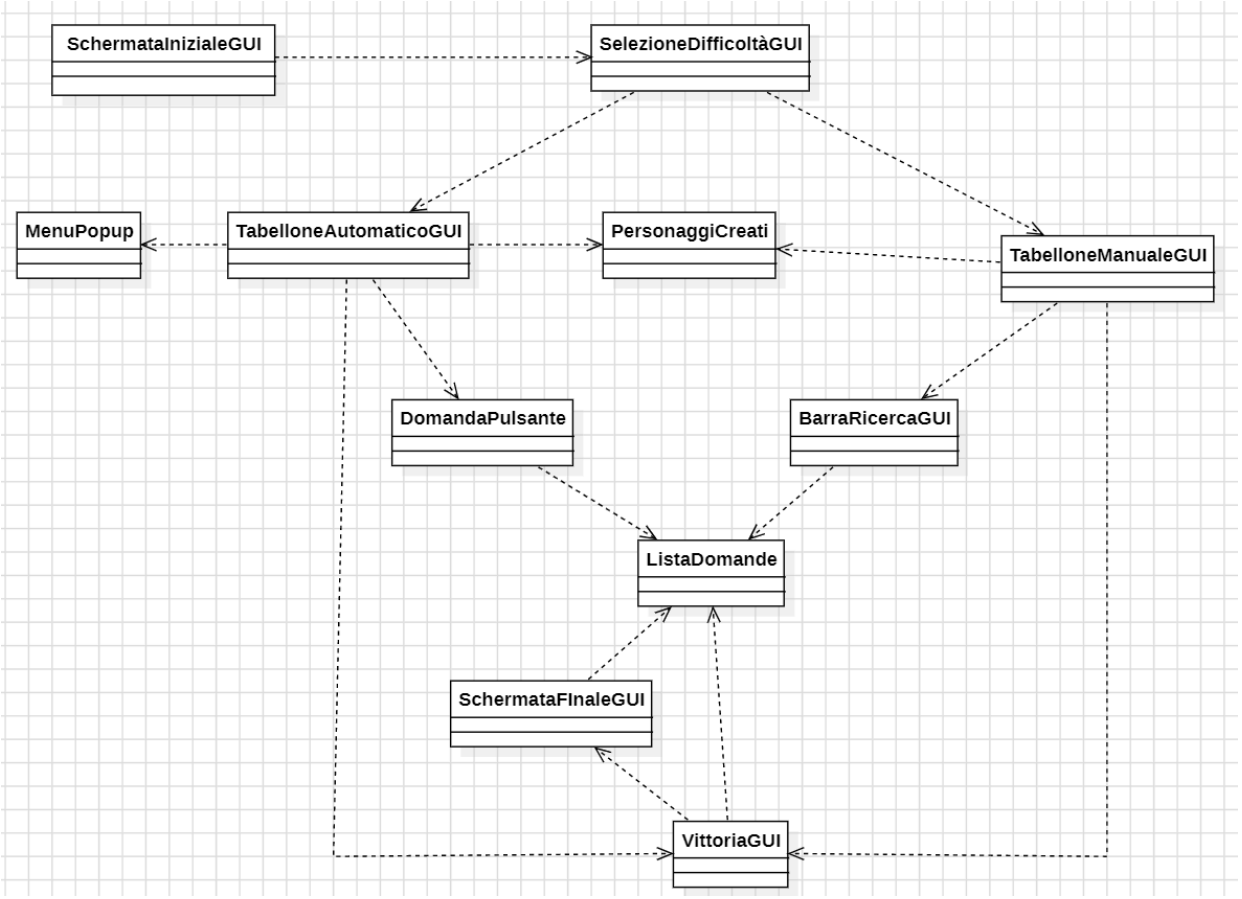
• All'avvio del gioco, verrà visualizzata una schermata iniziale che include un menu. Questo menu consentirà al giocatore di scegliere tra l'avvio di una nuova partita o la chiusura della sessione attuale.
•Il software comprenderà un menù per scegliere le modalità di gioco (automatica o manuale) e il livello di difficoltà (semplice, intermedio, difficile)
•Ogni livello offrirà una dimensione del tabellone diversa, aumentando così il numero di personaggi inclusi e rendendo più complesso indovinare la soluzione.
• Durante il gioco, il giocatore avrà accesso a un ulteriore menù che varierà a seconda della modalità selezionata. Questo menù consentirà di fare domande relative alle caratteristiche dei personaggi per avvicinarsi alla soluzione corretta.
• Durante una partita, il giocatore dovrà essere in grado di visualizzare il tabellone aggiornato e selezionare il personaggio che ritiene essere la soluzione, nel tentativo di indovinare correttamente.
• Dopo una sconfitta o una vittoria, verrà visualizzata una schermata con il risultato della partita. Da questa schermata, il giocatore avrà la possibilità di chiudere la sessione di gioco, visualizzare le statistiche della partita e la classifica, oppure tornare al menu di gioco iniziale.

Requisiti non funzionali

• Il software sarà progettato per essere portabile su una varietà di sistemi operativi, inclusi Windows, Linux e MacOS, garantendo così la compatibilità con i sistemi più diffusi.
• La finestra del programma sarà progettata in modo che possa essere ridimensionata senza compromettere la risoluzione e la qualità dell'immagine del gioco.

1.2 Analisi e modello del dominio

"Indovina Chi?" è un gioco da tavolo che si svolge su un tabellone suddiviso in caselle che rappresentano una serie di personaggi con determinate caratteristiche tra cui i giocatori possono scegliere per indovinare quello selezionato casualmente dal computer.
Ogni casella è contrassegnata con un'immagine e un nome per identificare univocamente il personaggio. Le carte personaggio sono disposte casualmente sul tabellone all'inizio del gioco, selezionandole tra tutti i personaggi disponibili e garantendo così un'esperienza sempre nuova e avvincente ad ogni partita.
Durante il gioco, i giocatori cercano di indovinare il personaggio segreto scelto casualmente dal computer. Ciò avviene attraverso una serie di domande poste dai giocatori stessi. Le domande sono progettate per identificare specifiche caratteristiche del personaggio segreto. Ad esempio, un giocatore potrebbe chiedere se il personaggio indossi degli occhiali o se sia di sesso maschile.
Le risposte alle domande vengono fornite attraverso una modalità automatica, attivata da un apposito pulsante e un menu, o attraverso una modalità manuale tramite una barra di ricerca. Se la caratteristica menzionata nella domanda è presente nel personaggio segreto, le caselle corrispondenti sul tabellone vengono chiuse. Altrimenti, vengono chiuse le caselle che non presentano quella caratteristica.
Attraverso un processo di deduzione e eliminazione, i giocatori cercano di restringere le possibilità fino a individuare il personaggio segreto. Quando il giocatore riesce a identificare correttamente il personaggio segreto vince la partita, ottenendo una schermata finale in cui potrà visualizzare il proprio punteggio nella classifica e un riassunto del gioco.



Schema UML dell’analisi del problema con rappresentate le entità principali ed i rapporti fra loro

Capitolo 2

Design

2.1 Architettura

Il gioco è concettualmente diviso in scene, ognuna dedicata a una fase specifica: una per il menu iniziale, una per la fase di gioco e una per i risultati di riepilogo e di classifica alla fine della partita.
La fase di gioco può essere divisa in due ambiti: il menù per formulare domande e il tabellone di gioco che rappresenta i personaggi.
Analizzeremo nel dettaglio l'architettura proposta nel codice in relazione al modello MVC (Model-View-Controller), si evidenzieranno come i concetti di Model, View e Controller siano rappresentati e si le ragioni della mancanza di eventuali parti.

Relativamente alla schermata iniziale viene utilizzato il modello MVC in parte.

- Model: la classe PersistentHashMap funge da modello. Gestisce la persistenza dei dati della mappa chiave-valore su disco. Incapsula la logica di accesso ai dati e fornisce metodi per aggiungere, rimuovere, ottenere e salvare elementi dalla mappa. Questa classe gestisce anche la persistenza dei dati dei giocatori.
- View: le classi SchermataInizialeGUI e SelezioneDifficoltaGUI fungono da vista. Gestiscono l'interfaccia utente e la presentazione grafica del gioco. SchermataInizialeGUI mostra l'immagine di benvenuto e fornisce pulsanti per selezionare la modalità di gioco. SelezioneDifficoltaGUI mostra i pulsanti per selezionare la difficoltà del gioco.
- Controller: la gestione degli eventi e la logica di controllo sono integrate nelle classi vista. I pulsanti presenti nelle interfacce utente (SchermataInizialeGUI e SelezioneDifficoltaGUI) sono associati agli ascoltatori di azioni che determinano il comportamento del programma in base all'input dell'utente. Ad esempio, la selezione della modalità di gioco e della difficoltà del gioco. Quindi, mentre la persistenza dei dati e la logica di gioco sono gestite dal modello, l'interfaccia utente e la gestione degli eventi sono gestite dalla vista, mantenendo il modello e la vista separati.

La parte di codice relativa ai personaggi è così suddivisa:

- Model: Le classi Personaggio e PersonaggioImpl costituiscono il modello dei personaggi nel gioco. Queste classi definiscono le caratteristiche e il comportamento dei personaggi. La classe PersonaggiCreati gestisce invece la creazione dei personaggi disponibili nel gioco. Questa classe agisce come

un'entità che si occupa della logica e fornisce metodi per ottenere una lista di personaggi in modo casuale.
- View: La vista non è esplicitamente rappresentata nel codice fornito.
- Controller: Nel codice fornito non c'è un controller definito. Tuttavia, il controllo sull'interazione con i personaggi (come la selezione e la gestione delle caratteristiche dei personaggi) avviene in altre parti del codice.

Relativamente al tabellone di gioco è possibile analizzare nel dettaglio l'uso del modello MVC.

- Model: Il modello è rappresentato principalmente dalla classe TabelloneImpl. Questa classe incarna la logica di base del tabellone di gioco, assumendo la responsabilità della gestione dei dati e della logica. Gestisce l'inizializzazione del tabellone con i personaggi, recupera il personaggio da indovinare e fornisce i metodi per accedere ai personaggi in una determinata posizione. In sostanza, rappresenta il nucleo della logica applicativa, mantenendo lo stato del gioco e definendo le operazioni eseguibili su di esso.
- View: La vista è rappresentata dalla classe TabelloneGUI. Questa classe si occupa dell'interfaccia utente del tabellone di gioco, gestendo la presentazione dei dati al giocatore e le interazioni utente. È responsabile della creazione dei pulsanti, del posizionamento dei personaggi sul tabellone e della gestione degli eventi di clic sui pulsanti. Tuttavia, la classe TabelloneGUI incorpora anche la logica di controllo, che va oltre il ruolo tradizionale della vista. Ciò introduce una mancanza di separazione tra il controllo e la vista, come discuteremo più avanti.
- Controller: Nel codice fornito, non c'è una chiara separazione del controller. Idealmente, il controller dovrebbe essere responsabile della gestione degli eventi e della logica di controllo, agendo come intermediario tra il modello e la vista. Tuttavia, la logica di controllo è incorporata direttamente nella classe TabelloneGUI, dove vengono gestiti gli eventi di clic sui pulsanti. Questo approccio viola il principio di separazione delle responsabilità. Tuttavia la mancanza di una separazione chiara tra il controllo e la vista può essere giustificata da diversi fattori: semplicità del sistema (in tale progetto introdurre una separazione tra controllo e vista avrebbe aggiunto complessità senza un chiaro beneficio), facilità di comprensione e manutenzione (incorporare la logica di controllo nella vista semplifica la comprensione del codice, riducendo la complessità complessiva e agevolando la manutenzione futura), scalabilità (una separazione rigorosa dei concetti MVC è per progetti complessi, ma potrebbe non essere necessaria inizialmente in quanto la separazione potrà essere introdotta in fasi successive dello sviluppo, se diventerà necessaria per gestire la complessità del sistema). Tuttavia, in scenari più complessi o in caso di requisiti futuri che richiedono una maggiore

modularità, potrebbe essere opportuno considerare l'introduzione di un controller dedicato per migliorare l'architettura complessiva del sistema.

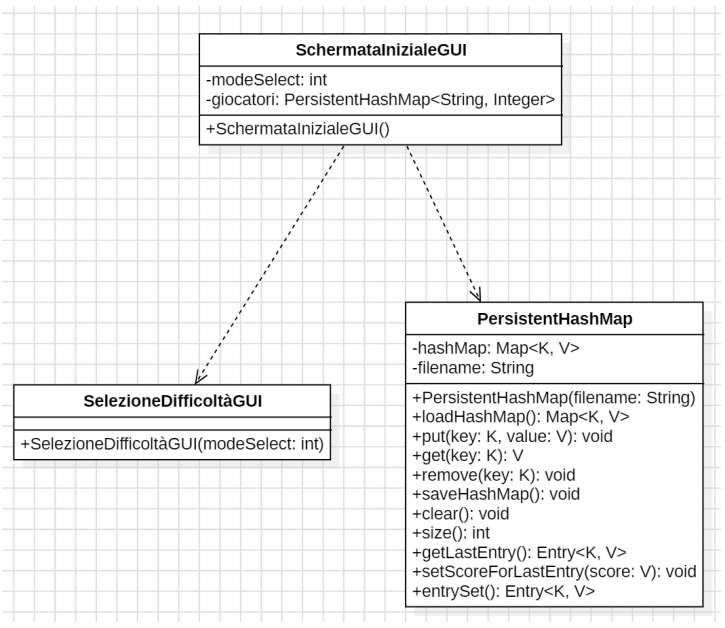
Relativamente alla schermata finale viene utilizzato il modello MVC in parte.

- Model: la classe PersistentHashMap funge da modello. Gestisce la persistenza dei dati dei giocatori e dei punteggi su disco. Incapsula la logica di accesso ai dati e fornisce metodi per aggiungere, rimuovere, ottenere e salvare i punteggi. La classe ListaDomande sembra gestire l'elenco delle domande. Anche se non è mostrato il suo codice, si presume che fornisca funzionalità per ottenere, aggiungere o rimuovere domande, fungendo da parte del modello.
- View: le classi SchermataFinaleGUI e VittoriaGUI fungono da viste. Gestiscono l'interfaccia utente e la presentazione grafica della schermata finale, mostrando riepiloghi delle domande, classifiche e risultati di vittoria o sconfitta. Queste classi creano finestre JFrame, aggiungono componenti Swing e gestiscono gli eventi utente tramite ActionListener, fornendo l'interfaccia utente per visualizzare le informazioni.
- Controller: la logica di controllo è integrata nelle classi vista. Gli ActionListener associati ai pulsanti gestiscono le azioni dell'utente, come giocare ancora, visualizzare dettagli, uscire dall'applicazione, ecc. Questi ActionListener influenzano il modello (ad esempio, aggiornando i dati dei punteggi dei giocatori) e la vista (ad esempio, creando nuove finestre o chiudendo quelle esistenti). In sintesi, anche se non c'è una netta separazione tra il modello, la vista e il controller, poiché la logica di controllo è incorporata direttamente nelle classi vista, il codice mantiene comunque una struttura che segue i principi del modello MVC.

2.2 Design dettagliato

2.2.1 Alex Petreti: Schermata Iniziale e Schermata Finale

- Schermata Iniziale



La schermata iniziale dell'applicazione è composta da due principali classi SchermataInizialeGUI e SelezioneDifficoltàGUI ed una classe di utilità PersistentHashMap.

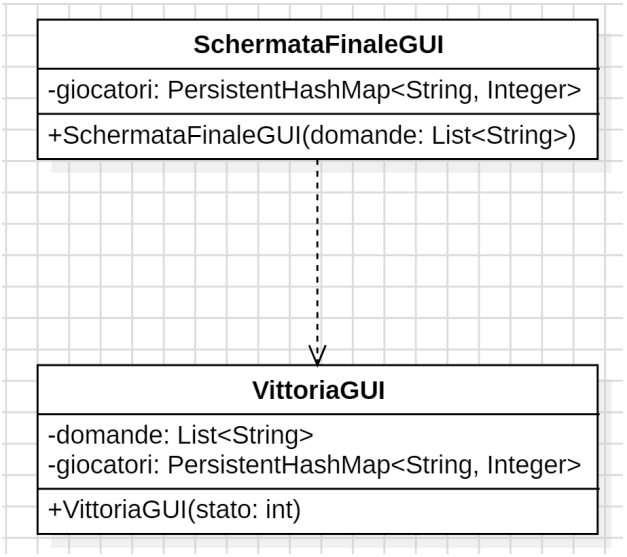
Queste prime due classi aprono una schermata simile dove è presente l'immagine del gioco e dei bottoni che permettono all'utente di interagire con l'interfaccia visiva.

SchermataInizialeGUI permette di scegliere la modalità di gioco e, dopo la scelta della modalità, si aprirà una finestra di dialogo dove verrà richiesto di inserire il proprio nome.
Ciò accade perché nella schermata finale sarà presente una classifica che mostrerà i nomi con il relativo numero di domande fatte prima di indovinare.

PersistentHashMap è la classe di utilità che si occupa di gestire la lista di giocatori con i relativi punteggi.
Questa classe, oltre ad avere metodi per modificare la lista a piacimento, si occupa di salvarla in un file serializzabile .ser in modo che possa essere sempre utilizzata anche ad avvii successivi dell'applicazione.

SelezioneDifficoltàGUI è la classe che si aprirà dopo aver scelto la modalità ed aver inserito il proprio nome.
questa classe è identica a SchermataInizialeGUI ma ha 3 bottoni che indicano le diverse difficoltà che si possono scegliere.
Una volta cliccata la difficoltà, questa classe si occuperà di invocare il tabellone con la grandezza relativa alla difficoltà scelta nella modalità di gioco selezionata nella classe prima.
Ho deciso di dividere la schermata iniziale dell'applicazione in due classi in modo che in futuro sia più semplice aggiungere nuove modalità o difficoltà.

- Schermata Finale



La schermata si compone di due classi VittoriaGUI e SchermataFinaleGUI.

VittoriaGUI si aprirà con un messaggio di vittoria in caso il personaggio sia stato indovinato, oppure con un messaggio di sconfitta in caso sia stato raggiunto il limite di domande.
In tutti e due i casi l'interfaccia grafica avrà 3 bottoni che permetteranno di: giocare ancora, uscire dall'applicazione oppure mostrare la classifica.

SchermataFinaleGUI è la classe che viene invocata da VittoriaGUI in caso si voglia mostrare la classifica.
Questa interfaccia grafica è divisa in due riquadri: il riquadro a sinistra mostra in un'area di testo tutte le domande fatte nel corso della partita, il riquadro di destra mostra la classifica dei giocatori con il numero di domande fatte prima di vincere.
In basso sono presenti due bottoni per giocare ancora oppure uscire dall'applicazione.

2.2.2 Gaia Pojaghi: Tabellone

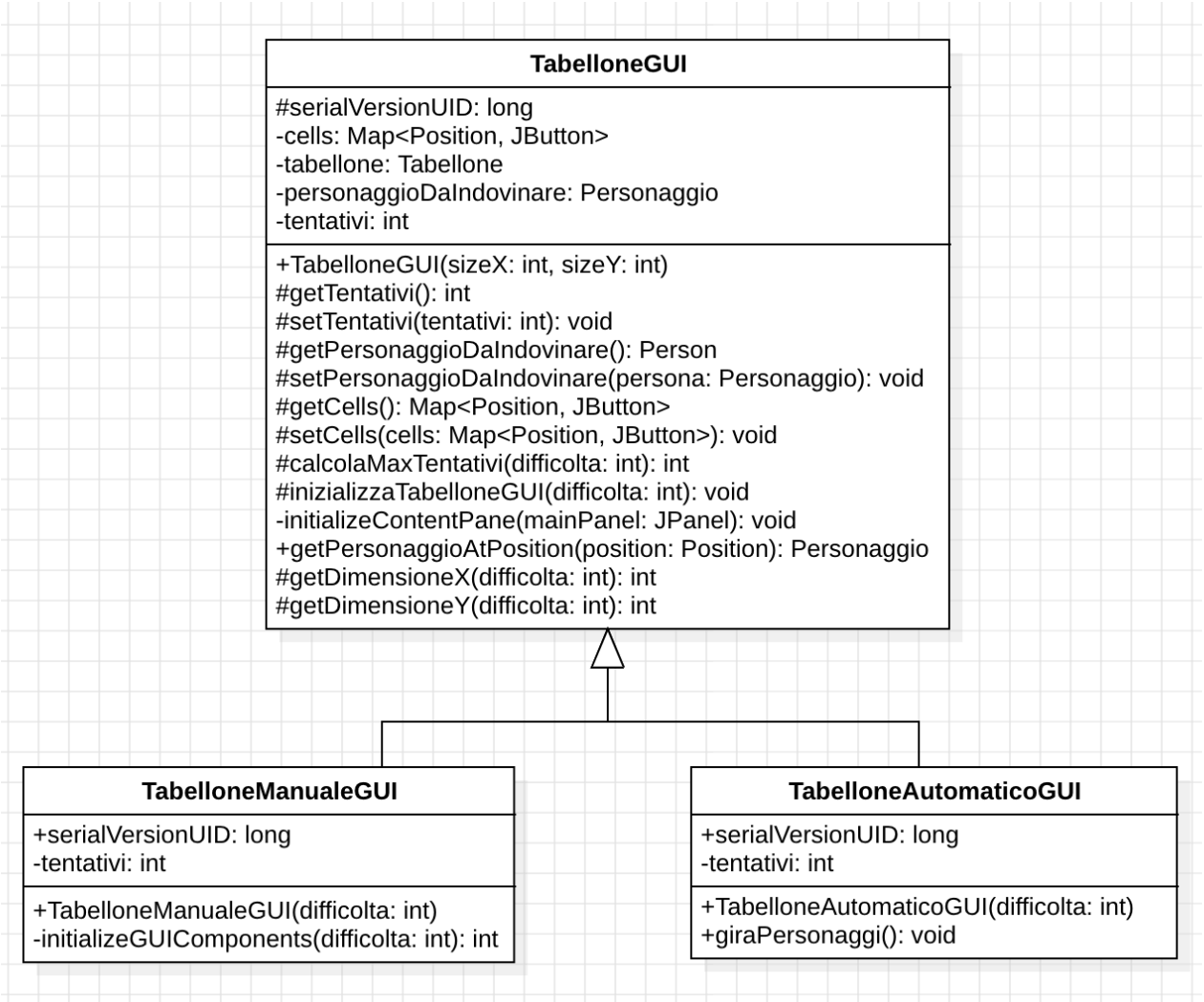
Nel progetto le classi TabelloneGUI e TabelloneImpl hanno responsabilità diverse all'interno dell'applicazione:

- TabelloneGUI gestisce l'interfaccia grafica del tabellone di gioco, occupandosi della creazione dei pulsanti, del posizionamento dei personaggi e della gestione degli eventi di clic sui pulsanti. In breve, si occupa di tutti gli aspetti visivi e interattivi dell'esperienza di gioco.
- TabelloneImpl, invece, si occupa della logica di base del tabellone di gioco. Questa classe gestisce operazioni come l'inizializzazione del tabellone con i personaggi, il recupero del personaggio da indovinare e il recupero dei personaggi in una specifica posizione. In altre parole, TabelloneImpl si occupa delle operazioni sottostanti che rendono possibile il funzionamento del gioco.

Ottimizzazione dell'interfaccia del tabellone di gioco con ereditarietà
Problema: Ho due classi GUI che implementano diverse modalità del tabellone di gioco. Queste classi hanno praticamente gli stessi metodi, portando quindi a una considerevole duplicazione di codice.

Risoluzione: Per risolvere questo problema, ho introdotto una classe denominata TabelloneGUI che estende JFrame e contiene tutti i metodi necessari per gestire il tabellone di gioco e fornirne le funzionalità di base. Questa classe contiene metodi per inizializzare l'interfaccia grafica del tabellone, gestire i pulsanti e le azioni associate ad essi, nonché per ottenere informazioni sul tabellone stesso, come il numero di tentativi effettuati e il personaggio da indovinare. Le classi già esistenti

sono state modificate per estendere questa nuova classe, sfruttando così l'ereditarietà dei metodi per evitare la duplicazione del codice. Le classi specifiche TabelloneAutomaticoGUI e TabelloneManualeGUI implementavano le diverse modalità del tabellone, ora estendono TabelloneGUI anziché implementare direttamente l'interfaccia grafica. In questo modo, possono ereditare tutti i metodi comuni forniti da TabelloneGUI, evitando la duplicazione del codice e migliorando la coesione del design complessivo.

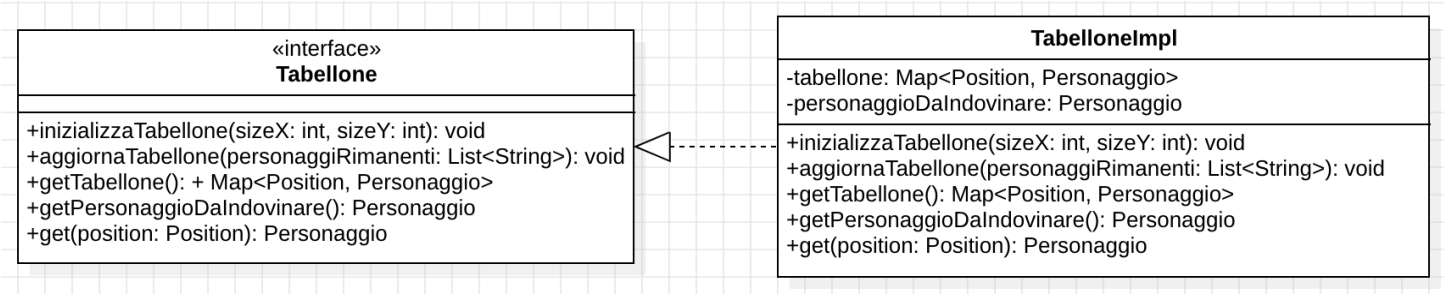


Riuso del codice con Template Method per livelli di difficoltà

Problema**:** Durante lo sviluppo del gioco da tavolo, è emersa la necessità di creare tre tipologie di tabellone corrispondenti ai livelli di difficoltà: "facile", "intermedio" e "difficile". Tuttavia, ho notato che queste tre tipologie di tabellone condividono molte caratteristiche comuni, come la gestione della disposizione dei personaggi, l'inizializzazione delle caselle e la gestione dei pulsanti. Questa situazione ha portato alla creazione di classi molto simili e alla duplicazione del codice.

Soluzione**:** Per risolvere il problema della duplicazione del codice e massimizzare il riuso del codice, ho adottato una soluzione basata sul design pattern "Template

Method". Questo approccio consente di definire uno scheletro in una classe base, con alcuni passaggi specifici implementati nelle classi derivate. L'interfaccia Tabellone e la classe TabelloneImpl sono l'implementazione di questa soluzione. Con questa soluzione è possibile creare diverse tipologie di tabellone per i diversi livelli di difficoltà, mantenendo allo stesso tempo il codice comune in una classe base e implementando solo le specifiche necessarie nelle classi derivate.



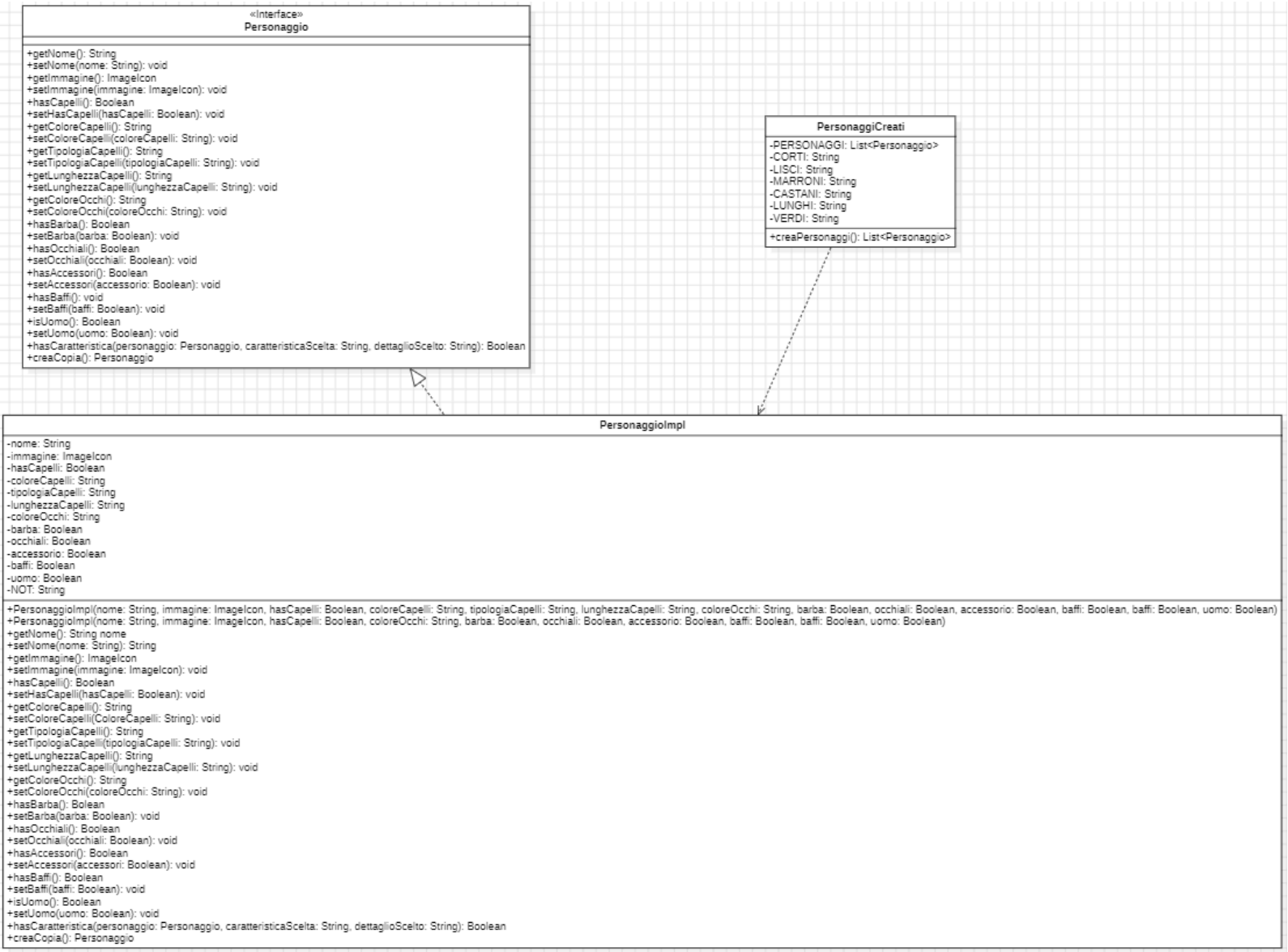
2.2.3 Mirco Ventaloro: Personaggi

Nel progetto la parte dei personaggi ha il compito di creare ogni personaggio con le relative caratteristiche e di associare ad ognuno un'immagine.
Per fare ciò sono partito dalla creazione di un'interfaccia che avesse al suo interno i metodi necessari per settare e restituire le caratteristiche dei personaggi.
Nella classe PersonaggioImpl questi metodi vengono implementati. Ho deciso di utilizzare tipi stringa per le caratteristiche che richiedono una scelta multipla (come il colore dei capelli), tipi booleani invece per quelle che richiedono una scelta binaria (come la presenza di occhiali).

Per fare in modo che ogni personaggio fosse disponibile per ogni modalità del gioco ho deciso di utilizzare una classe PersonaggiCreati che creasse ogni singolo personaggio con relativa immagine. Ho deciso di salvare ogni immagine in una propria variabile creando una copia dell'immagine originale.

Problema: nella creazione dei pulsanti per selezionare le caratteristiche nella modalità automatica, le caratteristiche sono state create in modo diverso da quello utilizzato nella parte relativa ai personaggi. Vengono create delle liste per ogni caratteristica composte dai dettagli selezionabili.
Soluzione: ho creato un metodo all'interno della classe PersonaggioImpl che prendesse in input il personaggio, il nome del pulsante selezionato nel menù corrispondente alla caratteristica e il dettaglio. A seconda dell'input vengono richiamati perciò metodi diversi creati dentro la classe.

Problema: utilizzando le immagini originali per comporre i personaggi, nel caso in cui esse avessero subito un'eventuale modifica sarebbero risultate poi inutilizzabili.
Soluzione: ho creato per ogni immagine una variabile che contenesse una copia di essa in modo che non dipendessero dalle originali le immagini che compongono la grafica di gioco.



2.2.4 Francesco Volpini: Menù

Il membro del team si è ritirato dal progetto prima del suo completamento, pertanto la sua parte di documentazione dettagliata sul design non è disponibile.

Capitolo 3

Sviluppo

3.1 Testing automatizzato

Numerose parti di codice sono state sottoposte a testing automatizzato. Nello specifico nel progetto si trovano:

- Il test TestMenuPopUp è una classe Java utilizzata per testare la classe MenuPopup. Nel metodo main, viene istanziata un'istanza di MenuPopup con un parametro 0. Questo rappresenta un test di base per la classe MenuPopup per verificare che l'istanza venga creata correttamente senza generare eccezioni.
- Il test PersonaggiCreatiTest è una classe di test per la classe PersonaggiCreati. Questa classe di test utilizza il framework JUnit Jupiter per eseguire test automatizzati sulla logica di creazione dei personaggi implementata nella classe PersonaggiCreati.
- Il test PersonaggioImplTest è una classe di test per la classe PersonaggioImpl. Questo test verifica diversi aspetti del comportamento dei personaggi implementati, tra cui la corretta impostazione e il recupero del nome, dell'immagine, delle caratteristiche fisiche come colore e lunghezza dei capelli, colore degli occhi e la presenza di barba, occhiali e accessori. La classe di test utilizza il framework JUnit Jupiter per definire e eseguire i test. I test includono anche metodi di utilità per la conversione di immagini e verificano varie proprietà e comportamenti dei personaggi, come la correttezza dei valori impostati e la presenza o assenza di determinate caratteristiche. Complessivamente, questo test fornisce fornisce una copertura esaustiva per assicurare il corretto funzionamento della classe PersonaggioImpl, coprendo diverse proprietà e comportamenti dei personaggi implementati.
- Il test VittoriaTest è una classe Java utilizzata per avviare l'interfaccia grafica della schermata finale di vittoria, rappresentata dalla classe VittoriaGUI. Nel metodo main, viene istanziata un'istanza di VittoriaGUI con un parametro 1, che indica la configurazione per la schermata di vittoria.
- Il test TestBarraRicercaGUI è una classe Java utilizzata per testare la classe BarraRicercaGUI. Nel metodo main, viene istanziata un'istanza di BarraRicercaGUI con il parametro "Natalie". Questo rappresenta un test di base per la classe, che verifica che la barra di ricerca possa essere istanziata correttamente con un termine di ricerca predefinito come "Natalie".
- Il test TabelloneAutomaticoTest è una classe Java utilizzata per avviare l'interfaccia grafica del tabellone di gioco automatico, rappresentata dalla

classe TabelloneAutomaticoGUI. Nel metodo main, viene istanziata un'istanza di TabelloneAutomaticoGUI con il parametro 3, che indica la dimensione del tabellone di gioco. Questo test può essere utilizzato per verificare che l'interfaccia grafica del tabellone di gioco automatico si avvii correttamente e che sia in grado di gestire un tabellone di dimensioni specifiche.

- Il test TabelloneManualeTest è una classe Java utilizzata per avviare l'interfaccia grafica del tabellone di gioco manuale, rappresentata dalla classe TabelloneManualeGUI. Nel metodo main, viene istanziata un'istanza di TabelloneManualeGUI con il parametro 3, che indica la dimensione del tabellone di gioco. Questo test può essere utilizzato per verificare che l'interfaccia grafica del tabellone di gioco manuale si avvii correttamente e che sia in grado di gestire un tabellone di dimensioni specifiche.

3.2 Note di sviluppo

Nel progetto del gioco da tavolo, sono implementate alcune caratteristiche avanzate del linguaggio e dell'ecosistema Java per garantire una progettazione efficiente e funzionale.

3.2.1 Alex Petreti

- Uso di generics: ho utilizzato le generics nella mia classe di utilità per gestire la classifica dei giocatori PersistentHashMap in modo che possa essere utilizzata nel modo più flessibile possibile.

    1. https://github.com/Ribenh/OOP23-indovinachi/blob/e73ba075ec06228b09652b4d2574bf08df4bb32e/src/main/java/schermatainiziale/PersistentHashMap.java#L1-L198+

- Utilizzo di un file serializable: il file .ser che viene generato da PersistentHashMap è serializzabile e rimane salvato anche dopo la chiusura dell'applicazione. Questo permette di poter utilizzare la classifica giocatori anche dopo la prima partita.

    1. https://github.com/Ribenh/OOP23-indovinachi/blob/e73ba075ec06228b09652b4d2574bf08df4bb32e/src/main/java/schermatainiziale/PersistentHashMap.java#L1-L198

3.2.2 Gaia Pojaghi

- Uso di generics: i generics vengono impiegate nell'interfaccia Tabellone e nella sua implementazione TabelloneImpl, consentendo la definizione di nuovi tipi generici e l'utilizzo di bounded generics per mantenere la flessibilità del

codice e favorire la riutilizzabilità delle classi.
https://github.com/Ribenh/OOP23-indovinachi/blob/418e30981ebf343ecf27fee75ccc1414b5b5e58e/src/main/java/tabellone/Tabellone.java#L29
https://github.com/Ribenh/OOP23-indovinachi/blob/418e30981ebf343ecf27fee75ccc1414b5b5e58e/src/main/java/tabellone/Tabellone.java#L35
https://github.com/Ribenh/OOP23-indovinachi/blob/418e30981ebf343ecf27fee75ccc1414b5b5e58e/src/main/java/tabellone/TabelloneImpl.java#L19

- Uso di lambda expressions: le lambda expressions sono usate negli ActionListener dei pulsanti presenti nell'interfaccia grafica del tabellone di gioco. Queste espressioni consentono di gestire in modo conciso gli eventi di click dei pulsanti, migliorando la leggibilità del codice e semplificando la gestione delle interazioni utente.
  https://github.com/Ribenh/OOP23-indovinachi/blob/418e30981ebf343ecf27fee75ccc1414b5b5e58e/src/main/java/tabellone/TabelloneManualeGUI.java#L31C5-L51C6
  https://github.com/Ribenh/OOP23-indovinachi/blob/418e30981ebf343ecf27fee75ccc1414b5b5e58e/src/main/java/tabellone/TabelloneAutomaticoGUI.java#L38C9-L71C1

Queste funzionalità avanzate del linguaggio Java contribuiscono significativamente alla progettazione di un sistema robusto, flessibile e manutenibile, offrendo allo sviluppatore strumenti potenti per realizzare un'applicazione efficiente e intuitiva.

3.2.3 Mirco Ventaloro

- Utilizzo della classe “ImageIcon” fornita da Java Swing per la gestione delle immagini dei personaggi che compongono la grafica di gioco relativa al tabellone. Ciò consente di il caricamento, l’accesso e la protezione delle immagini all’interno dell’applicazione, in modo che le immagini siano protette da modifiche esterne.
  https://github.com/Ribenh/OOP23-indovinachi/blob/e73ba075ec06228b09652b4d2574bf08df4bb32e/src/main/java/personaggi/PersonaggiCreati.java#L24-L47

Capitolo 4

Commenti finali

4.1 Autovalutazione e lavori futuri

3.2.1 Alex Petreti

Al termine del progetto, sono soddisfatto di aver completato con successo lo sviluppo di un'applicazione funzionale e divertente insieme al nostro team. Durante questo percorso, ho incontrato diverse sfide e riconosco che avrei potuto fare una ricerca più approfondita prima di immergermi nella programmazione, al fine di prendere decisioni di design più informate. Tuttavia, credo fermamente nell'importanza della pratica e nell'apprendimento attraverso gli errori, poiché ciò permette di migliorare nei settori in cui si è meno esperti.

Indipendentemente dall'esito finale, questa esperienza è stata estremamente positiva per il mio sviluppo personale. Mi ha sfidato in modi inaspettati e, anche quando ho incontrato momenti di difficoltà, sono riuscito a superarli e portare il progetto a termine con successo. Mi sento di aver compiuto un passo avanti significativo lungo il mio percorso nella programmazione.

Ho apprezzato molto la collaborazione con gli altri membri del team, che si sono dimostrati sempre disponibili e comprensivi. L'unica pecca è stata l'improvvisa partenza di un membro del team nei giorni precedenti la consegna finale. Questo ci ha costretti ad affrontare e comprendere il suo codice senza il suo contributo diretto, il che è stato indubbiamente complicato.

Nonostante le sfide incontrate, ritengo che questa esperienza sia stata estremamente gratificante. Ha rappresentato una prova impegnativa, ma mi ha fornito preziose lezioni e soddisfazioni.

3.2.2 Gaia Pojaghi

Al termine del progetto, mi sento estremamente soddisfatta per diversi motivi. In primis, questa è stata la mia prima esperienza di lavoro in un progetto di gruppo, e ritengo di aver imparato molto durante lo sviluppo di questo progetto. In secondo luogo perché le difficoltà incontrate, sia a livello personale sia a livello di gruppo, non sono state poche. Inoltre, devo ammettere che la gestione del tempo è stata una sfida, in particolare nel riuscire a bilanciare lo studio per gli esami con il lavoro sul progetto, specialmente durante una sessione intensa come quella in questione. Probabilmente molti aspetti del mio codice sarebbero potuti essere implementati meglio, ma questo mi avrebbe portato via molto tempo per approfondire le mie conoscenze e quindi ho preferito strutturare il mio codice con semplicità per favorirne

la chiarezza. Ritengo però che attraverso questo lavoro io abbia appreso molto sia a livello di linguaggio di programmazione ad oggetti sia a livello di relazioni all'interno di un gruppo di lavoro con persone non conosciute.
Come menzionato precedentemente, infatti, ci sono stati alcuni problemi all'interno del gruppo che si sono conclusi con l'abbandono del quarto integrante ormai giunti al termine del percorso. Questo avvenimento ha scombussolato l'andamento dei lavori perché io, in prima persona, mi sono trovata a dover lavorare su un codice sviluppato da terzi per poter completare la mia parte, senza però poter avere chiarimenti o aiuti nella comprensione del lavoro fatto, compito tutt'altro che agevole. Mi dispiace perché inizialmente il gruppo mi sembrava essere un buon ambiente di lavoro in cui c'era molta collaborazione e aiuto, ma questo è andato scemando man mano che ci si avvicinava al momento della consegna. Devo però dire che, tralasciato questo avvenimento, mi sono trovata molto bene con i restanti membri del gruppo, a mio parere molto disponibili e che ho reputato fondamentali nei momenti in cui niente sembrava funzionare. In conclusione posso affermare che nel complesso è stato un lavoro positivo che mi ha insegnato molto e mi ha fatto comprendere cosa significa davvero lavorare in gruppo.

3.2.3 Mirco Ventaloro

Al termine del progetto mi trovo soddisfatto del lavoro svolto sotto vari punti di vista. Avendo avuto diverse difficoltà durante le lezioni nella comprensione degli argomenti, metterli in pratica per risolvere dei problemi mi ha aiutato molto a sviluppare una conoscenza anche legata all'esperienza del lavoro. Questo è stato anche aiutato dal fatto che è stato necessario lavorare con altre persone e di conseguenza comunicare utilizzando termini specifici. I miei compagni si sono rivelati disponibili ad aiutare e a dare delucidazioni. Purtroppo questo però non sempre è stato possibile soprattutto dal momento in cui un membro del gruppo ha abbandonato il team di lavoro a progetto quasi completato. Mi è dispiaciuto e sembrato strano in quanto le motivazioni erano la consegna impossibile in breve termine, ma purtroppo essa è stata posticipata per via del fatto che la maggior parte dei membri del gruppo sono stati impegnati fino all'ultimo con la sessione d'esami non esattamente facile. Si è così creata una situazione non ottimale all'interno dell'ambiente di lavoro, in quanto noi membri restanti ci siamo trovato a dover comprendere e eventualmente modificare una parte di codice non scritta da noi senza poter ricevere alcuna delucidazione a riguardo. Personalmente ho trovato difficoltà e perplessità nel notare che il modo in cui sono state create le caratteristiche dei personaggi nella parte riguardante il menù di scelta nella modalità automatica era diverso da quello utilizzato da me nella parte dove i personaggi vengono creati. Si tratta soltanto di terminologia, ma avrei preferito utilizzare un linguaggio univoco in tutto lo sviluppo del codice. Oltre a questo gli altri membri del gruppo sono stati veramente bravi nel lavoro collettivo e nell'interazione tra di noi. Ho scoperto cosa significa sviluppare un progetto insieme ad un team e come poter porsi per migliorare anche l'esperienza degli altri all'interno dell'ambiente di lavoro.

Appendice A

Guida utente

Nel gioco “Indovina Chi” in modalità giocatore singolo l'obiettivo del giocatore sarà quello di indovinare il personaggio selezionato casualmente dal computer. Il giocatore potrà porre una domanda circa una caratteristica dell'aspetto del personaggio da indovinare: in base alla risposta, potrà eliminare tutti i personaggi che non si adattano alla caratteristica.

Il gioco può essere settato in due diverse modalità tramite un menù iniziale:

- Modalità automatica: le domande saranno fatte tramite un apposito menù al quale si accede per mezzo del pulsante “Fai una domanda”. Dopo aver fatto la domanda il giocatore dovrà cliccare il pulsante “Gira personaggi” per procedere con l’esclusione dei personaggi che non corrispondono alla caratteristica richiesta.
- Modalità manuale: le domande saranno fatte tramite un’apposita barra di ricerca alla quale si accede per mezzo del pulsante “Fai una domanda”. Dopo aver fatto la domanda e ricevuto la risposta il giocatore dovrà girare manualmente tutte le caselle che corrispondono ai personaggi che hanno o non hanno la caratteristica selezionata.

Il gioco può inoltre essere settato su tre diverse difficoltà (facile, intermedio, difficile): in base al livello scelto il tabellone sarà più o meno grande e quindi conterrà un diverso numero di personaggi. Inoltre la difficoltà determina i tentativi concessi al giocatore per indovinare il personaggio: chi indovinerà facendo meno domande vince!

Bibliografia

https://www.tes.com/teaching-resource/guess-who-powerpoint-game-12659873: immagini dei personaggi usati nel tabellone