Elaborato di Progetto -
Paradigmi di Programmazione e Sviluppo

Ruzzle

Matteo Mirisola
matteo.mirisola@studio.unibo.it
Marco Brighi
marco.brighi4@studio.unibo.it

12 marzo 2019

Indice

Capitolo 1

Processo di sviluppo

1.1 Metodologia di Sviluppo

La metodologia Agile rappresenta l'approccio adottato nell'intero processo di sviluppo. Più in dettaglio, è stato seguito un approccio Scrum-like con alcune piccole variazioni rispetto l'originale. Dato il numero ristretto dei componenti del gruppo, entrambi i membri sono posti allo stesso livello. L'unica eccezione è legata al ruolo di Product Owner, ricoperto da Marco Brighi data la conoscenza del progetto in questione.

In fase di setup del progetto, è stato delineato il Product Backlog nella quale sono contenuti tutti gli item che descrivono le principali macro attività da realizzare, ordinate secondo priorità. Ad ognuna di esse è associata una stima in termini di ore di lavoro necessarie. Ogni item è poi suddiviso in piccoli task assegnati ai membri del gruppo.

Il lavoro è stato suddiviso in una serie di sprint settimanali contenenti i task assegnati ad ogni membro del gruppo. Al termine di ogni sprint è eseguita una sprint review nella quale ogni componente condivide con gli altri il proprio lavoro. In particolare, si discute di eventuali problematiche riscontrate e possibili miglioramenti da apportare al progetto. Infine, si concordano i nuovi task da assegnare per lo sprint successivo. Naturalmente, ad ogni task è associato un valore che ne stima l'effort in termini di ore di lavoro.

Ogni sprint settimanale segue il modello di branching GitFlow. Ogni membro crea quindi il proprio branch feature all'interno del quale sono realizzati i singoli task assegnati. Infine, poco prima di eseguire lo sprint review, si effettua il merge sul branch principale e remoto condiviso da tutti.

Oltre all'approccio Agile, è stata adottata un'ulteriore metodologia di sviluppo, la Test Driven Development. Questa prevede che la costruzione del codice sia orientata completamente al superamento di test appositamente

predisposti. Questo permette inoltre di verificare molto semplicemente la correttezza del codice a seguito di una operazione di refactoring.
Per garantire una più flessibile gestione dei task, si è creata un'apposita bacheca su Trello. All'interno di essa, ogni componente, può organizzare autonomamente i suoi compiti e valutare così lo stato di avanzamento generale. Questo spazio risulta inoltre particolarmente utile in fase di sprint review poiché permette di rintracciare facilmente eventuali problematiche riscontrate.

1.2 Strumenti Utilizzati

I principali strumenti utilizzati durante l'intero processo di sviluppo sono:

- *git*, il lavoro di gruppo necessita dell'utilizzo di un sistema di versioning;
- *GitHub*, quale servizio di hosting nel quale creare il repository;
- *Travis CI*, quale servizio di continuous integration, abbinabile nativamente ai repository GitHub; permette di eseguire automaticamente compilazione e test dei sorgenti facilitando così lo sviluppo;
- *sbt*, quale sistema di build nativo per Scala assolutamente indispensabile per la gestione delle dipendenze di librerie esterne, esecuzione complessiva dei test e generazione della relativa documentazione di progetto.

Capitolo 2

Requisiti

Il progetto Ruzzle è nato con l'idea di sviluppare una versione in Scala del popolare gioco Ruzzle, con l'introduzione di piccole e dettagliate varianti. E' prevista la costruzione di una soluzione che sia distribuita e facilmente accessibile da piattaforme differenti.

Il gioco si presenta come una semplice matrice a due dimensioni, chiamata board, nella quale sono presenti caratteri appartenenti all'alfabeto della lingua inglese disposti casualmente. L'obiettivo dei giocatori è quello di scovare delle parole di senso compiuto, specificandone la categoria grammaticale, in un determinato lasso di tempo. Lo scopo finale è quindi quello di raccogliere il maggior numero di parole in modo tale da massimizzare il proprio score.

Il gioco può essere eseguito in due differenti modalità:

- single-player, l'utente gioca autonomamente senza confrontarsi con altri giocatori; il suo obiettivo è semplicemente quello di scovare parole per raggiungere il maggior punteggio possibile;

- multi-player, più utenti si sfidano alla ricerca di parole su un board comune; l'obiettivo consiste in questo caso nel catturare parole corrette che sono sfuggite agli altri avversari per guadagnare punti; al termine viene stilata una classifica contenente il punteggio raggiunto da tutti i giocatori.

Dalla descrizione precedente è evidente come la modalità multi-player prevede un vincolo aggiuntivo per i giocatori, quello della unicità delle parole scovate.

2.1 Requisiti Utente

Come in qualsiasi gioco, l'utente ricopre un ruole centrale ed essenziale per la buona riuscita del progetto.
Il requisito fondamentale per l'utente consiste nella capacità di registrarsi specificando un determinato username con il quale potrà essere contraddistinto.
Per quanto riguarda l'inizializzazione del gioco, l'utente deve poter scegliere se:

- iniziare una partita in solitaria;
- dare avvio ad una partita multi-player insieme ad altri;
- participare ad una partita multi-player già esistente.

Un altro aspetto essenziale riguarda la capacità dell'utente di regolare i parametri fondamentali della partita, come ad esempio il tempo di gioco concesso oppure il numero di giocatori che possono partecipare alla partita. Sempre all'interno di questo contesto, deve essere possibile modificare il sistema di punteggio utilizzato per determinare lo score finale del giocatore.
Infine, per poter valutare i proprio progressi, deve essere possibile accedere ad una classifica all'interno della quale sono salvati i punteggi ottenuti dall'utente nelle partite single-player giocate.
Tutti i requisiti precedentemente dettagliati sono sintetizzati in figura 2.1.

Figura 2.1: Diagramma dei casi d'uso UML.

2.2 Requisiti Funzionali

Dall'analisi del caso di studio, il progetto può essere suddiviso in diverse parti:

- gioco;
- multi-player;
- interfaccia utente;
- dizionario;
- sistema di punteggio;

2.2.1 Gioco

Per quanto riguarda il gioco, i requisiti derivano in gran parte dalle regole della versione originale del gioco stesso:

- il campo di gioco, detto anche board, è formato da una matrice 2D contentente caratteri facenti parte dell'alfabeto inglese, siano essi vocali o consonanti;

- l'inserimento dei caratteri nel board segue una strategia completamente casuale;
- ogni giocatore specifica il proprio username nel momento in cui crea o partecipa ad una nuova partita;
- il giocatore, entro un certo lasso di tempo, individua parole di senso compiuto, specificandone anche la tipologia grammaticale, sulla base dell'adiacenza dei caratteri presenti nel board;
- ogni parola può essere del tipo:
  - nome;
  - avverbio;
  - aggettivo;
  - verbo;
- ogni partita è caratterizzata da:
  - tempo;
  - la possibilità di incrementare il proprio punteggio sulla base della presenza di sinonimi;
  - il sistema di punteggio utilizzato;
- è possibile, da parte dell'utente, modificare tutti i parametri legati alla singola partita;
- la partita termina alla scadenza del tempo previsto;
- il punteggio dell'utente è dato dalla somma dei punteggi ottenuti dalle parole scovate e valide;
- il punteggio viene inserito all'interno della classifica generale, se rientra tra i primi 10 presenti.

2.2.2 Multi-player

Nella versione multi-player del gioco si assiste all'introduzione di nuove funzionalità richieste dal differente contesto:

- il creatore della partita, oltre alle classiche informazioni richieste, specifica il numero di giocatori che intende ospitare (<10);

- la partita ha inizio quando il numero di player richiesti sono collegati alla partita;
- affinché sia valida, una parola deve essere unica, ovvero deve essere scoperta da un solo giocatore;
- al termine, ogni giocatore visualizza la classifica con il punteggio proprio e dei suoi avversari;
- naturalmente è dichiarato vincitore chi presenta il punteggio maggiore.

2.2.3 Interfaccia Utente

Il compito principale dell'interfaccia utente è quello di visualizzare il board di gioco e tutti i suoi caratteri. Fornisce inoltre ai singoli utenti la capacità di inserire del testo rappresentante la parola scovata. Oltre a ciò, permette di selezionare la tipologia grammaticale della parola (nome, verbo, avverbio, aggettivo).
Naturalmente, alla creazione della partita, permette di editare i parametri ad essa connessa. Inoltre, attraverso apposite dialog, è mostrato, ed all'occorrenza modificato, l'intero sistema di punteggio del gioco che verrà utilizzare per definire lo score di ogni singola parola.
Infine, visualizza la classifica elencando, in ordine decrescente, il punteggio ottenuto da ogni singolo utente all'interno della partita.

2.2.4 Dizionario

Il dizionario è un componente dalle funzionalità essenziali per il ruolo che esso ricopre nel gioco. Data una parola, non deve far altro che chiarire se essa è presente nel dizionario della lingua inglese.

2.2.5 Sistema di Punteggio

Il sistema di punteggio ricopre un ruolo fondamentale nel determinare lo score da assegnare ad ogni parola valida scovata dal singolo utente.
E' basato sulle seguenti caratteristiche a cui sono associati i relativi valori di default:

- numero di caratteri (1pt per carattere);
- numero di vocali (2pt per vocale);
- numero di consonanti (1pt per consonante);

- numero di sinonimi presenti (se la funzionalità è abilitata dall'utente) (1pt per sinonimo);
- tipologia della parola (nome - 3pt, avverbio - 4p, verbo - 3pt, aggettivo - 4pt);

Prima della conclusione della partita, l'utente può modificare questo sistema di punteggio con altri valori. Nel caso di partita multi-player, è utilizzato il sistema di punteggio presente nel contesto del soggetto creatore.

2.3 Requisiti Non Funzionali

Nella modalità single-player i requisiti non funzionali richiesti sono:

- reattività, l'utente deve ottenere risposte dal sistema in tempi molto brevi rispetto ogni sua azione;
- performance, il gioco deve complessivamente scorrere con una certa fluidità.

Per quanto riguarda la modalità multi-player è invece richiesto:

- robustezza, nel caso in cui i singoli utenti abbiano problemi di qualsiasi genere, il gioco non deve finire in situazioni di blocco ma prevedere sempre una via d'uscita possibile;
- precisione, nonstante il contesto distribuito deve essere garantisco lo stesso lasso di tempo stabilito per ogni singolo giocatore.

2.4 Requisiti Implementativi

I requisiti implementativi legati allo sviluppo del progetto sono i seguenti:

- Scala, quale linguaggio di programmazione; attraverso questo linguaggio è infatti possibile sfruttare al massimo le caratteristiche dei paradigmi object-oriented e funzionale;
- Akka, quale framework ad attori in grado di fornire un substrato sul quale poi costruire lo scenario multi-player in un contesto distribuito;
- jar, il gioco, essendo costruito per la JVM, deve essere facilmente avviabile da un fat jar che contenga tutto il necessario.

Capitolo 3

Design Architetturale

Nella definizione ad alto livello dell'architettura si è sfruttato il modello VAT in grado di integrare il paradigma ad oggetti con la presenza di attori. Sulla base di questo approccio, l'intera logica del sistema è modellata sfruttando il classico paradigma object oriented. Il ruolo degli attori è invece quello di container di oggetti passivi.

Questa soluzione permette di realizzare un'elegante progettazione object oriented dei diversi componenti, integrandoli poi successivamente all'interno di attori-container.

La scelta di sfuttare il paradigma attori e le tecnologie ad esso collegate è legata alla possibilità di ususfruire di tutti quei vantaggi che il modello comporta. Tutte le problematiche dovute alla necessaria interazione all'interno di un sistema distribuito vengono così completamente demandante al framework utilizzato.

Inoltre, il modello ad attori fornisce un'astrazione, l'attore, particolarmente adatta per modellare la soluzione attuale.

3.1 Componenti del Sistema

Una visione ad alto livello dell'architettura è presente in figura 3.1. In quest'ultima sono infatti visibili i macro componenti della soluzione.

Gli attori sono identificabili dal relativo stereotipo. Le connessioni presenti tra di essi mostrano il flusso che segue lo scambio di messaggi.

Sulla base del modello VAT, precedentemente descritto, all'attore Game è associato un package contenente tutta la logica dell'applicazione strutturata secondo il paradigma object oriented.

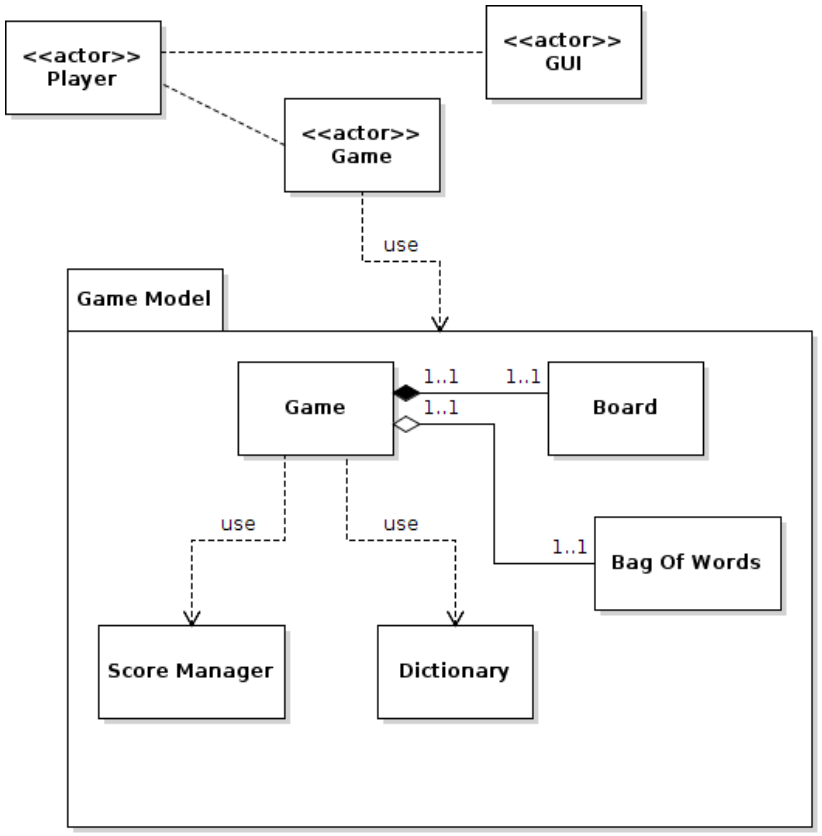

Figura 3.1: Architettura dell'applicazione.

3.1.1 Interazione

La comunicazione tra i 3 attori principali avviene seguendo le connessioni presenti in figura 3.1. Seguendo la logica applicativa del gioco, l'interazione dovrà occuparsi di:

- creazione della partita;
- gestione delle parole scovate dagli utenti;
- presentazione dei risultati.

Tutto ciò è mostrato in dettaglio all'interno della figura 3.2 che contiene un diagramma di sequenza. Osservandolo è inoltre possibile comprendere la logica con la quale gli attori vengono creati e sfruttati. In figura è mostrato il solo scenario single-player poichè la soluzione multi-player la si può semplicemente ottenere generalizzando la comunicazione verso più player.
Una volta terminato il tempo a disposizione, il gioco si conclude seguendo i passaggi raffigurati in figura 3.3.

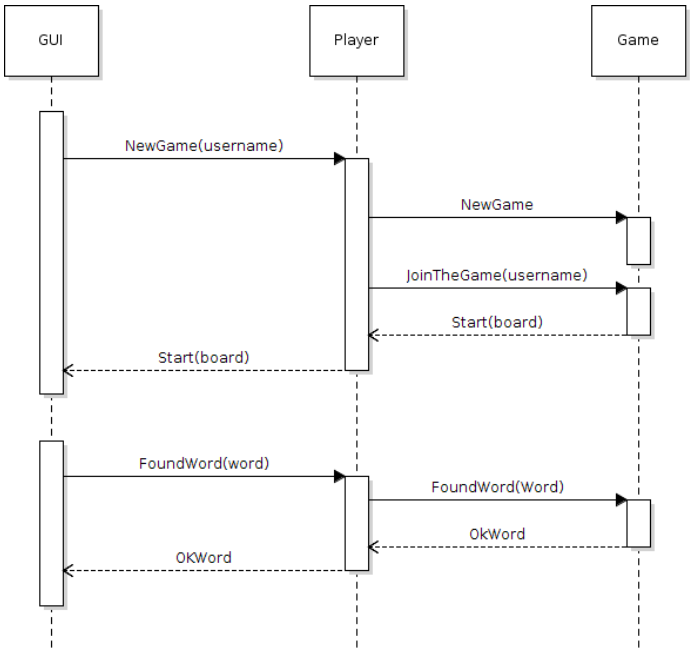

Figura 3.2: Diagramma di sequenza dell'applicazione.

3.1.2 Game Model

Il componente essenziale dell'architettura è certamente il Game Model. Come precedentemente descritto, al suo interno è presente tutta la logica di base dell'applicativo.

In particolare, ricopre un ruolo assolutamente centrale la classe Game. Attraverso quest'ultima si modella una singola partita del gioco, memorizzando al suo interno tutte le informazioni più importanti. Dato il legame di composizione presente, è evidente come sia essenziale per il singolo Game avere a disposizione il board dei caratteri su cui poi compiere tutte le verifiche necessarie sulle singole parole.

Bag Of Words rappresenta invece il contenitore delle parole trovate dai singoli giocatori che partecipano alla partita. Il componente è quindi particolarmente importante poichè memorizza le parole che andranno poi a definire lo score raggiunto dai giocatori.

Infine, Score Manager e Dictionary, rappresentano due elementi basilari per la verifica di correttezza della parola e la formulazione del punteggio associato.

In conclusione, la logica precedentemente illustrata determina lo svolgimento di una singola partita ed offre agli attori la possibilità di generare e scambiare messaggi sino a portare il gioco al suo termine.

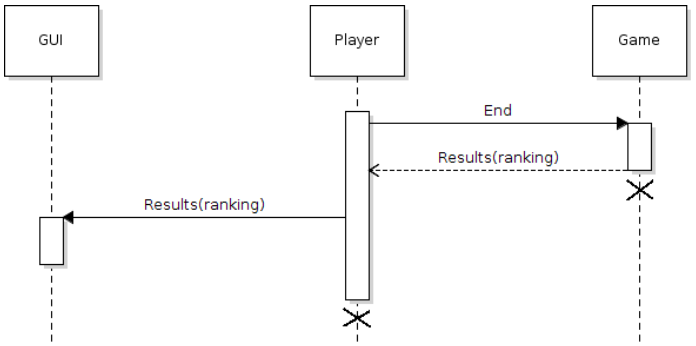

Figura 3.3: Diagramma di sequenza di fine partita.

3.2 Tecnologie

Di seguito sono elencate le tecnologie cruciali utilizzate nello sviluppo del progetto.

3.2.1 JavaFX

E' attualmente la libreria grafica più utilizzata nel mondo Java. Il principale vantaggio che questa soluzione offre è la forte separazione tra aspetti grafici e di controllo. Attraverso un apposito builder è possibile costruire l'interfaccia grafica descritta da uno specifico file con estensione fxml. Infine, attraverso la creazione di una specifica classe, è possibile costruirne la classe di controllo.

3.2.2 Akka

Akka rappresenta il più popolare framework ad attori disponibile in ambito JVM. E' costruito per essere utilizzato principalmente in Scala e questo rende il suo utilizzo particolarmente elegante. Le potenzialità di questa libreria sono enormi. Gestisce autonomamente alcuni degli aspetti più complessi legati ad un sistema distribuito, fornendo come principale astrazione l'attore. E' utilizzato sia per la modalità single-player sia per quella multi-player.

3.2.3 tuProlog

E' una delle più famose versioni JVM-based dell'engine Prolog. Permette di introdurre e sfruttare il paradigma di progettazione logica all'interno dell'applicazione. Il paradigma si presta particolarmente alla risoluzione di problemi di ricerca all'interno di un particolare spazio. TuProlog è utilizzato ampiamente nell'implementazione del board di gioco ed in particolare nella ricerca

di parole al suo interno. Nelle sezioni successive, il tema viene illustrato in maniera molto più dettagliata.

3.2.4 WordNet & Java Wordnet Interface

Come precedentemente illustrato, il dizionario ricopre un ruolo fondamentale all'interno del progetto. WordNet è un database semantico-lessicale per la lingua inglese creato dal linguista George Armitage Miller presso l'Università di Princeton. Si propone di organizzare, definire e descrivere i concetti espressi dai vocaboli. Contiene decine di migliaia di parole ed è utilizzabile liberamente. Java Wordnet Interface è una libreria Java che permette proprio di interfacciarsi con il dizionario WordNet. E' disponibile gratuitamente per qualsiasi tipo di scopo ed è mantenuta dal Massachusetts Institute of Technology (MIT).

Capitolo 4

Design di Dettaglio

In questo capitolo sono illustrate le scelte effettuate a livello di design di dettaglio. L'intera applicazione viene suddivisa in moduli e per ognuno di essi è fornita una spiegazione dettagliata rispetto le scelte effettuate ed i pattern utilizzati.

4.1 Organizzazione del Codice

Il codice è strutturato in 4 package che trattato determinati aspetti dell'applicazione:

- actors, è il package contenente il codice dei 3 attori principali e nel quale sono definiti i messaggi che essi si scambiano;
- prolog, è un package di utilità contenente tutti gli elementi per permettere una facile ed elegante integrazione tra tuProlog e Scala; è infatti fondamentale permettere una facile comunicazione con l'engine tuProlog poichè su di esso è basata l'implementazione del board;
- view, contiene tutto il codice di controllo dell'interfaccia grafica;
- model, rappresenta il package più importante poichè contiene l'intera logica del gioco.

4.2 Pattern Ricorrenti

Durante l'intero processo di sviluppo e integrazione sono stati impiegati diversi design pattern, tra questi alcuni si sono prestati meglio di altri ad un applicazione costante e ripetitiva.

Il pattern certamente più utilizzato, poichè si presta particolarmente bene all'uso con il linguaggio Scala, è il Factory Method. Per ogni classe si è quindi proceduto alla definizione del trait e del relativo companion object. Grazie a questo oggetto, tramite la chiamata ad opportuni metodi, è possibile costruire e richiede una specifica implementazione dell'interfaccia.
Ogni singola e specifica implementazione è quindi racchiusa all'interno del companion object ed inaccessibile dall'esterno.

4.3 Actors

Per quanto riguarda il package actors, la maggior parte delle informazioni sono già state illustrate in fase di design architetturale. All'interno dei singoli attori non è presente la logica applicativa ma essi si limitano esclusivamente a supportare, tramite lo scambio di messaggi, gli elementi di modello dell'applicazione, seguendo così la soluzione VAT. La comunicazione avviene sempre seguendo lo schema presente nelle sezioni precedenti. Le interazioni che intercorrono seguono sempre il percorso: GUI Actor -> Player Actor -> Game Actor oppure quello inverso. Questa soluzione permette di applicare adeguatamente il Single responsibility principle (SRP). Ai singoli attori sono infatti associati compiti e responsabilità ben precisi.
Nelle sezioni precedenti la robustezza è stata definita come un requisito non funzionale nel contesto multi-player dell'applicazione. Proprio per evitare situazioni di blocco, dovute a problematiche di rete, gli attori Player e Game compiono le loro operazioni all'interno di un interallo massimo di tempo, opportunamente settato in fase di creazione. Oltre la soglia, l'attore segnala l'errore di funzionamento permettendo così al gioco di superare la fase di stallo. Un esempio lampante di queste situazioni è rappresentato dall'improvvisa caduta del collegamento verso un player, durante lo svolgimento della partita. Senza alcun vincolo temporale, la partita è destinata a non concludersi mai nell'attesa di ricevere un messaggio di fine dal player disconnesso.
All'interno del package sono presenti le 3 classi relative ai 3 attori del gioco ed un sorgente contenente la definizione dei messaggi. Ogni messaggio ricopre uno specifico ruolo nel supportare la logica dell'applicazione.

4.4 View

Questo package contiene la classe adibita al controllo dell'interfaccia utente tramite il framework JavaFX. La vera e propria definizione della Graphical User Interface è invece contenuta in un file FXML. Quest'ultimo è un linguag-

gio di markup XML-based in grado di descrivere una qualunque interfaccia grafica. Rappresenta un'alternativa efficiente ad una costruzione procedurale e code-based ormai ampiamente abbandonata.

A seguito del suo caricamento, la vera e propria costruzione della GUI viene eseguita direttamente dal framework.

Come precedentemente dichiarato, tutte le operazioni che coinvologno l'interfaccia grafica, siano esse in entrata oppure in uscita, sono gestite e trattate dal GUI Actor.

4.5 Prolog

Questo package ricopre un ruolo essenziale nell'applicazione poichè si occupa di gestire l'intero engine Prolog, essenziale per il corretto funzionamento del board di gioco. Come precedentemente indicato, l'engine scelto per svolgere questo ruolo è quello fornito dalla libreria tuProlog, realizzata completamente in linguaggio Java. La complessità e l'insieme di funzionalità offerte da questa libreria risultano essere persino superiori a quelle richieste. Per questo motivo si è scelto di utilizzare il design pattern Adapter.

Questo pattern strutturale consiste nella costruzione di una nuova interfaccia (wrapper), tipicamente verso una libreria, in modo tale da renderla più semplicemente utilizzabile senza la necessità di riscrivere l'intero componente. In questo caso, l'obiettivo consiste nella costruzione di una più semplice API Scala-like che permetta di utilizzare l'engine logico fornito da tuProlog.

L'insieme delle classi principali costruite a questo scopo è presentato in figura 4.1. Nel diagramma sono infatti manipolati tutti i più semplici e basilari concetti della logica computazionale di Prolog: le costanti, le variabili e le liste. Tutti questi elementi possono essere quindi considerati dei termini logici.

Singoli termini logici possono poi essere utilizzati per comporre dei veri e propri predicati che, all'occorrenza, possono essere trattati come goal da risolvere. Per semplificare la costruzione di predicati si è scelto di utilizzare un altro famoso pattern, il Builder. E' quindi disponibile una classe che permette di costruire step-by-step il predicato, aggiundendo i termini necessari.

Infine, è presente il vero e proprio engine logico che ci permette di navigare lo spazio delle soluzioni. E' infatti possibile modificare la teoria corrente, inserendo nuovi fatti, oppure richiedere la risoluzione di un specifico goal. La risoluzione, se a buon termine, produrrà uno specifico solution set dal quale sarà poi possibile prelevare il valore delle singole variabili unificate.

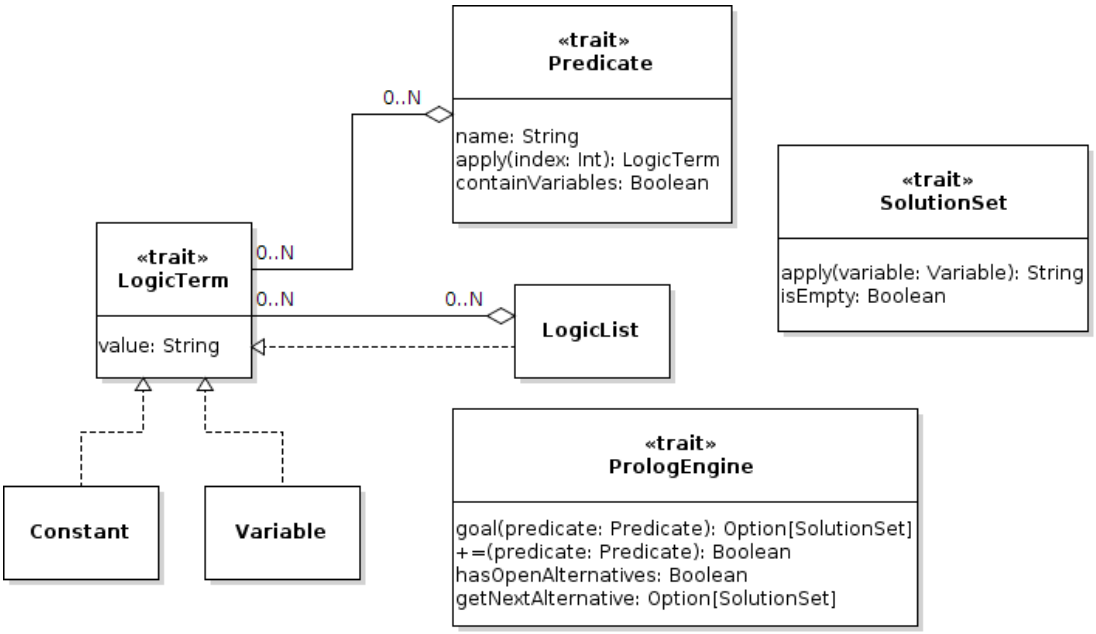

Figura 4.1: Diagramma delle classi relativo all'uso di tuProlog.

In conclusione, con poche e semplici classi, è possibile utilizzare una libreria complessa quale tuProlog per quanto riguarda tutte le funzionalità richieste dal progetto.

4.6 Model

Questo package rappresenta il vero e proprio cuore dell'intera applicazione. Al suo interno è racchiusa tutta la logica applicativa richiesta e sfruttata poi dai singoli attori.
L'obiettivo basilare del gioco consiste nel ricercare parole di senso compiuto all'interno del board. E' quindi essenziale modellare adeguatamente il concetto di parola e poterne valutare la validità lessicale attraverso un dizionario della lingua inglese. Gli elementi presentati dal diagramma in figura 4.2 permettono di fare proprio questo.
Come è facilmente intuibile, il tag rappresenta la tipologia grammaticale della parola come già descritto in fase di analisi dei requisiti.
Per quanto riguarda il dizionario, come già avvenuto per la libreria tuProlog, si è deciso di applicare il pattern Adapter. La libreria JWI, che permette di utilizzare il dizionario WordNet, risulta essere infatti non particolarmente semplice nell'utilizzo. Tutta la complessità è stata quindi nascosta all'inter-

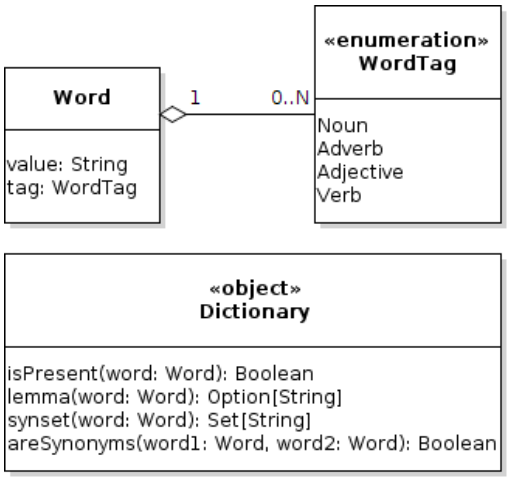

Figura 4.2: Diagramma delle classi che modella il dizionario e le parole.

no del relativo object. L'utilizzo di un object sottointende l'uso del design pattern Singleton.
Un elemento basilare per il funzionamento dell'applicativo è certamente il board di gioco. Le questioni essenziali per quest'ultimo riguardano le modalità di utilizzo e di costruzione. In un'ottica funzionale si è scelto di costruire il board attraverso delle funzioni generatrici dette generator. Queste non devono far altro che restituire, uno dopo l'altro, i caratteri in grado di comporre la matrice. Questo caso rispecchia inoltre l'applicazione di un altro design pattern estramamente comune, lo Strategy. In figura 4.3 è schematizzato l'elemento board.

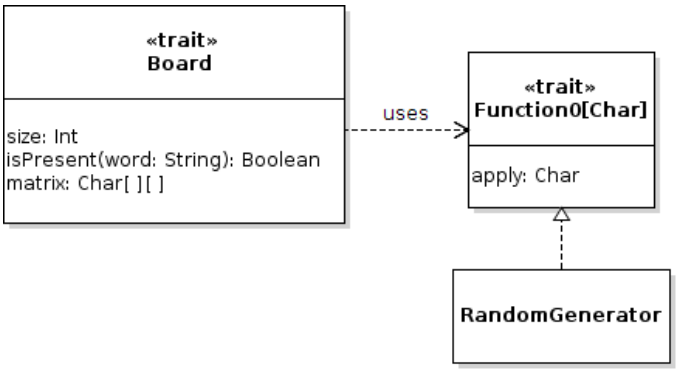

Figura 4.3: Diagramma delle classi che modella il board.

Come richiesto in fase di analisi è presente un generatore di caratteri appartenenti all'alfabeto della lingua inglese completamente casuale.

Dopo aver messo in campo questi elementi, è possibile fornire una rappresentazione estesa di quello che è il core dell'intera applicazione, come illustrato in figura 4.4.

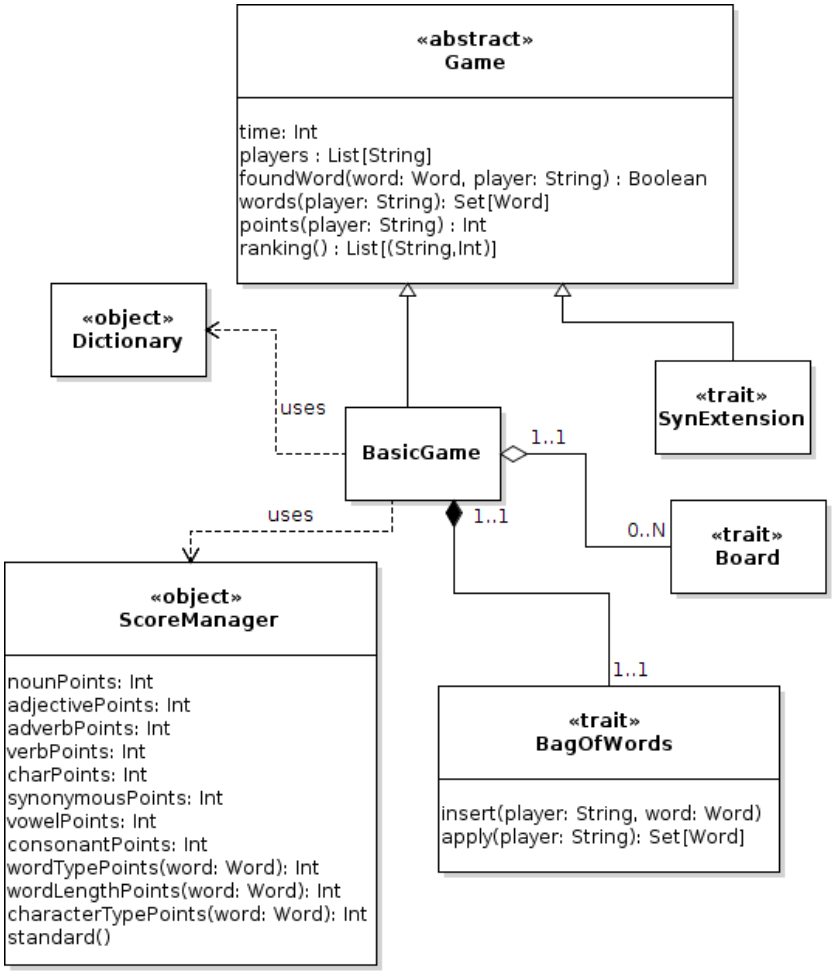

Figura 4.4: Diagramma delle classi dell'intero gioco.

Alcuni elementi sono già stati descritti precedentemente, come il dizionario oppure il board. Il funzionamento e l'utilizzo di questi è quindi già chiaro e conosciuto.
La Bag Of Words funge da semplice contenitore all'interno del quale memorizzare tutte le parole valide scelte dai singoli giocatori. Sarà suo compito, una volta terminato il gioco, restituire esclusivamente quei termini effettivamente validi per la costruzione dello score del singolo giocatore, ovvero quelli unici.

Lo Score Manager, realizzato come Singleton, è l'elemento depositario dei punteggi da assegnare ad ogni singolo termine. Dopo aver estratto le sole parole valide scovate da ogni singolo giocatore, queste vengono poi tradotte in punteggio sfruttando proprio questo componente. Al suo interno memorizza infatti il valore da assegnare ad ogni singola fattispecie riguardante le parole. Osservando l'intera struttura, è possibile notare come la soluzione sia stata strutturata in modo tale da essere arricchiata tramite mixins. La soluzione base può essere infatti integrata, secondo una logica stackable, attraverso la definizione di nuovi trait. In particolare, questo è avvenuto per quanto riguarda l'estensione del gioco che richiede la valutazione della presenza di sinonimi. Estensioni come questa, o simili, possono essere banalmente implementate andando ad agire sul metodo di formulazione dei punteggi.

Capitolo 5

Implementazione

All'interno di questo capitolo vengono analizzate e descritte le parti a cui ognuno dei componenti del gruppo ha contribuito. Ogni soggetto presenta un elenco esaustivo legato agli aspetti che lo hanno riguardato. Se necessario, alcuni di essi verranno poi descritti in dettaglio.

5.1 Brighi

Lo studente Marco Brighi si è occupato principalmente di implementare gli elementi appartententi al model dell'applicazione. In particolare si è occupato di:

- interfacciamento con tuProlog;
- costruzione dell'insieme di classi legate all'uso dell'engine logico;
- definizione ed implementazione del board di gioco;
- costruzione del generatore random per il board;
- interfacciamento con il dizionario WordNet;
- applicazione del pattern Adapter rispetto la libreria Java WordNet Interface;
- costruzione e modellazione delle parole;
- definizione ed implementazione dello Score Manager per la gestione dei punteggi;
- implementazione della Bag Of Words per memorizzare le parole dei singoli giocatori;

- costruzione dei relativi test rispetto tutto ciò che è stato direttamente sviluppato.

In linea si massima, nell'implementazione di tutti questi elementi, si è cercato di adottare un approccio funzionale alla risoluzione del problema sfruttando inoltre tutti i costrutti avanzati messi a disposizione da Scala. Si è fatto un largo uso della ricorsione e di tecniche quali gli impliciti. Tutte le strutture dati utilizzate sono immutabili.

5.1.1 Interfacciamento con tuProlog

Per un più semplice interfacciamento alla libreria tuProlog, si è scelto di utilizzare il pattern Adapter. Per una costruzione fluida e Scala-like di tutti gli elementi di base, si è fatto un grande uso di meccanismi quali gli impliciti. In particolare si è assunto che ogni Char rappresenti una variabile mentre gli altri dati di tipo primitivo rappresentino delle costanti. Sempre attraverso gli impliciti, le strutture List vengono automaticamente convertite in liste logiche per Prolog.
L'esempio mostrato in figura 5.1 mostra la semplicità con la quale è possibile definire un predicato, costruirlo tramite l'apposito builder e chiederne la risoluzione come goal.

```
val builder = PredicateBuilder("cell") += 1 += 1 += 'X'
val result = engine.goal(builder.create).get
val variableContent = result('X')
```

Figura 5.1: Esempio di risoluzione di un goal.

5.1.2 Implementazione del board di gioco

Come precedentemente descritto, il board di gioco è stato definito attraverso la costruzione di un teoria Prolog. L'uso della programmazione logica si presta particolarmente bene poichè l'unica operazione che deve essere svolta è una operazione di ricerca. Attraverso il predicato *cell* è quindi possibile definire le celle della matrice indicando i due indici ed il relativo carattere. A questo punto il board può dichiararsi costruito.
Per compiere adeguatamente un'operazione di ricerca è però necessario definire il concetto di vicinanza tra le singole celle. Tramite la costruzione di semplici regole è possibile esprimere logicamente questo concetto. Come mostrato in figura 5.2, le celle vicine sono semplicemente quelle presenti nell'ottetto circostante.

```
% Check if near a position there is an element K
% Input
% I: rowIndex, J: colIndex, K: value looking for
% Output
% X: rowIndex, Y: colIndex
near(I, J, X, Y, K) :- X is I - 1, Y is J - 1, cell(X, Y, K).
near(I, J, X, Y, K) :- X is I, Y is J - 1, cell(X, Y, K).
near(I, J, X, Y, K) :- X is I + 1, Y is J - 1, cell(X, Y, K).
near(I, J, X, Y, K) :- X is I - 1, Y is J, cell(X, Y, K).
near(I, J, X, Y, K) :- X is I + 1, Y is J, cell(X, Y, K).
near(I, J, X, Y, K) :- X is I - 1, Y is J + 1, cell(X, Y, K).
near(I, J, X, Y, K) :- X is I, Y is J + 1, cell(X, Y, K).
near(I, J, X, Y, K) :- X is I + 1, Y is J + 1, cell(X, Y, K).
```

Figura 5.2: Regole di vicinanza del board di gioco.

Attraverso questa modalità di implementazione risulta essere particolarmente semplice modificare le regole con le quali è possibile scovare parole all'interno del board di gioco. L'approccio logico-dichiarativo si dimostra quindi estremamente conciso ed efficace.
Infine, l'insieme delle classi presentate in fase di design di dettaglio non fanno altro che sfruttare la rappresentazione fornita dalla teoria logica.

5.1.3 Implementazione dello Score Manager

Anche per quanto riguarda lo Score Manager si è fatto uso del meccanismo degli impliciti. In particolare si è costruita una classe in grado di estendere l'insieme delle funzionalità standard del tipo Char. Sono stati aggiunti due nuovi metodi che permettono di conoscere se uno specifico carattere è una vocale oppure una consonante.
Questo risulta essere estremamente utili nell'ambito dell'implementazione delle funzioni di generazione dello score di una singola parola, semplificando il codice.

5.2 Mirisola

Lo studente Matteo Mirisola si è occupato principalmente di implementare gli elementi appartenenti alla view dell'applicazione. In particolare si è occupato di:

- creazione del progetto;
- configurazione Sbt, Git e Travis CI;
- interfacciamento con JavaFX SceneBuilder;

- costruzione del prototipo grafico;
- connessione interfaccia grafica-modello;
- costruzione del Game per la gestione del gioco e della modalità con i sinonimi;
- definizione ed implementazione del Ranking per la serializzazione dei dati e gestione del file della classifica;
- realizzazione delle varie form per le varie tipologie di partita;
- costruzione della richiesta inziale di username per l'utente;
- implementazione della modalità di modifica dei punteggi.

Per la creazione dell'interfaccia utente si è scelto di utilizzare il framework JavaFX. Il modello è stato realizzato attraverso SceneBuilder, il quale permette di effettuare un rapido design della view dell'applicazione senza scrivere codice. Il risultato è poi memorizzato all'interno di un apposito file con estensione FXML.
Per quanto riguarda la scrittura di codice si è cercato di usare un approccio funzionale alla risoluzione del problema sfruttando tutti i costrutti messi a disposizione da Scala, come ad esempio la tecnica del Mixins. Tutte le strutture dati utilizzate sono immutabili.

5.2.1 Interfacciamento Scala - FXML

Per poter gestire in maniera più semplificata la realizzazione della finestra principale dell'interfaccia grafica si è scelto di affidarsi a SceneBuilder, un costruttore di scene grafiche in grado di operare all'interno dell'ecosistema JavaFX. La costruzione della scena principale richiede la definizione di un file di layout FXML e di una classe che avvii l'intera applicazione.

```
val loaderDashboard: FXMLLoader = new FXMLLoader(getClass.getResource("/view/dashboardView.fxml"))
val sceneDashboard = new Scene(loaderDashboard.load())
primaryStage.setTitle("Ruzzle")
primaryStage.setScene(sceneDashboard)
primaryStage.show()
```

Figura 5.3: Esempio di creazione della scena con il layout definito.

È stato inoltre creato un controller aggiungendo alcuni campi ed alcuni metodi con la speciale annotazione "@FXML". Questa operazione è necessaria

al fine di avere accesso ai corrispettivi elementi privati del file FXML. Successivamente, l'applicazione popolerà automaticamente le variabili quando il file FXML sarà caricato.

5.2.2 Implementazione di Game

Game è la classe che ricopre un ruolo fondamentale per il gioco in quanto permette di modellare una partita. Al suo interno sono presenti il tempo di gioco, la lista dei giocatori ed il board su cui compiere tutte le operazioni. Per permettere l'estensione di una partita base con nuove caratteristiche, si è fatto uso della tecnica di Mixins. Attraverso questo approccio si è costruita la modalità di gioco che considera anche la presenza di sinonimi. Questa tecnica avanzata fornisce un maggior livello di flessibilità rispetto alla tradizionale ereditarietà singola

```
private class GameWithSinExtension(players : List[String], board : Board, time : Int)
   extends BasicGame(players, board, time) with SinExtension
```

Figura 5.4: Applicazione del mixins al Game di base.

5.2.3 Implementazione del Ranking

La classifica del gioco è stata realizzata attraverso l'implementazione dell'oggetto Ranking. Poichè queste informazioni devono essere mantenute e memorizzate nel tempo, si è fatto utilizzo della serializzazione.

Al primo utilizzo dell'oggetto, viene verificata la presenza del file contenente la classifica. Se necessario, quest'ultimo viene creato. Al termine di ogni singola partita single-player, il punteggio raggiunto dall'utente viene opportunamente aggiunto agli altri. Ogni punteggio è caratterizzato da una coppia di valori username - score. Con l'obiettivo di memorizzare esclusivamente punteggi significativi, la classifica garantisce persistenza ai soli primi dieci score ottenuti.

Per poter visualizzare adeguatamente i dati di classifica, username e score, si è dovuto creare una specifica classe, in linguaggio Java, chiamata Rank. Questa funge da contenitore di coppie username - score delle singole partite, normalmente memorizzate come tuple a 2 valori. Il suo utilizzo si è reso obbligatorio per questioni di compatibilità rispetto il framework JavaFX.

5.3 Componenti creati congiuntamente

Le operazioni che hanno coinvolto entrambi i membri del team sono quelle legate a scelte di progettazione. In particolare la scelta del design architetturale e della modalità d'uso del modello ad attori sono state concordate dalle due parti. Inoltre in fase di start up del progetto, si è convenuto su determinate scelte stilistiche legate alla costruzione del codice.

Capitolo 6

Retrospettiva

Nel complesso, lo sviluppo del processo si è svolto senza particolari intoppi o problematiche.
L'utilizzo del modello Test Driven Development (TDD) si è dimostrato una soluzione vincente ed in grado di creare codice sufficientemente robusto. La combinazione di quest'ultimo con tool automatici, quali sbt, ha reso il tutto pratico ed efficiente. Più di una volta, la presenza di test ha permesso di individuare tempestivamente bug sorti da piccoli cambiamenti nel codice altrimenti difficilmente catturabili.
Anche l'approccio Agile Scrum-based si è rilevato una buona scelta. In particolare, la presenza di una review settimanale ha permesso di risolvere tempestivamente eventuali problematiche che sarebbereo inevitablmente sorte legate all'integrazione delle diverse parti. La problematica più forte riscontrata è stata la determinazione di un corretto effort rispetto i singoli task. Già dalla prime settimane, le previsioni realizzate si sono dimostrate decisamente inattendibili. Procedendo nel lavoro anche questo aspetto è stato poi corretto ed aggiustato.
In conclusione, possiamo affermare che tutti i requisiti del progetto sono stati completati. L'unica eccezione rigurda la feature opzionale legata alla costruzione di una chat durante le partite.