MasterMind

Release 1.0.0

Francesco Pio Stelluti, Francesco Coppola

30 giu 2019

Indice

> *«Our education might stop, if we so choose. Our brains' never does. The brain will keep reacting to how we decide to use it. The difference is not whether or not we learn, but what and how we learn.»*
>
> *Maria Konnikova*[1]

All'interno delle seguenti pagine sarà possibile trovare la documentazione generata per il progetto **MasterMind**, realizzato per il corso di *Programmazione Avanzata* dell'anno 2018/2019.

Lo sviluppo di tale codice è da attribuire interamente agli studenti **Francesco Pio Stelluti** e **Francesco Coppola**.

[1] Maria Konnikova, Mastermind: How to Think Like Sherlock Holmes

CAPITOLO 1

Introduzione

Il progetto è stato indirizzato ad all'implementazione tramite linguaggio **Java** del gioco da tavolo **Mastermind**[1].

Nell'ideare la struttura del progetto si è puntato alla **massima modularità possibile**, per quanto non totale, ottenuta tramite l'applicazione di determinati design pattern.

1.1 Architettura fondamentale del progetto

L'avvio del programma è delegato ad una classe che estende `MainManager`, classe astratta contenente il funzionamento effettivo e a più alto livello del programma. La particolare estensione di tale classe è delegata a definire quali implementazioni delle classi `GameViewFactory` e `StartView` si è scelto di impiegare.

Le classi `GameViewFactory` e `StartView` sono fondamentali in quanto estendibili con classi mirate a fornire delle viste finalizzate all'interazione con gli **utenti fisici**. Il funzionamento di `MainManager` si basa sulla creazione, esecuzione e monitoraggio di istanze personalizzate di `SingleMatch`, rappresentanti singole partite di gioco.

La corrente implementazione di `MainManager` consente la gestione di una singola istanza di `SingleMatch` alla volta. All'interno dell'esecuzione effettiva del metodo di avvio presente in `SingleMatch` si ha poi l'interazione di due entità rappresentanti i giocatori, rispettivamente un `CodeMaker` *(colui che definisce la sequenza di ColorPegs da indovinare)* e un `CodeBreaker` *(colui che definisce sequenze di ColorPegs valide come tentativi)*, con l'entità `BoardController`, attraverso la quale viene aggiornata un'istanza di `BoardModel` *(rappresentante una plancia di gioco)*.

Lo svolgimento di un `SingleMatch` si conclude quando si è arrivati ad una delle tre condizioni di vittoria, rappresentate dalla sconfitta del `CodeBreaker` a causa di una sua resa o per l'esaurimento dei tentativi disponibili e dalla sconfitta del `CodeMaker` a causa della definizione di una corretta sequenza tentativa da parte del `CodeBreaker`. L'interazione con l'utente fisico all'interno del programma è svolta da istanze estensione di `StartView` *(mirate alla fase di preparazione dei singoli match)* e da istanze estensione di `GameView` *(mirate alla gestione delle azioni da eseguire durante i match)*.

[1] Mastermind

1.2 Estendibilità ed implementazioni fornite di default

L'estendibilità del progetto si sostanzia nella possibilità di definire nuove implementazioni per le seguenti responsabilità:

- **Gestione dell'avvio e del monitoraggio delle singole partite**, rappresentata da `MainManager`.
- **Gestione dell'interazione con l'utente fisico per l'avvio di nuove partite**, rappresentata da `StartView`.
- **Gestione dell'interazione con l'utente fisico per la gestione delle azioni all'interno di singole partite**, rappresentata da `GameView`.
- **Fornire istanze di implementazioni di GameView**, rapprsentata da `GameViewFactory`.
- **Rappresentazione di un giocatore che decide la sequenza da indovinare**, rappresentata da `CodeMaker`.
- **Fornire istanze di implementazioni di CodeMaker**, rappresentata da `MakerFactory`.
- **Rappresentazione di un giocatore che cerca di indovinare la sequenza**, rappresentata da `CodeBreaker`.
- **Fornire istanze di implementazioni di CodeBreaker**, rappresentata da `BreakerFactory`.

Esempi di implementazioni già incluse nella release attuale del progetto sono:

- **ConsoleMainManager**, ad estensione di `MainManager`.
- **ConsoleStartView**, implementazione di `StartView`.
- **ConsoleGameView**, estensione di `GameView`.
- **ConsoleGameViewFactory**, implementazione di `GameViewFactory`.
- **InteractiveMaker, RandomBotMaker**, estensioni di `CodeMaker`.
- **InteractiveMakerFactory, RandomBotMakerFactory**, implementazioni di `MakerFactory`.
- **InteractiveBreaker, RandomBotBreaker, DonaldKnuthBreaker**, estensioni di `CodeBreaker`.
- **InteractiveBreakerFactory, RandomBotBreakerFactory, DonaldKnuthBreakerFactory**, estensioni di `BreakerFactory`.

Per ulteriori informazioni circa le classi elencate si rimanda alle relative *sezioni*.

1.3 Informazioni fondamentali circa il primo avvio

Il caricamento a **runtime** delle informazioni relative alle classi factory, grazie alle quali ottenere istanze di classi che estendono `CodeBreaker` e `CodeMaker`, è stato reso possibile grazie alla definizione di classi implementazione `PlayerFactoryRegistry`, classi le cui istanze sono indirizzate alla lettura a runtime di file di input e al caricamento di istanze di `BreakerFactory` e `MakerFactory`. Il formato delle informazioni di tali file di input è molto importante ed in loro assenza ne vengono generati automaticamente altri *(all'interno della cartella GameResources)* contenenti le istruzioni necessarie per un corretto avvio del programma. Il caricamento a runtime di tali informazioni permette l'aggiunta di nuove funzionalità del programma, nei limiti di estendibilità già trattati, senza avere la necessità di ricompilare tutte le classi del progetto.

Si rimanda alle *sezioni* per ulteriori informazioni circa le implementazioni di `PlayerFactoryRegistry` fornite.

1.4 Responsabilità delle classi

Si rimanda alle *sezioni* riguardanti le implementazioni delle singole classi per ulteriori informazioni.

1.5 Design pattern impiegati

1. **Model View Controller**[2] Rappresenta la struttura alla base del funzionamento delle singole partite. È stata implementata tramite le classi `GameView`, `BoardModel` e `BoardCoordinator`, classi le cui istanze comunicano all'interno di `SingleMatch`.

2. **Observer**[3] Implementato fornendo come classe da osservare `BoardModel` e come classi che osservano `GameView` e `MatchState`, classi estensione di `BoardObserver`. Dalla versione 9 di Java l'interfaccia Observer, pensata nell'ottica di questo design pattern, risulta deprecata. La sua implementazione all'interno di questo progetto è quindi da vedere in un'ottica puramente accademica e finalizzata all'apprendimento del concetto alla base del pattern.

3. **Singleton**[4] Presente all'interno della classe `ConsoleStartView`, esso garantisce che siano presenti **singole** istanze di tali classe all'interno del progetto.

4. **Factory**[5] Implementato tramite le classi `PlayerFactory`, `MakerFactory`, `BreakerFactory` e le loro implementazioni per poter fornire istanze di giocatori `CodeMaker` e `CodeBreaker`. Lo stesso pattern è stato inoltre implementato con `GameViewFactory` per poter fornire istanze di `GameView` all'inizializzazione dei vari `SingleMatch`.

1.6 Testing

Sono stati ideati dei test, scritti sotto ambiente **JUnit 5**[6], per poter testare in modo mirato le singole *funzionalità* del progetto.

Per ulteriori informazioni si rimanda alle *sezioni* riguardanti le implementazioni di tali test.

1.7 Gradle

Nell'ottica di garantire continuità al progetto si è deciso anche di implementare il tool di building **Gradle**[7], in versione 5.4.1, per facilitare il deploy e la distribuzione di tale software all'interno di altri sistemi.

1.8 Continuous Integration

> *La Continuous Integration, proprio come la Continuous Delivery, viene apprezzata soprattutto nello sviluppo agile di software. L'obiettivo di questo moderno metodo è quello di suddividere il lavoro in porzioni più piccole per rendere il processo stesso di sviluppo più efficiente e poter reagire con maggiore flessibilità alle modifiche. La Continuous Integration è stata nominata per la prima volta nella descrizione della metodologia agile Extreme Programming di Kent Beck.*

Mediante l'implementazione di **Gradle**, illustrata in precedenza, si è riuscito a integrare all'interno della natura del progetto anche il software **Travis CI**[8].

[2] MVC
[3] Observer
[4] Singleton
[5] Factory
[6] JUnit
[7] Gradle
[8] Travis CI

Quest'ultimo garantisce all'intero progetto la possibilità di sviluppare una **integrazione continua** all'interno di un team di lavoro in primo luogo, *e di consegunza*, una seria di vantaggi non indifferenti, quali:

- **Resa del build auto-testante**
  - Ogni volta che il codice sorgente viene buildato ed impacchettato vengono eseguiti dei test sul sorgente affinché la qualità del codice venga tenuta sotto controllo ed eventuali bug vengano scoperti il prima possibile.
- **Ogni commit lancia una build**
  - Ogni modifica al codice sorgente condiviso potrebbe generare dei bug e quindi compilare e testare subito dà la possibilità di intervenire immediatamente su eventuali falle del sistema.
- **Esecuzione di test in un clone dell'ambiente di produzione**
  - L'ambiente di lavoro può differire in base all'OS adottato e dal hardware stesso della macchina che si adopera, per questo è fondamentale creare un clone del *workspace* che sia il medesimo per tutti i membri del progetto e incontro a tale evenienza viene in aiuto **Docker**.
- **Repository del codice sorgente**
  - Questo elemento è propedeutico a tutti gli altri principi descritti in precedenza, poichè senza avere un repository del codice è impossibile automatizzare il build ed i test.

Aver inserito anche una *feature* come quella del **CI** rende sicuramente l'intero parco software **robusto**, **elegante** e **flessibile**.

CAPITOLO 2

Guida al gioco

Mastermind o *Master Mind* è un gioco da tavolo astratto di crittoanalisi[1] per due giocatori, in cui un giocatore, il **«decodificatore»**, deve indovinare il codice segreto composto dal suo avversario, detto **«codificatore»**.

2.1 Regolamento

Nella versione **originale** di Mastermind, il codice segreto è di quattro cifre e il codificatore ha a disposizione, per comporlo, le dieci cifre del sistema decimale standard `(0,1,2,3,4,5,6,7,8,9)`.

Esistono numerose versioni successive, **la più famosa** è quella in cui al posto dei numeri si usano dei pioli di 6 colori differenti. Tale implementazione infatti è quella fornita all'interno del progetto proposto.

Dopo che il codificatore ha composto il codice, il decodificatore fa il suo primo tentativo, cercando di indovinare il codice. Il codificatore, appena il suo avversario ha completato il tentativo, fornisce degli aiuti comunicando:

- **Il numero di cifre giuste al posto giusto**, cioè le cifre del tentativo che sono effettivamente presenti nel codice al posto tentato, **con pioli neri**.
- **Il numero di cifre giuste al posto sbagliato**, cioè le cifre del tentativo che sono effettivamente presenti nel codice, ma non al posto tentato, **con pioli bianchi**.

Nella versione di gioco prodotta **la lunghezza della sequenza** da inserire può essere selezionata dall'utente, così come il **numero di tentativi disponibili** per provare ad indovinare il codice segreto.

Se il decodificatore riesce ad indovinare il codice entro il numero di tentativi predeterminati, *che nel caso di default sono pari a 9*, allora quest'ultimo vince la partita, altrimenti vince il codificatore.

2.2 Singola partita

Avviando il gioco si avrà accesso alla seguente schermata:

[1] *Per crittoanalisi (dal greco kryptós, «nascosto», e analýein, «scomporre»), o crittanalisi, si intende lo studio dei metodi per ottenere il significato di informazioni cifrate senza avere accesso all'informazione segreta che è di solito richiesta per effettuare l'operazione.*

Da quest'ultima si potrà scegliere quale giocatore selezionare per il effettuare il ruolo di **codificatore**. Una volta inserito il valore desiderato sarà possibile selezionare il giocatore **decodificatore** all'interno di tale interfaccia:

Come è possibile osservare nelle immagini precedenti il progetto ammette anche l'utilizzo di un **codificatore** e di un **decodificatore** aventi sembianze artificiali, ovvero controllati da *puri* e *meri* algoritmi matematici.

Ovviamente è possibile anche effettuare partite mediante il solo utilizzo di giocatori di natura **interactive**, cioè controllati da classici *player* umani.

È interessante inoltre notare come il parco software prodotto metta anche a disposizione un algoritmo di risoluzione del **Mastermind** più avanzato. Quest'ultimo prende il nome da un noto informatico statunitense **Donald Knuth**[2].

[2] *Knuth è appunto considerato il padre del campo di studio che studia in maniera rigorosa la parte algoritmica della teoria della complessità e ha dato fondamentali contributi in svariati rami dell'informatica teorica. Ha contribuito infatti con la sua analisi comparativa dei due algoritmi usati («first fit» e «best fit») per la frammentazione esterna della memoria segmentata dei calcolatori, dimostrando che l'algoritmo «first fit» risulta essere migliore in termini di prestazioni complessive rispetto al «best fit».*

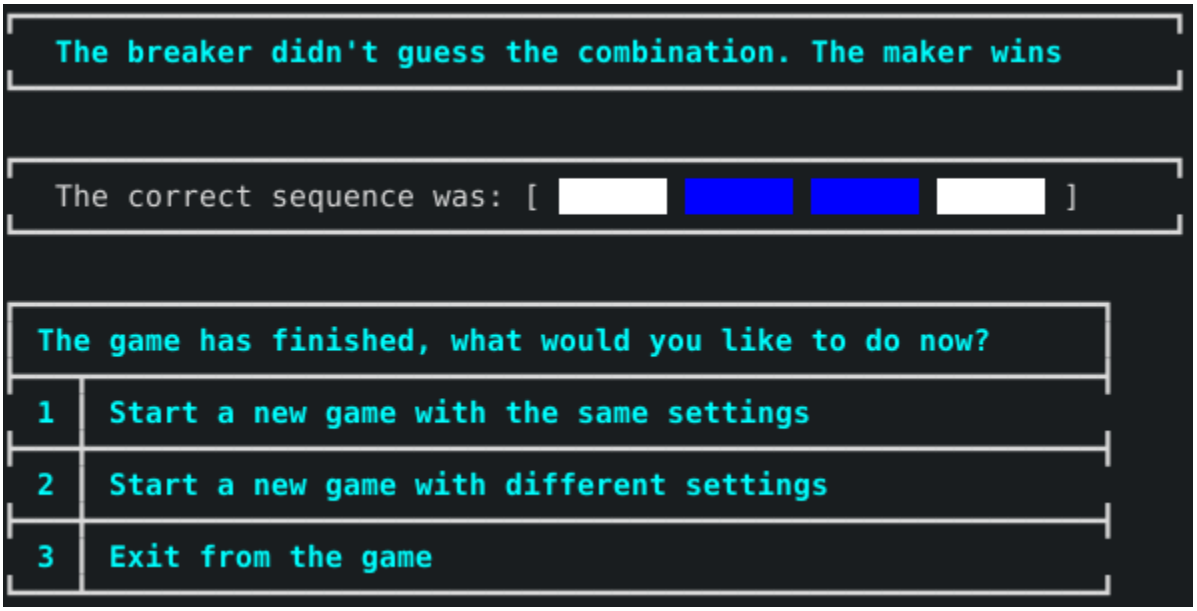
Esso afferma infatti di risolvere una classica partita con un numero di mosse *minori o pari* a 5, grazie ad un rubusto algoritmo che basa la sua **potenza computazionale** sugli indizi forniti e su un ampio numero di combinazioni possibili che genera a priori.

Una volta selezionati i *players* con i quali si vuole disputare la partita sarà possibile accedere alle impostazioni di gioco con le quali si desidera giocare, ovvero la lunghezza della sequenza segreto e il numero di tentativi messi a disposizione.

Esempio: seguendo la figura qui di seguito si avranno a disposizione 9 tentativi e la lunghezza segreta da indovinare avrà una lunghezza pari a 4 caselle colorate

La tabella di gioco avrà un aspetto di questa natura e sarà possibile andare a inserire il colore desiderato mediante dei numeri i quali rappresentano proprio quest'utlimi, *come è possibile infatti evincere dalla legenda riportata*:

La conclusione di una singola partita invece avrà un aspetto di questo tipo:



Come è possibile osservare una volta terminato un match l'utente avrà di fronte a se tre opzioni fondamentali:

1) Iniziare un nuovo match con le medesime impostazioni del precedente
2) Iniziare un nuovo match con un set di impostazioni differente dal precedente e quindi *settabile* nuovamente
3) Uscire dal gioco definitivamente

2.3 Struttura dell'interfaccia

L'interfaccia grafica usufrubile da console è stata realizzata utilizzando unicamente caratteri di tipo **UNICODE**[3] e decodifica **ANSI**[4].

Il primo è stato fondamentale per la creazione dei vari **box** contenenti le varie informazioni riportate nel gioco e soprattutto per la creazione delle **tabelle dinamiche**, le quali contengono i valori inseriti e gli indizi autogenerati.

Il secondo invece è stato necessario **per utilizzare i colori con i quali l'utente può interagire** e per rendere meno monotona l'interfaccia di gioco, **colorando** diversi contenuti all'interno dei vari menù presenti all'interno del gioco.

[3] *Unicode è un sistema di codifica che assegna un numero univoco ad ogni carattere usato per la scrittura di testi, in maniera indipendente dalla lingua, dalla piattaforma informatica e dal programma utilizzato.*

[4] *Creato nel 1918 con sede a New York (1430 Broadway) questo istituto privato senza fini di lucro raccoglie oltre 1300 aziende (tra cui tutti i principali fornitori di personal computer) che cooperano alla definizione e alla pubblicazione di standard facoltativi per il mondo dellinformatica e delle comunicazioni.*

CAPITOLO 3

Sphinx e le sue potenzialità

L'intera documentazione generata della quale si sta usufruendo è frutto dell'unione tra Sphinx e JavaDoc, due strumenti dedicati alla generazione di testi a partire da del mero e puro codice.

3.1 Strumenti con cui è stata realizzata

Solitamente per documentare in maniera *raffinata* un progetto **Java** viene utilizzato lo strumento fornito dall'IDE di sviluppo stesso **JavaDoc**[1].

Esso offre degli incredibili vantaggi, come la facilità d'utilizzo e soprattutto un layout ben noto all'interno della community dei developers Java che permette di trovare informazioni in maniera decisamente veloce.

La pecca più grande di tale strumento però resta la datazione dei vari stili che compongono i file CSS e l'assenza di un'eleganza generale complessiva.

Per risolvere tale mancanza quindi si è pensato di ricorrere a **Sphinx**[2].

Poi mediante l'utilizzo di un'estensione nominata `javasphinx`[3] è stato possibile convertire i vari commenti **JavaDoc** secondo lo standard perseguito da Sphinx stesso, e così facendo abbiamo ottenuto sia una documentazione piacevole per la vista che facile ed intutiva da poter seguire.

3.2 Autogenerazione della sintassi convertita da JavaDoc a Sphinx

Per fare questa operazione è necessario innanzitutto installare `javasphinx` sulla propria macchina, attraverso il seguente comando:

```
$ pip install javasphinx
```

[1] *Javadoc è un applicativo incluso nel Java Development Kit della Sun Microsystems, utilizzato per la generazione automatica della documentazione del codice sorgente scritto in linguaggio Java*

[2] *Software Open Source per l'autogenerazione di documentazioni a partire da un codice sorgente generico*

[3] Javasphinx

Una volta effettuato ciò sarà necessario inserire l'estensione `javasphinx` appena installata nel file `conf.py` generato da Sphinx.

A questo bisognerà definire lo standard Java da seguire, all'interno del file `conf.py`, nel seguente modo:

```
javadoc_url_map = { '<namespace_here>' :  ('<base_url_here>', 'javadoc') }
```

Arrivati a questo punto basterà lanciare il comando:

```
$ javasphinx-apidoc -o docs/source/ --title='<name_here>' ../path/to/java_dirtoscan
```

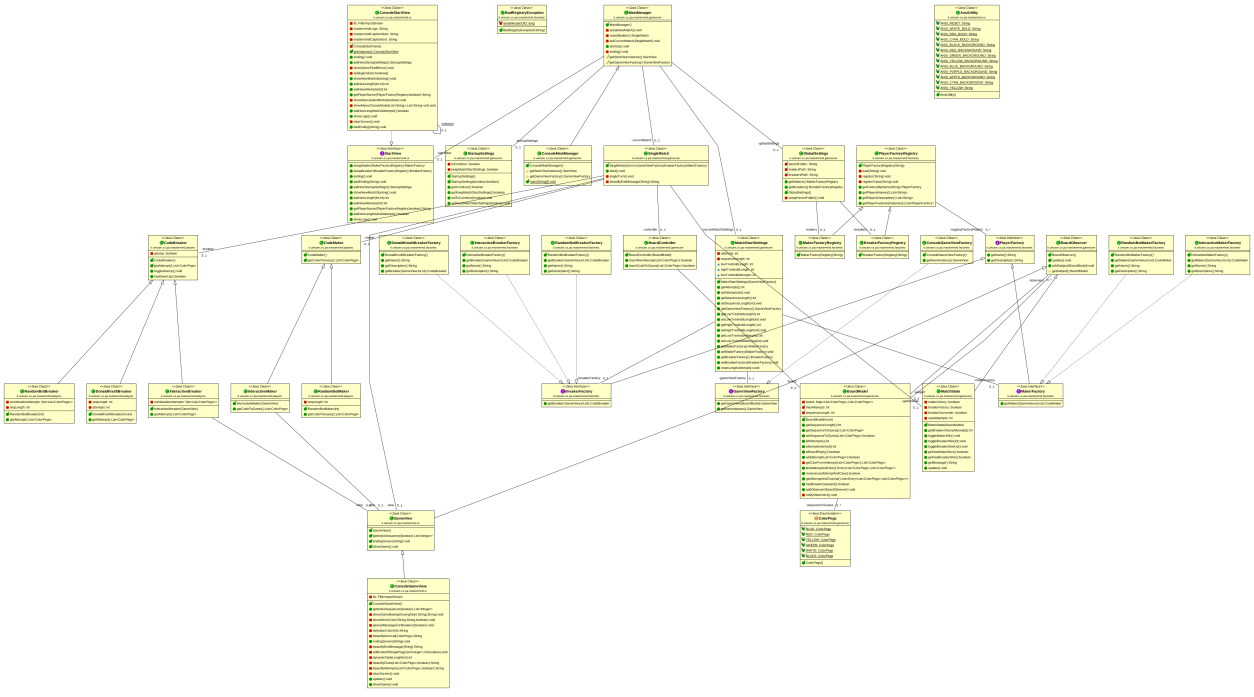
La documentazione quindi sarà pronta per essere usata nei vari file con estensione `.rst` che, attraverso il comando `make`, diventaranno file `.html`.

CAPITOLO 4

Documentazione del codice

Nella seguente pagina sarà possibile accedere alle informazioni che descrivono in maniera *dettagliata* ogni **classe** ed ogni **package** appartenente al parco software prodotto.

Per rendere più chiara la composizione della struttura del progetto, e quindi comprendere più dettagliatamente quello che è stato realizzato, abbiamo reso disponibile un **diagramma UML** il quale è possibile visualizzare qui di seguito.



4.1 it.unicam.cs.pa.mastermind.factories

Il package contiene le varie factory che hanno il compito di generare nuovi player durante il processo di esecuzione in maniera dinamica ed efficiente. All'interno del package sono contenute anche le classi che hanno la funzione di registro per tenere traccia di tali classi factory.

4.1.1 BadRegistryException

public class **`BadRegistryException`** extends Exception

Eccezione personalizzata impiegata in tutti quei casi in cui ci sia stato un problema nell'inizializzazione di istanze di `PlayerFactoryRegistry`.

Author Francesco Pio Stelluti, Francesco Coppola

Constructors

BadRegistryException

public **`BadRegistryException`**(String *message*)

4.1.2 BreakerFactory

public interface **`BreakerFactory`** extends *PlayerFactory*

Responsabilità: fornire istanze di implementazioni di `CodeBreaker`. Interfaccia finalizzata all'implementazione di classi factory per le particolari implementazioni dei giocatori `CodeBreaker`.

Author Francesco Pio Stelluti, Francesco Coppola

Methods

getBreaker

public *CodeBreaker* **`getBreaker`**(*GameView* *view*, int *seqLength*, int *attempts*)

Ottenimento di un'istanza di un giocatore `CodeBreaker`.

Parametri

- **`view`** – vista per l'interazione con l'utente fisico
- **`seqLength`** – lunghezza della sequenza di `ColorPegs` da trattare
- **`attempts`** – numero di tentativi per vincere il gioco

Ritorna CodeBreaker istanza di un giocatore `CodeBreaker`

4.1.3 BreakerFactoryRegistry

public class **`BreakerFactoryRegistry`** extends *PlayerFactoryRegistry*

Estensione di `PlayerFactoryRegistry` per poter contenere informazioni circa le implementazioni di `BreakerFactory`.

Author Francesco Pio Stelluti, Francesco Coppola

Constructors

BreakerFactoryRegistry

public **BreakerFactoryRegistry**(String *path*)

Parametri

- **path** – associato al file da cui recuperare le informazioni sulle classi da caricare dinamicamente

Solleva

- **BadRegistryException** – in caso le istanze caricate non siano appartenenti a classi implementazione di `BreakerFactory`

4.1.4 ConsoleGameViewFactory

public class **ConsoleGameViewFactory** implements *GameViewFactory*

Classe factory estensione di `GameViewFactory` impiegata per ottenere istanze di `ConsoleGameView`.

Author Francesco Pio Stelluti, Francesco Coppola

Methods

getNewInstance

public *GameView* **getNewInstance**()

4.1.5 DonaldKnuthBreakerFactory

public class **DonaldKnuthBreakerFactory** implements *BreakerFactory*

Classe factory implementazione di `BreakerFactory` impiegata per ottenere istanze di `DonaldKnuthBreaker`.

Author Francesco Pio Stelluti, Francesco Coppola

Methods

getBreaker

public *CodeBreaker* **getBreaker**(*GameView* *view*, int *seqLength*, int *attempts*)

getDescription

public String **getDescription**()

getName

public String **getName**()

4.1.6 GameViewFactory

public interface **GameViewFactory**

Interfaccia finalizzata all'implementazione di classi factory per le particolari implementazioni della vista `GameView`.

Author Francesco Pio Stelluti, Francesco Coppola

Methods

getGameView

public *GameView* **getGameView**(*BoardModel subject*)

Ottenimento di un'istanza di una vista `GameView`. **Contratto**: il metodo deve avere come argomento un riferimento ad un oggetto BoardModel in quanto l'istanza restituita appartiene al pattern **Observer** in cui è coinvolto BoardModel.

Parametri

- **subject** – l'istanza fondamentale per il pattern **Observer**

Ritorna GameView istanza richiesta

getNewInstance

public *GameView* **getNewInstance**()

Ritorna GameView nuova istanza di `GameView` a cui non è stato aggiunto il soggetto da osservare secondo il pattern **Observer**

4.1.7 InteractiveBreakerFactory

public class **InteractiveBreakerFactory** implements *BreakerFactory*

Classe factory implementazione di `BreakerFactory` impiegata per ottenere istanze di `InteractiveBreaker`.

Author Francesco Pio Stelluti, Francesco Coppola

Methods

getBreaker

public *CodeBreaker* **getBreaker**(*GameView view*, int *seqLength*, int *attempts*)

getDescription

public String **getDescription**()

getName

public String **getName**()

4.1.8 InteractiveMakerFactory

public class **InteractiveMakerFactory** implements *MakerFactory*

Classe factory implementazione di `MakerFactory` impiegata per ottenere istanze di `InteractiveMaker`.

Author Francesco Pio Stelluti, Francesco Coppola

Methods

getDescription

public String **getDescription**()

getMaker

public *CodeMaker* **getMaker**(*GameView* *view*, int *seqLength*, int *attempts*)

getName

public String **getName**()

4.1.9 MakerFactory

public interface **MakerFactory** extends *PlayerFactory*

Responsabilità: fornire istanze di implementazioni di `CodeMaker`. Interfaccia finalizzata all'implementazione di classi factory per le particolari implementazioni dei giocatori `CodeMaker`.

Author Francesco Pio Stelluti, Francesco Coppola

Methods

getMaker

public *CodeMaker* **getMaker**(*GameView* *view*, int *seqLength*, int *attempts*)

Ottenimento di un'istanza di un giocatore `CodeMaker`.

Parametri

- **view** – vista per l'interazione con l'utente fisico
- **seqLength** – lunghezza della sequenza di `ColorPegs` da trattare
- **attempts** – numero di tentativi per vincere il gioco

Ritorna CodeMaker istanza di un giocatore `CodeMaker`

4.1.10 MakerFactoryRegistry

public class **MakerFactoryRegistry** extends *PlayerFactoryRegistry*

Estensione di `PlayerFactoryRegistry` per poter contenere informazioni circa le implementazioni di `MakerFactory`.

Author Francesco Pio Stelluti, Francesco Coppola

Constructors

MakerFactoryRegistry

public **`MakerFactoryRegistry`**(String *path*)

Parametri

- **`path`** – associato al file da cui recuperare le informazioni sulle classi da caricare dinamicamente

Solleva

- **`BadRegistryException`** – in caso le istanze caricate non siano appartenenti a classi implementazione di `MakerFactory`

4.1.11 PlayerFactory

public interface **`PlayerFactory`**

Responsabilità: fornire istanze di implementazioni dei giocatori. Interfaccia finalizzata all'implementazione di classi factory per le particolari implementazioni dei giocatori.

Author Francesco Pio Stelluti, Francesco Coppola

Methods

getDescription

String **`getDescription`**()

Ritorna String descrizione della particolare implementazione di un giocatore

getName

String **`getName`**()

Ritorna String nome della particolare implementazione di un giocatore

4.1.12 PlayerFactoryRegistry

public abstract class **`PlayerFactoryRegistry`**

Responsabilità: gestione dinamica delle implementazioni delle classi factory implementazione di `PlayerFactory`. Classe astratta estendibile da classi rappresentanti registri contenenti informazioni sulle classi factory impiegate per istanziare le implementazioni dei giocatori.

Author Francesco Pio Stelluti, Francesco Coppola

Constructors

PlayerFactoryRegistry

public **PlayerFactoryRegistry**(String *pathLettura*)
Costruttore di `PlayerFactoryRegistry`.

Parametri

- **pathLettura** – associato al file da cui leggere informazioni da inserire all'interno di `registryFactoryPlayers`.

Solleva

- **BadRegistryException** – in caso ci siano stati errori nell'inizializzazione del registro

Methods

getFactoryByName

public *PlayerFactory* **getFactoryByName**(String *name*)
Ottenimento di un'istanza di `PlayerFactory` dalla struttura dati di base conoscendo il suo nome.

Parametri

- **name** – della particolare `PlayerFactory` richiesta

Solleva

- **BadRegistryException** – in caso la particolare `PlayerFactory` con il nome specificato tramite argomento non sia presente

Ritorna PlayerFactory richiesta

getPlayerFactoriesInstances

public List<*PlayerFactory*> **getPlayerFactoriesInstances**()

Ritorna List contenente tutte le istanze `PlayerFactory` caricate

getPlayersDescription

public List<String> **getPlayersDescription**()

Ritorna List contenente tutte le descrizioni delle istanze `PlayerFactory` caricate

getPlayersNames

public List<String> **getPlayersNames**()

Ritorna List contenente tutti i nomi delle istanze `PlayerFactory` caricate

4.1.13 RandomBotBreakerFactory

public class **RandomBotBreakerFactory** implements *BreakerFactory*

Classe factory implementazione di `BreakerFactory` impiegata per ottenere istanze di `RandomBotBreaker`.

Author Francesco Pio Stelluti, Francesco Coppola

Methods

getBreaker

public *CodeBreaker* **getBreaker** (*GameView* *view*, int *seqLength*, int *attempts*)

getDescription

public String **getDescription** ()

getName

public String **getName** ()

4.1.14 RandomBotMakerFactory

public class **RandomBotMakerFactory** implements *MakerFactory*

Classe factory implementazione di `MakerFactory` impiegata per ottenere istanze di `RandomBotMaker`.

Author Francesco Pio Stelluti, Francesco Coppola

Methods

getDescription

public String **getDescription** ()

getMaker

public *CodeMaker* **getMaker** (*GameView* *view*, int *seqLength*, int *attempts*)

getName

public String **getName** ()

4.2 it.unicam.cs.pa.mastermind.gamecore

Il package contiene le componenti chiave relative all'intera gestione del gioco, quali gli attori delegati alla creazione e alla gestione di nuovi match e gli attori delegati allo svolgimeto vero e proprio di tali match.

4.2.1 BoardController

public class **BoardController**

Responsabilità: gestire le interazioni dall'esterno e dirette alla modifica di un'istanza `BoardModel`. Rientra nel pattern **MVC**.

Author Francesco Pio Stelluti, Francesco Coppola

Constructors

BoardController

public **BoardController**(*BoardModel newBoard*)

Costruttore

Parametri

- **newBoard** – la `BoardModel` che si desidera gestire

Methods

insertCodeToGuess

public boolean **insertCodeToGuess**(List<*ColorPegs*> *toGuess*)

Metodo che consente l'inserimento di una sequenza da indovinare all'interno della `BoardModel`.

Parametri

- **toGuess** – la `List` di `ColorPegs` contenente i valori che si vogliono inserire come sequenza da indovinare.

Ritorna boolean a rappresentazione dell'esito dell'inserimento

insertNewAttempt

public boolean **insertNewAttempt**(List<*ColorPegs*> *attempt*)

Metodo che consente l'inserimento di un nuovo tentativo all'interno della `BoardModel`.

Parametri

- **attempt** – la `List` di `ColorPegs` contenente i valori che si vogliono inserire all'interno della `BoardModel`

Ritorna boolean a rappresentazione dell'esito dell'inserimento

4.2.2 BoardModel

public class **BoardModel**

Responsabilità: gestire le informazioni relative ad una plancia di gioco. Rientra nei pattern **MVC** e **Observer**.

Author Francesco Pio Stelluti, Francesco Coppola

Constructors

BoardModel

public **BoardModel** (int *sequenceLength*, int *maxAttempts*)

Costruttore di una plancia. L'impiego di una LinkedHashMap quale particolare struttura dati per tenere traccia delle sequenze inserite permette di tenere conto anche dell'ordine di inserimento.

Parametri

- **sequenceLength** – massima delle sequenze presenti in questa plancia
- **maxAttempts** – numero massimo di tentativi possibili per indovinare la `sequenceToGuess`

Methods

addAttempt

public boolean **addAttempt** (List<*ColorPegs*> *attempt*)

Aggiunge alla plancia una nuova sequenza di pioli tentativo e la relativa sequenza di pioli indizio, calcolata all'interno del metodo

Parametri

- **attempt** – la sequenza da inserire

Solleva

- **IllegalArgumentException** – in caso di inserimento illegale

Ritorna boolean relativo alla riuscita dell'inserimento

addObserver

public void **addObserver** (*BoardObserver observer*)

Metodo il quale registra un nuovo `BoardObserver` e notifica tutti i `BoardObserver` attualmente associati all'istanza di `BoardModel`.

Parametri

- **observer** – nuova istanza di `BoardObserver` da aggiungere

attemptsInserted

public int **attemptsInserted**()

Ritorna int numero di tentativi inseriti fino ad ora

getAttemptAndClueList

public List<Map.Entry<List<*ColorPegs*>, List<*ColorPegs*>>> **getAttemptAndClueList**()

Ottenimento di una `List` contenente tutta le coppie sequenza tentativo - sequenza indizio inserite nella plancia.

Ritorna List contenenti Map.Entry con le sequenze di `ColorPegs` inserite come tentativo e le relative sequenze indizio

getSequenceLength

public int **getSequenceLength**()

Ritorna int lunghezza massima delle sequenze presenti in questa plancia

getSequenceToGuess

public List<*ColorPegs*> **getSequenceToGuess**()

Ritorna List di `ColorPegs` da indovinare.

hasBreakerGuessed

public boolean **hasBreakerGuessed**()

Ritorna boolean che indica se il giocatore Breaker ha indovinato o meno la sequenza del Maker in base alle informazioni contenute nella plancia

isBoardEmpty

public boolean **isBoardEmpty**()

Ritorna boolean che indica se sono stati inseriti o meno tentativi nella plancia

lastAttemptAndClue

public Map.Entry<List<*ColorPegs*>, List<*ColorPegs*>> **lastAttemptAndClue**()
Ottenimento dell'ultima coppia sequenza tentativo - sequenza indizio inserita nella plancia.

Ritorna Map.Entry contenente l'ultima sequenza di `ColorPegs` inserita come tentativo e la relativa sequenza indizio.

leftAttempts

public int **leftAttempts**()

Ritorna int numero di tentativi rimasti

removeLastAttemptAndClue

public boolean **removeLastAttemptAndClue**()
Rimozione dell'ultima coppia sequenza tentativo - sequenza indizio inserita nella plancia.

Ritorna boolean relativo alla riuscita della rimozione.

setSequenceToGuess

public boolean **setSequenceToGuess** (List<*ColorPegs*> *toGuess*)
Imposta la sequenza di pioli da indovinare.

Parametri

- **toGuess** – lista di `ColorPegs` della sequenza da indovinare

Solleva

- **IllegalArgumentException** – se la lunghezza della sequenza inserita non è valida

Ritorna un booleano a seconda della riuscita o meno dell'inserimento nella plancia di gioco

4.2.3 BoardObserver

public abstract class **BoardObserver**
Classe astratta estendibile da tutte quelle classi coinvolte nel design pattern **Observer**, aventi quindi necessità di osservare e adattarsi in tempo reale ai cambiamenti di stato di oggetti di tipo BoardModel.

Author Francesco Pio Stelluti, Francesco Coppola

Methods

addSubject

public void **addSubject** (*BoardModel* *subject*)
Metodo per il quale viene aggiunto un altro elemento da osservare alla lista interna.

Parametri

- **subject** – il soggetto che si vuole osservare

getSubject

protected *BoardModel* **getSubject** ()
Restituito il riferimento alla `BoardModel` osservata

Ritorna BoardModel il riferimento richiesto

update

public abstract void **update** ()
Aggiornamento dello stato interno dell'oggetto.

4.2.4 ColorPegs

public enum **ColorPegs**
Responsabilità: rappresentare gli elementi alla base delle sequenze trattate durante le partite di gioco.

Author Francesco Pio Stelluti, Francesco Coppola

Enum Constants

BLACK

public static final *ColorPegs* **BLACK**

BLUE

public static final *ColorPegs* **BLUE**

GREEN

public static final *ColorPegs* **GREEN**

RED

public static final *ColorPegs* **RED**

WHITE

public static final *ColorPegs* **WHITE**

YELLOW

public static final *ColorPegs* **YELLOW**

4.2.5 ConsoleMainManager

public class **ConsoleMainManager** extends *MainManager*

Implementazione di `MainManager` correlata ad implementazioni di `GameView` e `StartView` basate su interazione via console.

Author Francesco Pio Stelluti, Francesco Coppola

Methods

getGameViewFactory

protected *GameViewFactory* **getGameViewFactory**()

getStartViewInstance

protected *StartView* **getStartViewInstance**()

main

public static void **main**(String[] *args*)
Metodo main fondamentale per l'avvio

Parametri

- **args** –

4.2.6 GlobalSettings

public class **GlobalSettings**
Responsabilità: tenere traccia delle impostazioni globali del gioco, comuni a tutte le partite. **Contratto**: le istanze vengono gestite all'interno di `MainManager`.

Author Francesco Pio Stelluti, Francesco Coppola

Constructors

GlobalSettings

public **GlobalSettings**()
Inizializzazione con generazione dei registri

Solleva

- **BadRegistryException** – in caso di errori con la generazione dei `PlayerFactoryRegistry`.

Methods

getBreakers

public *BreakerFactoryRegistry* **getBreakers**()

getMakers

public *MakerFactoryRegistry* **getMakers**()

4.2.7 MainManager

public abstract class **MainManager**
Responsabilità: permettere il corretto svolgimento del gioco, monitorando e tenendo traccia di una partita di MasterMind alla volta

Author Francesco Pio Stelluti, Francesco Coppola

Constructors

MainManager

public **`MainManager`**`()`

Methods

getGameViewFactory

protected abstract *GameViewFactory* **`getGameViewFactory`**`()`
Ottenimento dell'istanza di `GameViewFactory` che si desidera impiegare all'interno di tutti i match per poter generare istanze di `GameView` utili per l'interazione con l'utente fisico durante il loro svolgimento. **Contratto**: il metodo deve risultare coerente con la particolare estensione di `MainManager` in cui viene definito.

Ritorna GameViewFactory da impiegare in `SingleMatch`

getStartViewInstance

protected abstract *StartView* **`getStartViewInstance`**`()`
Ottenimento dell'istanza di `StartView` che si desidera impiegare con l'istanza di `MainManager` corrente. **Contratto**: il metodo deve risultare coerente con la particolare estensione di `MainManager` in cui viene definito.

Ritorna StartView da impiegare nel `MainManager`

startUp

public void **`startUp`**`()`
Gestione continua di nuovi match, creati, gestiti ed avviati uno alla volta.

4.2.8 MatchStartSettings

public class **`MatchStartSettings`**
Responsabilità: tenere traccia delle informazioni necessarie per poter iniziare una nuova partita e da impiegare all'interno di essa. **Contratto**: le istanze vengono gestite all'interno di `MainManager`.

Author Francesco Pio Stelluti, Francesco Coppola

Fields

highTresholdLength

int **`highTresholdLength`**

lowTresholdAttempts

int **`lowTresholdAttempts`**

lowTresholdLength

int **lowTresholdLength**

Constructors

MatchStartSettings

public **MatchStartSettings** (*GameViewFactory gameViewFactory*)

Methods

getAttempts

public int **getAttempts** ()

getBreakerFactory

public *BreakerFactory* **getBreakerFactory** ()

getGameViewFactory

public *GameViewFactory* **getGameViewFactory** ()

getHighTresholdLength

public int **getHighTresholdLength** ()

getLowTresholdAttempts

public int **getLowTresholdAttempts** ()

getLowTresholdLength

public int **getLowTresholdLength** ()

getMakerFactory

public *MakerFactory* **getMakerFactory** ()

getSequenceLength

public int **getSequenceLength** ()

resetLengthAttempts

public void **resetLengthAttempts**()

setAttempts

public void **setAttempts**(int *attempts*)

setBreakerFactory

public void **setBreakerFactory**(*BreakerFactory breakerFactory*)

setHighTresholdLength

public void **setHighTresholdLength**(int *highTresholdLength*)

setLowTresholdAttempts

public void **setLowTresholdAttempts**(int *lowTresholdAttempts*)

setLowTresholdLength

public void **setLowTresholdLength**(int *lowTresholdLength*)

setMakerFactory

public void **setMakerFactory**(*MakerFactory makerFactory*)

setSequenceLength

public void **setSequenceLength**(int *sequenceLength*)

4.2.9 MatchState

public class **MatchState** extends *BoardObserver*

Responsabilità: tenere traccia delle informazioni necessarie per poter decretare se una partita è ancora in corso o meno. Rientra nel pattern **Observer**.

Author Francesco Pio Stelluti, Francesco Coppola

Constructors

MatchState

public **MatchState**(*BoardModel subject*)
Inizializzazione con valori di default.

Parametri

- **subject** – BoardModel coinvolta nel pattern **Observer**

Methods

getBreakerVictoryAttempts

public int **getBreakerVictoryAttempts**()
Metodo attraverso il quale vengono restituiti i tentativi usati fino ad ora dal `CodeBreaker` in caso abbia vinto.

Ritorna int numero di tentativi che sono stati necessari al Breaker per vincere.

getHasBreakerWon

public boolean **getHasBreakerWon**()
Metodo che stabilisce la vittoria del giocatore Breaker o meno.

Ritorna boolean che indica se il Breaker ha vinto o meno.

getHasMakerWon

public boolean **getHasMakerWon**()
Metodo che stabilisce la vittoria del giocatore Maker o meno.

Ritorna boolean che indica se il Maker ha vinto o meno.

getMessage

public String **getMessage**()
Metodo che comunica l'esito finale della partita corrente.

Ritorna String che comunica il vincitore attuale della partita

toggleBreakerGiveUp

public void **toggleBreakerGiveUp**()
Toggle sulle variabili private per indicare la resa del Breaker.

toggleBreakerWin

public void **toggleBreakerWin**(int *attempts*)
Toggle sulle variabili private per indicare la vittoria del Breaker.

Parametri

- **attempts** – il numero di tentativi impiegati dal Breaker per vincere

toggleMakerWin

public void **toggleMakerWin**()
Toggle sulle variabili private per indicare la vittoria del Maker.

update

public void **update**()

4.2.10 SingleMatch

public class **SingleMatch**
Responsabilità: gestione dello svolgimento di una singola partita di gioco.

Author Francesco Pio Stelluti, Francesco Coppola

Fields

gameState

MatchState **gameState**
Oggetto contenente informazioni relative al vincitore della partita in corso.

Constructors

SingleMatch

public **SingleMatch**(int *sequenceLength*, int *attempts*, *GameViewFactory viewFactory*, *BreakerFactory bFactory*, *MakerFactory mFactory*)
Costruttore di una singola partita

Parametri

- **sequenceLength** – relativa alle sequenze di `CodePegs` impiegate nella partita.
- **attempts** – massimi per il giocatore Breaker per indovinare.
- **view** – Istanza della particolare implementazione di `InteractionView` scelta per l'istanza di partita in corso.
- **bFactory** – istanza della `BreakerFavctory` relativa al giocatore `CodeBreaker` selezionato per la partita.

- **mFactory** – istanza della `MakerFactory` relativa al giocatore `CodeMaker` selezionato per la partita.

Methods

start

public void **start**()
Avvio e gestione completa di una singola partita di gioco.

4.2.11 StartupSettings

public class **StartupSettings**
Responsabilità: tenere traccia delle informazioni necessarie per decidere se iniziare una nuova partita e se impostare nuove impostazioni di avvio. **Contratto**: le istanze vengono gestite all'interno di `MainManager`.

Author Francesco Pio Stelluti, Francesco Coppola

Constructors

StartupSettings

public **StartupSettings**()
Nel costruttore senza parametri si manifesta la volontà di continuare a giocare sin dall'inizio e di non voler mantenere impostazioni. Il costruttore è nello specifico finalizzato ad un utilizzo dell'istanza di `StartupSettings` sin dall'avvio del gioco, dove si presume si voglia avviare un nuovo match e di fatto non esistono impostazioni passate.

StartupSettings

public **StartupSettings**(boolean *toContinue*, boolean *keepSettings*)
Costruttore in cui è possibile specificare la volontà di effettuare nuove partite e di mantenere o meno le impostazioni per il loro avvio.

Parametri

- **toContinue** –
- **keepSettings** –

Methods

getContinue

public boolean **getContinue**()

Ritorna boolean volontà dell'utente umano di continuare a giocare o meno.

getKeepMatchStartSettings

public boolean **`getKeepMatchStartSettings`**()

Ritorna boolean volontà dell'utente umano di continuare a giocare con le medesime impostazioni o meno.

setKeepMatchStartSettings

public void **`setKeepMatchStartSettings`**(boolean *keepSettings*)

Impostazione valore personalizzato della volontà di mantenere le impostazioni per l'avvio di nuove partite.

Parametri

- **`keepSettings`** – volontà

setToContinue

public void **`setToContinue`**(boolean *toContinue*)

Impostazione valore personalizzato della volontà di continuare a giocare.

Parametri

- **`toContinue`** – volontà

4.3 it.unicam.cs.pa.mastermind.players

Nel seguente package sono definiti i due principali attori del gioco, il Maker, colui che decide la sequenza da indovinare, e il Breaker, colui che deve cercare di indovinare la sequenza decisa dal Maker. All'interno del medesimo package è possibile trovare le implementazioni per queste due entità coinvolte nel gioco.

4.3.1 CodeBreaker

public abstract class **`CodeBreaker`**

Responsabilità: rappresentazione di un giocatore `CodeBreaker`, il cui compito è quello di indovinare la sequenza di `ColorPegs` decisa dal giocatore `CodeMaker`.

Author Francesco Pio Stelluti, Francesco Coppola

Methods

getAttempt

public abstract List<*ColorPegs*> **`getAttempt`**()

Ritorna List contenente i `ColorPegs` validi come sequenza tentativo.

hasGivenUp

public boolean **`hasGivenUp`**`()`

Ritorna la volontà del giocatore `CodeBreaker` di arrendersi o meno

toggleGiveUp

public void **`toggleGiveUp`**`()`

Imposta la volontà del giocatore `CodeBreaker` di arrendersi.

4.3.2 CodeMaker

public abstract class **`CodeMaker`**

Responsabilità: rappresentazione di un giocatore `CodeMaker`, il cui compito è quello di decretare una sequenza di `ColorPegs` che il giocatore `CodeBreaker` deve indovinare.

Author Francesco Pio Stelluti, Francesco Coppola

Methods

getCodeToGuess

public abstract List<*ColorPegs*> **`getCodeToGuess`**`()`

Ritorna List contenente i `ColorPegs` validi come sequenza da indovinare

4.3.3 DonaldKnuthBreaker

public class **`DonaldKnuthBreaker`** extends *CodeBreaker*

Estensione di `CodeBreaker` mirata ad una gestione del comportamento è basato sull'algoritmo di risoluzione teorizzato dal matematico Donald Knuth, il quale attesta di risolvere il gioco del Mastermind in cinque mosse al massimo mediante una precisa serie di passaggi.

Author Francesco Pio Stelluti, Francesco Coppola

Constructors

DonaldKnuthBreaker

public **`DonaldKnuthBreaker`**(int *seqLength*, int *attempts*)

Methods

getAttempt

public List<*ColorPegs*> **`getAttempt`**`()`

4.3.4 InteractiveBreaker

public class **InteractiveBreaker** extends *CodeBreaker*

Particolare estensione di `CodeBreaker`, rappresentante un utente fisico. Nello specifico l'utente umano può effettuare decisioni ed impartire comandi passando da un'istanza di `GameView`.

Author Francesco Pio Stelluti, Francesco Coppola

Constructors

InteractiveBreaker

public **InteractiveBreaker**(*GameView newView*)

Methods

getAttempt

public List<*ColorPegs*> **getAttempt**()

L'utente fisico può decidere di voler reinserire una sequenza di `ColorPegs` già inserita precedentemente. In tal caso ripeterà l'azione di definizione di una nuova sequenza. **Contratto**: se dalla vista `GameView` viene restuito il valore 0 allora tale valore viene interpretato come la volontà dell'utente fisico di arrendersi.

4.3.5 InteractiveMaker

public class **InteractiveMaker** extends *CodeMaker*

Particolare estensione di `CodeMaker`, rappresentante un giocatore umano. Nello specifico l'utente umano può effettuare decisioni ed impartire comandi passando da un'istanza di `GameView`.

Author Francesco Pio Stelluti, Francesco Coppola

Constructors

InteractiveMaker

public **InteractiveMaker**(*GameView newView*)

Methods

getCodeToGuess

public List<*ColorPegs*> **getCodeToGuess**()

4.3.6 RandomBotBreaker

public class **RandomBotBreaker** extends *CodeBreaker*

Estensione di `CodeBreaker` mirata ad una gestione del comportamento del giocatore in maniera casuale.

Author Francesco Pio Stelluti, Francesco Coppola

Constructors

RandomBotBreaker

public **RandomBotBreaker**(int *seqLength*)

Methods

getAttempt

public List<*ColorPegs*> **getAttempt**()
Potrebbe capitare che la generazione casuale delle sequenze porti ad una sequenza di `ColorPegs` già inserita precedentemente. In tal caso verrà ripetuta l'azione di definizione di una nuova sequenza.

4.3.7 RandomBotMaker

public class **RandomBotMaker** extends *CodeMaker*
Estensione di `CodeMaker` mirata ad una gestione del comportamento del giocatore in maniera casuale.

Author Francesco Pio Stelluti, Francesco Coppola

Constructors

RandomBotMaker

public **RandomBotMaker**(int *seqLength*)

Methods

getCodeToGuess

public List<*ColorPegs*> **getCodeToGuess**()

4.4 it.unicam.cs.pa.mastermind.ui

Il seguente package contiene le classi relative a tutto ciò che concerne l'interfaccia di gioco con la quale comunicherà l'utente fisico, sia esso o meno un giocatore attivo nel gioco. Attraverso le classi di questo package è possibile avere le operazioni di Input/Output iniziali con il programma e le interazioni di Input/Output durante lo svolgimento delle partite.

4.4.1 AnsiUtility

public class **AnsiUtility**
La seguente classe ha il solo scopo di rendere la console di gioco più accattivante e user-friendly andando ad aggiungere una nota di colore ai vari ColorPegs che verranno inseriti.

Author Francesco Pio Stelluti, Francesco Coppola

Fields

ANSI_BLACK_BACKGROUND

public static final String **ANSI_BLACK_BACKGROUND**

ANSI_BLUE_BACKGROUND

public static final String **ANSI_BLUE_BACKGROUND**

ANSI_CYAN_BACKGROUND

public static final String **ANSI_CYAN_BACKGROUND**

ANSI_CYAN_BOLD

public static final String **ANSI_CYAN_BOLD**

ANSI_GREEN_BACKGROUND

public static final String **ANSI_GREEN_BACKGROUND**

ANSI_PURPLE_BACKGROUND

public static final String **ANSI_PURPLE_BACKGROUND**

ANSI_RED_BACKGROUND

public static final String **ANSI_RED_BACKGROUND**

ANSI_RED_BOLD

public static final String **ANSI_RED_BOLD**

ANSI_RESET

public static final String **ANSI_RESET**

ANSI_WHITE_BACKGROUND

public static final String **ANSI_WHITE_BACKGROUND**

ANSI_WHITE_BOLD

public static final String **ANSI_WHITE_BOLD**

ANSI_YELLOW

public static final String **ANSI_YELLOW**

ANSI_YELLOW_BACKGROUND

public static final String **ANSI_YELLOW_BACKGROUND**

4.4.2 ConsoleGameView

public class **ConsoleGameView** extends *GameView*

Implementazione di una vista con interazione via console della classe `GameView`.

Author Francesco Pio Stelluti, Francesco Coppola

Constructors

ConsoleGameView

public **ConsoleGameView**()

Inizializzazione della vista con un `FilterInputStream` che non porta alla chiusura di `System.in` all'interno del suo metodo `close()`.

Methods

endingScreen

public void **endingScreen**(String *gameEndingMessage*)

getIndexSequence

public List<Integer> **getIndexSequence**(boolean *isBreaker*)

showGame

public void **showGame**()

update

public void **update**()

4.4.3 ConsoleStartView

public class **`ConsoleStartView`** implements *StartView*

Implementazione con interazione via console della classe `StartView`. Integra il pattern **Singleton**.

Author Francesco Pio Stelluti, Francesco Coppola

Methods

askNewAttempts

public int **`askNewAttempts`** (int *lowTreshold*)

askNewLength

public int **`askNewLength`** (int *lowTreshold*, int *highTreshhold*)

askNewLengthsAndAttempts

public boolean **`askNewLengthsAndAttempts`** ()

askNewStartupSettings

public *StartupSettings* **`askNewStartupSettings`** ()

badEnding

public void **`badEnding`** (String *reason*)

ending

public void **`ending`** ()

getInstance

public static *ConsoleStartView* **`getInstance`** ()

Ritorna ConsoleStartView istanza **Singleton** di `ConsoleStartView`.

getPlayerName

public String **`getPlayerName`** (*PlayerFactoryRegistry* *registry*, boolean *isBreaker*)

showLogo

public void **`showLogo`** ()

showNewMatchStarting

public void **showNewMatchStarting**()

4.4.4 GameView

public abstract class **GameView** extends *BoardObserver*

Responsabilità: fornire agli utenti fisici coinvolti in una singola partita operazioni di Input/Output. Rientra nel pattern **Observer** per poter fornire in output all'utente fisico una rappresentazione di quelle che sono le azioni effettuate dai giocatori nel gioco. Rientra nel pattern **MVC**.

Author Francesco Pio Stelluti, Francesco Coppola

Methods

endingScreen

public abstract void **endingScreen**(String *gameEndingMessage*)

Interazione finale con l'utente fisico relativa al termine di una partita

Parametri

- **gameEndingMessage** – stringa con il messaggio finale da mostrare all'utente

getIndexSequence

public abstract List<Integer> **getIndexSequence**(boolean *toGuess*)

Interazione con l'utente fisico per poter ottenere gli indici associati ai diversi valori di `ColorPegs`. Se il valore restituito contiene l'`Integer` 0 è stata rappresentata la volontà di un giocatore `CodeBreaker` di arrendersi.

Parametri

- **toGuess** – flag che indica se la sequenza di indici interi da ottenere si riferisce alla sequenza da indovinare o meno

Ritorna List contenente gli indici interi associati all'enum ColorPegs

showGame

public abstract void **showGame**()

Interazione con l'utente fisico per mostrare la situazione di gioco.

4.4.5 StartView

public interface **StartView**

Responsabilità: fornire agli utenti fisici coinvolti nel gioco l'interazione per poter iniziare nuove partite.

Author Francesco Pio Stelluti, Francesco Coppola

Methods

askNewAttempts

public int **askNewAttempts** (int *lowTres*)

Gestione dell'interazione con l'utente fisico per ottenere un nuovo valore relativo al numero di tentativi utili all'interno del gioco.

Parametri

- **lowTres** – limite inferiore al valore da scegliere

Ritorna int valore scelto

askNewLength

public int **askNewLength** (int *lowTres*, int *highTres*)

Gestione dell'interazione con l'utente fisico per ottenere un nuovo valore relativo alla lunghezza delle sequenze impiegate nel gioco.

Parametri

- **lowTres** – limite inferiore al valore da scegliere
- **highTres** – limite superiore al valore da scegliere

Ritorna int valore scelto

askNewLengthsAndAttempts

public boolean **askNewLengthsAndAttempts** ()

Gestione dell'interazione con l'utente fisico circa le decisioni per l'impostazione di nuovi valori di lunghezza delle sequenze e di numero di tentativi per un nuovo match.

Ritorna boolean volontà dell'utente fisico di decidere nuove impostazioni per un nuovo match.

askNewStartupSettings

public *StartupSettings* **askNewStartupSettings** ()

Gestione dell'interazione con l'utente fisico circa le decisioni per l'inizio di un nuovo match o meno dopo che uno è stato concluso.

Ritorna StartupSettings contenente informazioni utili per iniziare o meno nuovi match

badEnding

public void **badEnding** (String *reason*)

Gestione anticipata della conclusione dell'intero gioco, richiamata ad esempio per il sollevamento di errori importanti.

Parametri

- **reason** – da presentare all'utente fisico

ending

public void **ending**()
Gestione della conclusione dell'intero gioco dopo la fine di ogni singola partita.

getPlayerName

public String **getPlayerName**(*PlayerFactoryRegistry* *registry*, boolean *isBreaker*)
Gestione dell'interazione dell'utente fisico per la scelta della particolare implementazione dei giocatori che verranno coinvolti nella nuova partita.

Parametri

- **registry** – registro contenente le informazioni sulle classi `PlayerFactory` relative alle implementazioni dei giocatori.
- **isBreaker** – flag che indica se la scelta è relativa ad una factory finalizzata alla generazione di un giocatore `CodeBreaker` o meno.

Ritorna String rappresentante l'implementazione del giocatore scelta per la nuova partita.

setupBreaker

public *BreakerFactory* **setupBreaker**(*BreakerFactoryRegistry* *registry*)
Gestione dell'interazione con l'utente fisico circa la particolare implementazione di `CodeBreaker` da impiegare nel gioco.

Parametri

- **registry** – da cui recuperare le informazioni

Ritorna BreakerFactory per la generazione di nuovi giocatori `CodeBreaker`

setupMaker

public *MakerFactory* **setupMaker**(*MakerFactoryRegistry* *registry*)
Gestione dell'interazione con l'utente fisico circa la particolare implementazione di `CodeMaker` da impiegare nel gioco.

Parametri

- **registry** – da cui recuperare le informazioni

Ritorna MakerFactory per la generazione di nuovi giocatori `CodeMaker`

showLogo

public void **showLogo**()
Gestione di interazione con l'utente fisico per mostrare il logo di gioco.

showNewMatchStarting

public void **showNewMatchStarting**()
Gestione dell'interazione con l'utente fisico circa l'inizio di un nuovo match

CAPITOLO 5

Test realizzati in JUnit

Di seguito è possibile analizzare in maniera dettagliata e scrupolosa quelli che sono i **test** che sono stati prodotti per mostrare il corretto funzionamento del progetto.

Essi infatti **garantiscono oggettivamente** che il codice si comporti come previsto.

5.1 it.unicam.cs.pa.mastermind.test

Il seguente package contiene i vari test che andaranno effettuati all'interno del progetto, per testarne la qualità, la bontà e soprattutto l'efficenza.

5.1.1 GameCoreBoardControllerTest

class **`GameCoreBoardControllerTest`**
Test di controllo utili alle meccaniche del coordinatore di gioco.

Author Francesco Pio Stelluti, Francesco Coppola

Fields

attempt

List<*ColorPegs*> **`attempt`**

toGuess

List<*ColorPegs*> **`toGuess`**

Methods

setUp

void **setUp**()
Setup of the board runned before each other test.

testBoardController

void **testBoardController**()
Test method for it.unicam.cs.pa.mastermind.gamecore.BoardController.BoardController(it.unicam.cs.pa.mastermind.gamecore.BoardModel).

testGetSequenceLength

void **testGetSequenceLength**()
Test method for it.unicam.cs.pa.mastermind.gamecore.BoardController.getSequenceLength().

testGetSequenceToGuess

void **testGetSequenceToGuess**()
Test method for it.unicam.cs.pa.mastermind.gamecore.BoardController.getSequenceToGuess().

testInsertCodeToGuess

void **testInsertCodeToGuess**()
Test method for it.unicam.cs.pa.mastermind.gamecore.BoardController.insertCodeToGuess(java.util.List).

testInsertNewAttempt

void **testInsertNewAttempt**()
Test method for it.unicam.cs.pa.mastermind.gamecore.BoardController.insertNewAttempt(java.util.List).

5.1.2 GameCoreBoardModelTest

class **GameCoreBoardModelTest**
Test di controllo all'interno della board.

Author Francesco Pio Stelluti, Francesco Coppola

Fields

attempt

List<*ColorPegs*> **attempt**

toGuess

List<*ColorPegs*> **toGuess**

Methods

setUp

void **setUp**()
Setup of the board runned before each other test.

testAddAttempt

void **testAddAttempt**()
Test method for `it.unicam.cs.pa.mastermind.gamecore.BoardModel.addAttempt(java.util.List,java.util.List)`.

testAttemptsInserted

void **testAttemptsInserted**()
Test method for *it.unicam.cs.pa.mastermind.gamecore.BoardModel.attemptsInserted()*.

testBoard

void **testBoard**()
Test method for `it.unicam.cs.pa.mastermind.gamecore.BoardModel.Board(int,int)`.

testIsEmpty

void **testIsEmpty**()
Test method for `it.unicam.cs.pa.mastermind.gamecore.BoardModel.isEmpty()`.

testLastAttemptAndClue

void **testLastAttemptAndClue**()
Test method for *it.unicam.cs.pa.mastermind.gamecore.BoardModel.lastAttemptAndClue()*.

testLeftAttempts

void **testLeftAttempts**()
Test method for *it.unicam.cs.pa.mastermind.gamecore.BoardModel.leftAttempts()*.

testSetSequenceToGuess

void **testSetSequenceToGuess**()
Test method for it.unicam.cs.pa.mastermind.gamecore.BoardModel.setSequenceToGuess(java.util.List).

5.1.3 PlayersFactoryRegistry

class **PlayersFactoryRegistry**
Test di controllo utili alla generazione delle factory relativi ai player.

Author Francesco Pio Stelluti, Francesco Coppola

Fields

playersFactory

List<String> **playersFactory**

Methods

testBreakerFactoryRegistry

void **testBreakerFactoryRegistry**()
Test method for it.unicam.cs.pa.mastermind.factories.BreakerFactoryRegistry.BreakerFactoryRegistry().

Solleva

- **BadRegistryException** –

testCheckRightPathName

void **testCheckRightPathName**()
Test method for the check of the existence of the path name passed in the constructor.

Solleva

- **BadRegistryException** –
- **IOException** –

testGetFactoryByName

void **testGetFactoryByName**()
Test method for `it.unicam.cs.pa.mastermind.factories.PlayerFactoryRegistry.getFactoryByName(java.lang.String)`.

Solleva

- **BadRegistryException** –

testGetPlayersNames

void **testGetPlayersNames**()
Test method for *it.unicam.cs.pa.mastermind.factories.PlayerFactoryRegistry.getPlayersNames()*.

Solleva

- **BadRegistryException** –

testMakerFactoryRegistry

void **testMakerFactoryRegistry**()
Test method for `it.unicam.cs.pa.mastermind.factories.MakerFactoryRegistry.MakerFactoryRegistry()`.

Solleva

- **BadRegistryException** –

5.1.4 PlayersInteractiveBreakerTest

class **PlayersInteractiveBreakerTest**
Test di controllo utili alla generazione di un player decodficatore di natura umana.

Author Francesco Pio Stelluti, Francesco Coppola

Methods

testGetAttempt

void **testGetAttempt**()
Test method for `it.unicam.cs.pa.mastermind.players.InteractiveBreaker.getAttempt(int,it.unicam.cs.pa.mastermind.ui.InteractionView)`.

testInteractiveBreaker

void **testInteractiveBreaker**()
Test method for `it.unicam.cs.pa.mastermind.players.InteractiveBreaker.InteractiveBreaker()`.

5.1.5 PlayersInteractiveMakerTest

class **PlayersInteractiveMakerTest**
Test di controllo utili alla generazione di un player codficatore di natura umana.

Author Francesco Pio Stelluti, Francesco Coppola

Methods

testGetCodeToGuess

void **testGetCodeToGuess**()
Test method for `it.unicam.cs.pa.mastermind.players.InteractiveMaker.getCodeToGuess(int,it.unicam.cs.pa.mastermind.ui.InteractionView)`.

5.1.6 PlayersRandomBotBreakerTest

class **PlayersRandomBotBreakerTest**
Test di controllo utili alla generazione di un player decodficatore di natura bot.

Author Francesco Pio Stelluti, Francesco Coppola

Methods

testGetAttempt

void **testGetAttempt**()
Test method for `it.unicam.cs.pa.mastermind.players.RandomBotBreaker.getAttempt(int,it.unicam.cs.pa.mastermind.ui.InteractionManager)`.

5.1.7 PlayersRandomBotMakerTest

class **PlayersRandomBotMakerTest**
Test di controllo utili alla generazione di un player codficatore di natura bot.

Author Francesco Pio Stelluti, Francesco Coppola

Methods

testGetCodeToGuess

void **testGetCodeToGuess**()
Test method for `it.unicam.cs.pa.mastermind.players.RandomBotMaker.getCodeToGuess(int,it.unicam.cs.pa.mastermind.ui.InteractionManager)`.

5.1.8 SimulationGame

class **SimulationGame**
Il seguente test simula il corretto funzionamento di una singola partita.

Author Francesco Pio Stelluti, Francesco Coppola

Methods

testSimulationGame

void **testSimulationGame**()

5.1.9 UIConsoleStartViewTest

class **UIConsoleStartViewTest**
Test di controllo utili al check dell'unica instanza della classe sotto esamina.

Author Francesco Pio Stelluti, Francesco Coppola

Methods

testGetIstance

void **testGetIstance**()
Test method for *it.unicam.cs.pa.mastermind.ui.ConsoleStartView.getInstance()*.

Indice

H

I

L

M

P

R

S

T

U

W

Y