MasterMind

Release 1.0.0

Francesco Pio Stelluti, Francesco Coppola

06 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**.

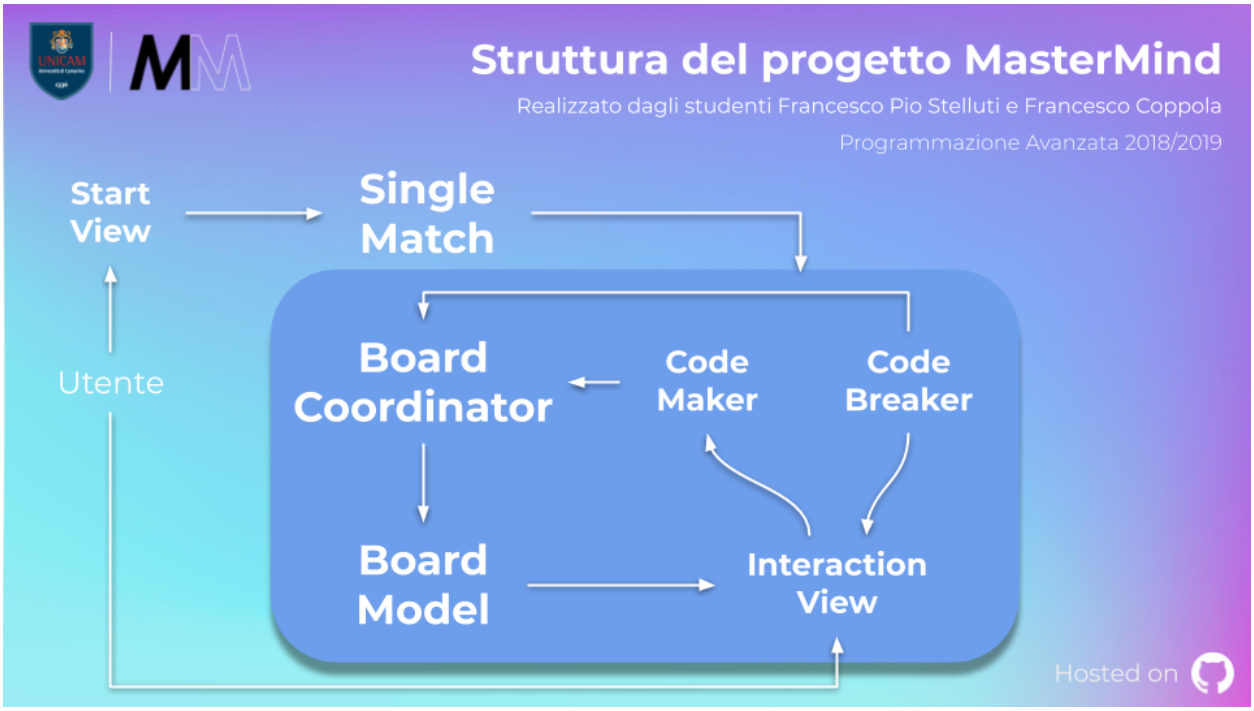
[1] Maria Konnikova, Mastermind: How to Think Like Sherlock Holmes

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 Struttura fondamentale del progetto



[1] Mastermind

L'idea alla base della struttura del gioco riguarda le *interazioni* tra l'utente umano ed un'istanza di una classe che estende `StartView`. Tramite questa interazione è possibile decidere quali impostazioni e quali implementazioni dei giocatori, rispettivamente un `CodeBreaker` ed un `CodeMaker`, impiegare all'interno di singole partite. I giocatori potranno poi interagire all'interno della partita comunicando con una istanza di una classe che estende `InteractionView`, dalla quale ottengono informazioni sulla partita in corso e grazie alla quale hanno una possibile interazione con l'utente umano, e con una istanza di `BoardController`, alla quale **comunicano** decisioni di gioco quali la sequenza da indovinare o le sequenze valide come tentativo per poter indovinare tale sequenza.

Alla base della sequenza, a rappresentazione dei pioli impiegati nel gioco originale, sono presenti valori della classe enum `ColorPegs`, contenente otto colori.

1.2 Estendibilità ed implementazioni fornite di default

Si è deciso di adottare una struttura molto rigida per quanto riguarda la rappresentazione dei pioli e della plancia di gioco, non offrendo possibilità di aggiungere ulteriori implementazioni o diversificazioni di quelle che sono le classi **ColorPegs**, **BoardModel** e **BoardCoordinator**. Diverso il discorso sul piano delle implementazioni di giocatori o delle interfacce di comunicazione con l'utente umano. È infatti possibile aggiungere classi che estendono `CodeMaker` e `CodeBreaker`, fornendo anche le relative classi *factory* che estendono rispettivamente `MakerFactory` e `BreakerFactory`, senza che il codice venga ricompilato. Per fare ciò si è deciso di implementare una classe astratta `PlayerFactoryRegistry`, estesa nel progetto in questione da `MakerFactoryRegistry` e `BreakerFactoryRegistry`, classi che permettono di **collezionare a runtime** informazioni riguardo le factory puntate a generare istannze di classi estensione di `CodeMaker` e `CodeBreaker`. Analogamente è possibile aggiungere classi estensione di `StartView` per fornire particolari *viste* indirizzate **all'interazione** con l'utente fisico durante l'impostazione e l'avvio di nuove partite. Ad ogni `StartView` si richiede di associare anche una classe che estenda `InteractionView` che sia **coerente** con la particolare estensione di StartView trattata e di includere il metodo **main** per permettere l'avvio effettivo del programma.

Di default sono fornite delle implementazioni di quelle che sono le classi rappresentanti i giocatori e l'interazione con l'utente umano:

- **ConsoleStartView**: estensione di `StartView`, fornisce un'interazione con l'utente fisico per l'impostazione e l'avvio di nuove partite tramite console.
- **ConsoleInteractionView**: estensione di `InteractionView`, è strettamente associata con la classe `ConsoleStartView` fornisce un'interazione con l'utente fisico in caso siano necessarie per *impartire* nuove decisioni durante lo svolgimento di una partita.
- **InteractiveMaker**: estensione di `CodeMaker`, fornisce l'implementazione di un giocatore comandato dall'utente umano attraverso l'interazione fornita da una classe estensione di `InteractionView`. È possibile ottenere istanze di questa estensione tramite la classe `InteractiveMakerFactory`.
- **InteractiveBreaker**: estensione di `CodeBreaker`, fornisce l'implementazione di un giocatore comandato dall'utente umano attraverso l'interazione fornita da una classe estensione di `InteractionView`. È possibile ottenere istanze di questa estensione tramite la classe `InteractiveBreakerFactory`.
- **RandomBotMaker**: estensione di `CodeMaker`, fornisce l'implementazione di un giocatore comandato da un **IA** che agisce fornendo sequenze randomiche. È possibile ottenere istanze di questa estensione tramite la classe `RandomBotMakerFactory`.
- **RandomBotBreaker**: estensione di `CodeBreaker`, fornisce l'implementazione di un giocatore comandato da un **IA** che agisce fornendo sequenze randomiche. È possibile ottenere istanze di questa estensione tramite la classe `RandomBotBreakerFactory`.

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 lettura di specifici file testuali. In loro assenza il software creerà dei file standard, comunicando all'utente questa decisione, da modificare **obbligatoriamente** con le giuste informazioni per avere un corretto avvio ed una corretta esecuzione del programma.

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 dell'intero gioco. È stata implementata tramite le classi `StartView`, `InteractionView`, `BoardModel` e `BoardCoordinator`.

2. **Observer**[3] Implementato fornendo come classe da osservare `BoardModel` e come classi che osservano `InteractionView` e `CurrentGameStats`. Dalla versione 9 di Java l'interfaccia Observer, pensata nell'ottica di questo design pattern, risulta deprecata. La sua implementazione 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 delle classi `ConsoleStartView` e `ConsoleInteractionView`, esso garantisce che siano presenti **singole** istanze di tali classi all'interno del progetto.

4. **Factory**[5] Implementato tramite l'interfaccia `PlayerFactory`, implementata da `BreakerFactory` e `MakerFactory`, **classi astratte** da estendere tramite classi factory che forniscano istanze di classi estensione rispettivamente di `CodeBreaker` e `CodeMaker`.

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 5.1.1, per facilitare il deploy e la distribuzione di tale software all'interno di altri sistemi.

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

CAPITOLO 2

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.

2.1 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.

2.2 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
```

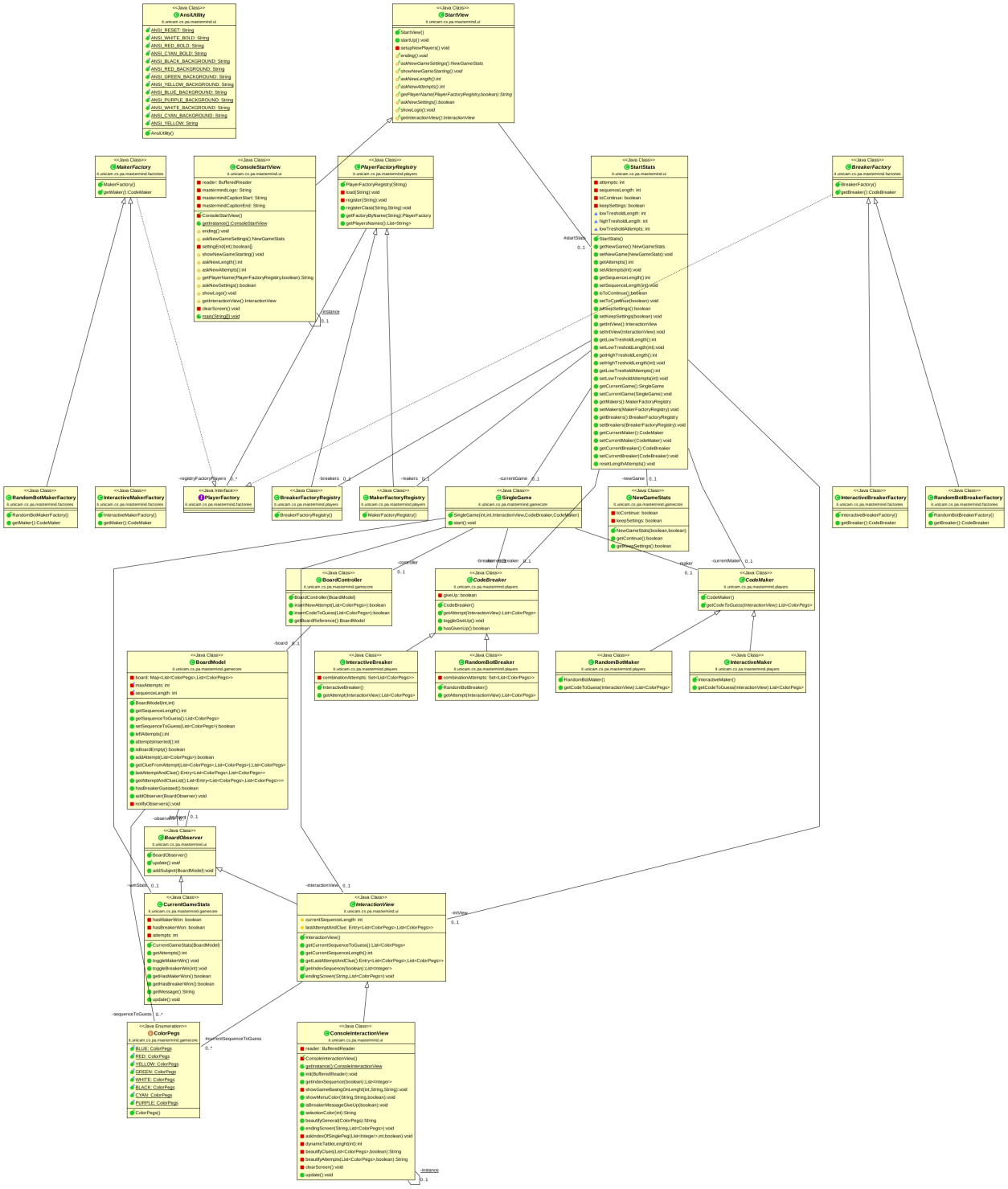
La documentazione quindi sarà pronta per essere usata nei vari file con estensione `.rst` che, attraverso il comando `make`, diventaranno file `.html`.

CAPITOLO 3

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.



3.1 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.

3.1.1 BreakerFactory

public abstract class **BreakerFactory** implements *PlayerFactory*

Classe factory astratta estensione di `PlayerFactory` da estendere con classi factory concrete finalizzate all'ottenimento di istanze di `CodeBreaker`.

Author Francesco Pio Stelluti, Francesco Coppola

Methods

getBreaker

public abstract *CodeBreaker* **getBreaker**()

3.1.2 InteractiveBreakerFactory

public class **InteractiveBreakerFactory** extends *BreakerFactory*

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

Author Francesco Pio Stelluti, Francesco Coppola

Methods

getBreaker

public *CodeBreaker* **getBreaker**()

3.1.3 InteractiveMakerFactory

public class **InteractiveMakerFactory** extends *MakerFactory*

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

Author Francesco Pio Stelluti, Francesco Coppola

Methods

getMaker

public *CodeMaker* **getMaker**()

3.1.4 MakerFactory

public abstract class **MakerFactory** implements *PlayerFactory*

Classe factory astratta estensione di `PlayerFactory` da estendere con classi factory concrete finalizzate all'ottenimento di istanze di `CodeMaker`.

Author Francesco Pio Stelluti, Francesco Coppola

Methods

getMaker

public abstract *CodeMaker* **getMaker**()

3.1.5 PlayerFactory

public interface **PlayerFactory**

Interfaccia `PlayerFactory` che consente l'implementazione da parte delle altri classi ed usata quale label da applicare alle classi che la implementano.

Author Francesco Pio Stelluti, Francesco Coppola

3.1.6 RandomBotBreakerFactory

public class **RandomBotBreakerFactory** extends *BreakerFactory*

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

Author Francesco Pio Stelluti, Francesco Coppola

Methods

getBreaker

public *CodeBreaker* **getBreaker**()

3.1.7 RandomBotMakerFactory

public class **RandomBotMakerFactory** extends *MakerFactory*

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

Author Francesco Pio Stelluti, Francesco Coppola

Methods

getMaker

public *CodeMaker* **getMaker**()

3.2 it.unicam.cs.pa.mastermind.gamecore

Il package contiene le componenti chiave relative ad una singola partita, quali la plancia di gioco, il controllore di tale plancia e l'istanza della partita correntemente attiva.

3.2.1 BoardController

public class **BoardController**

Responsabilità: gestire le interazioni dall'esterno e dirette alla modifica di un'istanza `BoardModel`.

Author Francesco Pio Stelluti, Francesco Coppola

Constructors

BoardController

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

Costruttore

Parametri

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

Methods

getBoardReference

public *BoardModel* **getBoardReference**()

Ritorna BoardModel su cui agisce l'istanza di `BoardCoordinator` corrente.

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

3.2.2 BoardModel

public class **BoardModel**

Responsabilità: gestire le informazioni relative ad una plancia di gioco.

Author Francesco Pio Stelluti, Francesco Coppola

Constructors

BoardModel

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

Costruttore di una plancia

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**()

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

getClueFromAttempt

public List<*ColorPegs*> **getClueFromAttempt**(List<*ColorPegs*> *attempt*)

Calcolo della sequenza di `ColorPegs` indizio a fronte di una sequenza di `ColorPegs` assicurata valida come tentativo.

Parametri

- **attempt** – la lista che si inserisce come tentativo di risoluzione.
- **toGuess** – la lista che contiene la sequenza da indovinare.

Ritorna List di indizi generata a partire dalla lista di tentativi.

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**()

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

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

3.2.3 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**

CYAN

public static final *ColorPegs* **CYAN**

GREEN

public static final *ColorPegs* **GREEN**

PURPLE

public static final *ColorPegs* **PURPLE**

RED

public static final *ColorPegs* **RED**

WHITE

public static final *ColorPegs* **WHITE**

YELLOW

public static final *ColorPegs* **YELLOW**

3.2.4 CurrentGameStats

public class **CurrentGameStats** extends *BoardObserver*

Responsabilità: tenere traccia delle informazioni necessarie per poter decretare se una partita è terminata o meno.

Author Francesco Pio Stelluti, Francesco Coppola

Constructors

CurrentGameStats

public **CurrentGameStats** (*BoardModel* *board*)

Costruttore.

Methods

getAttempts

public int **getAttempts** ()

Metodo attraverso il quale vengono restituiti i tentativi rimanenti al player per vincere il game corrente.

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

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**()

3.2.5 NewGameStats

public class **NewGameStats**
Responsabilità: tenere traccia delle informazioni necessarie per poter iniziare una nuova partita dopo che ne è stata conclusa una.

Author Francesco Pio Stelluti, Francesco Coppola

Constructors

NewGameStats

public **NewGameStats**(boolean *toContinue*, boolean *keepSettings*)
Costruttore

Parametri

- **toContinue** – volontà dell'utente umano di continuare a giocare o meno.
- **keepSettings** – volontà dell'utente umano di continuare a giocare con le medesime impostazioni o meno.

Methods

getContinue

public boolean **getContinue**()

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

getKeepSettings

public boolean **getKeepSettings**()

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

3.2.6 SingleMatch

public class **SingleMatch**

Responsabilità: gestione dello svolgimento di una singola partita di gioco.

Author Francesco Pio Stelluti, Francesco Coppola

Fields

gameStats

CurrentGameStats **gameStats**

Oggetto contenente informazioni relative al vincitore della partita in corso.

Constructors

SingleMatch

public **SingleMatch**(int *sequenceLength*, int *attempts*, *InteractionView* *view*, *CodeBreaker* *currentBreaker*, *CodeMaker* *currentMaker*)

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.
- **currentBreaker** – istanza del giocatore che decodifica.
- **currentMaker** – istanza del giocatore che codifica.

Methods

start

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

3.3 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 responsabilità.

3.3.1 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

Constructors

BadRegistryException

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

3.3.2 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*)

3.3.3 CodeBreaker

public abstract class **CodeBreaker**
Responsabilità: gestire le interazioni del giocatore Breaker

Author Francesco Pio Stelluti, Francesco Coppola

Methods

getAttempt

public abstract List<*ColorPegs*> **getAttempt** (*InteractionView intView*)
Restituisce la sequenza di `ColorPegs` valida come singolo tentativo.

Parametri

- **intView** – necessario per ottenere informazioni riguardo il gioco

Ritorna List di `ColorPegs` valida come singolo tentativo

hasGivenUp

public boolean **hasGivenUp** ()

Ritorna la volontà del giocatore `CodeBreaker` di arrendersi

toggleGiveUp

public void **toggleGiveUp** ()
Imposta la volontà del giocatore `CodeBreaker` di arrendersi.

3.3.4 CodeMaker

public abstract class **CodeMaker**
Responsabilità: gestire le interazioni del giocatore Maker

Author Francesco Pio Stelluti, Francesco Coppola

Methods

getCodeToGuess

public abstract List<*ColorPegs*> **getCodeToGuess** (*InteractionView intView*)
Restituisce la sequenza di `ColorPegs` valida come sequenza da indovinare.

Parametri

- **intView** – necessario per ottenere informazioni riguardo il gioco

Ritorna List di `ColorPegs` valida come sequenza da indovinare

3.3.5 InteractiveBreaker

public class **InteractiveBreaker** extends *CodeBreaker*
Estensione di `CodeBreaker` mirata ad una gestione del comportamento del giocatore tramite interazioni con l'utente umano.

Author Francesco Pio Stelluti, Francesco Coppola

Constructors

InteractiveBreaker

public **InteractiveBreaker**()

Methods

getAttempt

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

3.3.6 InteractiveMaker

public class **InteractiveMaker** extends *CodeMaker*

Estensione di `CodeMaker` mirata ad una gestione del comportamento del giocatore tramite interazioni con l'utente umano.

Author Francesco Pio Stelluti, Francesco Coppola

Methods

getCodeToGuess

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

3.3.7 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*)

3.3.8 PlayerFactoryRegistry

public abstract class **PlayerFactoryRegistry**

Responsabilità: gestione dinamica delle implementazioni delle classi factory di `CodeMaker` e `CodeBreaker`. 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 istanze di `PlayerFactory`.

Parametri

- **name** – nome associato all'istanza di `PlayerFactory`

Ritorna PlayerFactory associato al nome

getPlayerFactoriesInstances

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

Ritorna List contenente le istanze di `PlayerFactory` presenti in `registryFactoryPlayers`

getPlayersNames

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

Ritorna List contenente i nomi associati alle istanze di `PlayerFactory` presenti in `registryFactoryPlayers`

3.3.9 RandomBotBreaker

public class **RandomBotBreaker** extends *CodeBreaker*
Estensione di `CodeBreaker` mirata ad una gestione del comportamento del giocatore parzialmente random.

Author Francesco Pio Stelluti, Francesco Coppola

Constructors

RandomBotBreaker

public **RandomBotBreaker**()

Methods

getAttempt

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

3.3.10 RandomBotMaker

public class **RandomBotMaker** extends *CodeMaker*

Estensione di `CodeMaker` mirata ad una gestione del comportamento del giocatore totalmente random.

Author Francesco Pio Stelluti, Francesco Coppola

Methods

getCodeToGuess

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

3.4 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, sia esso o meno un giocatore attivo nel gioco. Attraverso le classi di questo package è possibile avere le interazioni iniziali con il programma e le interazioni durante lo svolgimento delle partite.

3.4.1 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**

3.4.2 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

Fields

subject

protected *BoardModel* **subject**

L'oggetto che viene osservato.

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

update

public abstract void **update** ()

Aggiornamento dello stato interno dell'oggetto.

3.4.3 ConsoleInteractionView

public class **ConsoleInteractionView** extends *InteractionView*

Implementazione con interazione via console della classe `InteractionView`.

Author Francesco Pio Stelluti, Francesco Coppola

Methods

endingScreen

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

getIndexSequence

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

getInstance

public static *ConsoleInteractionView* **getInstance** ()

Ritorna ConsoleStartView istanza singleton di `ConsoleInteractionView`.

init

public void **init** (BufferedReader *newReader*)
Inizializzazione del reader associato all'istanza di `ConsoleInteractionView`.

Parametri

- **newReader** – reader da associare all'istanza.

update

public void **update** ()

3.4.4 ConsoleStartView

public class **ConsoleStartView** extends *StartView*
Implementazione con interazione via console della classe `StartView`.

Author Francesco Pio Stelluti, Francesco Coppola

Methods

askNewAttempts

protected int **askNewAttempts** ()

askNewGameSettings

protected *NewGameStats* **askNewGameSettings** ()

askNewLength

protected int **askNewLength** ()

askNewSettings

protected boolean **askNewSettings** ()

badEnding

protected void **badEnding**(String *reason*)

ending

protected void **ending**()

getInstance

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

Ritorna ConsoleStartView istanza singleton di `ConsoleStartView`.

getInteractionView

protected *InteractionView* **getInteractionView**()

getPlayerName

protected String **getPlayerName**(*PlayerFactoryRegistry registry*, boolean *isBreaker*)

main

public static void **main**(String[] *args*)

showLogo

protected void **showLogo**()

showNewGameStarting

protected void **showNewGameStarting**()

3.4.5 InteractionView

public abstract class **InteractionView** extends *BoardObserver*

Responsabilità: fornire ai giocatori coinvolti in una singola partita interazioni con quest'ultima.

Author Francesco Pio Stelluti, Francesco Coppola

Fields

currentSequenceLength

protected int **currentSequenceLength**
La lunghezza della sequenza da indovinare.

currentSequenceToGuess

protected List<*ColorPegs*> **currentSequenceToGuess**
La sequenza da indovinare.

lastAttemptAndClue

protected Map.Entry<List<*ColorPegs*>, List<*ColorPegs*>> **lastAttemptAndClue**
Singola entry di una mappa, contenente l'ultima lista di ColorPegs inseriti e la relativa sequenza indizio.

Methods

endingScreen

public abstract void **endingScreen**(String *gameEndingMessage*)
Interazione finale con il giocatore relativa al termine di una partita

Parametri

- **gameEndingMessage** – stringa con il messaggio finale da mostrare al giocatore

getCurrentSequenceLength

public int **getCurrentSequenceLength**()
Metodo getter che restituisce la lunghezza della sequenza da indovinare.

Ritorna int il valore intero di tale lunghezza

getCurrentSequenceToGuess

public List<*ColorPegs*> **getCurrentSequenceToGuess**()
Metodo getter che restituisce la sequenza da indovinare.

Ritorna List la lista di ColorPegs da indovinare

getIndexSequence

public abstract List<Integer> **getIndexSequence**(boolean *toGuess*)
Interazione con l'utente fisico o altra entità 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 interi da ottenere si riferisce alla sequenza da indovinare o meno

Ritorna List contenente gli indici da 1 a currentSequenceLength, associati all'enum ColorPegs

getLastAttemptAndClue

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

Metodo getter che restituisce la entry di mappa contenente l'ultima lista di ColorPegs inseriti e la relativa sequenza indizio.

Ritorna Map.Entry contenente l'ultima lista di ColorPegs inseriti e la relativa sequenza indizio.

3.4.6 StartStats

public class **StartStats**

Responsabilità: tenere traccia delle informazioni necessarie per poter iniziare una nuova partita.

Author Francesco Pio Stelluti, Francesco Coppola

Fields

highTresholdLength

int **highTresholdLength**

lowTresholdAttempts

int **lowTresholdAttempts**

lowTresholdLength

int **lowTresholdLength**

Constructors

StartStats

public **StartStats**()

Methods

getAttempts

public int **getAttempts**()

getBreakers

public *BreakerFactoryRegistry* **getBreakers**()

getCurrentBreaker

public *CodeBreaker* **getCurrentBreaker**()

getCurrentGame

public *SingleMatch* **getCurrentGame**()

getCurrentMaker

public *CodeMaker* **getCurrentMaker**()

getHighTresholdLength

public int **getHighTresholdLength**()

getIntView

public *InteractionView* **getIntView**()

getLowTresholdAttempts

public int **getLowTresholdAttempts**()

getLowTresholdLength

public int **getLowTresholdLength**()

getMakers

public *MakerFactoryRegistry* **getMakers**()

getNewGame

public *NewGameStats* **getNewGame**()

getSequenceLength

public int **getSequenceLength**()

isKeepSettings

public boolean **isKeepSettings**()

isToContinue

public boolean **isToContinue**()

resetLengthAttempts

public void **resetLengthAttempts**()
Vengono impostati i valori standard del numero di tentativi e della lunghezza delle sequenze

setAttempts

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

setBreakers

public void **setBreakers**(*BreakerFactoryRegistry* *breakers*)

setCurrentBreaker

public void **setCurrentBreaker**(*CodeBreaker* *currentBreaker*)

setCurrentGame

public void **setCurrentGame**(*SingleMatch* *currentGame*)

setCurrentMaker

public void **setCurrentMaker**(*CodeMaker* *currentMaker*)

setHighTresholdLength

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

setIntView

public void **setIntView**(*InteractionView* *intView*)

setKeepSettings

public void **setKeepSettings**(boolean *keepSettings*)

setLowTresholdAttempts

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

setLowTresholdLength

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

setMakers

public void **setMakers** (*MakerFactoryRegistry* *makers*)

setNewGame

public void **setNewGame** (*NewGameStats* *newGame*)

setSequenceLength

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

setToContinue

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

3.4.7 StartView

public abstract class **StartView**

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

Author Francesco Pio Stelluti, Francesco Coppola

Fields

startStats

protected *StartStats* **startStats**

Istanza della classe `StartStats`.

Constructors

StartView

public **StartView** ()

Methods

askNewAttempts

protected abstract int **askNewAttempts**()

Gestione dell'interazione con l'utente fisico per l'impostazione di un nuovo valore di numero di tentativi massimi richiesti al `CodeBreaker` all'interno della nuova partita.

Ritorna int numero di tentativi massimi richiesti al `CodeBreaker` all'interno della nuova partita.

askNewGameSettings

protected abstract *NewGameStats* **askNewGameSettings**()

Interazione con l'utente fisico a fronte della conclusione di una singola partita.

Ritorna NewGameStats contenente informazioni relative all'inizio di una nuova partita e alle impostazioni correlate.

askNewLength

protected abstract int **askNewLength**()

Gestione dell'interazione con l'utente fisico per l'impostazione di un nuovo valore della lunghezza delle sequenze di elementi presenti nella nuova partita.

Ritorna int valore della lunghezza delle sequenze di elementi presenti nella nuova partita.

askNewSettings

protected abstract boolean **askNewSettings**()

Gestione dell'interazione con l'utente fisico per l'impostazione o meno di nuove impostazioni relative alla nuova partita.

Ritorna boolean volontà dell'utente fisico di decidere nuove impostazioni per la nuova partita.

badEnding

protected abstract void **badEnding**(String *reason*)

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

Parametri

- **reason** –

ending

protected abstract void **ending**()

Gestione della conclusione dell'intero gioco dopo la fine di ogni singola partita.

getInteractionView

protected abstract *InteractionView* **getInteractionView**()
Ottenimento dell'oggetto `InteractionView` associato alla particolare implementazione di `StartView`.

Ritorna InteractionView associata all'oggetto `StartView`.

getPlayerName

protected abstract 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 un giocatore `CodeBreaker` o meno.

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

showLogo

protected abstract void **showLogo**()
Gestione del logo di avvio del gioco.

showNewGameStarting

protected abstract void **showNewGameStarting**()
Gestione del messaggio di avvio di una singola partita.

startUp

public void **startUp**()
Gestione completa dell'interazione con l'utente fisico per poter iniziare una nuova partita.

CAPITOLO 4

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.

4.1 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.

4.1.1 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).

testGetBoardReference

void **testGetBoardReference**()
Test method for *it.unicam.cs.pa.mastermind.gamecore.BoardController.getBoardReference()*.

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).

4.1.2 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).

4.1.3 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.players.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.players.PlayerFactoryRegistry.getFactoryByName(java.lang.String)`.

Solleva

- ***BadRegistryException*** –

testGetPlayersNames

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

Solleva

- ***BadRegistryException*** –

testMakerFactoryRegistry

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

Solleva

- ***BadRegistryException*** –

4.1.4 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()`*.

4.1.5 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)`.

4.1.6 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)`.

4.1.7 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)`.

4.1.8 SimulationGame

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

Author Francesco Pio Stelluti, Francesco Coppola

Methods

testSimulationGame

void **testSimulationGame**()

4.1.9 UIConsoleInteractionViewTest

class **UIConsoleInteractionViewTest**
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.ConsoleInteractionView.getInstance()*.

4.1.10 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

N

P

R

S

T

U

W

Y