ESAME DI FONDAMENTI DI INFORMATICA T-2 del 13/1/2021
Proff. E. Denti – R. Calegari – A. Molesini
Tempo a disposizione: 3 ore

NOME PROGETTO ECLIPSE: CognomeNome-matricola (es. RossiMario-0000123456)
NOME CARTELLA PROGETTO: CognomeNome-matricola (es. RossiMario-0000123456)
NOME ZIP DA CONSEGNARE: CognomeNome-matricola.zip (es. RossiMario-0000123456.zip)
NOME JAR DA CONSEGNARE: CognomeNome-matricola.jar (es. RossiMario-0000123456.jar)

Si devono consegnare DUE FILE: *l'intero progetto Eclipse* e *il JAR eseguibile*

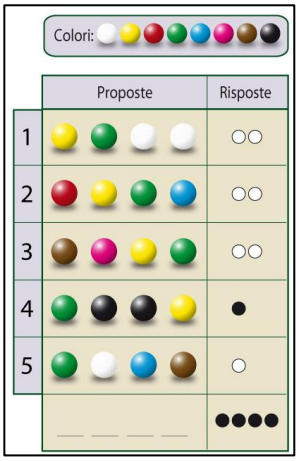
Si ricorda che compiti *non compilabili* o *palesemente lontani da 18/30* NON SARANNO CORRETTI e causeranno la verbalizzazione del giudizio "RESPINTO"

Si vuole sviluppare un'app per giocare a Master Mind (detto anche Codice Segreto).

DESCRIZIONE DEL DOMINIO DEL PROBLEMA

Master Mind è un gioco di abilità fra due giocatori, il *codificatore* e il *risolutore.* Preliminarmente, il codificatore deve inventare una *combinazione* di colori (il codice segreto) che il risolutore deve poi indovinare entro un prefissato *numero massimo* di *tentativi.*

A tal fine sono disponibili *piolini colorati* di diversi colori. I colori disponibili sono di norma 6, ma possono essere 8 o anche più nelle versioni per giocatori esperti; analogamente, la combinazione è di norma costituita da 4 piolini colorati, ma può essere di 5 nelle versioni per esperti. La figura a lato illustra la versione a 8 colori con combinazione da 4.



EVOLUZIONE DEL GIOCO

A ogni tentativo del risolutore, il codificatore risponde con una serie di *piolini risposta* bianchi o neri, collocati a lato del tentativo del risolutore, che assumono il seguente significato:

- Per ogni colore indovinato nella giusta posizione: un piolino nero
- Per ogni colore presente ma in posizione errata: un piolino bianco
- Per ogni colore non presente nella combinazione segreta: nessun piolino.

Pertanto la risposta può comprendere da 0 a 4 piolini, con un mix di bianchi e neri: 4 piolini neri equivalgono a combinazione indovinata, nel qual caso la vittoria è del risolutore; se invece questi non riesce a indovinare la combinazione segreta entro il prefissato numero massimo di tentativi, la vittoria va al codificatore.

L'ordine dei piolini di risposta non è significativo: conta unicamente la quantità di piolini neri e bianchi. Per questo, spesso si conviene di elencare prima i neri e poi i bianchi, così da favorire il confronto fra i tentativi e il ragionamento.
NB: il codificatore non ha limiti nell'uso dei colori, che possono essere anche ripetuti più volte.

CURIOSITÀ

Il gioco può essere fatto anche sostituendo ai colori dei numeri, o ai piolini risposta due simboli ("O"/"X", o altro); alcune varianti prevedono piolini risposta bianchi e rossi (anziché neri); alcuni giochi al computer usano rettangoli colorati invece di cerchi; etc. Un simulatore online è disponibile sul sito del CNR al link http://mastermind.itd.cnr.it/index.php.

ESEMPI

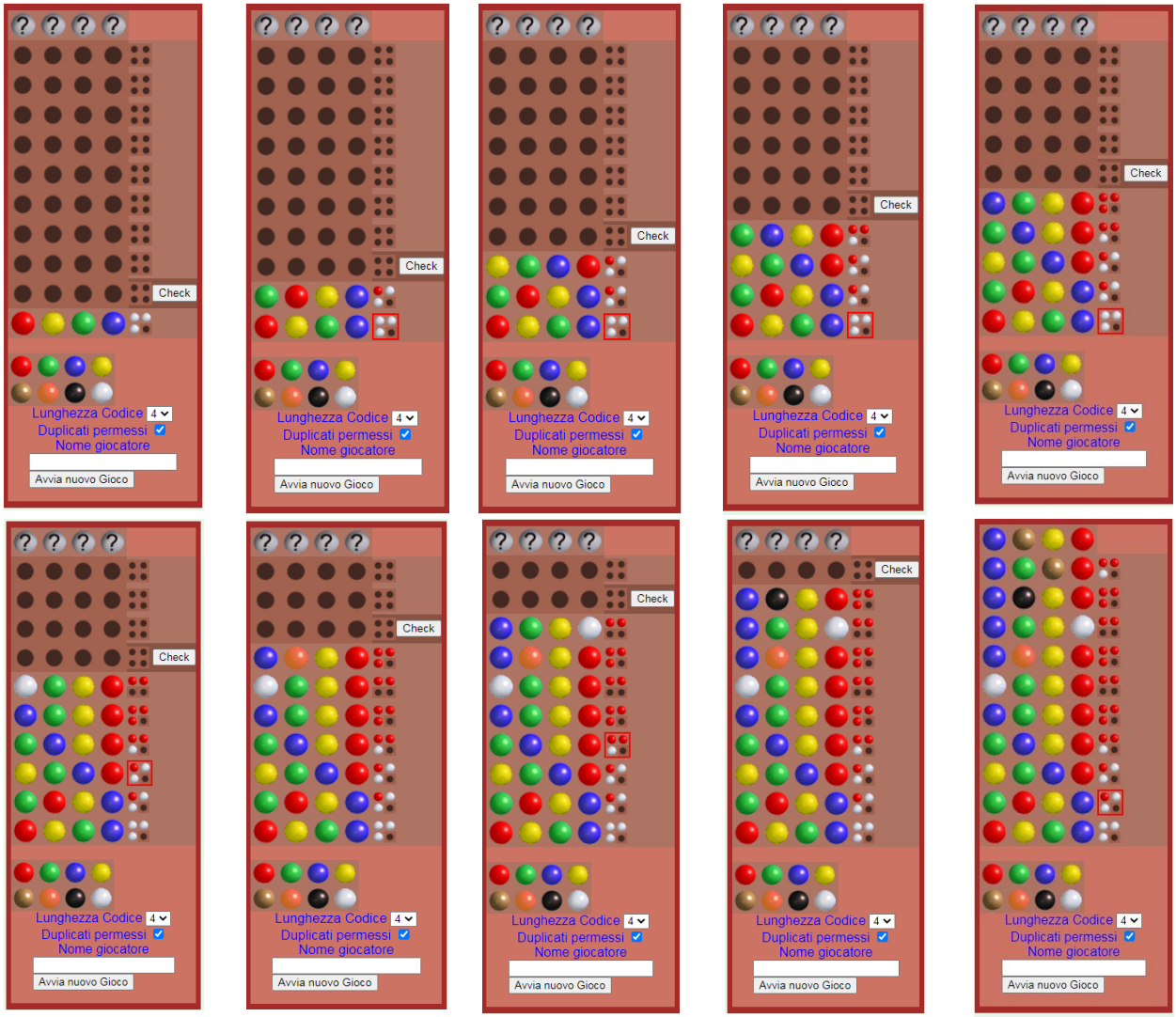
Si riportano di seguito alcune partite al simulatore, una persa dal risolutore, le altre vinte in divere situazioni: i piolini risposta in questo caso sono bianchi e rossi, in quanto il nero è usato per indicare il vuoto.

Nell'applicazione da sviluppare, per semplicità, la descrizione dei colori sarà testuale ("ROSSO", "VERDE", etc.) e lo stesso varrà per i piolini-risposta ("BIANCO", "NERO", "VUOTO"): in particolare si userà il termine "VUOTO" per indicare una casella senza piolini (v. oltre). Solo nella grafica si utilizzeranno combo con sfondo colorato (fornito), così da dare maggior gradevolezza al gioco (vedere figure in coda al testo).

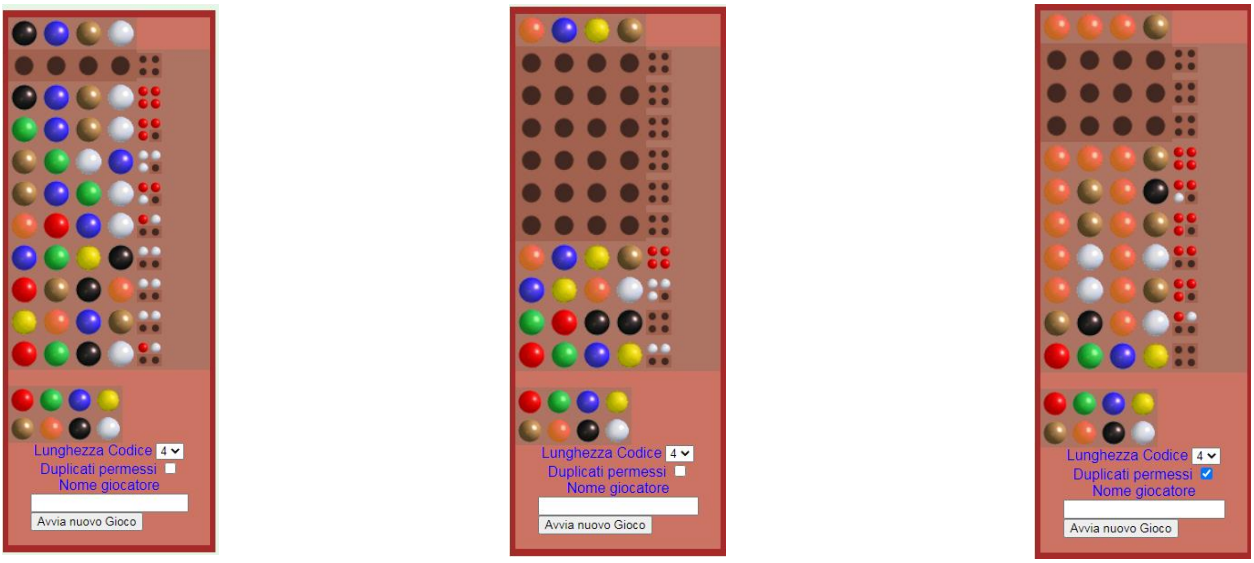
Cose da ricordare

- salva costantemente il tuo lavoro: l'informatica a volte può essere "subdolamente ostile"
- se ora compila e stai per fare modifiche, salva la versione attuale (non si sa mai…)

ESEMPIO PASSO-PASSO: partita persa dal risolutore



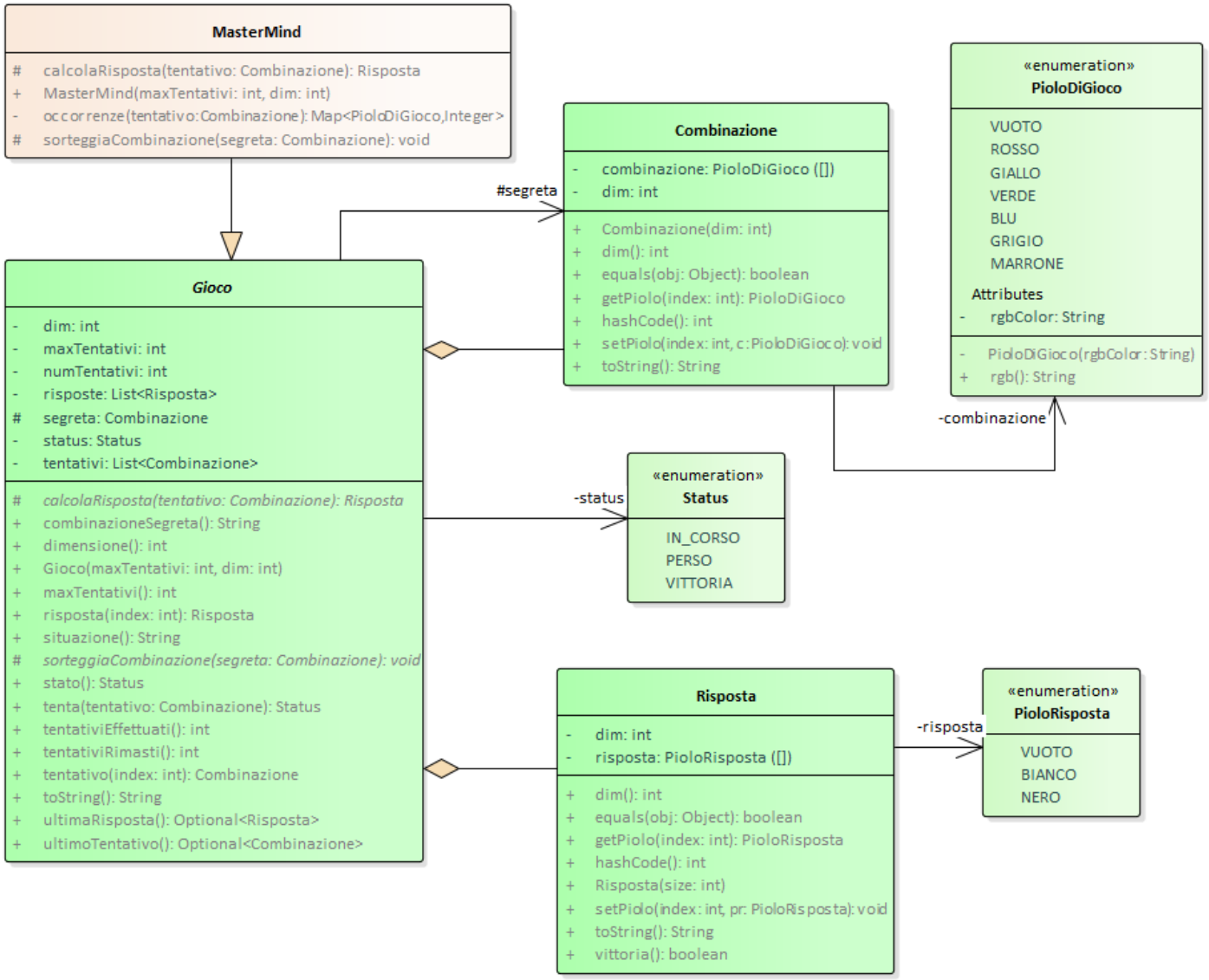
ALTRI ESEMPI: partite vinte dal risolutore



Parte 1 (punti: 18)

Dati (namespace mastermind.model) *(punti: 13)*

Il modello dei dati deve essere organizzato secondo il diagramma UML di seguito riportato:



SEMANTICA:

a) L'enumerativo ***PioloDiGioco*** (fornito) definisce i colori possibili: a ciascuno è associato il codice rgb, utilizzato nella grafica per colorare opportunamente lo sfondo delle caselle. A fini di test è previsto anche il colore "VUOTO".

b) L'enumerativo ***PioloRisposta*** (fornito) definisce i tre casi possibili – nero, bianco, vuoto.

c) L'enumerativo ***Status*** (fornito) definisce i tre stati possibili del gioco – in corso, partita persa, partita vinta.

d) La classe ***Combinazione*** (fornita) modella una combinazione di colori: il costruttore ne riceve la dimensione (ossia il numero di pioli-colore di cui è composta) e la inizializza con posizioni tutte vuote. Gli accessor ***setPiolo***/***getPiolo*** permettono di impostare l'i-esimo piolo della combinazione (numerati da 0 a N-1, da sinistra a destra). Il metodo ***toString*** emette la sequenza di colori separati da virgole, mentre ***equals*** ed ***hashcode*** sono definite in modo tale da consentire un facile confronto fra combinazioni.

e) La classe ***Risposta*** (fornita), analoga alla precedente, modella invece una risposta costituita da una sequenza di piolini neri, bianchi o vuoti. Come sopra, gli accessor ***setPiolo***/***getPiolo*** permettono di impostare l'i-esimo piolo-risposta della combinazione (numerati da 0 a N-1, da sinistra a destra). Il metodo ***toString*** emette la sequenza di pioli-risposta separati da virgole, mentre ***equals*** ed ***hashcode*** sono definite in modo tale da consentire un facile confronto fra combinazioni. Il metodo ***vittoria*** è vero se la risposta è costituita da tutti piolini neri.

f) La classe astratta ***Gioco*** (fornita) implementa lo schema generale di gioco: il costruttore riceve il numero massimo di tentativi e la dimensione della combinazione. Sono forniti molti metodi per recuperare il numero massimo di

tentativi, il numero di tentativi effettuati e quelli rimasti, il tentativo i-esimo e la risposta i-esima, l'ultimo tentativo e l'ultima risposta, lo stato del gioco (uno ***Status***) e la combinazione segreta. Metodi rilevanti:

- Il metodo ***situazione*** restituisce una stringa descrittiva dell'intera sequenza di tentativi e rispettive risposte, mentre ***toString*** completa tali informazioni con alcune stringhe descrittive e lo stato del gioco
- Il metodo ***tenta*** effettua una giocata con la binazione tentativo ricevuta come argomento: internamente calcola la risposta, quindi aggiorna e restituisce lo stato del gioco dopo tale giocata.

I due metodi astratti (protetti) ***sorteggiaCombinazione*** e ***calcolaRisposta*** astraggono la specifica logica di generazione della combinazione segreta e dell'algoritmo di calcolo della risposta.

g) La classe concreta **MasterMind** (da realizzare) deve specializzare ***Gioco*** nel caso di Mastermind, e pertanto:

- Il metodo ***sorteggiaCombinazione*** deve sorteggiare i colori richiesti (anche ripetuti)
- Il metodo ***calcolaRisposta*** deve calcolare la risposta secondo le regole di Master Mind descritte nel Dominio del Problema.

SUGGERIMENTO: per il calcolo dei piolini bianchi, che richiede attenzione in particolare in presenza di colori ripetuti, può convenire calcolare preliminarmente il totale dei piolini (bianchi+neri) della risposta, sottraendo poi quelli neri (facili da calcolare). A tal fine sono possibili diverse strategie, quali ad esempio:

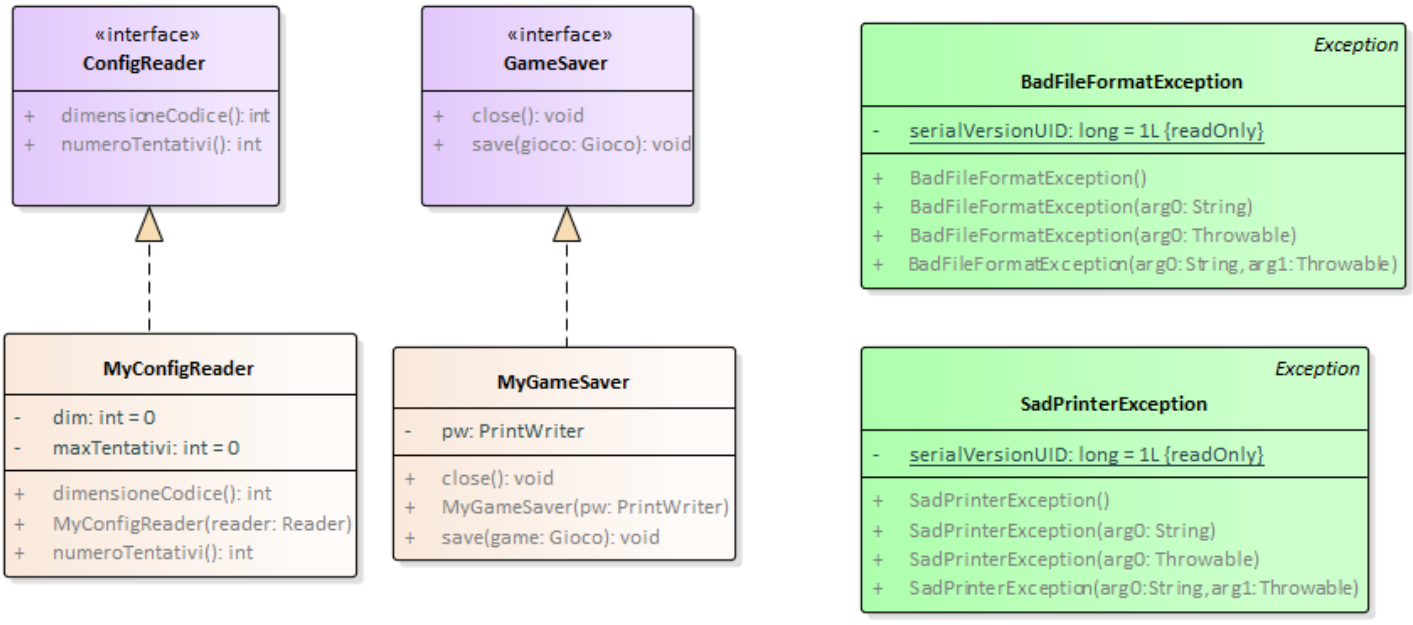
a) Operando su una copia della combinazione segreta, considerare un piolo per volta, svuotando via via quelli già considerati, in modo da non contarli più volte

b) Calcolare preventivamente il numero di occorrenze di ogni colore rispettivamente nella combinazione segreta e nel tentativo corrente, agendo poi per differenza.

Non vi sono invece, evidentemente, particolari difficoltà per il calcolo dei piolini neri.

Persistenza (namespace mastermind.persistence) *(punti: 5)*

Questo package definisce due componenti: ***GameSaver*** per stampare su file (*gameover.txt*) lo stato di una partita, e ***ConfigReader*** per leggere da un file di testo (*config.txt*) la configurazione iniziale del gioco (ovvero, la dimensione della combinazione segreta e il numero massimo di tentativi ammessi).



SEMANTICA:

a) l'interfaccia ***ConfigReader*** (fornita) dichiara i due metodi ***dimensioneCodice*** e ***numeroTentativi***, dall'ovvio significato

b) la classe **MyConfigReader** (da realizzare) implementa tale interfaccia: il costruttore riceve un ***Reader*** già aperto, da cui legge le due righe di configurazione, nel formato sotto specificato. L'ordine delle due righe non è prestabilito:

potrebbe esserci prima la riga relativa alla dimensione o prima quella relativa ai tentativi, non è dato sapere. È il costruttore a svolgere tutto il lavoro di lettura, memorizzando infine i due valori letti in due proprietà interne, che vengono poi restituite dai due metodi ***dimensioneCodice*** e ***numeroTentativi***. In caso di problemi di I/O o nel formato delle righe, il costruttore deve lanciare ***BadFileFormatException*** (fornita) con apposito messaggio d'errore.

FORMATO DEL FILE: ogni riga contiene una delle due dichiarazioni "Tentativi" o "Lunghezza combinazione", scritte con qualunque sequenza di maiuscole e/o minuscole, seguita dal carattere "=" e poi, dopo eventualmente spazi o tabulazioni intermedi, il valore intero. Da notare che fra le due parole "Lunghezza" e "combinazione" dev'esservi invece esattamente uno e un solo spazio, altrimenti la dichiarazione non sarà riconosciuta valida.

ESEMPI DI FILE LECITI:

```
Tentativi = 10
Lunghezza combinazione = 4
```

```
Tentativi                 = 10
Lunghezza combinazione    =  4
```

c) l'interfaccia ***GameSaver*** (fornita) dichiara i metodi ***save,*** che salva un ***Gioco***, e ***close***, che chiude il writer di uscita

d) la classe **MyGameSaver** (da realizzare) implementa tale interfaccia: il costruttore deve ricevere un ***PrintWriter***, che poi utilizza nei metodi ***save*** (per salvare lo stato attuale del gioco) e ***close*** (per chiudere il writer stesso).
NB: successive chiamate a ***save*** causano l'accodamento (append) delle diverse fasi del gioco nello stesso file.

Parte 2 (punti: 12)

Controller (namespace mastermind.ui)

Il controller (il cui UML è fornito sotto insieme alla UI) – articolato in interfaccia e implementazione – è fornito già pronto: il suo stato interno crea e gestisce il ***Gioco*** con tutto lo stato della partita, nonché il ***GameSaver*** da usare per i salvataggi su file. Conseguentemente, i suoi metodi fanno da ponte con entrambe tali entità, quasi sempre richiamando gli omonimi metodi di ***Gioco***. Da notare:

- il costruttore separa nel metodo accessorio ***init*** la logica di inizializzazione della partita, così da poterla più facilmente riutilizzare nel metodo ***restart***
- ***restart*** reinizializza lo stato del controller per una nuova partita (quindi nuovo ***Gioco*** e nuovo ***GameSaver***)

L'interfaccia ***Controller*** offre altresì il metodo statico ***alert*** per far comparire una finestra di dialogo utile a segnalare errori all'utente (in questa applicazione, solo nel caso in cui la stampa su file fallisca per qualche motivo).

GUI (namespace mastermind.ui) *(punti: 12)*

In caso di malfunzionamento del ConfigReader, usare come main MasterMindAppMock, che utilizza valori di configurazione di default anziché leggerli da file.

L'interfaccia grafica dev'essere simile (non necessariamente identica) a quella sotto illustrata (Fig.1): in alto sono presenti tante combobox quanta la dimensione della combinazione segreta specificata nel file di configurazione (nell'esempio, 4), tutte precaricate con i colori possibili ad esclusione del colore "VUOTO", che avendo solo fini di test non deve essere incluso. In basso, la barra di stato indica il numero massimo di tentativi, quelli rimasti e lo stato del gioco.

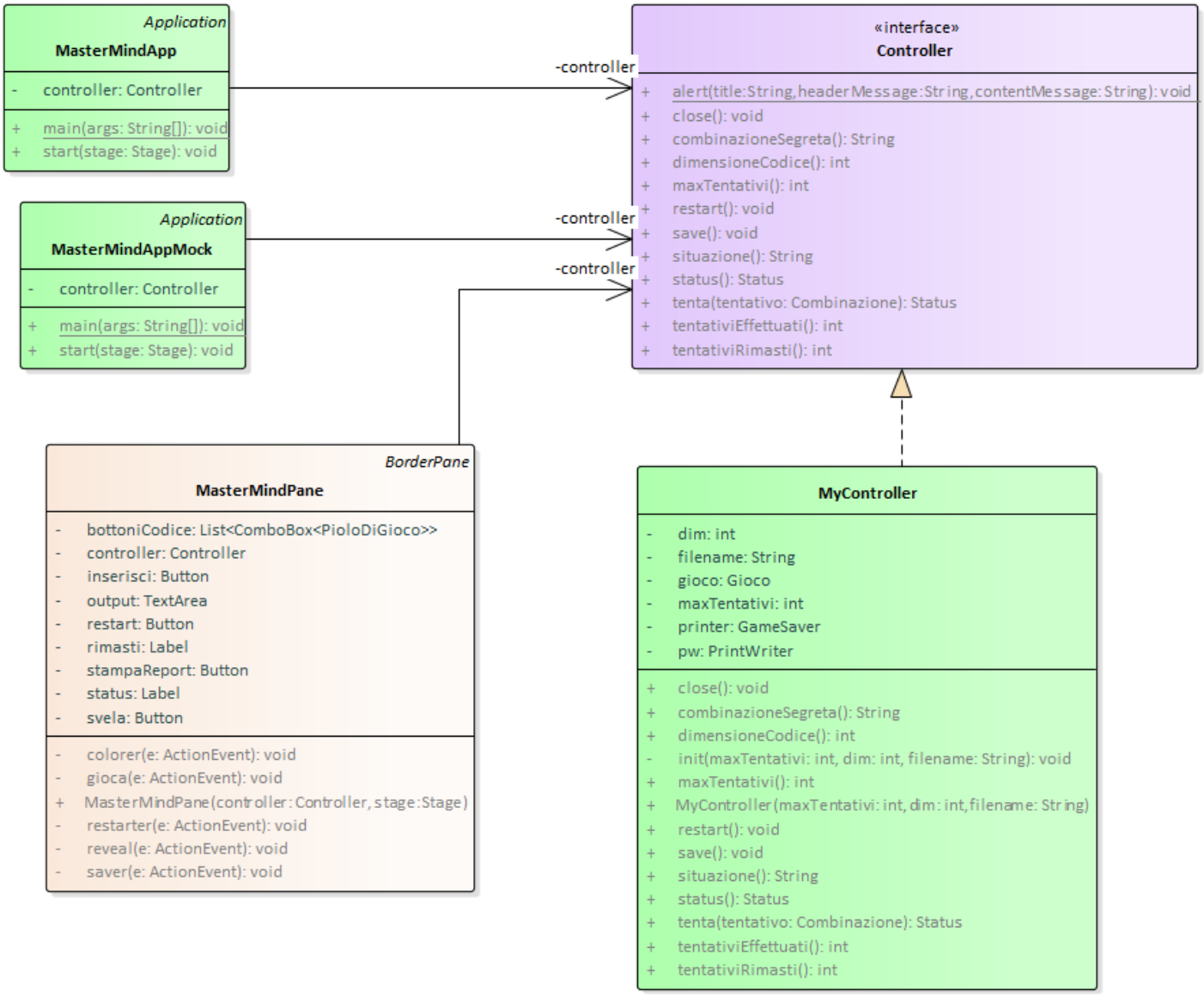
Per effettuare una giocata occorre in primis selezionare la nuova combinazione di tentativo tramite le combo (Fig.2), indi premere il pulsante "Inserisci" (Fig.3): ciò causa il calcolo dalla risposta e la visualizzazione della situazione aggiornata. Analogamente si procedere per le giocate successive (Figg.4,5,6): se, entro il numero massimo di tentativi, il solutore indovina la combinazione segreta, la partita è vinta e lo stato aggiornato opportunamente (Fig.6).

In qualunque momento:

- il tasto "Svela", sulla sinistra, consente di svelare la combinazione segreta (Fig. 7). Senza però interrompere il gioco, che prosegue comunque regolarmente
- il tasto "Restart", sulla sinistra, riporta tutto allo stato iniziale, per fare una nuova partita

- il tasto “Stampa report”, sulla sinistra, salva sul file di testo “gameover.txt” lo stato attuale del gioco.

Se si preme il tasto “Inserisci” senza aver prima selezionato tutti i colori, compare la finestra di Alert con opportuno messaggio di errore (Fig. 8).



a) La classe ***MasterMindApp*** (fornita) contiene il main di partenza dell’applicazione
(la sua variante ***MasterMindAppMock*** bypassa il reader configurando la GUI con valori di default)

b) La classe ***MasterMindPane*** (da completare) è una specializzazione di ***BorderPane***:

- in alto, contiene una lista di Combobox opportunamente popolate e, sotto di esse, il pulsante “Inserisci”
- a sinistra, i tre pulsanti di controllo
- a destra, l’area di uscita, dove viene costantemente visualizzata la situazione del gioco
- in basso, la barra di stato con le informazioni sullo stato del gioco.

Nello scheletro fornito sono già presenti alcune parti, e precisamente:

- il metodo ***colorer***, per la gestione del colore di sfondo delle combo
- il metodo ***restarter***, per la gestione del pulsante **Restart**

mentre vanno completati il costruttore e i metodi ***gioca***, ***reveal*** e ***saver*** che gestiscono rispettivamente i pulsanti **Inserisci**, **Svela** e **Stampa Report**.

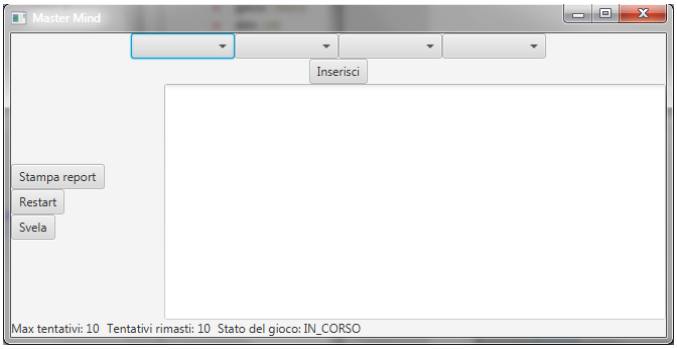


Figura 1

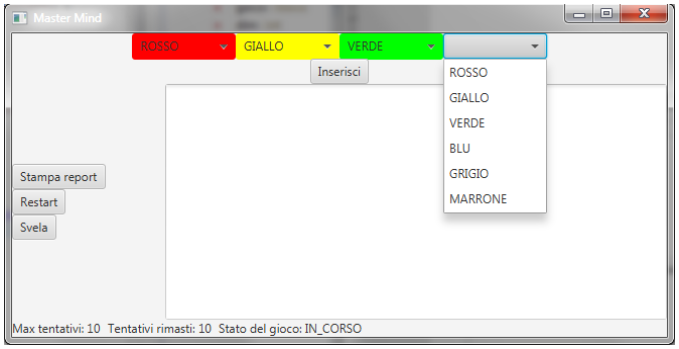


Figura 2

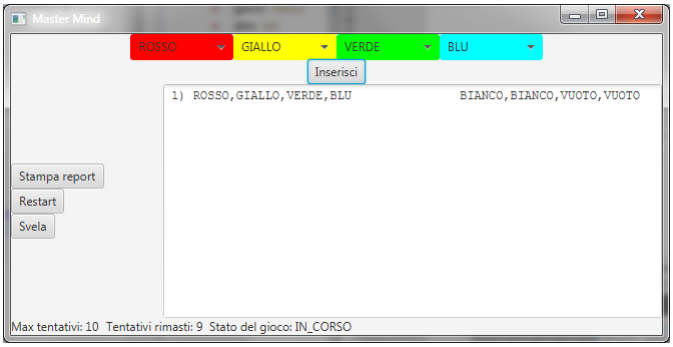


Figura 3

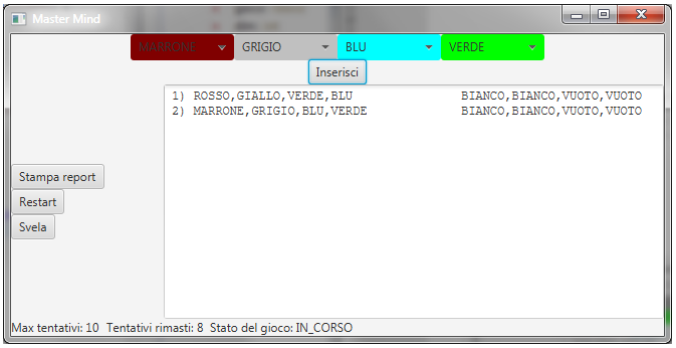


Figura 4

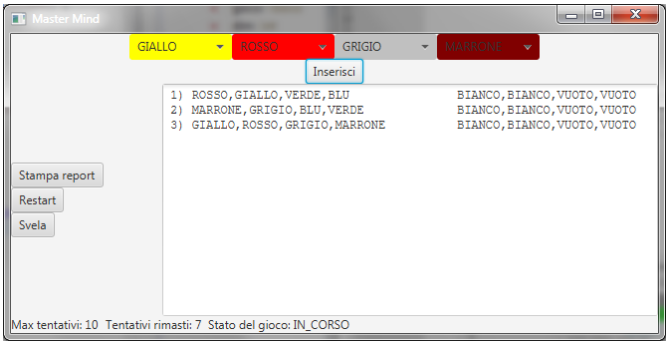


Figura 5

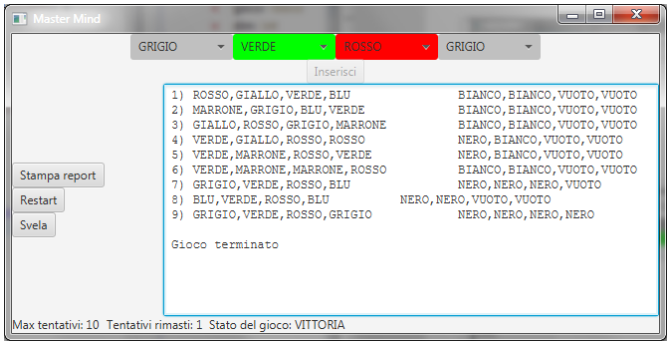


Figura 6

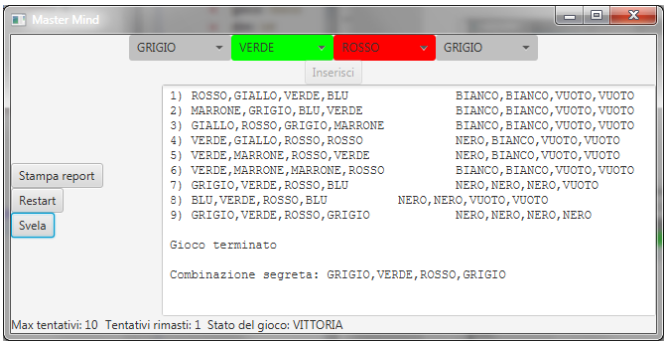


Figura 7

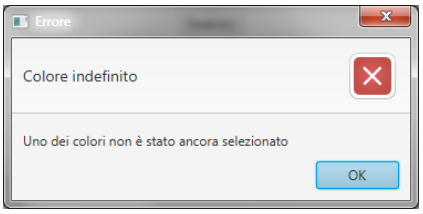


Figura 8

Checklist di consegna

- Hai fatto un **JAR eseguibile**, che contenga cioè l'indicazione del main?
- Hai controllato che **si compili** e **ci sia tutto**? [NB: non includere il PDF del testo]
- Hai **rinominato** IL PROGETTO, lo ZIP e il JAR esattamente come richiesto?
- Hai **chiamato** la cartella del progetto esattamente come richiesto?
- Hai fatto un **unico file ZIP (NON .7z, rar o altri formati)** contenente l'intero progetto? In particolare, ti sei assicurato di aver incluso tutti i file .java (e non solo i .class)?
- **Hai consegnato DUE file distinti, ossia lo ZIP col progetto e il JAR eseguibile?**
- Su EOL, hai **premuto** il tasto "CONFERMA" per inviare il tuo elaborato?