ESAME DI FONDAMENTI DI INFORMATICA T-2 del 5/02/2013
Proff. E. Denti – G. Zannoni
Tempo a disposizione: 4 ore MAX

NB: il candidato troverà nell'archivio ZIP scaricato da Esamix anche il software "Start Kit"

NOME PROGETTO ECLIPSE: CognomeNome-matricola (es. RossiMario-0000123456)

Un editore di giochi on line ha richiesto lo sviluppo di un'applicazione per il classico gioco *Master Mind*.

DESCRIZIONE DEL DOMINIO DEL PROBLEMA.

Il gioco Master Mind consiste nell'indovinare un codice segreto rappresentato da 4 piolini colorati, che possono avere diversi colori. Si distinguono due ruoli:

- il codificatore, che inventa il codice segreto e a ogni turno dà allo sfidante una "risposta";
- lo sfidante, che cerca di indovinare il codice segreto per tentativi successivi, basandosi sulla "risposta" data dal codificatore a ogni suo tentativo.

Inizialmente, il codificatore inventa un codice, all'insaputa dello sfidante; tale codice può comprendere anche colori ripetuti (al limite anche tutti uguali).

Lo sfidante inizia quindi il gioco, proponendo un codice (presumibilmente casuale). Il codificatore deve rispondere indicando quanti piolini sono stati indovinati esattamente sia come colore che come posizione, e quanti invece sono stati indovinati solo come colore, ma non come posizione. A tal fine, il codificatore dispone di chiodini bianchi e neri, che utilizza come segue:

- un chiodino nero per ogni piolino colorato indovinato esattamente sia come colore che come posizione;
- un chiodino bianco per ogni piolino colorato indovinato come colore ma NON come posizione.

Ovviamente, i piolini già contati come "neri" non vengono ri-contati come "bianchi". Inoltre, l'ordine dei chiodini-risposta non ha alcun significato particolare.

Lo sfidante tenterà quindi di proporre un nuovo codice, facendo tesoro delle risposte precedenti. Il gioco termina o quando lo sfidante indovina esattamente il codice segreto, o al più dopo 10 tentativi: in questo caso, se il codice non è stato indovinato, la vittoria è del codificatore.

Esempi (usando numeri anziché colori per comodità di rappresentazione)

Numero segreto: 0482
1° Tentativo: 1**2**3**4** - Risposta: 2 chiodini bianchi
2° Tentativo: 56**2**1 - Risposta: 1 chiodino bianco
3° tentativo: 719**8** - Risposta: 1 chiodino bianco
4° tentativo: **2**7**0**3 - Risposta: 2 chiodini bianchi
5° tentativo: **8042** - Risposta: 1 chiodino nero + 3 bianchi
6° tentativo: **0482** - Risposta: 4 chiodini neri = VITTORIA

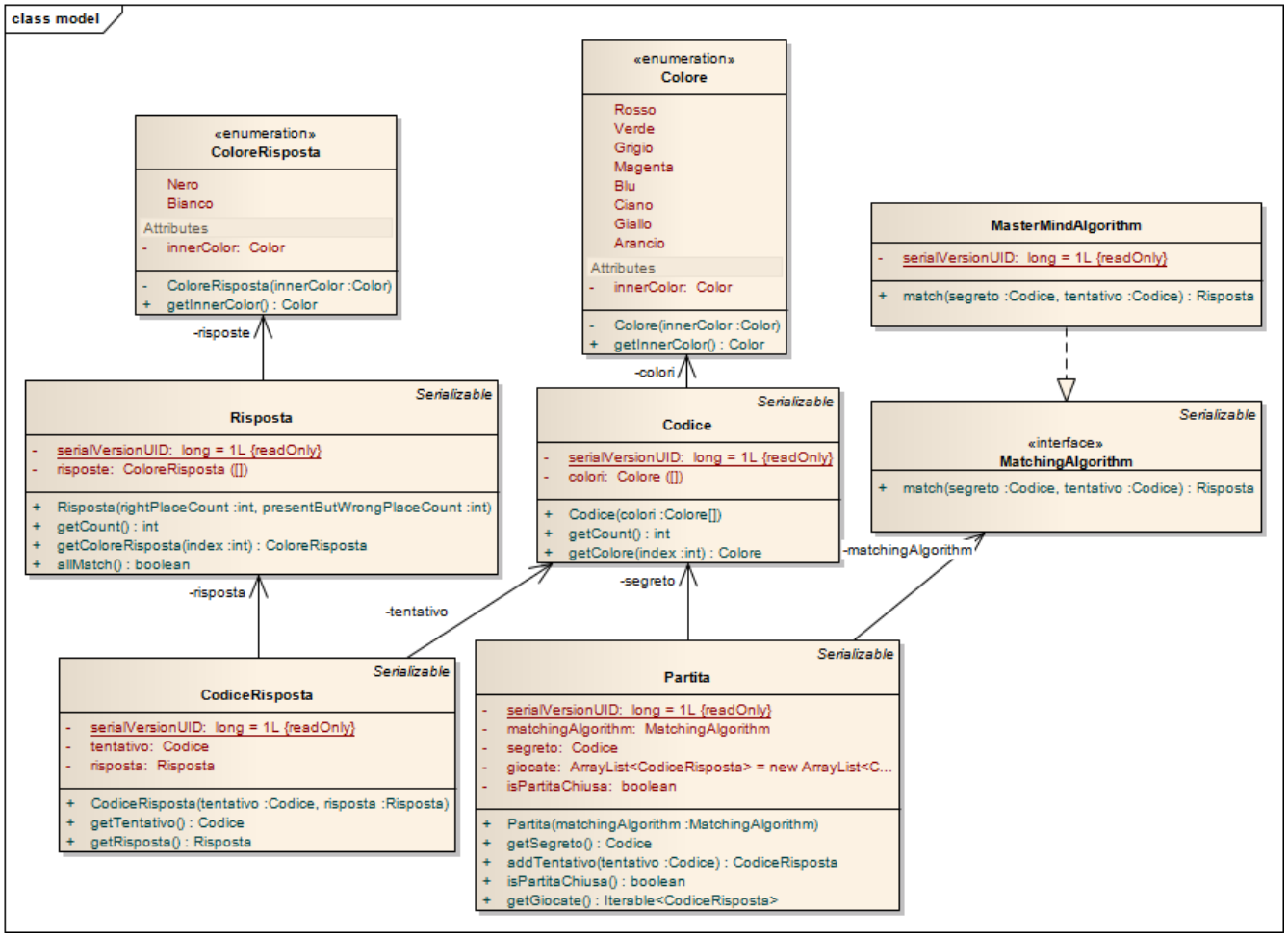
Numero segreto: 9437
1° tentativo: **9**8**7**4 - Risposta: 1 chiodino nero + 2 bianchi
2° tentativo: **9**18**7** - Risposta: 2 chiodini neri
3° tentativo: **9**2**4**8 - Risposta: 1 chiodino nero + 1 bianco
4° tentativo: **9437** - Risposta: 4 chiodini neri = VITTORIA

Il sistema si appoggia al file binario Partita.dat per catturare lo stato di una partita e al file Partita.txt per rappresentare in formato leggibile una partita conclusa (per il formato si veda più oltre).

Parte 1 (punti: 17)

Dati (namespace mastermind.model) *(punti: 10)*

Il modello dei dati deve essere organizzato secondo il diagramma UML più sotto riportato.



SEMANTICA:

a) la classe ***Configuration*** (fornita nello start kit) contiene le costanti ***LunghezzaCodice*** che determina la lunghezza del codice (default: 4) e ***GiocateMassime*** che determina il numero massimo di tentativi dello sfidante; al fine di mantenere il sistema parametrico è indispensabile utilizzare tali costanti evitando di codificare numeri magici.

b) la classe ***Codice*** (fornita nello start kit) rappresenta un codice di 4 ***Colori***;

c) l'enumerativo ***Colore*** (fornita) rappresenta i colori disponibili;

d) la classe ***Risposta*** (fornita nello start kit) rappresenta una risposta del codificatore allo sfidante, in termini di array di 0-4 ***ColoreRisposta***; il costruttore prende in ingresso il numero di colori indovinati che si trovano nella posizione corretta e il numero di colori indovinati che si trovano nella posizione sbagliata, il metodo ***allMatch*** permette di verificare se la risposta data è vincente (tutti neri);

e) l'enumerativo ***ColoreRisposta*** (fornito) rappresenta i colori dei piolini di risposta (tipicamente, ***Bianco*** e ***Nero***);

f) l'interfaccia ***MatchingAlgorithm*** (fornita nello start kit) cattura la struttura generale di un algoritmo di confronto fra un codice di tentativo e quello effettivo, astraendo dallo specifico algoritmo di matching usato; il metodo ***match*** restituisce un oggetto ***Risposta*** che incapsula la risposta del codificatore;

g) la classe ***MasterMindAlgorithm*** **(da realizzare)** concretizza ***MatchingAlgorithm*** implementando il metodo ***match*** in modo da usare l'algoritmo descritto nella sezione "Dominio del problema"; la classe non lancia MAI eccezioni.

> SUGGERIMENTO: confrontare i due codici contando sia i colori coincidenti come posizione (piolini neri nella risposta) sia i colori genericamente presenti in ambo i codici (piolini bianchi + neri nella risposta), ottenendo i piolini bianchi per differenza.

h) La classe ***CodiceRisposta*** (fornita), rappresenta la coppia ⟨Codice tentato, Risposta⟩ dell'algoritmo di match;

i) La classe ***Partita*** (fornita) mantiene lo stato della partita:

- il costruttore prende in ingresso un ***MatchingAlgorithm*** e crea e memorizza un nuovo codice segreto (mediante il metodo statico ***GeneratoreCodice.creaCodice***);

- il metodo ***addTentativo*** prende in ingresso un ***Codice*** proposto dallo sfidante, ottiene una ***Risposta*** mediante l'algoritmo di ***match***, crea la coppia ***CodiceRisposta*** e la aggiunge alla lista delle giocate, verifica se la partita è terminata (***Risposta***.***allMatch***) o è stato raggiunto il numero massimo di giocate (***Configuration***.***GiocateMassime***), restituendo il ***CodiceRisposta*** precedentemente costruito;
- il metodo ***isPartitaChiusa*** indica se la partita è chiusa;
- il metodo ***getGiocate*** restituisce l'elenco delle coppie ***CodiceRisposta*** che rappresentano la partita.

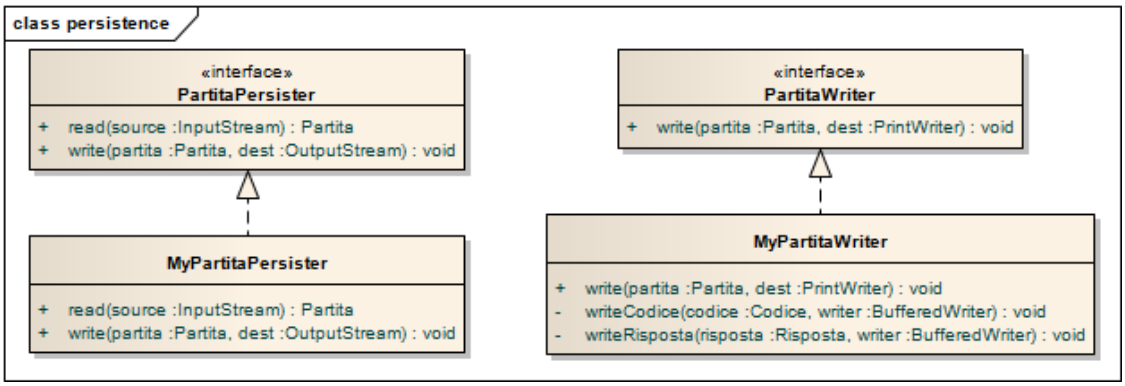
Lo Start Kit contiene anche i test (da includere nel progetto) per verificare il funzionamento di queste classi.

Persistenza (namespace mastermind.persistence) *(punti 7)*

Come anticipato, deve poter essere possibile salvare lo stato del gioco in un qualunque momento (tranne, ovviamente, a partita terminata), per riprenderlo poi in un momento successivo: a tal fine, lo stato del gioco, rappresentato da un'istanza di ***Partita***, viene salvato a richiesta nel file binario Partita.dat, da cui deve poter essere anche riletto a richiesta dell'utente. A fine partita, invece, l'intera storia della partita (compreso il codice segreto, ormai svelato) può essere salvata nel file di testo Partita.txt – una riga per il codice segreto, seguita da una linea **vuota**, poi una riga per ogni tentativo dello sfidante; ogni tentativo è espresso come sequenza di colori separati da virgole (stampare il ***toString*** dell'enumerativo).

L'interfaccia ***PartitaPersister*** dichiara i due metodi per leggere e scrivere il file binario. Questa classe essa dev'essere implementata dalla classe ***MyPartitaPersister*** **(da realizzare)** effettuando i necessari controlli sul contenuto del file binario, lanciando ***BadFileFormatException*** (fornita) quando opportuno.

L'interfaccia ***PartitaWriter*** dichiara il metodo per stampare su un apposito ***PrintWriter*** la storia della partita e deve essere implementata dalla classe ***MyPartitaWriter*** **(da realizzare)** come sopra specificato.



Lo Start Kit contiene anche i test (da includere nel progetto) per verificare il funzionamento di questa classe.

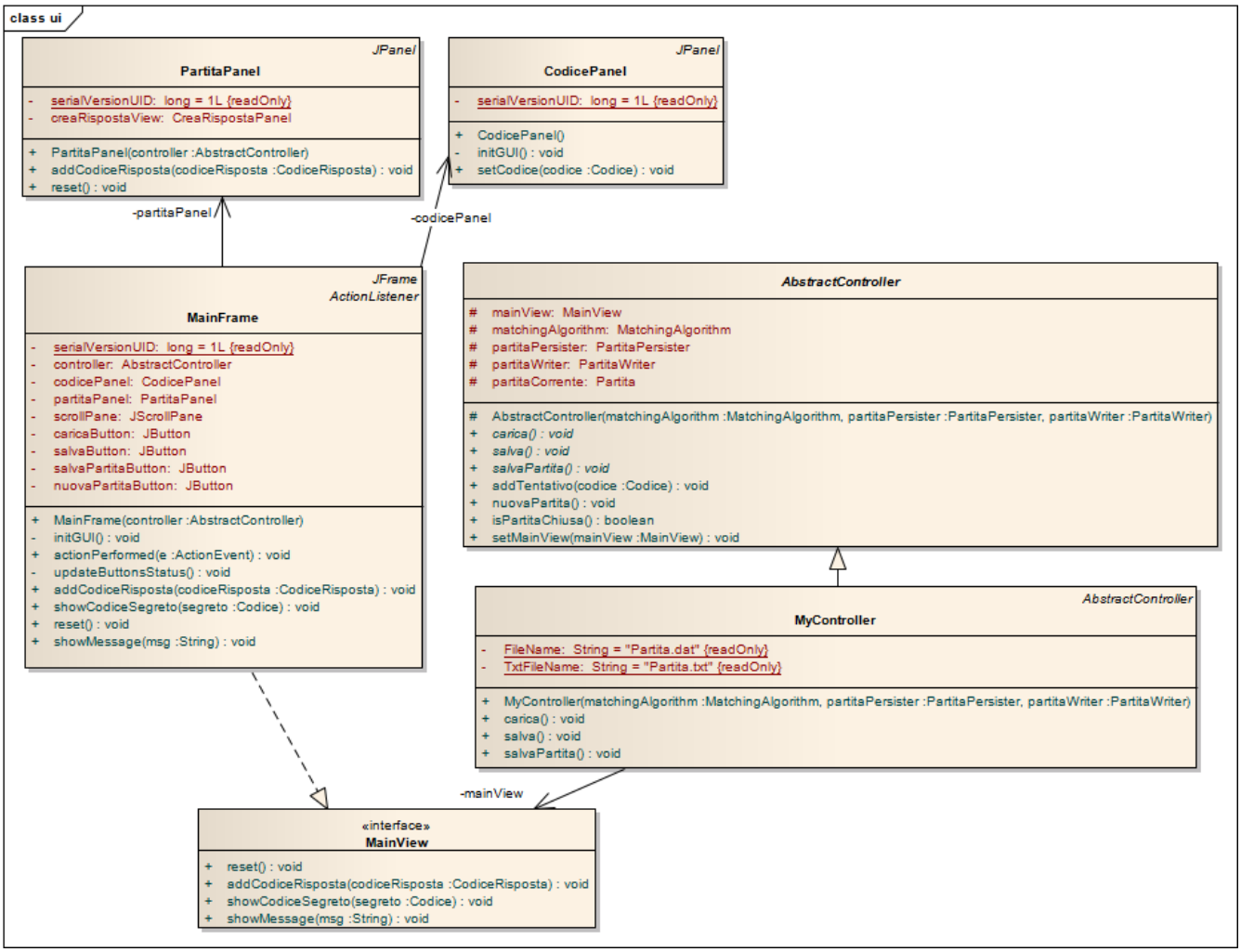
Parte 2 (punti: 13)

Controller (namespace mastermind.ui) *(punti 5)*

La classe ***MyController*** **(da realizzare)** concretizza la classe astratta ***AbstractController*** (fornita nello start kit).

- Il **costruttore (da realizzare)** prende in ingresso un'implementazione di ***MatchingAlgorithm***, un'implementazione di ***PartitaPersister*** e un'implementazione di ***PartitaWriter*** e invoca il costruttore della classe base con gli stessi parametri.
- Il metodo ereditato ***setMainView*** permette di impostare la vista sulla quale il controller lavora e dalla quale riceve i comandi; tale metodo memorizza la view ricevuta e ne invoca il metodo di reset.
- Il metodo ***carica*** **(da realizzare)** (eseguibile solamente se esiste una vista collegata) consente di caricare una partita lasciata in sospeso: esso carica la partita dal file binario mediante il ***PartitaPersister***, resetta la view e aggiunge alla view ogni giocata contenuta nella partita. In caso d'errore esso usa il metodo ***showMessage*** della view per avvisare l'utente dell'accaduto.

- Il metodo ***salva*** **(da realizzare)** (eseguibile solamente se esiste una vista collegata e se la partita non è chiusa) permette di salvare una partita lasciata in sospeso; la partita è salvata sul file binario mediante il ***PartitaPersister***. In caso d'errore usa il metodo ***showMessage*** della view per dare un feedback all'utente.
- Il metodo ***salvaPartita*** **(da realizzare)** (eseguibile solamente se esiste una vista collegata e se la partita è chiusa), salva la partita sul file di testo con l'ausilio del ***PartitaWriter***. In caso d'errore usa il metodo ***showMessage*** della view per avvisare l'utente.
- Il metodo ereditato ***addTentativo*** (eseguibile solamente se esiste una vista collegata e se la partita non è chiusa) prende in ingresso un ***Codice*** e aggiunge il tentativo alla partita corrente (metodo ***addTentativo*** di ***Codice***), ottenendo una coppia ***CodiceRisposta***; poi aggiunge la coppia ottenuta alla vista principale e, in caso di partita chiusa, mostra il codice segreto (***showCodiceSegreto*** sulla vista principale) ottenibile dalla Partita in corso.
- Il metodo ereditato ***nuovaPartita*** (eseguibile solamente se esiste una vista collegata) crea una nuova ***Partita***, la memorizza nel campo d'istanza opportuno e resetta la vista principale.
- Il metodo ereditato ***isPartitaChiusa*** restituisce il valore recuperato dal metodo omonimo nella ***Partita*** corrente.



Interfaccia utente (namespace mastermind.ui) *(punti 8)*

La classe ***Program*** (non mostrata nel diagramma UML, ma fornita nello start kit) contiene il ***main*** di partenza dell'intera applicazione.

L'interfaccia utente deve essere simile (non necessariamente identica) all'esempio mostrato in figura.

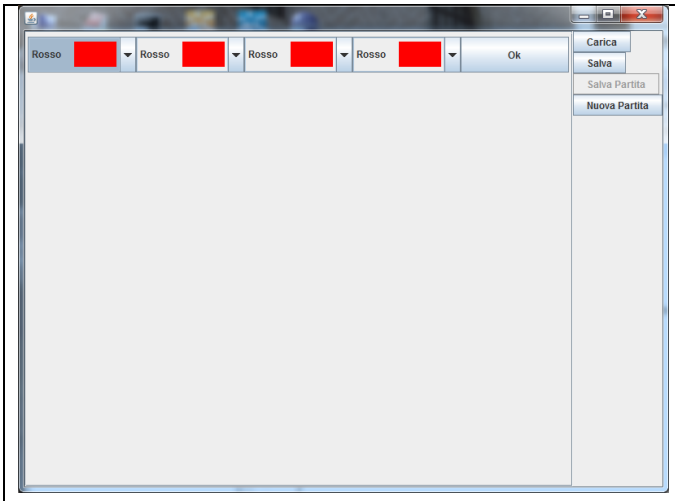


Figura 1: situazione iniziale

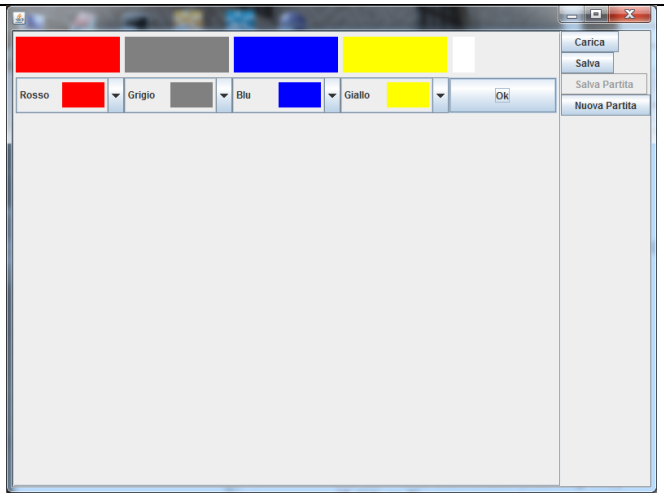


Figura 2: dopo il primo tentativo

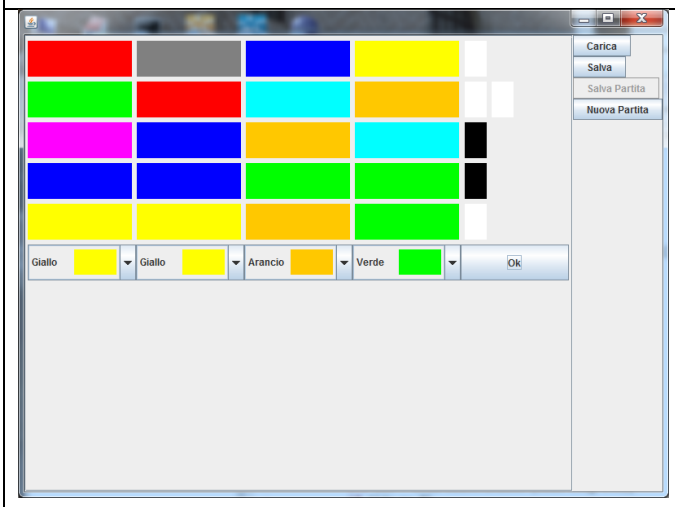


Figura 3: dopo cinque tentativi

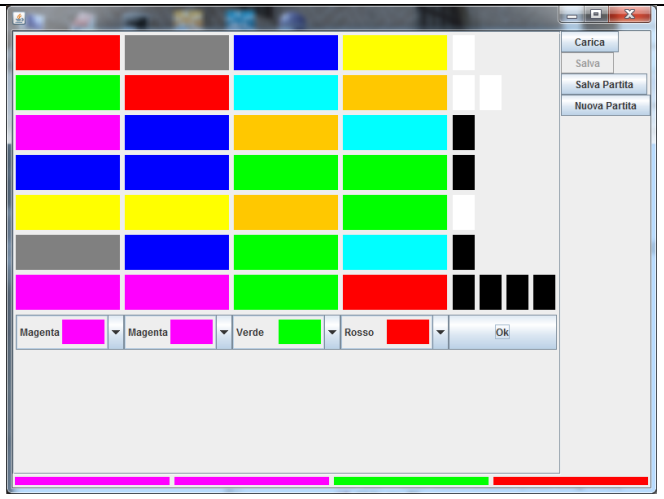


Figura 4: codice svelato dopo sette tentativi (vittoria)

La classe ***MainFrame*** **(da realizzare)** realizza la finestra principale (che deve essere realizzata in modo simile a quanto mostrato nella figura) e implementa l'interfaccia ***MainView*** (fornita nello start kit):

- sulla destra sono ospitati i bottoni **CARICA** (caricamento da file binario), **SALVA** (salvataggio su file binario), **SALVA PARTITA** (salvataggio su file di testo), **NUOVA PARTITA**: il gestore di tali bottoni deve invocare i corrispondenti metodi nel controller. Tali bottoni devono essere abilitati o disabilitati in modo consistente nei diversi momenti della partita, in base alla possibilità di eseguire o meno i metodi corrispondenti del controller; pertanto, alla pressione di un bottone occorre aggiornare lo stato di tutti i bottoni;
- in basso è ospitato un ***CodicePanel*** che consente, a fine partita, di vedere svelato il codice segreto; ***CodicePanel*** è dotato di un metodo ***setCodice*** che consente di impostare il ***Codice*** da visualizzare (se non si vuole visualizzare alcun codice, basta passare il valore ***null***);
- al centro, all'interno di un apposito scroll pane, è contenuto un ***PartitaPanel*** che consente di visualizzare ed effettuare le giocate; ***PartitaPanel*** è dotato del metodo ***addCodiceRisposta*** che consente di aggiungere una giocata e del metodo reset che consente di azzerare le giocate.

Poiché la classe ***MainFrame*** implementa l'interfaccia ***MainView***, deve implementare in modo opportuno i metodi di tale inferfaccia, ovvero:

- ***addCodiceRisposta*** aggiunge una giocata (***CodiceRisposta*** ricevuta come parametro) a ***PartitaPanel*** e aggiorna lo stato dei bottoni;
- ***showCodiceSegreto*** fa mostrare il codice segreto (***Codice*** ricevuto come parametro) al ***CodicePanel***;
- ***reset*** nasconde il ***CodicePanel*** (passa ***null*** al corrispondente metodo ***setCodice***) e resetta il ***PartitaPanel***;
- ***showMessage*** usa ***JOptionPane.showMessageDialog*** (si veda la documentazione) per mostrare un messaggio.