# ESAME DI FONDAMENTI DI INFORMATICA T-2 del 10/9/2019
## Proff. E. Denti, R. Calegari, A. Molesini – Tempo: 4 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)

**NB: l'archivio ZIP da consegnare deve contenere l'intero progetto Eclipse**

**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 *Campo Minato* (detto anche *Campo Fiorito*, *Mine Sweeper* o *Mines Sweeper*).

**DESCRIZIONE DEL DOMINIO DEL PROBLEMA**

Campo Minato è un solitario che unisce una componente di fortuna a una, prevalente, di abilità. Si gioca su una scacchiera NxN (la dimensione N dipende dal grado di difficoltà desiderato) su cui il computer dispone M mine ($M<<N^2$), in posizioni sconosciute al giocatore. Lo scopo del gioco è sminare il campo minato, rivelando via via tutte le caselle ma senza toccare mai quelle con le mine: se ciò malauguratamente accade, il giocatore salta per aria e la partita è persa. Se, invece, si riescono a rivelare tutte la caselle evitando quelle con le mine, il giocatore vince la partita.
Per guidare il giocatore nell'attività di sminamento, ogni casella senza mine contiene un *numero che indica quante mine ci sono nelle (max 8) caselle adiacenti*: spesso, al posto dello 0 la casella viene lasciata vuota. Nel caso si riveli una casella con zero mine adiacenti (ossia vuota), *vengono svelate anche le caselle numeriche a loro volta adiacenti a quest'ultima*, ricorsivamente, così da rendere più veloce e accattivante il gioco.

**ALGORITMO DI GIOCO**

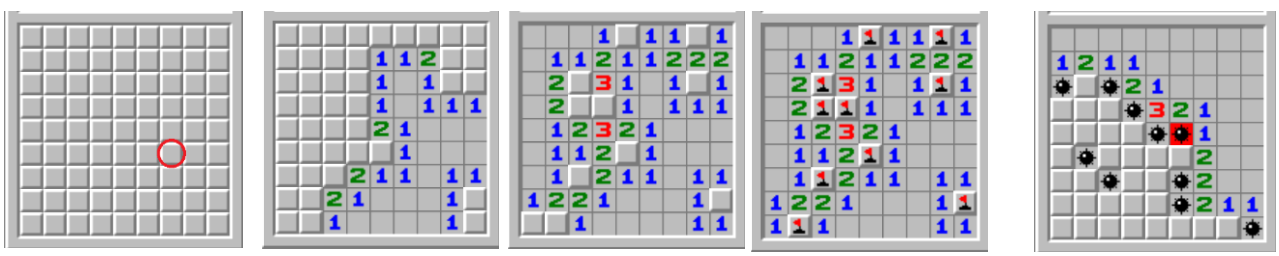
Inizialmente il giocatore svela (facendoci clic sopra) una casella, sperando di non incappare subito in una mina: qui chiaramente la componente fortuna è assoluta. Se la casella contiene un valore numerico, che indica il numero di mine adiacenti, il giocatore può sfruttare tale informazione per decidere quale casella svelare alla prossima mossa (ossia dove cliccare successivamente), operando con abilità. In ogni momento, se il giocatore malauguratamente svela una casella con una mina, salta per aria e perde la partita; se, invece, il giocatore riesce via via a svelare tutte le caselle evitando tutte quelle che contengono mine, vince la partita. Il numero M di mine è naturalmente noto a priori.

**ALGORITMO DI RIEMPIMENTO INIZIALE DELLA SCACCHIERA**

La parte più interessante del gioco è, paradossalmente, quella che il giocatore non vede – l'algoritmo con cui si calcolano i valori da porre nelle caselle numeriche (quelle che non contengono mine): essi indicano il numero di mine (da 0 a 8) presenti nelle (max 8) caselle adiacenti. A tal fine occorre considerare, per ogni cella di posizione (c,r), le caselle che la circondano, dalla riga r-1 alla riga r+1, e dalla colonna c-1 alla colonna c+1, avendo però cura di considerare i casi particolari relativi alla caselle nelle prime/ultime righe o colonne (che hanno ovviamente meno di 8 caselle adiacenti).

**ESEMPIO**

Si riportano di seguito alcuni screenshot di una partita su una scacchiera 9x9 con 10 mine. I primi quattro screenshot mostrano una partita vinta, l'ultimo – totalmente distinto dai precedenti – una partita persa. Nel primo caso, dopo varie mosse, nell'ultima azione (terzo screenshot) il giocatore svela l'ultima caselle senza mine: il sistema risponde concedendo la vittoria e mostrando (quarto screenshot), a conferma, la posizione di tutte le mine. Nel secondo caso invece il giocatore, mal interpretando le indicazioni numeriche, svela erroneamente una caselle con una mina (sfondo rosso) e salta per aria: la partita à persa e il sistema mostra la posizione delle altre mine, per completezza.

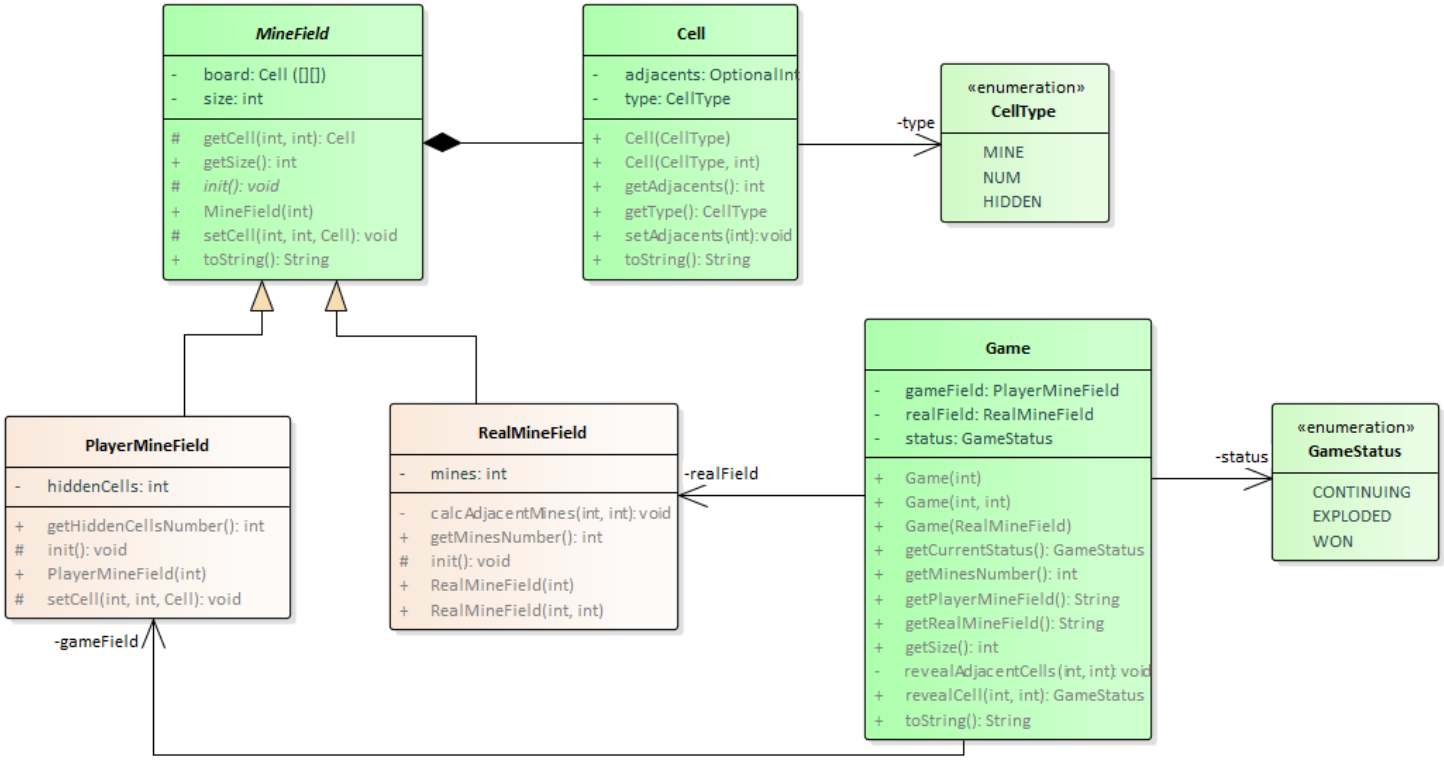


Nell'applicazione da sviluppare, per semplicità grafica, al posto delle icone delle mine si userà la lettera '**M**', mentre al posto delle caselle vuote si utilizzerà il carattere '**?**': inoltre, non si utilizzeranno i colori (vedere figure in coda al testo).

## Parte 1 (punti: 24)

### *Dati (namespace minesweeper.model)* *(punti: 17)*

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




SEMANTICA:

a) L'enumerativo ***CellType*** (fornito) definisce i tre stati possibili per ogni cella della scacchiera: MINE, NUM, HIDDEN. (Come si vedrà meglio nel seguito, la scacchiera nascosta creata dal computer ha solo celle di tipo MINE e NUM, mentre quella del giocatore ha inizialmente celle tutte HIDDEN, destinate a essere poi sostituite da celle NUM o MINE via via che le celle vengono svelate.)

b) La classe ***Cell*** (fornita) modella una cella della scacchiera: è caratterizzata dal suo tipo (***CellType***) e opzionalmente da un intero (utile nel solo caso in cui sia di tipo NUM). Il costruttore a singolo argomento è destinato principalmente a costruire celle di tipo MINE o HIDDEN, mentre quello a due argomenti è specifico per le celle di tipo NUM. Appositi metodi accessor (***getAdjacents***/***setAdjacents***) permettono di recuperare o impostare il valore numerico per le celle di tipo NUM, o di recuperarne il tipo (***getType***). Il metodo ***toString*** emette, come da specifica del dominio del problema, rispettivamente la stringa "**M**" per le celle di tipo MINE, "**?**" per le celle di tipo HIDDEN, e il valore numerico corrispondente per le celle di tipo NUM.

c) La classe astratta ***MineField*** (fornita) implementa una scacchiera di dimensione *size*: questa classe funge da base sia per la scacchiera del giocatore sia per quella del computer, implementate da due classi derivate (v. oltre). Il costruttore riceve la dimensione della scacchiera e alloca la corrispondente matrice *size* x *size* di ***Cell***.
Resta astratto solo il metodo (protetto) di inizializzazione ***init***, destinato a essere invocato dal costruttore delle sottoclassi per specializzare la logica di riempimento iniziale della scacchiera – che è diversa per la scacchiera nascosta del computer e per quella del giocatore (per questo tale metodo non è invocato nel costruttore di ***MineField***). Appositi accessor (***getCell***/***setCell***) permettono di recuperare o impostare la cella situata in una data posizione di riga e colonna, o la dimensione della scacchiera (***getSize***): il metodo ***toString*** produce una descrizione della scacchiera per righe, con le varie celle separate da tabulazioni e un fine riga dopo ogni riga.

d) La classe concreta **PlayerMineField** (da realizzare) specializza ***MineField*** nel caso della scacchiera del giocatore:

- poiché essa deve essere inizialmente tutta nascosta, il metodo di inizializzazione ***init*** deve riempirla tutta con celle di tipo HIDDEN.

- la classe deve definire e mantenere inoltre la proprietà *hiddenCellsNumber* – recuperabile con un apposito metodo accessor ***getHiddenCellsNumber*** – che rappresenta il numero di celle ancora nascoste (inizialmente tutte): tale valore dev'essere poi decrementato ogni volta che una cella HIDDEN viene sostituita da una cella MINE o NUM (tramite la versione specializzata del metodo ***setCell***, descritta al punto seguente)
- poiché nella scacchiera del giocatore l'unica azione di modifica prevista è la sostituzione di una cella HIDDEN con una di tipo NUM o MINE, il metodo ***setCell*** ereditato da ***MineField*** dev'essere sovrascritto da una versione specializzata, che:
    - richiami il metodo ***setCell*** ereditato
    - verifichi che non si stia sostituendo una cella HIDDEN con un'altra HIDDEN (cosa manifestamente assurda), lanciando in tal caso ***UnsupportedOperationException*** con apposito messaggio d'errore
    - infine, decrementi di 1 il valore della proprietà *hiddenCellsNumber*

      In tal modo, il gestore del gioco potrà facilmente identificare quando la partita sia terminata con successo (quando il numero di celle rimaste nascoste è uguale al numero di mine), o quando invece debba continuare (se il numero di celle rimaste nascoste è superiore al numero di mine).

e) la classe concreta **RealMineField** (da realizzare) specializza ***MineField*** nel caso della scacchiera del computer:
- il costruttore a un argomento accetta solo la dimensione della scacchiera (*size*), poi rimpalla l'opera sul costruttore a due argomenti, passando per default il valore 10 come numero di mine
- il costruttore a due argomenti accetta la dimensione della scacchiera (*size*) e il numero di mine (*mines*): dopo aver impostato tali proprietà, richiama il metodo ***init*** per effettuare il popolamento della scacchiera
- il metodo di inizializzazione ***init***
    - prima, sorteggia *mines* distinte posizioni – ossia coppie (riga, colonna) – in cui piazzare altrettante celle di tipo MINE
    - poi, percorre tutta la scacchiera e inizializza tutte le altre celle con celle di tipo NUM, avendo preventivamente cura di calcolare il numero di mine adiacenti a ciascuna, tramite il metodo privato ***calcAdjacentMines(row, col)***
- il metodo privato ausiliario ***calcAdjacentMines*** calcola il numero di mine adiacenti a una cella situata in una data posizione di riga (r) e colonna (c): a tal fine, percorre le (max 8) celle che la circondano, dalla riga r-1 alla riga r+1, e dalla colonna c-1 alla colonna c+1, contando le celle di tipo MINE. Nel farlo deve naturalmente tenere conto dei casi particolari in cui la cella data sia nella prima o ultima riga e/o colonna, nei quali ci sono ovviamente solo 5 o 3 celle adiacenti.
    - SUGGERIMENTO: si consiglia di determinare preventivamente l'escursione massima degli indici di riga (di base, da r-1 a r+1) e di colonna (di base, da c-1 a c+1) onde escludere la riga/colonna precedente nel caso la cella data sia sulla prima riga/colonna, e dualmente la riga/colonna successiva nel caso la cella data sia sull'ultima riga/colonna.
    - NB: naturalmente, nello scorrere gli indici, occorre evitare di considerare la cella stessa!
- Il metodo accessor ***getMinesNumber*** restituisce il numero di mine nella scacchiera

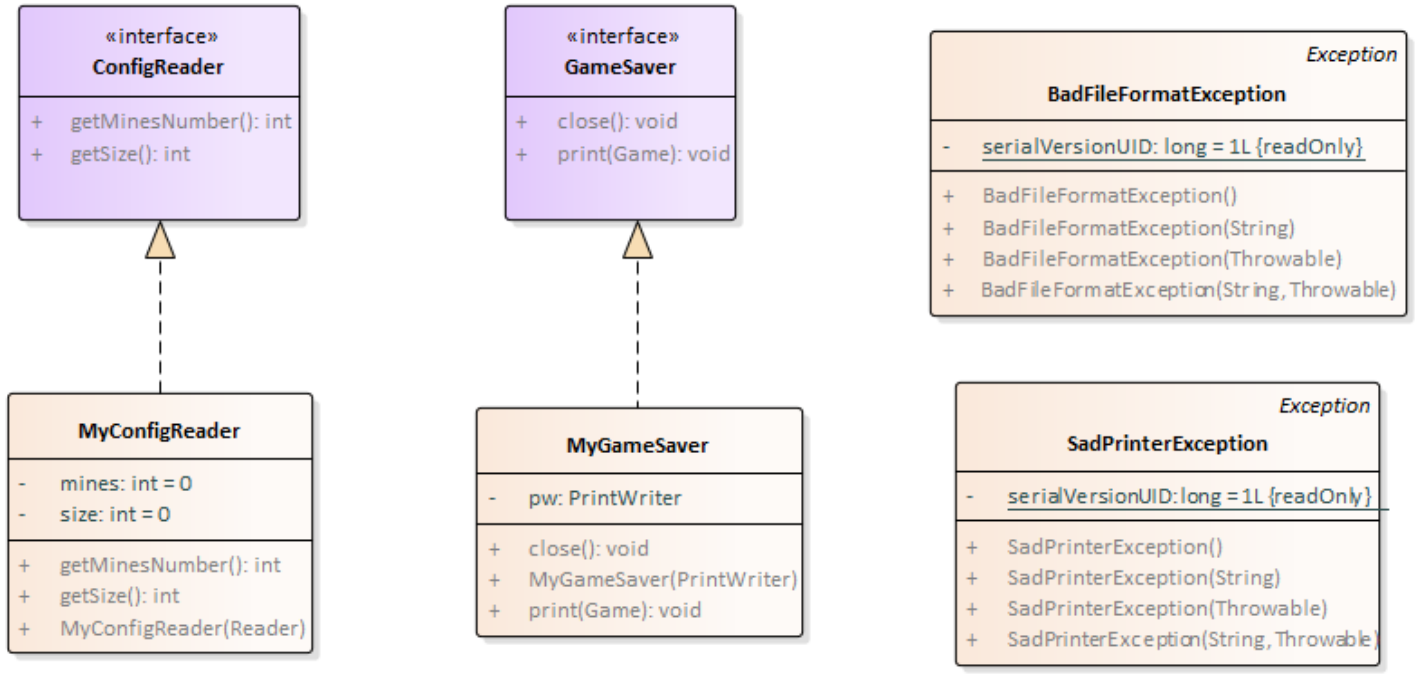
f) L'enumerativo ***GameStatus*** (fornito) definisce i tre stati possibili del gioco: CONTINUING, EXPLODED, WON.

g) La classe concreta ***Game*** (fornita) contiene tutta la logica di gioco:
- i costruttori creano le due scacchiere, una di tipo ***RealMinesField*** per il computer e una di tipo ***PlayerMinesField*** per il giocatore, e impostano lo stato iniziale del gioco a CONTINUING.
- Vari accessor consentono di recuperare lo stato attuale del gioco (***getCurrentStatus***), la dimensione delle scacchiere (***getSize***), il numero di mine (***getMinesNumber***) e lo stato attuale delle due scacchiere sotto forma di stringa (***getPlayerMineField*** e ***getRealMineField***).

- Il metodo cruciale, ***revealCell***, rivela la cella di coordinate (riga,colonna) specificate, recuperando la cella reale dalla scacchiera del computer e riportandola su quella del giocatore, e aggiornando quindi lo stato del gioco (EXPLODED se la cella rivelata era una mina, CONTINUING o WON in caso contrario). In ossequio alle regole, se la casella rivelata era numerica e conteneva uno 0, si procede ricorsivamente a rivelare anche le celle adiacenti a quest'ultima (tramite il metodo privato ***revealAdjacentCells***): se, dopo tale operazione, il numero di celle ancora nascoste è uguale al numero di mine, la partita è vinta e lo stato del gioco diventa WON: altrimenti, la partita deve continuare e lo stato è CONTINUING.
- Il metodo ***toString*** emette una descrizione completa dello stato attuale del gioco, riportando entrambe le scacchiere con opportune descrizioni.

### *Persistenza (namespace minesweeper.persistence)* *(punti: 7)*

Questo package definisce due componenti: il ***GameSaver*** per stampare su file (*gameover.txt*) lo stato di una partita, e il ***ConfigReader*** per leggere da file di testo (*config.txt*) la configurazione iniziale (dimensione scacchiera e numero mine).




SEMANTICA:

a) l'interfaccia ***ConfigReader*** (fornita) dichiara i due metodi ***getSize*** e ***getMinesNumber***, 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 alle mine, 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 ***getSize*** e ***getMinesNumber***. 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 DI CONFIGURAZIONE: ogni riga contiene una delle due parole "mines" o "size", scritte con qualunque sequenza di caratteri maiuscoli e/o minuscoli, seguita dal carattere "**:**" e poi, dopo eventualmente spazi o tabulazioni intermedi, il valore intero corrispondente.

ESEMPI DI FILE LECITI:

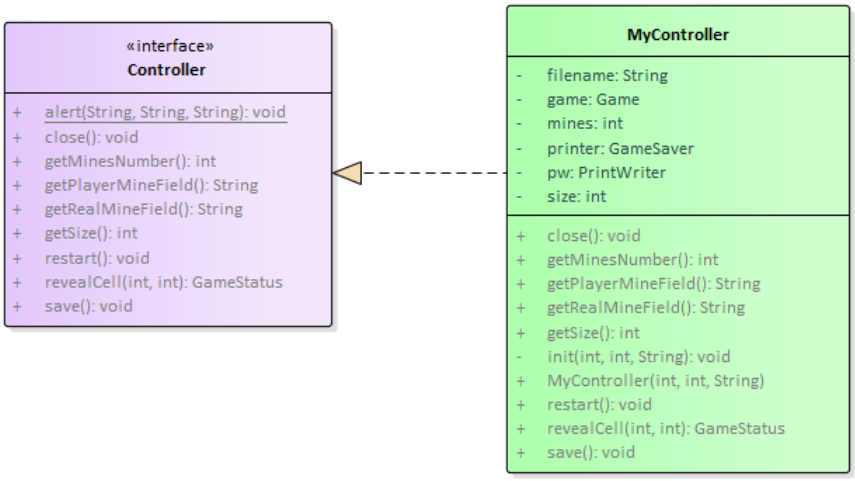
| | | |
|---|---|---|
| `MINES:  3`<br>`Size:  6` | `Mines: 3`<br>`Size:  6` | `size:  6`<br>`mines:3` |

c) l'interfaccia ***GameSaver*** (fornita) dichiara i metodi ***print,*** che stampa un ***Game***, 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 ***print*** (per stampare lo stato attuale del gioco) e ***close*** (per chiudere il writer stesso).
NB: successive chiamate a ***print*** causano l'accodamento (append) delle diverse fasi del gioco nello stesso file.

## Parte 2 (punti: 6)

### *Controller (namespace minesweeper.controller)*




Il controller – articolato in interfaccia e implementazione – è fornito già pronto: il suo stato interno crea e gestisce il ***Game*** con tutto lo stato della partita, nonché il ***GameSaver*** da usare per le stampe. Conseguentemente, i suoi metodi fanno da ponte con entrambe tali entità. In particolare:

- ***save*** e ***close*** richiamano i metodi ***print*** e ***close*** del ***GameSaver***
- svariati accessor rendono disponibili le proprietà utili del ***Game***, richiamando gli omonimi metodi: in particolare, ***revealCell*** richiama il corrispondente metodi di ***Game***, permettendo l'avanzare della partita
- ***restart*** reinizializza lo stato del controller per una nuova partita (quindi nuovo ***Game*** e nuovo ***GameSaver***)

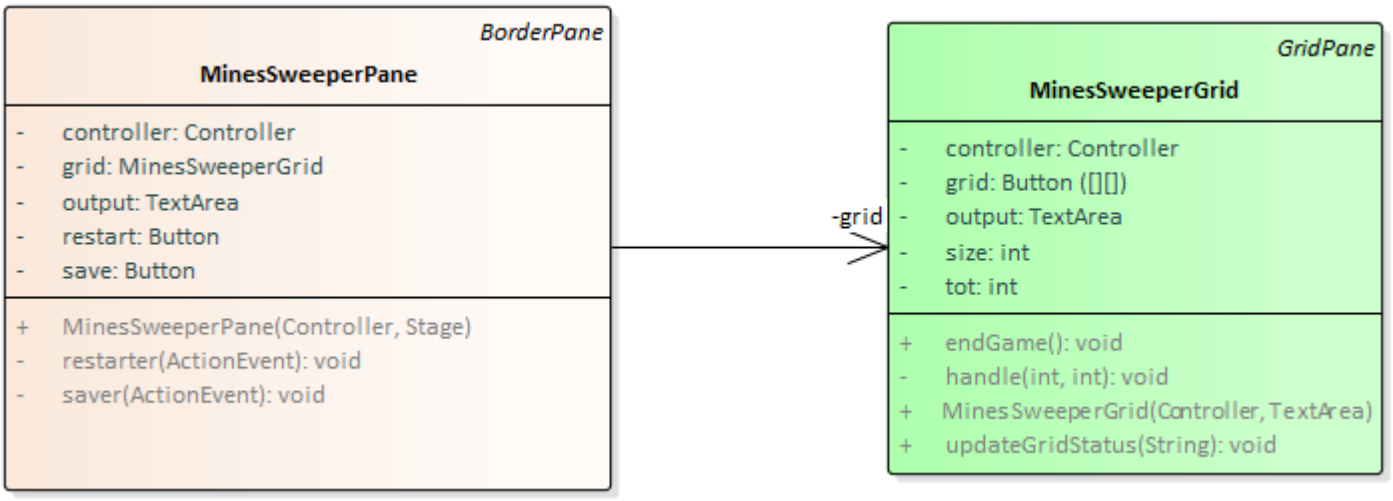
L'interfaccia ***Controller*** offre altresì il metodo statico ***alert*** per far comparire una finestra di dialogo utile a segnalare errori all'utente (in questa applicazione, solo nel caso in cui la stampa su file fallisca per qualche motivo).

### *GUI (namespace minesweeper.ui)* *(punti: 6)*

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

L'interfaccia grafica dev'essere simile (non necessariamente identica) a quella sotto illustrata.

All'inizio (Fig.1), la GUI mostra a sinistra la griglia del giocatore (in questo caso 5x5), a destra l'area di testo, sopra le mine (in questo caso 3) e sotto i pulsanti. Svelando una cella (Fig. 2), il sistema reagisce mostrando la cella cliccata e le le eventuali adiacenti (in questo caso no): l'area di testo ripete l'informazione e invita a continuare. Proseguendo con perizia, il giocatore arriva prima o poi a vincere la partita (Fig. 3): in tal caso tutte le celle vengono svelate e l'area di testo mostra, per conferma, anche la griglia reale del computer. Se, invece, il giocatore è sfortunato e clicca su una mina (Fig. 4, sfortunato alla primissima mossa!), il sistema reagisce disabilitando tutta la griglia, mostrando a lato la situazione attuale e la griglia reale, e dando il triste esito dell'esplosione dell'esploratore.




a) La classe ***MinesSweeperApp*** (fornita) contiene il main di partenza dell'applicazione.

b) La classe ***MinesSweeperGrid*** (fornita) contiene tutta la griglia completa di algoritmo di gioco

- il costruttore riceve il controller e la textarea (definita nel pane, v. oltre) su cui scrivere
- il metodo ***updateGridStatus*** aggiorna la visualizzazione della griglia in base alla stringa ricevuta come argomento (che si suppone nel formato restituito dalla ***toString*** di ***MineField***)
- il metodo ***endGame*** termina la partita, disabilitando tutti i pulsanti della griglia e richiamando poi i metodi ***save*** e ***close*** del controller per salvare lo stato finale della partita.

c) La classe ***MinesSweeperPane*** (da realizzare) è una specializzazione di ***BorderPane***:

- in alto, una label indica al giocatore il numero di mine
- al centro, una ***MinesSweeperGrid*** costituisce il pannello di gioco
- in basso, due pulsanti *"Save status"* e *"Restart"*, agganciati a due distinti metodi di gestione, consentono rispettivamente di salvare lo stato attuale del gioco e di resettarlo per fare una nuova partita.

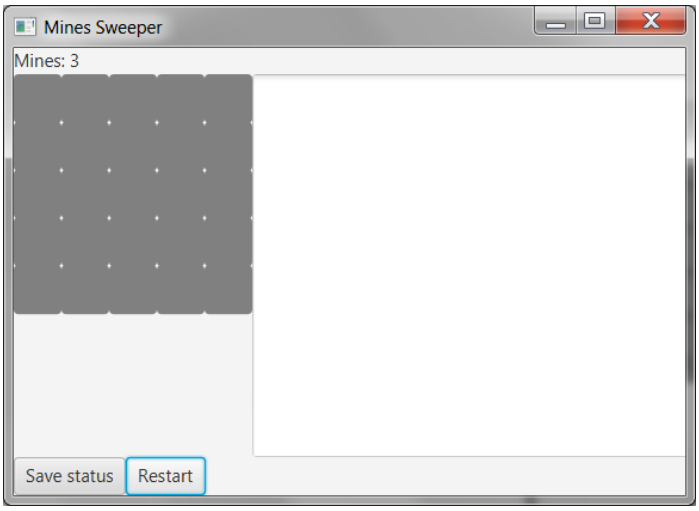



Figura 1

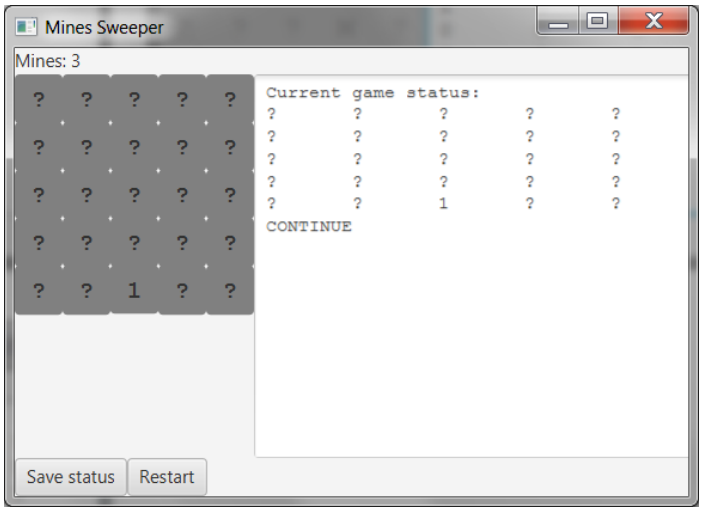



Figura 2

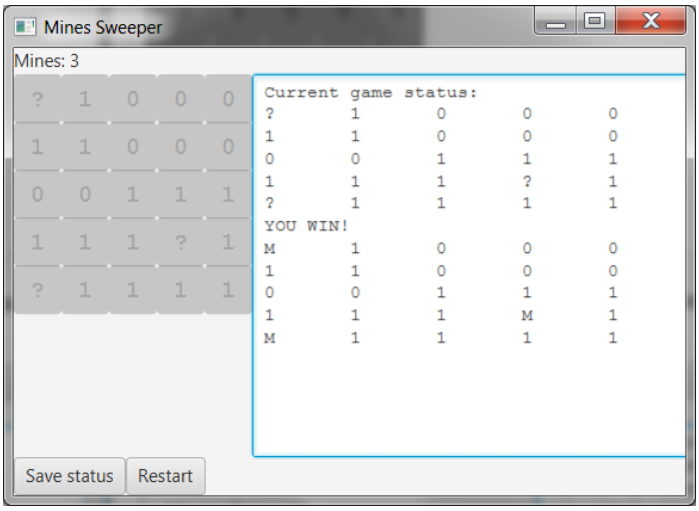



Figura 3

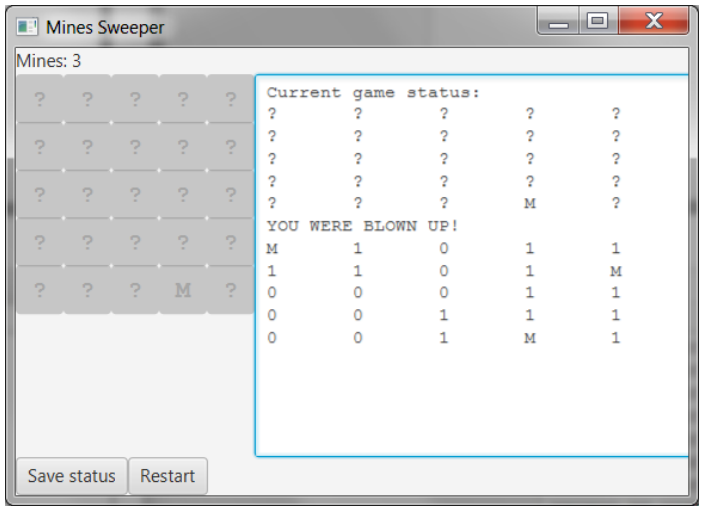



Figura 4

**Cose da ricordare**

- salva costantemente il tuo lavoro, facendo ZIP parziali e consegne parziali (vale l'ultima)
- in particolare, se ora compila e stai per fare modifiche, salva la versione attuale (non si sa mai..)

**Checklist di consegna**

- Hai fatto un unico file ZIP **(non .7z!!!)** contenente l'intero progetto?
  In particolare, ti sei assicurato di aver incluso tutti i file .java (e non solo i .class)?
- Hai controllato che si compili e ci sia tutto ? [NB: non serve includere il PDF del testo]
- Hai rinominato IL PROGETTO esattamente come richiesto?
- Hai chiamato IL FILE ZIP esattamente come richiesto?
- Hai chiamato la cartella del progetto esattamente come richiesto?
- Dopo aver caricato il file su Esamix, hai premuto il tasto "CONFERMA", ottenendo il messaggio "Hai concluso l'esame"?