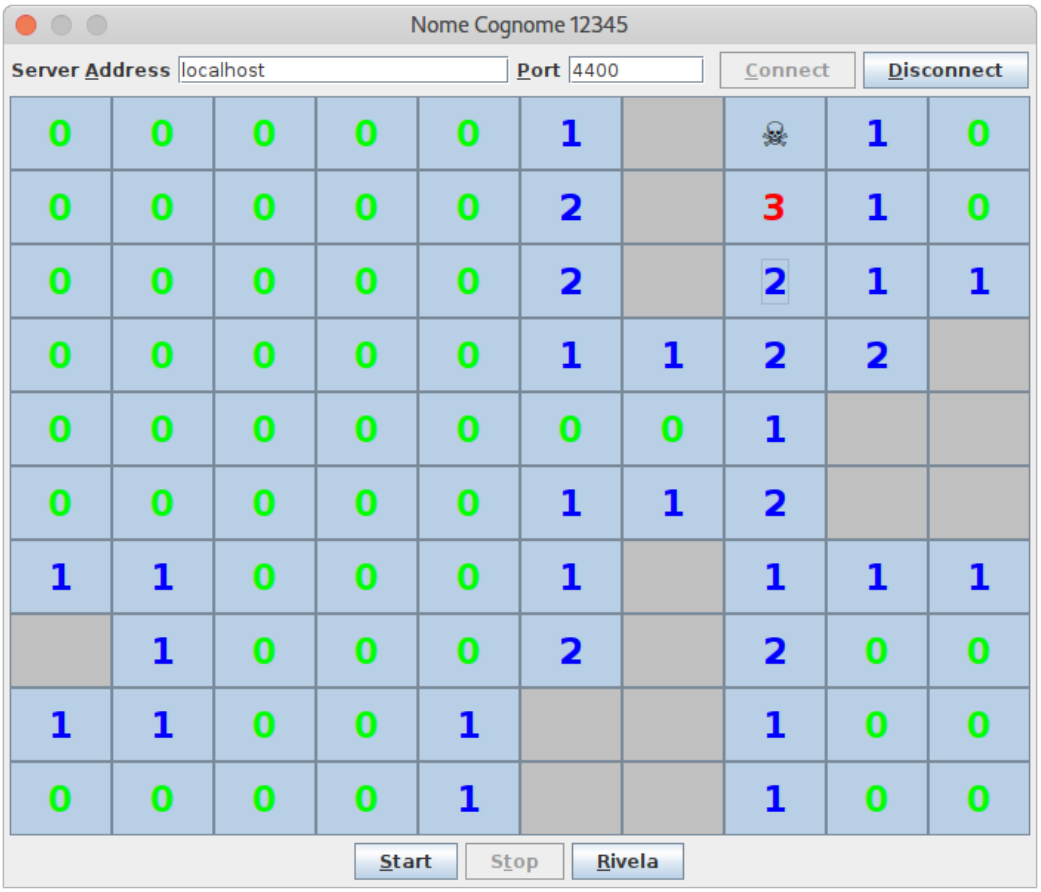
Progettazione del Software
Prova Pratica

18/06/2020 - tempo a disposizione 2h

Si vuole realizzare un'applicazione *client-server* che permetta di giocare ad una variante semplificata del gioco MineSweeper, (o *campo minato*). Il gioco si svolge su un tabellone composto da $n \times m$ caselle, il cui contenuto è inizialmente nascosto. Tra queste caselle, alcune vengono inizialmente popolate con delle mine, e lo scopo del gioco è quello di scoprire, una alla volta, tutte le caselle del tabellone, senza scoprire quelle contenenti una mina. Ogni volta che una casella viene selezionata, se questa contiene una mina allora la partita termina con la sconfitta del giocatore, altrimenti la casella viene scoperta, mostrando come suggerimento il numero di caselle direttamente adiacenti ad essa (comprese le diagonali) contenenti delle mine.

Il gioco si svolge nel seguente modo: *a*) Lato client, l'utente si connette al server e richiede di generare una nuova partita; *b*) Il server, ricevuta la richiesta, genera in maniera casuale una disposizione del tabellone, inviandola al client. *c*) Il client, ricevuto il contenuto del tabellone, popola l'interfaccia grafica e permette all'utente di iniziare la partita.

Il server è *multithreading* ed accetta connessioni da più client, ed ogni client connesso gestisce una partita diversa . La comunicazione è basata unicamente su scambio di stringhe. Tutte le stringhe sono inviate da client a server e viceversa utilizzando il carattere di fine linea come separatore.



Durante l'esame il server sarà raggiungibile al seguente indirizzo:

- Indirizzo IP: 80.211.232.219
- Porta: 4400

Si richiede la realizzazione del client, con interfaccia grafica e networking, in grado di comunicare con il server multithreading (fornito).

L'interfaccia grafica del client dovrà essere composta da un frame che abbia come titolo *nome cognome matricola* dello studente, da un pannello centrale che visualizzi lo stato attuale del gioco, da due campi testuali editabili per l'indirizzo e la porta del server, e da cinque pulsanti che realizzino le seguenti funzioni: *1*) *Connect*: permette di connettersi al server utilizzando indirizzo e porta specificati nei campi testuali; *2*) *Disconnect*: permette di chiudere la connessione al server; *3*) *Start*: permette di avviare la richiesta di una nuova partita; *4*) *Stop*: permette di interrompere la richiesta in corso; *5*) *Rivela*: permette di scoprire tutte le caselle del tabellone nel pannello centrale, rinunciando però alla possibilità di continuare la partita. Il pannello centrale che rappresenta il tabellone del gioco, dovrà essere composto da una matrice di 100 istanze della classe `BoardButton` (fornita), disposte secondo una griglia 10×10. La classe `BoardButton` estende `JToggleButton` e realizza la singola casella del campo minato tramite un pulsante, rappresentando graficamente all'utente lo stato ed il contenuto, una volta scoperta, della casella (come da figura). I pulsanti istanza di `BoardButton` implementano un meccanismo che gli permette di essere premuti solamente una volta, ad evitare che l'utente nasconda il contenuto di caselle già selezionate. Una volta premuto, per riportare il pulsante nello stato non selezionato, è necessario chiamare su di esso il metodo `reset()`.

La presenza o meno di una mina in una casella è rappresentata da un campo booleano nella classe `BoardButton`, accessibile tramite i metodi `hasMine()` e `setMine(boolean)`, mentre il numero di caselle adiacenti contenenti mine viene memorizzato nel campo `adjacentMines`, accessibile tramite i metodi `getAdjacentMinesCount()` e `setAdjacentMinesCount(int)`. La visualizzazione o meno del contenuto viene gestita dalla classe `BoardButton` in base al suo stato (selezionato o meno). La selezione di una casella può essere controllata attraverso i metodi `isSelected()` e `setSelected(boolean)`, ereditati da `JToggleButton`.

Si suggerisce di realizzare il tabellone di gioco (il pannello centrale) completando l'implementazione presente nella classe `Board` fornita, tuttavia lo studente è libero di realizzare la propria soluzione se preferisce o modificare quella fornita come più ritiene appropriato. La classe `Board`, sottoclasse di `JPanel`, ha il compito di gestire le istanze della classe `BoardButton`, mantenendo lo stato del gioco e facendo da *listener* per queste ultime, in modo da rispondere all'interazione dell'utente.

Suggerimento: Per la memorizzazione delle caselle che compongono il tabellone si suggerisce di utilizzare una matrice come struttura di dati.

Suggerimento: Al fine di semplificare lo svolgimento e la consegna dell'esame è fortemente consigliato l'utilizzo del package di *default* per tutte le classi dell'applicazione.

Si implementi il seguente protocollo:

- All'avvio solamente il pulsante *Connect* deve essere abilitato. Tutte le istanze di `BoardButton` devono essere disattivate (tramite chiamata al metodo `setEnabled(boolean)`).
- Alla pressione del pulsante *Connect*, il client invia una richiesta di connessione al server utilizzando indirizzo e porta indicati negli appositi campi.
- Una volta stabilita correttamente la connessione, il client deve abilitare i pulsanti *Start*, *Disconnect* e *Rivela*. Alla pressione del pulsante *Start*, il client deve inviare la stringa "start" al server, abilitare il pulsante *Stop*, e disabilitare i pulsanti *Start*, *Disconnect*, e *Rivela*. Il client dovrà inoltre effettuare il reset dell'eventuale partita attiva, chiamando il metodo `reset()` su ciascuna delle istanze di `BoardButton`.
- Una volta ricevuta la stringa "start", il server genererà una configurazione del tabellone di gioco, ed inizia ad inviare, ad intervalli regolari, stringhe al client corrispondenti al contenuto delle caselle, nel seguente formato:

  ```
  indice-riga:indice-colonna:n
  ```

  dove:

- `indice-riga` rappresenta *l'indice della riga* a cui si riferisce il valore, che può assumere i valori interi nell'intervallo $[0, 9]$, dove 0 rappresenta la prima riga partendo dall'alto.
- `indice-colonna` rappresenta *l'indice della colonna* a cui si riferisce il valore, che può assumere i valori interi nell'intervallo $[0, 9]$, dove 0 rappresenta la colonna più a sinistra.
- `n` è un *intero* che rappresenta il contenuto della casella (`indice-riga`, `indice-colonna`), e può assumere o il valore -1, ad indicare che la casella contiene una mina, o un intero non negativo, nel qual caso rappresenta il numero di caselle direttamente adiacenti contenenti mine.

Alla ricezione di ciascuna stringa il client deve:

1. Tradurre la posizione comunicata dal server negli indici (o indice) corrispondenti nella struttura dati utilizzata per memorizzare le caselle del tabellone (i.e. le istanze di `BoardButton`).
2. Impostare il valore del campo *mine* dell'opportuna casella al valore *true* se `n` è -1 tramite la chiamata al metodo `setMine(true)`, oppure impostare il conteggio del numero di caselle adiacenti con mine, attraverso la chiamata al metodo `setAdjacentMines(int)`, passandogli il valore ricevuto `n`, tradotto in intero.

N.B.: Durante la fase di download dal server, il contenuto delle caselle non sarà direttamente visibile nell'interfaccia, in quanto questo sarà visibile, al fine del gioco, solamente dopo che l'utente ha selezionato la casella. A scopo di debug, si consiglia quindi di produrre delle stampe a terminale ogni volta che viene ricevuta una stringa dal server.

- La comunicazione termina quando il server invia la stringa "done" al client, ad indicare che lo stato di tutte le caselle è stato inviato. Ricevuta la stringa "done", il client deve interrompere la ricezione, abilitare tutti i pulsanti istanze di `BoardButton`, e mostrare un messaggio in popup all'utente ad indicare che la partita può iniziare. Il client dovrà inoltre disabilitare il pulsante *Stop*, e riabilitare i pulsanti *Start*, *Disconnect* e *Rivela*.
  A questo punto l'utente può iniziare ad interagire con i pulsanti che rappresentano le caselle del campo minato, la cui pressione rivela il contenuto della casella. A seconda del contenuto, l'esito può essere uno tra i seguenti:
  - Se la casella contiene una mina, allora la partita termina con la sconfitta dell'utente. Il client deve mostrare un messaggio in popup e disabilitare tutti i pulsanti del tabellone.
  - Se la casella non contiene una mina, allora il client deve verificare se ci siano ancora caselle non contenenti mine che non sono state scoperte, nel qual caso la partita continua. Nel caso le caselle senza mine siano tutte scoperte, la partita termina con la vittoria dell'utente. Il client deve scoprire le restanti caselle, disabilitare tutti i pulsanti del tabellone, e mostrare un messaggio in popup per informare l'utente.
- Alla pressione del pulsante *Stop*, mentre la ricezione delle stringhe è ancora in corso, il client deve inviare al server la stringa "stop". Alla ricezione della stringa "stop", il server interromperà immediatamente l'invio delle stringhe, inviando la stringa "interrupted" ad indicare la terminazione. Alla ricezione di quest'ultima, il client dovrà smettere di ricevere stringhe, e comunicare, tramite un messaggio in popup, l'avvenuta interruzione. Il client dovrà inoltre disabilitare tutti i pulsanti del tabellone ed il pulsante *Stop*, e riabilitare *Start*, *Disconnect*, e *Rivela*. Nel caso in cui si avvii una nuova generazione dopo l'interruzione (ovvero si prema di nuovo *Start*), preliminarmente, il client deve reinizializzare tutti i pulsanti del tabellone, chiamando su ciascuno il metodo `reset()`.
- Alla pressione del pulsante *Disconnect*, il client deve inviare al server la stringa "disconnect", chiudere tutti i canali di comunicazione generati in fase di connessione, e deve inoltre abilitare nuovamente il pulsante *Connect* in quanto deve essere possibile instaurare una nuova connessione senza che sia necessario il riavvio del client. Il client deve inoltre disattivare tutte le caselle del tabellone.
- Alla pressione del pulsante *Rivela*, il client deve scoprire (tramite il metodo `setSelected(boolean)`) e disattivare tutte le caselle del tabellone, in quanto la partita si considera abbandonata dall'utente.