Progetto di Paradigmi di Programmazione (Programmazione ad Oggetti)

Introduzione
È stato richiesto di implementare in linguaggio Java un programma che simuli il gioco da tavolo "Battaglia Navale" seguendo il pattern di programmazione *Model-View-Controller (MVC)*. Lo sviluppo dell'applicazione ha tenuto conto delle **parti opzionali** elencate nella consegna del progetto. Sono quindi state implementate anche le seguenti funzionalità:

- salvataggio della partita su file binario;
- caricamento di una partita esistente da file binario;
- una modalità *difficile,* in cui il Computer continua a mirare una nave del giocatore dopo averla colpita;
- una modalità *a tempo,* in cui il giocatore è dichiarato sconfitto allo scadere del tempo.

Struttura dell'applicazione
Il package **upo.battleship.rossi** contiene *BattleshipMVC*, che consiste nel *main* dell'applicazione. Ci sono poi altri quattro packages, i cui contenuti sono mostrati nella seguente tabella.

controller	**model**	**utils**	**view**
BattleController.java BattleshipController.java CountdownController.java NewGameController.java SetShipsController.java StartLoadController.java	BattleshipModel.java Computer.java Player.java Ship.java	BattleshipState.java ComputerType.java Coordinates.java PlayerState.java ShipDirection.java ShipLength.java ShipType.java Utility.java	BattlePanel.java BattleshipView.java CountdownPanel.java ExitFrame.java NewGamePanel.java SetShipsPanel.java StartLoadPanel.java WindowDestructor.java

L'applicazione comprende un ultimo package **test**, in cui sono collocate le classi di test per ciascuna delle classi del package **model**. Tutti i packages, ad eccezione di **test**, sono corredati di JavaDOC.

L'applicazione è stata sviluppata estendendo il pattern MVC, in modo che ogni elemento del package **view** sia di tipo *Observer* sia gestito dal proprio personale oggetto **controller**, che modificherà o gli elementi del package **model** o la **view** a cui è collegato. Gli elementi del package **model** saranno tutti di tipo *Observable* e saranno "osservati" dagli elementi del package **view**, che aggiorneranno il proprio contenuto a seconda del **model** che osservano.

controller
Questo package contiene essenzialmente classi che estendono *ActionListener*. La classe principale è *BattleshipController*, che si comporta come gestore di controllori, fornendo il controllore adeguato alla gestione di una particolare vista tramite i propri metodi *giveXXXController*, dove *XXX* è il nome della vista da gestire (ad esempio, *giveBattleController()* assegnerà un *BattleController* ad un *BattlePanel*). Ogni altro **controller** di questo package gestisce esclusivamente la vista per cui è programmato.

model
Questo package contiene i modelli di nave, giocatore, computer e partita. In un'istanza di *BattleshipModel* sono mantenuti un *Player*, un *Computer*, la dimensione delle griglie di gioco ed un timer con la durata del conto alla rovescia.

Un *Player* mantiene **tre matrici booleane** tutte inizializzate a *true*, due delle quali verranno "riempite" con valori *false* durante il posizionamento delle navi (se in una cella c'è una nave, il valore della batrice booleana alle stesse coordinate sarà *false*) e la restante che verrà riempita con i colpi ricevuti dal *Computer* (quando *Player* subisce un colpo a certe coordinate, la matrice booleana dei colpi renderà *false* le celle colpite). Delle due matrici booleane di posizionamento delle navi, una rimarrà intonsa dopo l'inizializzazione, mentre l'altra sarà il risultato della sovrapposizione tra la matrice iniziale delle navi e la matrice booleana dei colpi subiti, in modo da semplificare il controllo del metodo *isDefeated()* di *Player,* che valuta se tutte le sue navi sono state affondate. *Player* mantiene anche **tre liste di elementi di tipo *Ship***: la prima con le navi disponibili e la cui dimensione è variabile a seconda della dimensione della griglia di gioco, la seconda con le navi piazzate (che verrà riempita con le navi della prima lista al posizionamento) e la terza con le navi completamente distrutte (che verrà riempita con le navi della seconda lista quando queste sono state affondate). Un *Player* offrirà metodi con cui impostare le coordinate intere a cui colpire il *Computer* e di metodi per posizionare le proprie navi.

Un *Computer* è una classe derivata da *Player*, che ne estende quindi il comportamento. In

particolare, un *Computer* implementa dei metodi per **mirare casualmente** le celle del *Player*. Qualora *Computer* fosse inizializzato come "intelligente", analizzando lo stato di *Player* e gli ultimi colpi andati a segno, esso sarebbe in grado di continuare a mirare celle adiacenti a quella colpita fino ad affondare una nave di *Player*.

Un elemento di tipo *Ship* mantiene una matrice booleana, della dimensione della griglia di gioco, che rappresenta la sua posizione assoluta sulla griglia di gioco stessa. Una *Ship* fornisce metodi per posizionare sé stessa su di una griglia booleana, e offre metodi per verificare che ci sia spazio sufficiente per il posizionamento. In questa classe, vengono anche forniti metodi per controllare che la *Ship* corrente sia stata mancata, colpita o colpita e affondata e per comunicarlo alle classi che utilizzano oggetti *Ship*.

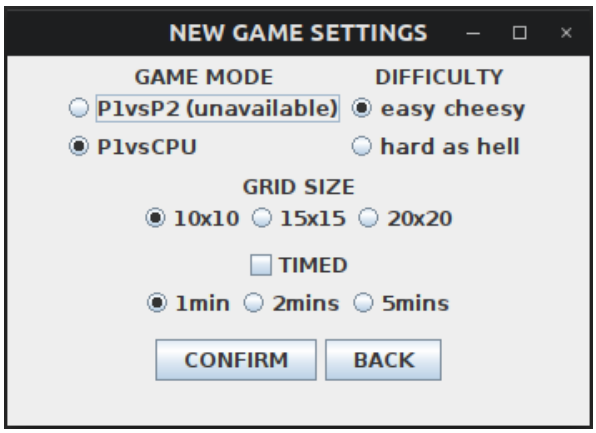
utils

Il package **utils** contiene delle classi o degli elementi *enum* che rappresentano delle costanti o delle labels utili ai fini del funzionamento dell'applicazione. Tra queste, ci sono i tipi di nave e le loro direzioni (*ShipType* e *ShipDirection*), le loro lunghezze (*ShipLength*), gli stati della partita e del giocatore (*BattleshipState* e *PlayerState*), il tipo di *Computer* (*ComputerType*) a seconda della difficoltà scelta e la classe *Coordinates*, che gestisce coppie di coordinate come se fossero oggetti.

view

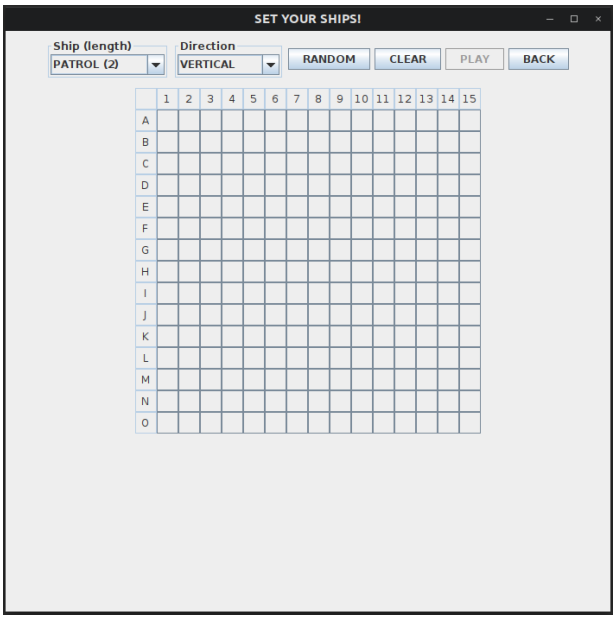
Il package **view** contiene *BattleshipView*, classe derivata da *JFrame*, che gestisce la finestra principale della vista e, a seconda del *BattleshipState* del modello *BattleshipModel*, deciderà quale dei pannelli (sostanzialmente i rimanenti elementi del package ad eccezione di *ExitFrame* e *WindowDestructor*) verranno resi visibili (in mutua esclusione). Ogni pannello avrà il compito di aggiornarsi in funzione di cambiamenti di stato o di attributo degli elementi del package *model*.

StartLoadPanel consiste nella vista iniziale, in cui l'utente può decidere se iniziare una nuova partita premendo il tasto "New game" o, se esiste e se il file non è "corrotto", continuare una partita salvata premendo il tasto "Load game". Premendo "New game" nello *StartLoadPanel*, verrà innescato un cambio di stato nel *BattleshipModel* che abiliterà la visibilità di *NewGamePanel* e renderà *StartLoadPanel* invisibile.

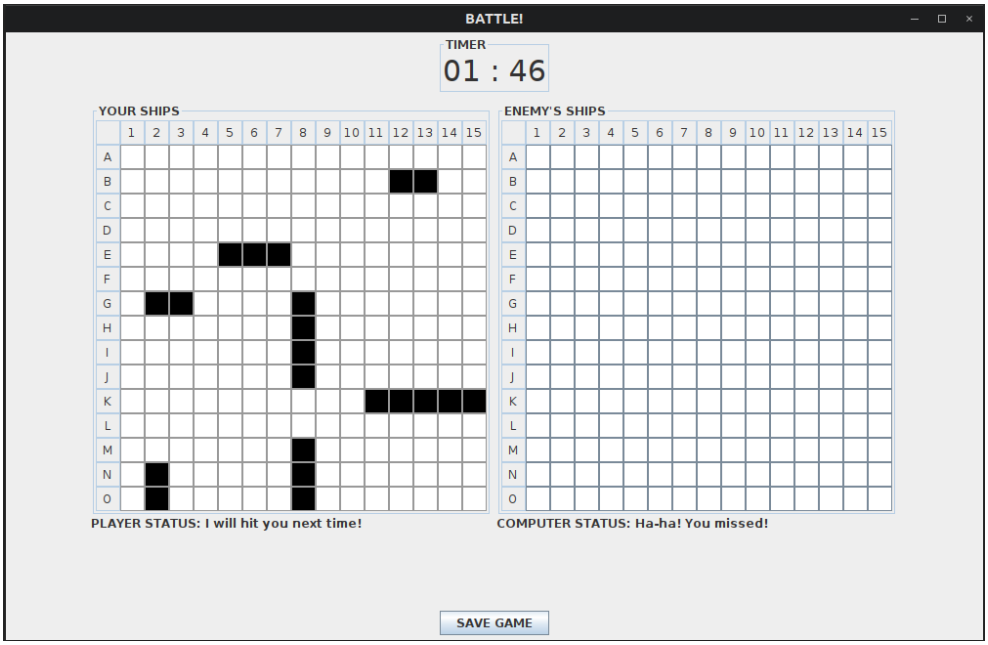


NewGamePanel è la vista in cui l'utente sceglie i parametri con cui impostare la propria partita, quindi il *BattleshipModel*. Tra questi parametri vi sono la modalità di gioco*****, la difficoltà del *Computer*, la dimensione della griglia ed un'opzione "TIMED" che, se spuntata, permette di tenere conto della selezione di un timer tra le opzioni visibili, e quindi di rendere la partita "a tempo". A causa della elevata velocità di risposta del *Computer*, si è deciso di non arrestare il timer durante l'azione del *Computer* stesso, poiché impercettibile all'occhio umano. Una volta selezionate le opzioni desiderate, premendo su "CONFIRM" il *BattleshipModel* cambierà stato ed innescherà la visibilità di *SetShipsPanel*. Premendo invece "BACK" si tornerà allo *StartLoadPanel*.

***Premendo "CONFIRM" dopo aver selezionato "P1vsP2", il NewGameController riceverà l'ActionEvent corrispondente, ma non sono state implementate azioni per tale configurazione. Si riserva tale funzionamento a future implementazioni.*

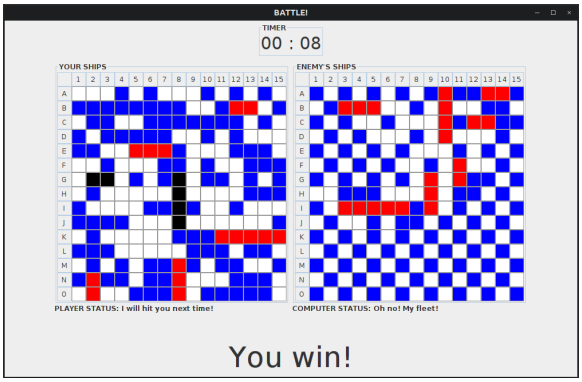


SetShipsPanel è la vista che permette all'utente di posizionare navi in due modi: una alla volta e tutte randomicamente. La seconda opzione è resa possibile dalla pressione del tasto "RANDOM", che posiziona ogni nave, in una direzione casuale tra "verticale" e "orizzontale", in una casella estratta casualmente con sufficiente spazio attorno. La prima opzione è invece resa possibile selezionando la nave e la direzione dai menù a tendina e, successivamente, premendo su una delle caselle della griglia. Se non ci fosse sufficiente spazio sulla griglia, la nave semplicemente non verrebbe posizionata e risulterebbe ancora presente nel menù a tendina.
Se l'utente non fosse soddisfatto del posizionamento, potrebbe annullare tutti i posizionamenti effettuati premendo il tasto "CLEAR".
Se l'utente volesse modificare altri parametri di gioco tornando a *NewGamePanel*, basterebbe premere il tasto "BACK". Quando tutte le navi sono state posizionate, viene abilitato il pulsante "PLAY", alla cui pressione viene innescato il cambio di stato in *BattleshipModel* che renderà invisibile *SetShipsPanel* e visibile *BattlePanel*.

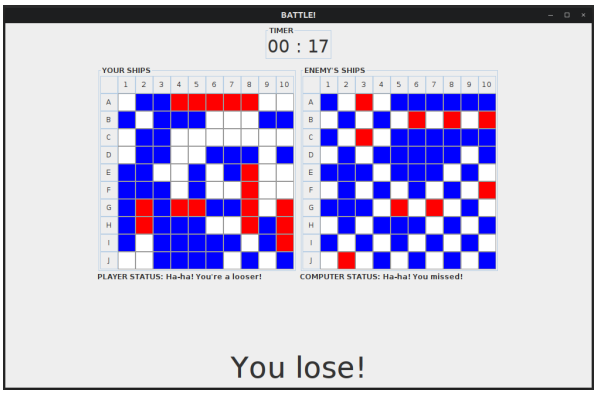


BattlePanel è il pannello di gioco che contiene le due griglie: una di *Player*, con le navi visibili, e una di *Computer*, con le navi nascoste. Entrambe sono griglie di bottoni, ma mentre nella prima tali bottoni sono disabilitati, nella seconda l'utente può decidere dove *Player* andrà a colpire *Computer* premendo il bottone corrispondente alle coordinate mirate. La pressione di uno di tali bottoni innescherà la seguente sequenza di azioni: il *Computer* viene colpito sulla cella di coordinate corrispondenti a quelle del bottone premuto e, dopo aver subito il colpo, sempre *Computer* colpirà *Player* su una delle sue celle. Tale sequenza è resa possibile dal metodo *hitAndGetHit()* di *BattleshipModel*. A livello grafico, una cella non selezionata risulterà bianca, una cella selezionata ma in cui non era presente una nave sarà blu e simboleggerà un "mancato" e una cella selezionata e in cui era presente una nave verrà contrassegnata come rossa, evidenziando un "colpito". Se la partita è stata impostata come "a tempo", verrà anche visualizzato un timer inizializzato con il conto alla rovescia scelto dall'utente.

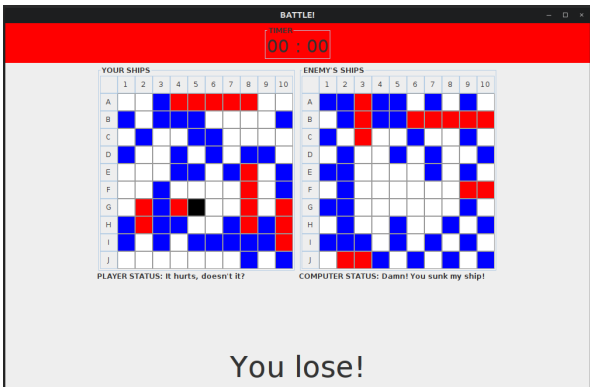
Al termine del tempo o quando tutte le navi di *Player* sono state affondate da *Computer*, verrà mostrata un'etichetta "You lose!". Al contrario, se tutte le navi di *Computer* vengono affondate e, se la partita è a tempo, se il conto alla rovescia non è scaduto, viene invece visualizzata un'etichetta "You win!". Per ogni nave mancata, colpita o affondata sia in una sia nell'altra griglia, verranno visualizzate etichette che evidenzieranno lo stato di Player e Computer (mancato, colpito, colpito e affondato).



Vittoria per aver affondato tutte le navi di *Computer*.



Sconfitta per non avere più navi in gioco.



Sconfitta per tempo scaduto.

Funzionamento

Lanciando *BattleshipMVC* viene inizializzato un oggetto *BattleshipModel* ed un *BattleshipView*. Quest'ultimo, prendendo *BattleshipModel* come parametro, inizializzerà i pannelli contenuti nel package *view* ed un oggetto *BattleshipController* che assegnerà ad ognuno di essi un controllore specifico. Ogni sotto-controllore viene assegnato alla propria vista solo nel momento in cui tale vista è resa visibile da *BattleshipView*, e l'assegnazione di un controllore consiste nell'assegnazione di un sotto-controllore appena inizializzato. Il primo pannello visibile all'interno di *BattleshipView* sarà quindi *StartLoadPanel*. In sequenza, compiendo le azioni spiegate in **view**, sarà possibile passare a *NewGamePanel* o a *BattlePanel* a seconda che si prema rispettivamente "New game" o "Load game" (ammesso che, in questo caso, il file di salvataggio esista e non sia corrotto o vuoto). *BattleshipModel* passerà rispettivamente dallo stato *STARTLOAD* allo stato *NEWGAME* o *BATTLE,* a seconda del bottone premuto, grazie al metodo *setState()* di *BattleshipModel*.

In *NewGamePanel* l'utente può selezionare tutti i parametri di gioco desiderati. La pressione di "CONFIRM" consiste nell'assegnazione di tali parametri a *BattleshipModel*, che inizializzerà di conseguenza i suoi campi interni (*Player, Computer,* dimensione della griglia di gioco, impostazioni del timer) grazie al metodo *newGame()* di *BattleshipModel*. *BattleshipModel* passerà poi dallo stato *NEWGAME* allo stato *SETSHIPS*, oppure tornerà allo stato *STARTLOAD* alla pressione di "BACK".

In *SetShipsPanel* l'utente può posizionare manualmente ogni nave, oppure tutte le navi casualmente premendo "RANDOM". Quando le navi disponibili sono state tutte posizionate, verrà abilitato il pulsante "PLAY" che permetterà a *BattleshipModel* di passare dallo stato *SETSHIPS* allo stato *BATTLE*, oppure allo stato *NEWGAME* alla pressione di "BACK".

Il cambio di stato di *BattleshipModel* in *BATTLE* determina l'inizio della partita tra *Player* e *Computer*, nonché la visibilità di *BattlePanel*. Oltre alle funzionalità spiegate in **view**, in questa vista è possibile salvare il modello della partita corrente, quindi l'istanza corrente di *BattleshipModel*, su un file binario *.dat*.

Qualora si passasse da *StartLoadPanel* a *BattlePanel* premendo "Load game", ammesso che il file di salvataggio sia esistente e valido, si potrà riprendere una partita da dove la si era salvata. La costruzione del *BattlePanel* verrà fatta in base ad un *BattleshipModel* inizializzato con gli stessi parametri dell'oggetto letto dal file binario.

Il comportamento di *Computer* varia a seconda della difficoltà impostata in *NewGamePanel*. Se è impostata la difficoltà "facile", allora ogni cella che il *Computer* colpirà sarà estratta casualmente. Se invece è selezionata la difficoltà "difficile", il *Computer* colpirà casualmente delle celle finché lo stato di *Player* non cambierà da *WATER* a *HIT*, e colpirà le celle attorno alla casella colpita per individuare l'orientamento verticale o orizzontale della nave. Scovato l'orientamento, *Computer* colpirà le celle allineate alle due colpite con successo finché *Player* non si troverà in uno stato *HITANDSUNK*, corrispondente a "colpito e affondato". Leggendo questo stato, *Computer* tornerà a mirare casualmente, alla ricerca di una nuova nave da affondare.

Player è decretato vincitore se affonda tutte le navi di *Computer* (entro il tempo limite, se la partita era a tempo). *Player* viene decretato sconfitto quando invece è *Computer* ad affondare tutte le navi di *Player* (o quando il tempo a disposizione scade, se la partita era a tempo).

Futuri sviluppi

Come testimoniato dalla presenza del radio button "P1vsP2", sicuramente una futura implementazione di questa applicazione prevedrebbe la modalità multigiocatore, in cui il *Computer* è sostituito da un altro *Player*. Questo comporterebbe sicuramente l'aggiunta o, quantomeno, la modifica di *BattlePanel* per visualizzare quattro griglie (due per giocatore) invece di due (solo per l'utente), di cui due opportunamente nascoste durante il turno dell'avversario.

Essendo *Computer* una classe derivata da *Player* ed avendo *Player* degli stati corrispondenti ai colpi subiti (*WATER* per mancato, *HIT* per colpito e *HITANDSUNK* per colpito e affondato), *Player* e *Computer* iniziano entrambi in stato *WATER*: questo è il motivo della presenza di etichette "PLAYER STATUS" e "COMPUTER STATUS" già definite sotto alle griglie di gioco quando non è ancora stata fatta la prima mossa. Si può aggirare questo piccolo ostacolo aggiungendo uno stato "START" in *PlayerState*, in modo che *Player* e *Computer* siano in quello stato all'inizio della partita.