Programmazione ad Oggetti e Ingegneria del Software
Anno Accademico 2019/2020
Sessione Estiva

BATTAGLIA NAVALE

Giacomo Cavalieri [Matricola 272094]
Pavel Bucsanu [Matricola 271426]

1. Specifica del problema

Si vuole realizzare un software che riproduca una partita del gioco “Battaglia Navale” scrivendo un’applicazione con interfaccia utente Windows Forms.
In particolare, si intende sviluppare una versione del gioco in cui è possibile sfidare unicamente un avversario “non-umano”, ovvero l’avversario del giocatore sarà il computer stesso. Le regole del gioco sono quelle della versione classica a carta e penna :

- Ogni giocatore dispone di 10 navi così suddivise:
    - 1 nave di classe **Portaerei;**
    - 2 navi di classe **Incrociatori;**
    - 3 navi di classe **Torpedinieri;**
    - 4 navi di classe **Sottomarini;**

- Nella fase iniziale del gioco, ogni giocatore disporrà segretamente le proprie navi nella griglia a lui associata. Ogni nave occuperà un numero diverso di caselle, più precisamente:
    - La Portaerei occupa **4 caselle**;
    - Gli Incrociatori occupano **3 caselle**;
    - I Torpedinieri occupano **2 caselle**;
    - I Sottomarini occupano **1 casella**;

- Ad ogni turno, il giocatore dovrà decidere in quale punto della griglia avversaria dovrà sparare sperando di colpire una delle navi nemiche e, in caso, **affondarla**;

- Quando una nave viene colpita in tutte le sue parti (ovvero tutte le caselle occupate) verrà affondata.

- Il gioco continua fino a quando uno dei due giocatori non avrà affondato tutte le navi avversarie e quindi vinto la partita.

A queste regole vengono aggiunte però delle modifiche:

- Il giocatore non-umano, diversamente dal suo avversario disporrà le proprie navi in maniera pseudo-casuale;
- Il giocatore non-umano, diversamente dal suo avversario, deciderà le coordinate per lo sparo in maniera pseudo-casuale;

- Ogni mossa dei 2 giocatori, verrà accompagnata da una scritta a schermo che indicherà l’esito dell’attacco:

- “**Colpito!**” nel caso si sia colpita una parte della nave avversaria;
- “**Colpito e Affondato!**” nel caso si sia affondata una nave avversaria;
- “**Mancato!**” nel caso non si sia colpito nulla;

2. Studio del problema

Di seguito vengono elencati i maggiori problemi per lo sviluppo del software:

Come viene rappresentato il campo dei due giocatori?
Il campo dei vari giocatori verrà rappresentato attraverso l’utilizzo di un array bidimensionale (una matrice) di caselle. Una casella è una classe in cui sono descritti due attributi di tipo intero: una per la coordinata della riga e un’altra per la coordina della colonna. L’utente, invece, vedrà il campo rappresentato nel form attraverso una griglia di bottoni da cliccare.

Come verranno rappresentate le navi?
Ogni nave viene rappresentata attraverso la descrizione di una classe. Ogni nave eredita attributi e metodi da una classe madre (abstract) per essere inserita nel campo da gioco e per essere affondata dai giocatori. Ogni nave ha una lunghezza diversa e un array di caselle di dimensione pari alla propria lunghezza. L’array è necessario per poter salvare la posizione nella griglia, utile per poter gestire i danni derivanti dagli spari dei due giocatori. Nel form le navi vengono rappresentate attraverso delle Picturebox in cui verranno caricate delle immagini per distinguere al meglio ogni nave.

In che modo verranno inserite le navi?
Le navi vengono inserite attraverso un metodo descritto nella classe “Navi” in cui sono date in ingresso delle coordinate e un campo da gioco in cui essere inserite. Ogni nave, inoltre possiede una proprietà (property) in cui verranno salvate la relativa posizione, ovvero le coordinate dell’array bidimensionale descritto in precedenza. L’inserimento viene descritto nella classe madre e grazie all’ereditarietà e il polimorfismo ogni nave riuscirà a inserirsi correttamente in base alla sua lunghezza diversa. L’inserimento viene rappresentato attraverso la stampa a schermo nei rispettivi bottoni del simbolo della nave.

In che modo verranno rappresentati i due giocatori?

I giocatori vengono rappresentati attraverso una classe. La classe possiede un campo da gioco e una lista di navi fondamentali per la gestione della partita. Il giocatore possiede inoltre dei metodi per poter gestire lo sparo verso il campo nemico.

In che modo le navi verranno affondate?

I giocatori dispongono di un metodo che ad ogni turno verrà invocato dal giocatore che dovrà far fuoco in una casella del campo avversario e se verrà colpita una nave, verrà invocata un metodo descritto nella classe madre delle navi che controllerà se il danno ricevuto è sufficiente per farla affondare andando a controllare il proprio array in cui è salvata la posizione.

Quando finirà la partita?

La partita terminerà quando uno dei due giocatori avrà tutte le proprie navi affondate. Nello specifico, appena una nave viene affondata, verrà rimossa dalla lista delle navi del giocatore. Successivamente, verrà effettuato un controllo sulla lista del giocatore e, se la lista è vuota, il giocatore avrà perso la partita, consegnando la vittoria all'avversario.

Come vengono rappresentati gli spari nel form?

All'interno del form l'utente vede l'esito degli spari attraverso la colorazione dei bottoni nei campi da gioco bersagliate dai giocatori. Se il bottone si colora di rosso, allora significa che una nave è stata colpita e/o affondata. Se, invece, il bottone si colora di blu, significa che è stato mancato il bersaglio. Inoltre, l'esito dello sparo viene comunicato all'utente anche attraverso un messaggio che compare a schermo.

3. Scelte architetturali

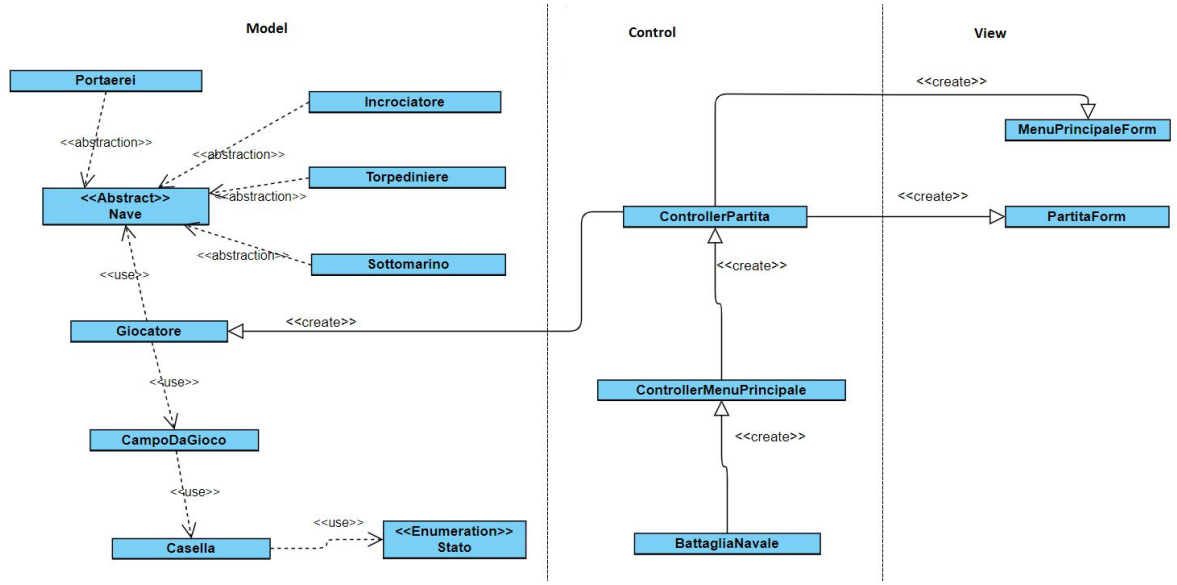
Per questo programma si è deciso di utilizzare un particolare pattern architetturale chiamato **MVC** (Model View Controller). L'utilizzo di un pattern architetturale, se ben implementato e utilizzato, permette al software di avere maggiore modularità e un'indipendenza funzionale che normalmente non avrebbe. In particolare l'MVC separa i dati (Model) dall'interfaccia utente (View) in maniera tale che l'interfaccia non influenzi la gestione dei dati e che i dati possano essere riorganizzati senza dover apportare cambiamenti all'interfaccia utente. Di seguito vengono descritte le caratteristiche dei 3 campi che caratterizzano il pattern:

1. **Model:** è la specifica rappresentazione dell'informazione sulla quale il modello opera. Racchiude dati ed algoritmi e costituisce il nucleo dell'applicazione. Il model è l'unico dei tre macro-componenti che risulta indipendente dagli altri due: infatti è possibile riutilizzarlo in altri progetti software se necessario senza dover preoccuparsi della struttura in quanto è l'unico componente a possedere totale indipendenza funzionale. In questo progetto il model viene rappresentato tramite una libreria di classi che provvederanno a descrivere i giocatori, le loro rispettive navi e i loro rispettivi campi da gioco. Elenco delle classi che rappresenteranno il Model:
   - `Casella.cs;`
   - `CampoDaGioco.cs;`
   - `Giocatore.cs;`
   - `Incrociatore.cs;`
   - `Nave.cs;`
   - `Portaerei.cs;`
   - `Sottomarino.cs;`
   - `Torpediniere.cs`
2. **View:** rappresenta il modello in una forma adatta all'interazione, di solito attraverso un'interfaccia grafica (form, bottoni, menu, ecc.). In questo progetto il componente View viene rappresentato da due classi di oggetti che si limiteranno a stampare a schermo bottoni e menu in modo che l'utente possa interagire con il software. Elenco delle classi che rappresenteranno il View:

- `MenuPrincipaleForm.cs;`
- `PartitaForm.cs;`

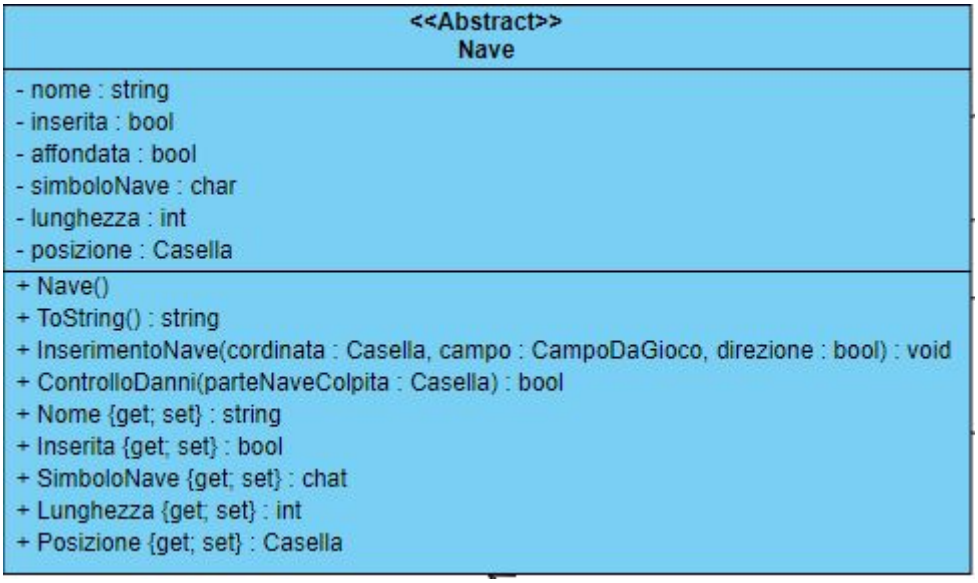
3. **Controller:** Processa e risponde agli eventi, come azioni che svolge l'utente e può comportare cambiamenti al modello. In questo progetto il Controller viene rappresentato da due classi di oggetti molto importanti che avranno il compito di dare un significato ai click che l'utente farà utilizzando il programma e comunicherà inoltre con il modello in modo da poter far visionare all'utente i cambiamenti che il modello apporterà durante l'esecuzione. Elenco delle classi che rappresenteranno il Controller:
   - `ControllerMenuPrincipale.cs;`
   - `ControllerPartita.cs;`

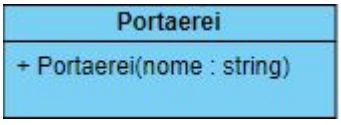
Il seguente diagramma vuole mettere in evidenza le relazioni che ci sono tra il modello, i controller e la vista. Come è possibile notare, i controller comunicano con la vista e il modello. Quest'ultimo rimane indipendente senza comunicare con le altre classi:

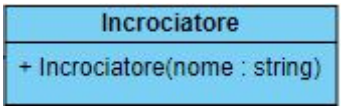


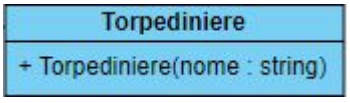
Di seguito, invece, si elencano le varie classi UML che compaiono nel progetto:

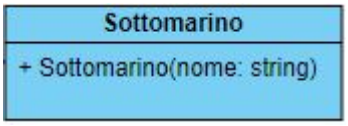
Model

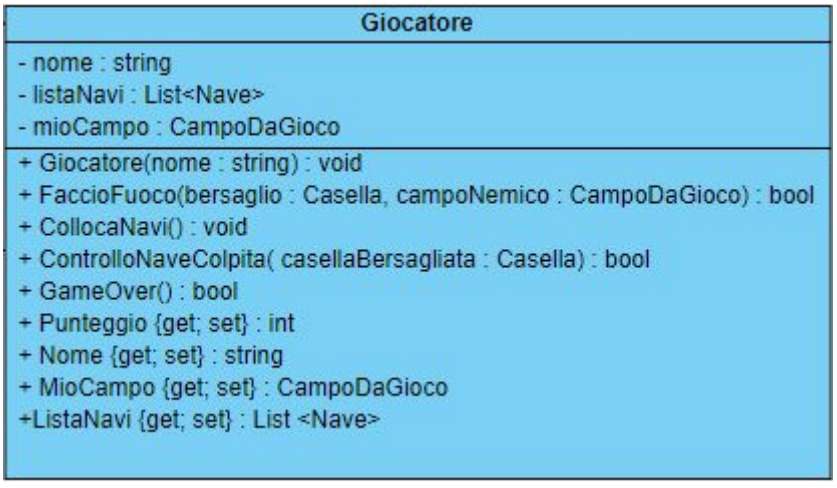


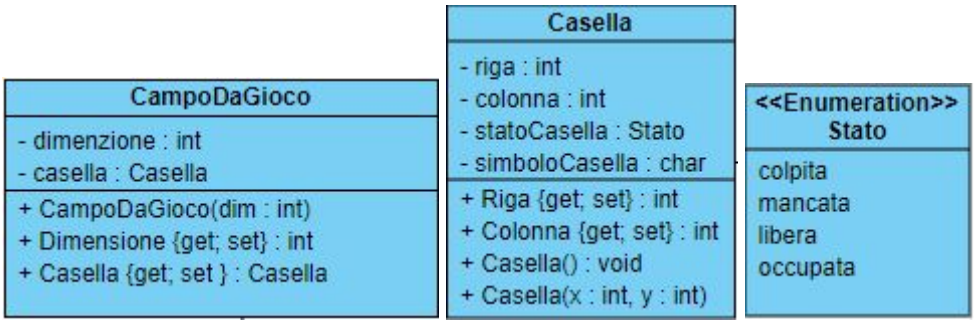




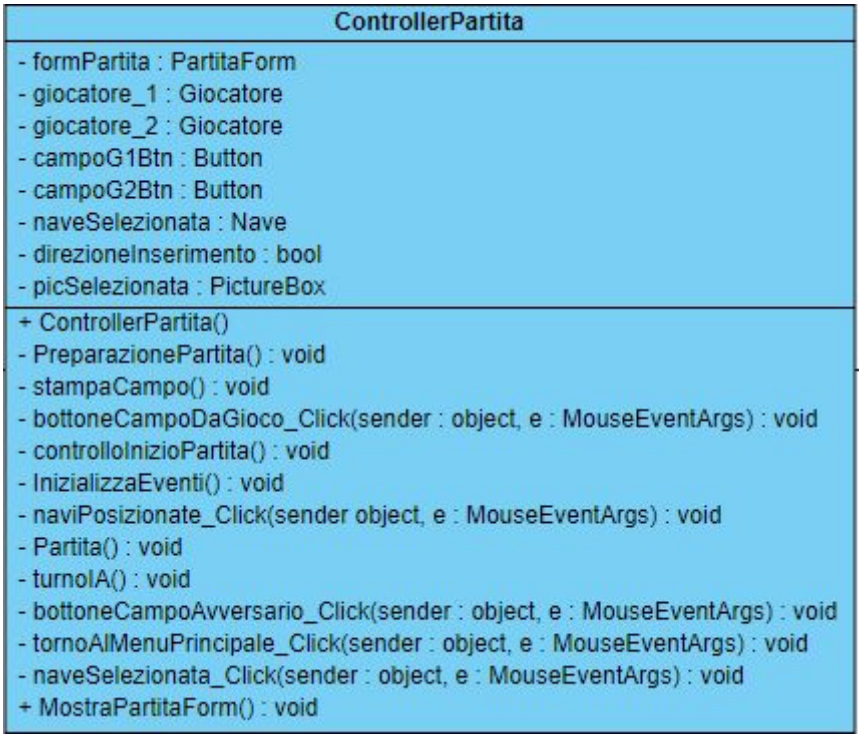


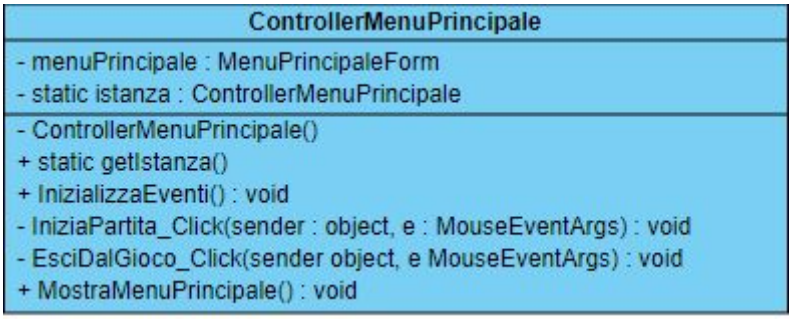




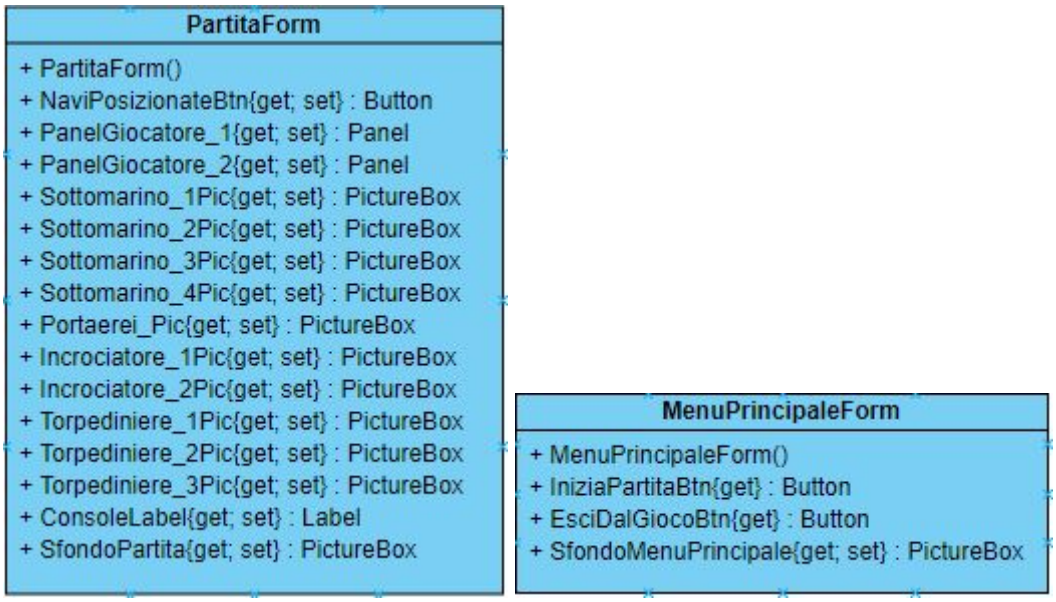

CampoDaGioco
- dimenzione : int
- casella : Casella
+ CampoDaGioco(dim : int)
+ Dimensione {get; set} : int
+ Casella {get; set } : Casella
Casella
- riga : int
- colonna : int
- statoCasella : Stato
- simboloCasella : char
+ Riga {get; set} : int
+ Colonna {get; set} : int
+ Casella() : void
+ Casella(x : int, y : int)
<<Enumeration>>
Stato
colpita
mancata
libera
occupata

Control


ControllerPartita
- formPartita : PartitaForm
- giocatore_1 : Giocatore
- giocatore_2 : Giocatore
- campoG1Btn : Button
- campoG2Btn : Button
- naveSelezionata : Nave
- direzioneInserimento : bool
- picSelezionata : PictureBox
+ ControllerPartita()
- PreparazionePartita() : void
- stampaCampo() : void
- bottoneCampoDaGioco_Click(sender : object, e : MouseEventArgs) : void
- controlloInizioPartita() : void
- InizializzaEventi() : void
- naviPosizionate_Click(sender object, e : MouseEventArgs) : void
- Partita() : void
- turnoIA() : void
- bottoneCampoAvversario_Click(sender : object, e : MouseEventArgs) : void
- tornoAlMenuPrincipale_Click(sender : object, e : MouseEventArgs) : void
- naveSelezionata_Click(sender : object, e : MouseEventArgs) : void
+ MostraPartitaForm() : void


ControllerMenuPrincipale
- menuPrincipale : MenuPrincipaleForm
- static istanza : ControllerMenuPrincipale
- ControllerMenuPrincipale()
+ static getIstanza()
+ InizializzaEventi() : void
- IniziaPartita_Click(sender : object, e : MouseEventArgs) : void
- EsciDalGioco_Click(sender object, e MouseEventArgs) : void
+ MostraMenuPrincipale() : void

View



Inoltre, per questo progetto, si è fatto uso del design pattern **Singleton**. Questo pattern assicura che una classe abbia al più un'istanza e fornisce un punto di accesso globale per essa. Si è fatto utilizzo nella classe `ControllerMenuPrincipale.cs` per fare in modo che, una volta che la partita è finita, il `controllerpartita` evocherà un'istanza del menu principale senza che venga istanziato un nuovo menu principale ma usando solamente quello istanziato all'avvio del software. Il design pattern è stato implementato all'interno della classe nel seguente modo:

```
// Metodo per l'implementazione del Design Pattern Singleton
public static ControllerMenuPrincipale getIstanza()
{
   if (istanza == null)
   {
      istanza = new ControllerMenuPrincipale();
   }
   return istanza;
}
```

Quando la classe viene creata nel metodo main della classe `program.cs` viene fatto nel seguente modo:

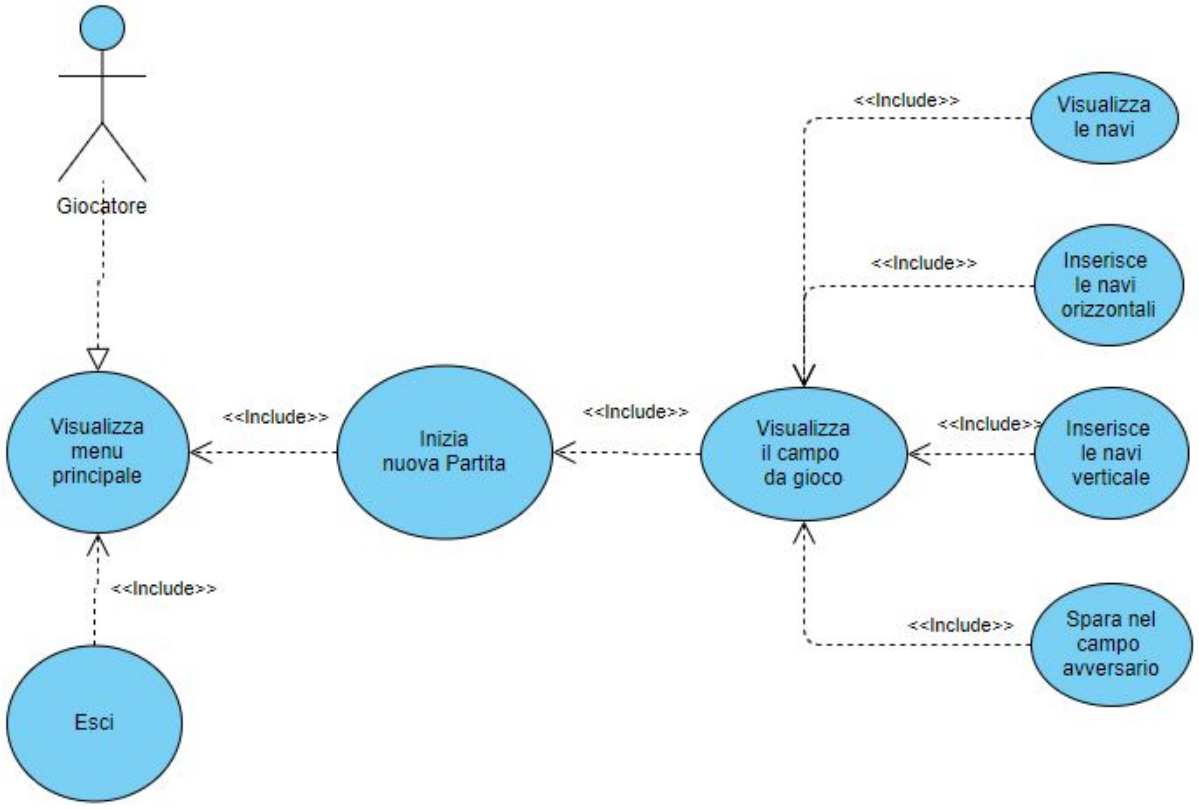
```
// Dichiaro il controller del menù Principale tramite il design pattern Singleton
ControllerMenuPrincipale menuPrincipale = ControllerMenuPrincipale.getIstanza();
```

4. Documentazione sull'utilizzo

Il software non ha bisogno di particolari passi per poter essere eseguito e non necessita di impostare parametri dopo la compilazione.

5. Use cases con relativo schema UML

Di seguito viene rappresentato il diagramma dei casi d'uso. Il diagramma ha evidenziato la partecipazione di un unico attore: il giocatore, il quale potrà avviare la partita tramite il bottone "INIZIA PARTITA" o uscire dal gioco tramite il bottone "ESCI":



Si mostrano di seguito le descrizione dei casi d'uso individuati.

- Il giocatore avvia l'applicazione GUI e visualizza un form che dà la possibilità di decidere se iniziare la partita oppure uscire dall'applicazione.
- Inizia nuova partita

Una volta che il giocatore clicca sul pulsante "Inizia Partita", visualizza il suo campo da gioco con le navi da inserire e il campo del nemico.

Le operazioni che il giocatore può effettuare sono le seguenti:

- ❖ Cliccare sulle immagini delle navi.
  - ➢ Il giocatore sulle immagini delle navi.
  - ➢ il programma mette in evidenza la nave selezionata.
- ❖ Inserire la nave all'interno del proprio campo.
  - ➢ Il giocatore clicca sul campo con la nave selezionata.
  - ➢ Il programma mostra in evidenza la nave selezionata all'interno del campo.
- ❖ Cambiare la direzione della nave (es. da verticale a orizzontale e viceversa).
  - ➢ Il giocatore clicca mouse destro per cambiare la direzione
  - ➢ Il programma mostra in evidenza la nave selezionata all'interno del campo cambiata di direzione.
- ❖ Cliccare sul pulsante "Iniziamo!".
  - ➢ Il giocatore inserisce tutte e 10 le navi.
  - ➢ Il programma mostra il pulsante "Iniziamo!"
- ❖ Cliccare sul campo da gioco del nemico per colpire/affondare le navi.
  - ➢ Il giocatore clicca sul campo avversario.
  - ➢ Il programma mostra se la nave è stata mancata/colpita/affondata.
- ❖ Se vince/perde la partita, cliccare sul pulsante "torna al menu principale".
  - ➢ Il giocatore può vincere o perdere la partita.
  - ➢ Il programma mostra il pulsante "torna al menu principale"
- ❖ Decidere se fare una nuova partita o uscire dall'applicazione.
  - ➢ Il giocatore può iniziare una nuova partita.
  - ➢ Il programma mostra la possibilità di iniziare una partita o uscire.