Progetto d’esame

SOFTWARE ENGINEERING

Simulatore di trasporto per sbarco anfibio

Anno Accademico 2013/2014

Proposto da: **Alberto Arvizzigno**

Matricola n. 237953
E-mail: albertoa.arvizzigno@gmail.com

1 INDICE

2 TERMINOLOGIA IMPIEGATA

Si è ritenuto necessario specificare inizialmente una nomenclatura contenente i termini di uso comune in ambito strettamente militare e adottati nel programma e nella simulazione, in modo che possa essere chiaro anche ai non "addetti ai lavori" l'obiettivo che si vuole ottenere e i formalismi tecnici a volte utilizzati.

Eliassalto: è lo sbarco di truppe e mezzi a mezzo elicottero in territorio controllato dal nemico.

Nave portaeromobili: è una unità navale militare destinata al trasporto, al lancio ed al recupero di aeromobili.

Spot: è il punto di decollo o di appontaggio per un elicottero in grado di operare da una portaeromobili, viene impiegato anche per rifornire l'elicottero. Ogni spot è in grado di ospitare differenti categorie di elicotteri, ad esempio uno spot per una categoria di elicotteri leggeri non è in grado di ospitare un elicottero per una categoria di elicotteri pesanti, inoltre gli spot possono essere limitati nell'operare di giorno o notte. Se uno spot può operare di notte può comunque operare anche di giorno, ma non viceversa.

Ponte di Volo: sulle navi portaeromobili è il ponte sul quale decollano e atterrano gli aerei imbarcati, spesso è angolato rispetto all'asse della nave e dotato di catapulte a vapore. Su altre unità militari il ponte di volo è quello destinato alle operazioni degli elicotteri. Anche questi sono spesso dotati di coppelle.

Holding Point (HP): punto di attesa in volo, è un punto attorno al quale un elicottero in volo si può mettere ad orbitare in attesa che si verifichi un determinato evento.

Landig Zone(LZ): è il punto in cui le truppe ed il cargo deve essere rilasciato a terra.

Raid Anfibio: particolare tipo di operazione anfibia che consiste nella rapida incursione in un obiettivo o l'occupazione temporanea dello stesso, va effettuata con il massimo numero di forze disponibili. Se ad esempio dispongo di 2 elicotteri che possono trasportare 10 uomini a testa, ma ho 30 uomini impiegabili in tutto il raid consiste nel caricare le massime forze disponibili (quindi 20 uomini) per inserirli sulla LZ. I restanti 10 uomini verranno prelevati successivamente.

Hangar: è il luogo all'interno della nave, sotto il ponte di volo, nel quale gli elicotteri permangono fino a quando non vi è la necessità di un impiego effettivo.

Elevatore: è il mezzo di trasporto che viene impiegato per poter spostare gli elicotteri dall'Hangar al Ponte di volo.

Nautical Mile (NM): distanza espressa in miglia marine (1852 metri).

Ingaggio elicottero: è il momento in cui inizia la rotazione delle pale, diverso dalla messa in moto dell'elicottero in cui vengono accesi i motori. L'energia dei motori in moto viene inizialmente sfruttata per poter aprire le pale dell'elicotteri (unfolding).

Rotore ingaggiato: stato di rotore in movimento (in cui le pale sono rotanti). Il moto di rotazione dai motori alle pale viene poi trasferito tramite una trasmissione.

Appontaggio: atterraggio di un elicottero su una nave.

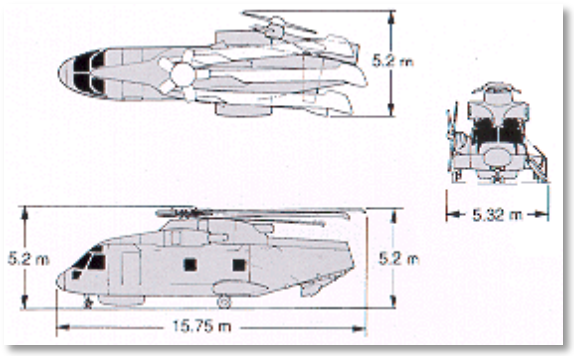
Ammaraggio: procedura di emergenza per cui un elicottero deve atterrare immediatamente, ma trovandosi in mezzo al mare è costretto a ad adagiarsi in acqua in modo che l'equipaggio possa salire

sulle zattere di salvataggio per il successivo recupero. L'ammaraggio comporta la perdita dell'elicottero.

Truppe anfibie: personale militare specializzata in attività di sbarco lungo costa o in operazioni che interessano terra e acqua.

Elicottero a Pale chiuse: gli elicotteri di tipo militare marina hanno in generale la capacità di chiudere le pale per ridurre lo spazio d'ingombro.

Le pale del rotore si possono si possono disporre per chiglia eliminando l'ingombro del rotore.



In figura viene mostrato un elicottero (tipo EH10) a pale piegate comunemente utilizzato nelle operazioni militari di sbarco anfibio. La rappresentazione di quest'elicottero è stata usata per gli sprite della simulazione.

Hot-Ref: rifornimento a caldo, è una procedura che permette di lasciare l'elicottero in moto sul ponte, col rotore ingaggiato e contemporaneamente effettuare operazioni di rifornimento eli (a caldo).

Carburante, peso truppe, peso cargo: è importante capire che l'elicottero ha un peso di decollo massimo che non può essere superato, quindi i dati di missione devono essere calibrati in funzione della distanza, maggiore è la distanza che devo percorrere, maggiore è il carburante che devo avere a bordo e conseguentemente sarà necessario sottrarre peso destinato alle truppe o al cargo.

3 SPECIFICA DEL PROBLEMA

Con questo progetto si vuole realizzare la simulazione di uno sbarco di personale anfibio e di cargo a mezzo elicotteri di trasporto da un'unità navale posta ad una determinata distanza dal punto di rilascio posto a terra.

La simulazione vuole far prendere atto delle tempistiche e della sequenza di eventi che andranno ad intercorrere in una manovra coordinata al fine di poter imbarcare truppe e cargo a bordo di uno o più elicotteri che devono essere rilasciati su un'apposita Landing Zone a terra. Ogni elicottero deve essere sequenzialmente movimentato dall'hangar al ponte di volo e deve inoltre: effettuare rifornimento di carburante se necessario, mettere in moto i motori, effettuare l'apertura delle pale, ingaggiare il rotore, caricare le truppe ed il cargo a bordo, effettuare il trasporto sulla LZ e tornare a bordo della nave eventualmente per ricominciare il ciclo se ci sono ancora truppe o cargo che devono essere trasportate. La sequenza di eventi necessari sopradescritta deve inoltre essere rappresentata e visualizzata per ogni elicottero disponibile al termine della simulazione, qualora questa abbia avuto successo, cosa che ad esempio non può avvenire se parte delle truppe o del cargo non viene rilasciato sulla LZ. Le motivazioni che possono far fallire la missione sono essenzialmente derivanti dalla mancanza di elicotteri in grado di portarla a termine.

La scomparsa di un elicottero di trasporto dall'elenco di elicotteri disponibili per l'attività può essere causata da differenti motivazioni quali:

- Ammaraggio forzato per un errato calcolo di carburante
- Impossibilità di poter effettuare lo spostamento degli elicotteri sul ponte di volo in quanto gli spot a disposizione non permettono il decollo per quella categoria di elicottero
- Il peso disponibile in elicottero è inferiore a qualunque cargo o truppa da imbarcare

Lo sbarco anfibio o raid prevede inoltre che si svolga con il numero massimo di forze disponibili concentrate in un'unica prima ondata di volo, ossia tutti gli elicotteri utilizzabili, qualora ci sia materiale o truppe da trasportare devono arrivare a destinazione insieme. Dopo la prima ondata, terminato l'effetto sorpresa, non sarà più necessario che tutti gli elicotteri siano raggruppati in trasporti congiunti. Sarà quindi necessario far attendere in volo in un holding point i primi elicotteri pronti a partire, in modo che si possano ricongiungere agli altri e sia in grado in questo modo d'iniziare la wave di sbarco.

E' necessario inserire da parte dell'utente le informazioni necessarie per procedere con l'attivazione della simulazione quali:

Elicotteri impiegabili: tipologia, caratteristiche, pesi disponibili per l'operazione, carburante richiesto per effettuare l'attività.

Truppe: numero di truppe da sbarcare e peso medio di un singolo soldato

Spot: numero e caratteristiche degli spot da impiegare

Cargo: è necessario conoscere il Cargo, ammesso che esista, che deve essere trasportato sulla LZ, a tal fine è solitamente necessario impiegare del personale che si deve occupare fisicamente del trasporto (ad esempio per trasportare una macchina tipo Hammer all'interno di un elicottero è necessario trasportare l'autista che poi dovrà guidarla).

Distanza LZ-SHIP: è necessario sapere le distanze che devono essere percorse in NM (nautical miles) dal decollo per poter raggiungere la LZ.

Una volta inserite le informazioni necessarie, definiti elicotteri, spot, truppe, cargo e distanze in gioco, la simulazione può iniziare con una visualizzazione grafica degli elicotteri che si spostano dal ponte di volo per poi decollare e raggiungere la LZ per il rilascio delle truppe e del cargo.

Gli elicotteri devono poter seguire una propria logica operazionale sopra descritta, nonché rispettare le tempistiche reali bloccandosi temporalmente onde completare l'attività in corso. Ad esempio per spostare un elicottero dall'hangar allo Spot tramite l'elevatore del ponte ci possono volere circa 15 minuti, per poter effettuare un rifornimento possono occorrere 6 minuti, ogni tempistica deve essere calibrata e tenuta in considerazione in modo che possa essere rappresentata al termine della simulazione.

La determinazione di un punto d'attesa HP, come accennato, deve permettere una temporanea sosta in volo per gli elicotteri che vanno verso la LZ in modo che tutte le forze disponibili possano essere impiegate in un'unica ondata, inoltre tale HP fornirà un punto di sosta anche in rientro dalla LZ qualora il ponte di volo non offra spot disponibili per un appontaggio immediato.

La maggior parte delle informazioni in input viene inserita per mezzo di barre valori, la scelta è stata effettuata per due principali ragioni, la prima è per evitare possibili scorrettezze che l'utente potrebbe compiere nell'inserimento dei dati, la seconda è per fornire all'utente dei limiti chiari entro cui i dati possono essere realisticamente accettabili.

I dati inseriti devono inoltre essere controllati in base alla fattibilità della missione, se ad esempio vengono inserito tre elicotteri per trasportare una persona sola è chiaro mi potrà bastare solo uno dei tre velivoli.

Inoltre devono essere effettuati anche i calcoli sul carburante necessari per l'effettuazione della missione. Se ad esempio in fase d'inserimento dati il carburante inserito non è in gradi di portare l'elicottero a destinazione sarà necessario correggere il valore, tuttavia il peso totale di un elicottero per problemi di portanza massina non è modificabile, quindi per aumentare carburante sarà anche necessario eliminare peso destinato a truppe o cargo.

La simulazione avrà termine quando tutte le truppe e tutto il cargo in grado di alloggiare per peso all'interno di un elicottero scarico saranno state rilasciate sulla LZ.

Come risultato saranno espresse le tempistiche necessarie ad effettuare l'operazione in una schermata finale in modo da rendere chiaro il film degli eventi che si sono svolti nella simulazione.

Per semplicità di sviluppo del programma sono stati mantenuti costanti i parametri di tempo e spazio:

- Parametri temporali: 1 secondo di funzionamento della simulazione pari viene considerato 1 minuto di tempo reale (rapporto 1/60).
- Parametri spaziali: lo schermo grafico su cui si lavora, anche per incompletezza di una gestione delle librerie grafiche open source che non permettono una efficiente riproduzione grafica a finestra, è stato mantenuto quindi a schermo intero con una risoluzione grafica abbastanza comune 1024x768. Tutti i calcoli di spazio e velocità sono stati quindi presi a misura su tali coordinate.

L'utente potrà inoltre visualizzare alcune informazioni specifiche su ogni singolo elicottero inserito tramite *l'uso del pad numerico*, ogni elicottero corrisponde infatti ad un numero sul pad numero e le informazioni verranno visualizzate.

Al termine della simulazione dovranno essere visualizzate le informazioni rappresentanti gli elicotteri impiegati e le tempistiche salienti per atterrare o decollare dalla nave, holding point e landing zone.

4 SPECIFICA DEI REQUISITI

Viene riportata una descrizione dettagliata delle funzionalità dell'applicazione espresse mediate diagrammi dei CASI d'USO.

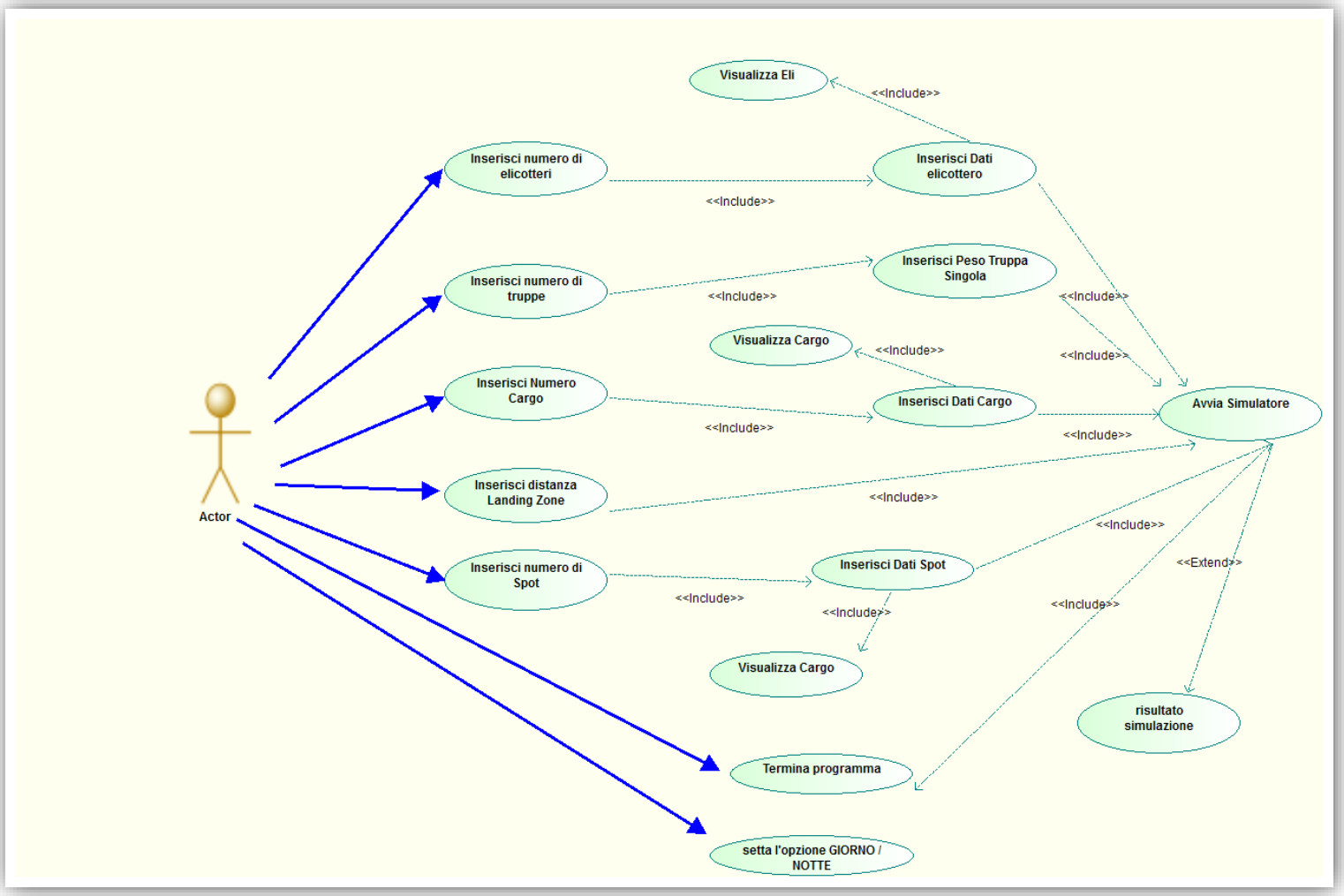
4.1 USE CASES

L'utente che dovrà utilizzare il programma è l'attore principale del software che dopo aver avviato il programma dovrà inserire le informazioni necessari per poter avvisare la simulazione.

Con gli USE CASES sotto descritti vengono mostrate le richieste funzionali del sistema.

La descrizione di un caso d'uso definisce cosa accade nel sistema in seguito all'evento di innesco. Ogni caso d'uso risponde ad un solo attore principale e cerca di realizzarne la richiesta.

Dal grafico è possibile verificare le principali interazioni che l'attore compie per l'inserimento dei dati, tali informazioni sono inoltre una precondizione per l'avvio del simulatore.



4.2 DESCRIZIONE DEI CASI D'USO

Tutti i casi d'uso sono attuati da un unico attore principale che avvia il programma.

I dati necessari alla simulazione vengono inseriti tramite interfacce grafiche, tutti gli inserimenti dati numerici vengono effettuati tramite barre valori ed hanno dei limiti di massimo o minimo che non possono essere cambiati, questa soluzione rende la simulazione realistica con dati in input che siano sempre entro dei valori accettabili.

4.2.1 Visualizza informazioni andamento programma

Nessuna precondizione necessaria.

- Una finestra informativa mostra costantemente lo stato delle operazioni effettuate e l'andamento del programma, la finestra serve a mantenere l'utente aggiornato e consapevole su quello che è stato fatto. La finestra può inoltre essere attivata o disattivata unicamente dalle opzioni del menu principale.

Post condizione: le informazioni salvate all'interno della finestra vengono anche salvate su un file di log alla fine del programma, in modo che i dati possano essere rivisti e confrontati in situazioni successive.

4.2.2 Inserisci numero di elicotteri

Nessuna Precondizione necessaria.

- Viene mostrata la schermata GENERALE d'inserimento dati.
- L'Attore inserisce il numero di elicotteri richiesto per l'operazione con l'utilizzo di una barra valori.
- Al termine dell'inserimento del dato, il dato deve essere accettato tramite l'apposito tasto CONFERMA del numero

Post condizione: il numero di elicotteri verrà fornito come informazione generale, determina il numero di record che devono essere creati per l'inserimento dei dati attivabile con l'apposito tasto OPZIONI. Il mancato inserimento delle informazioni richieste non permette la prosecuzione della simulazione.

4.2.3 Inserisci numero di truppe

Nessuna Precondizione necessaria.

- Viene mostrata la schermata GENERALE d'inserimento dati.
- L'Attore inserisce il numero di truppe richiesto per l'operazione con l'utilizzo di una barra valori.
- Al termine dell'inserimento del dato, il dato deve essere accettato tramite l'apposito tasto di CONFERMA del numero

Post condizione: il numero di truppe verrà fornito come informazione necessaria al simulatore per l'avvio dello stesso. Il mancato inserimento non permette la prosecuzione.

4.2.4 Inserisci numero cargo

Nessuna Precondizione necessaria.

- Viene mostrata la schermata GENERALE d'inserimento dati.
- L'Attore inserisce il numero di cargo richiesto per l'operazione con l'utilizzo di una barra valori.
- Al termine dell'inserimento del dato, il dato deve essere accettato tramite l'apposito tasto di CONFERMA del numero

Post condizione: il numero di cargo verrà fornito come informazione generale, determina il numero di record che devono essere creati per l'inserimento dei dati attivabile con l'apposito tasto OPZIONI. Il mancato inserimento delle informazioni non è una limitazione alla normale prosecuzione del programma, infatti non è necessario che esista del cargo in un assalto anfibio, quanto piuttosto non è possibile fare a meno delle truppe.

4.2.5 Inserisci distanza Landing Zone

Nessuna Precondizione necessaria.

- Viene mostrata la schermata GENERALE d'inserimento dati.
- L'Attore inserisce la distanza Nave-Landing Zone richiesta per l'operazione con l'utilizzo di una barra valori.
- Al termine dell'inserimento, il dato deve essere accettato tramite l'apposito tasto di CONFERMA del numero

Post condizione: il numero di cargo verrà fornito come informazione generale, determina il numero di record che devono essere creati per l'inserimento dei dati attivabile con l'apposito tasto OPZIONI. Il parametro di distanza LZ-NAVE è un dato necessario alla simulazione che non può essere avviata senza.

4.2.6 Inserisci numero di spot

Nessuna Precondizione necessaria.

- Viene mostrata la schermata GENERALE d'inserimento dati.
- L'Attore inserisce il numero di spot richiesto per l'operazione con l'utilizzo di una barra valori.
- Al termine dell'inserimento, il dato deve essere accettato tramite l'apposito tasto di CONFERMA del numero

Post condizione: il numero di cargo verrà fornito come informazione generale, determina il numero di record che devono essere creati per l'inserimento dei dati attivabile con l'apposito tasto OPZIONI. L'inserimento del numero di spot è un dato necessario alla simulazione che non può essere avviata senza.

4.2.7 Inserisci dati elicottero

Precondizione: deve essere stato inserito un numero di RECORD tramite il tasto di CONFERMA elicotteri dal menu principale.

- Premendo il Tasto OPZIONI per elicotteri dalla schermata principale appare la schermata d'inserimento delle informazioni per gli elicotteri. L'inserimento delle informazioni sugli elicotteri è un dato necessario per l'avvio della simulazione.
- L'attore deve inserire le informazioni richieste:
    - IDENTIFICATIVO ELICOTTERO: indica il nome attribuito all'elicottero per l'operazione. Il valore deve essere accettato tramite il tasto IDENTIFICATIVO ELICOTTERO.
    - CATEGORIA ELICOTTERO: indica la classe di appartenenza dell'elicottero che può essere LEGGERA (classe 1), PESANTE (classe 2), ALTRA CLASSE (classe 3). L'inserimento viene effettuato tramite la barra valori. Il valore deve essere accettato tramite il tasto CATEGORIA ELICOTTERO
    - PESO CARBURANTE: indica il carburante di partenza con cui è rifornito inizialmente l'elicottero. Il dato viene inserito tramite barra valori. Il dato deve essere accettato tramite il tasto PESO CARBURANTE.
    - PESO TRUPPE IMBARCABILI: indica il peso libero che viene lasciato all'imbarco delle truppe. (Ad esempio 2000Kg indica che possono essere imbarcati 2000 Kg di Truppe).

Il dato viene inserito tramite barra valori. Il dato deve essere accettato tramite il tasto PESO TRUPPE IMBARCABILI.

- PESO CARGO IMBARCABILE: indica il peso libero che viene lasciato all'imbarco del Cargo. (Per Cargo s'intende il peso effettivo del Materia ed il peso delle truppe necessarie al suo trasporto che andranno sull'elicottero, la sommatoria dei due dati costituisce il perso del cargo imbarcabile). Il dato deve essere accettato tramite il tasto PESO CARGO IMBARCABILE.
- PESO MASSIMO AL DECOLLO: indica il peso massimo che l'elicottero può sostenere al suo decollo. Il valore viene inserito tramite l'apposita barra valori e non deve superare la sommatoria di: peso elicottero scarico, peso truppe imbarcabili, peso cargo imbarcabile, peso carburante. Il dato deve essere accettato tramite il tasto PESO MASSIMO AL DECOLLO.
- PESO ELICOTTERO SCARICO: indica il peso dell'elicottero senza carburante, truppe o cargo. Il valore viene inserito tramite l'apposita barra valori e deve essere accettato tramite il tasto PESO ELICOTTERO SCARICO.
- CONSUMO CARB/H: indica il consumo orario di carburante (misurato in Kg per ora). Il valore viene inserito tramite l'apposita barra valori e deve essere accettato tramite il tasto CONSUMO CARB/H.
- HANGAR/SUL PONTE: switch che indica la posizione iniziale dell'elicottero che può essere sul ponte di volo oppure in hangar.
- PALE CHIUSE/PALE APERTE: switch che indica lo stato iniziale dell'elicottero che può essere a pale aperte oppure a pale chiuse.
- ACCESO/SPENTO: switch che indica lo stato iniziale dell'elicottero che può essere a motori in moto quindi acceso oppure spento (lo stato non indica che l'elicottero è ingaggiato).
- EFFICIENTE/INEFFICIENTE: switch che indica lo stato iniziale dell'elicottero che può essere efficiente oppure no. Un elicottero inefficiente è inutilizzabile per l'operazione.

- SALVA RECORD E RESETTA: dopo l'inserimento i dati richiesti devono essere salvati con il tasto SALVA RECORD E RESETTA che inoltre avrà l'effetto di azzerare i campi per l'inserimento del dato successivo.
- PRECEDENTE: viene visualizzato il record precedentemente inserito tramite il tasto PRECEDENTE.
- SUCCESSIVO: viene visualizzato il record successivo a quello attuale, se esiste, tramite il tasto SUCCESSIVO.
- ELIMINA RECORD: viene eliminato il record corrente tramite il tasto ELIMINA RECORD.
- ACCETTA DATI – TORNA AL MENU PRINCIPALE: i record inseriti vengono accettati e sarà in questo modo possibile tornare al menu precedente della schermata generale per l'inserimento delle informazioni successive.
- L'attore può modificare i dati inseriti precedentemente selezionando il record di interesse, modificandone il dato e salvando nuovamente il record.

4.2.8 Inserisci dati truppe

- Premendo il Tasto OPZIONI per truppe dalla schermata principale appare la schermata d'inserimento delle informazioni per le truppe: INSERISCI PESO BASICO DI OGNI SOLDATO IN KG.
- Nella finestra INSERISCI PESO BASICO DI OGNI SOLDATO IN KG viene inserito il peso in KG (peso medio) di un singolo soldato. Il Dato viene accettato con il tasto ACCETTA DATO E TORNA AL MENU PRINCIPALE.

4.2.9 Inserisci dati cargo

Precondizione: deve essere stato inserito un numero di RECORD tramite il tasto di CONFERMA per il Cargo dal menu principale.

- Premendo il Tasto OPZIONI per cargo dalla schermata principale appare la schermata d'inserimento delle informazioni per il cargo. L'inserimento delle informazioni sul Cargo NON è un dato necessario per l'avvio della simulazione. Il cargo potrebbe infatti anche non esistere per effettuare la simulazione. Sono indispensabili invece le truppe.
- L'attore deve inserire le informazioni richieste:
    - MATERIALE TRASPORTATO: indica il nome attribuito al cargo per l'operazione (Può ad esempio trattarsi di acqua, armi e munizioni, zaini…).
    - PESO COMPLESSI MATERIALE: indica il peso del solo CARGO (ossia del materiale da caricare a bordo dell'elicottero escluso il peso delle truppe necessarie che viene calcolato a parte).
    - PERSONALE NECESSARIO AL TRASPORTO: indica il numero di persone necessarie alla movimentazione del cargo che dovranno essere caricate a bordo dell'elicottero. In base al peso del singolo elicottero inserito nella finestra d'inserimento dei dati delle truppe viene calcolato infatti il peso totale delle truppe necessarie al cargo che poi andranno a far parte del peso TOTALE destinato al CARGO e sottratto dal peso destinato al cargo settato sull'elicottero.
- SALVA RECORD E RESETTA: dopo l'inserimento i dati richiesti devono essere salvati con il tasto SALVA RECORD E RESETTA che inoltre avrà l'effetto di azzerare i campi per l'inserimento del dato successivo.
- PRECEDENTE: viene visualizzato il record precedentemente inserito tramite il tasto PRECEDENTE.
- SUCCESSIVO: viene visualizzato il record successivo a quello attuale, se esiste, tramite il tasto SUCCESSIVO.
- ELIMINA RECORD: viene eliminato il record corrente tramite il tasto ELIMINA RECORD.
- ACCETTA DATI – TORNA AL MENU PRINCIPALE: i record inseriti vengono accettati e sarà in questo modo possibile tornare al menu precedente della schermata generale per l'inserimento delle informazioni successive.
- L'attore può modificare i dati inseriti precedentemente selezionando il record di interesse, modificandone il dato e salvando nuovamente il record.

4.2.10 Inserisci dati spot

Precondizione: deve essere stato inserito un numero di RECORD tramite il tasto di CONFERMA per gli SPOT dal menu principale.

- Premendo il Tasto OPZIONI per spot dalla schermata principale appare la schermata d'inserimento delle informazioni per gli spot.
- L'attore deve inserire le informazioni richieste:
    - EFFICIENZA SPOT: tramite una checkbox viene inserito lo stato di efficienza dello spot.
    - OPERAZIONI DIURNE: tramite le checkbox visualizzate viene inserita la disponibilità dello spot ad accettare per operazioni diurne:
        - ELICOTTERI LEGGERI
        - ELICOTTERI PESANTI
        - ALTRE CLASSI DI ELICOTTERO
    - OPERAZIONI NOTTURNE: tramite le checkbox visualizzate viene inserita la disponibilità dello spot ad accettare per operazioni notturne:
        - ELICOTTERI LEGGERI
        - ELICOTTERI PESANTI

        - ALTRE CLASSI DI ELICOTTERO
- SALVA RECORD E RESETTA: dopo l'inserimento i dati richiesti devono essere salvati con il tasto SALVA RECORD E RESETTA che inoltre avrà l'effetto di azzerare i campi per l'inserimento del dato successivo.
- RECORD PRECEDENTE: viene visualizzato il record precedentemente inserito tramite il tasto RECORD PRECEDENTE.
- RECORD SUCCESSIVO: viene visualizzato il record successivo a quello attuale, se esiste, tramite il tasto SUCCESSIVO.
- ELIMINA RECORD: viene eliminato il record corrente tramite il tasto ELIMINA RECORD.
- ACCETTA DATI – TORNA AL MENU PRINCIPALE: i record inseriti vengono accettati e sarà in questo modo possibile tornare al menu precedente della schermata generale per l'inserimento delle informazioni successive.
- L'attore può modificare i dati inseriti precedentemente selezionando il record di interesse, modificandone il dato e salvando nuovamente il record.

4.2.11 Visualizza elicotteri

L'attore ha la possibilità di monitorare l'elenco degli elicotteri inseriti grazie all'uso di una finestra informazioni. Tramite il bottone VISUALIZZA ELENCO posto nella sezione elicotteri verranno visualizzate le principali informazioni inserite.

4.2.12 Visualizza cargo

L'attore ha la possibilità di monitorare l'elenco del cargo inserito grazie all'uso di una finestra informazioni. Tramite il bottone VISUALIZZA ELENCO posto nella sezione cargo verranno visualizzate le principali informazioni inserite.

4.2.13 Visualizza spot

L'attore ha la possibilità di monitorare l'elenco degli spot inseriti grazie all'uso di una finestra informazioni. Tramite il bottone VISUALIZZA ELENCO posto nella sezione spot verranno visualizzate le principali informazioni inserite.

4.2.14 Inserisci opzione GIORNO/NOTTE

L'attore inserisce l'opzione GIORNO/NOTTE dal menu si simulazione. Alcuni spot sono utilizzabili sono di giorno mentre altri giorno e notte, il parametro è utile per identificare l'utilizzabilità dello spot.

4.2.15 Avvia simulazione

Precondizione: devono venire inseriti tutti i parametri necessari alla simulazione quali:

Numero e dati elicotteri disponibili, numero e dati truppe disponibili, numero e dati cargo disponibili, numero e dati spot disponibili, distanza Landing zone.

L'attore preme il tasto START oppure utilizza il menu di simulazione per avviare la simulazione. Se i dati sono stati inseriti correttamente verrà visualizzata la schermata grafica che avvierà la sequenza temporale di eventi per elicotteri, truppe e cargo. La simulazione avrà termine quando tutte le truppe e tutto il cargo saranno trasportate sulla LZ.

4.2.16 Visualizza risultato simulazione

Precondizione: deve essere stato effettuato l'avvio della simulazione che deve avere avuto un esito positivo.

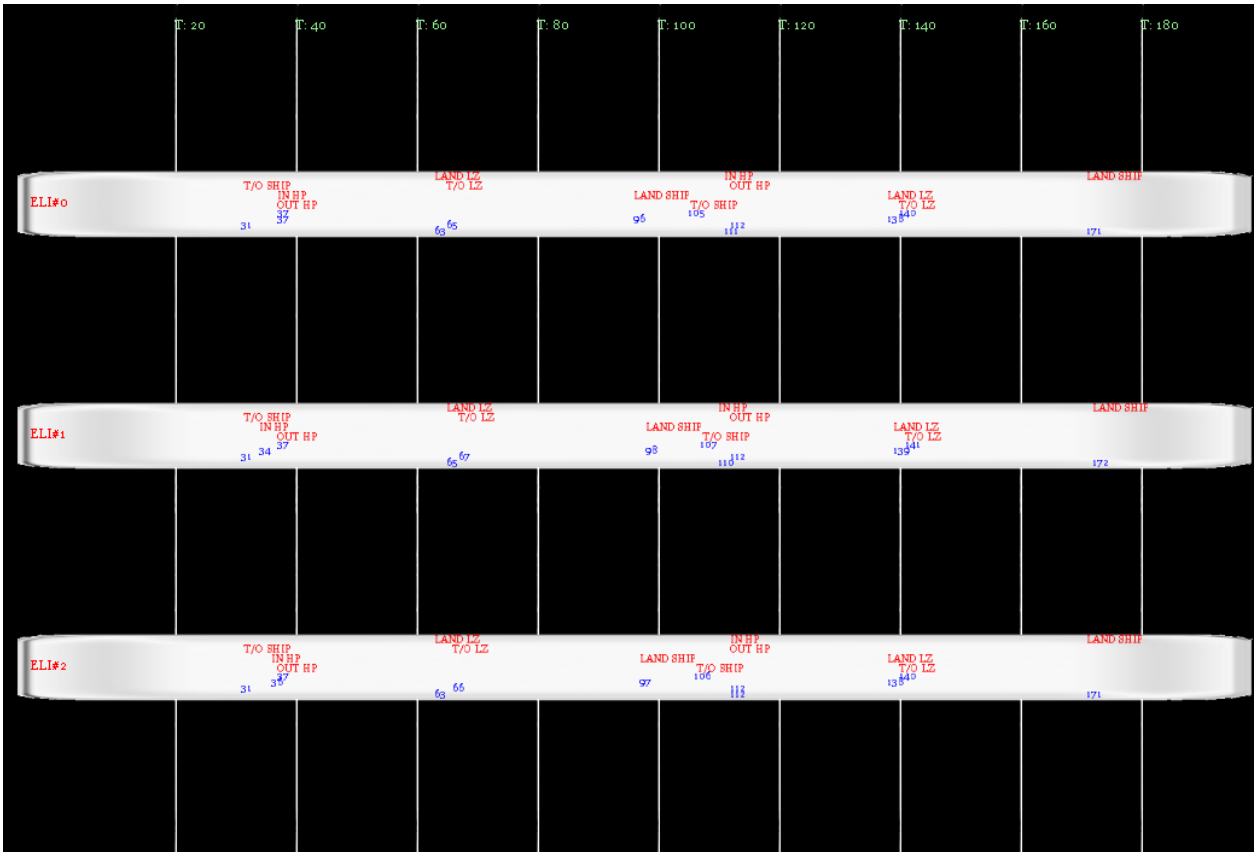
Al termine della simulazione è possibile visualizzare per ogni elicottero le tempistiche e i dati salienti intercorsi. La visualizzazione avviene tramite barre rappresentanti gli elicotteri con le tempistiche in minuti (in blu) in cui è stata effettuata una determinata azione con l'azione svolta (in rosso). Qualora la simulazione non abbia avuto un termine verrebbe visualizzato il messaggio di simulazione fallita.

Viene mostrata una schermata di un possibile risultato a titolo di esempio

In alto in verde sono intervallati i minuti, suddivisi in archi da 20, le barre bianche indicano gli elicotteri ELI#. In rosso per ogni elicottero vengono espressi gli eventi salienti intercorsi, tali eventi sono allineati con la tempistica in minuti espressa in blu.

Ad esempio per l'ELI#0 possiamo dire che sono avvenuti due atterraggi sulla landing zone (LAND LZ) il primo dopo 63 minuti, il secondo dopo 138 minuti.

Vengono inoltre espresse le entrate in holding point (IN HP) le uscite dagli holding point (OUT HP), gli atterraggi sulla nave (LAND SHIP) e sulla Landing Zone (LAND LZ).



5 ANALISI E PROGETTAZIONE

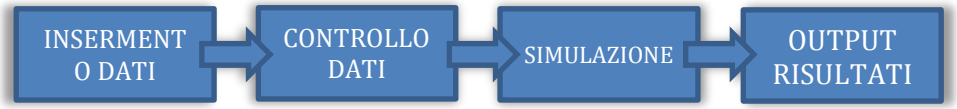
Vengono ora espresse le principali scelte inerenti la progettazione del simulare.

Per lo sviluppo del software è prevalso l'utilizzo di un modello di processo incrementale in cui ad ogni livello, partendo da un prodotto base sono state aggiunte iterativamente funzioni più specifiche e complesse.

Il progetto viene essenzialmente diviso in due grosse fasi progettabili separatamente

- Inserimento delle informazioni necessari (tramite interfaccia GUI Gtk)
- Avvio della simulazione (uso di MONOGAME e librerie grafiche openAL)

Naturalmente i dati inseriti poi devono essere controllati ed al termine della simulazione deve essere riportato all'utente un output.



Tra le principali decisioni affrontate riguardano l'utilizzo di un'interfaccia GUI per la creazione del menu principale e per l'inserimento di tutti i dati necessari, inoltre si è dovuto scegliere un metodo di visualizzazione grafica della simulazione in se, avvalendosi in quest'ultimo caso di framework comunemente usate nel campo dei video giochi.

A tal scopo sono state utilizzati:

- *Gtk# (Gimp Toolkit)* versione 2.12 per la creazione delle interfacce grafiche.
- *MonoGame* versione 3.0.1: implementazione open source del framework Microsoft XNA.

Durante la fase di progettazione sono state sviluppate le classi che meglio descrivo il problem space e ne permettono la risoluzione.

Gli oggetti istanza delle classi che interagiscono tra loro sono innumerevoli e vengono differenziati nelle classi principali che costituiscono gli attori principali quali: ELICOTTERI, TRUPPE, SPOT, CARGO, HP, LZ e classi di supporto che garantiscono una corretta interazione quali ELIMANAGER, CARGOMANAGER, TROOPSMANAGER e infine le classe di gestione grafica che permettono la visualizzazione delle schermate.

Viene divisa l'interfaccia utente, creata tramite il supporto fornito da Xamarin Studio (la parte View), dalla parte che rappresenta i dati dell'applicazione e le regole che governano le modalità con cui questi dati vengono acceduti e modificati (Model) e dalla parte che gestisce le richieste dell'utente e le azioni che devo essere intraprese sul Model (Controller).

Durante la fase di progettazione sono state sviluppate prima le classe degli attori principali, a seguire le classi per l'inserimento delle informazioni e le GUI, infine le classi di gestione delle interazioni tra gli attori e la classe di gestione grafica con il supporto di MONOGAME.

Viene ora fornita una descrizione delle principali classi implementate. La descrizione generale del diagramma delle classi è stato suddiviso, per motivi di spazio in tre blocchi principali quali: le classi fisiche, le classi di visualizzazione GUI e di inserimento dati ed infine le classi di gestione grafica della simulazione.

5.1 CLASSI PRINCIPALI: GLI ATTORI FISICI

Sono state determinate delle classi di attori che corrispondono fisicamente nel mondo reale gli oggetti che devono intervenire nella simulazione. Ad esempio per poter effettuare uno sbarco occorreranno degli elicotteri, delle truppe da movimentare, probabilmente del cargo, il tutto andrà spostato su una LZ e probabilmente gli elicotteri dovranno rimanere in holding su un HP. Gli oggetti sopra definiti sono le classi le cui istanze costituiscono gli attori della simulazione.

Nessuno degli oggetti fisici è stato ritenuto unico (come classe singleton), anche se formalmente al livello di logica di programmazione non è stata prevista ad esempio una seconda LZ o un secondo HP. Tuttavia non è detto che per principio di riusabilità non si possa prevedere in futuro di poterne creare altri.

5.1.1 CLASSE ELICOTTERO

La classe elicottero contiene attributi e metodi per definire l'istanza, eredita dalla classe *IOBserver* il metodo Update il cui *override* produce un aggiornamento del timer di bordo (pattern *Observer*)

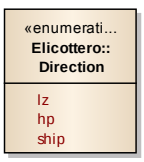
Gli attributi della classe vengono differenziati in differenti tipologie che determinano:

- Stato dell'oggetto: ne fanno parte gli attributi che determinano uno stato dell'elicottero, ad esempio isFlying indica un elicottero in volo, isFull indica un elicottero carico di truppe e Cargo, isLZ indica un elicottero che è sulla LZ. Gli attributi di stato hanno il prefisso "is".
- Determinazione dell'oggetto: ne fanno parte quegli attributi che svolgono una funzione quantitativa e sull'oggetto. Ad esempio un elicottero possiede una velocità, un timer, una direzione, una quantità di carburante, delle truppe a bordo. Questi attributi sono riportabili a quelli che effettivamente e fisicamente possiamo riscontrare nel mondo reale.

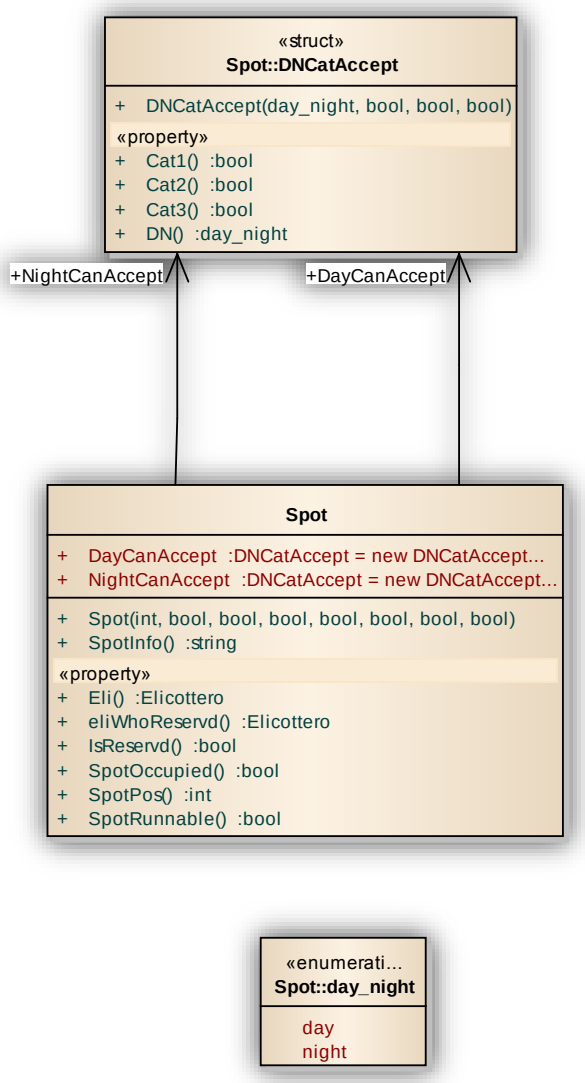
Dato che stato e determinazione dell'oggetto possono essere modificati dal manager gli attributi risultano pubblici e non privati alla classe.

Tra i metodi implementati dalla classe elicottero di particolare importanza ritroviamo:

Update(subject) che sfruttando il pattern observer *è* utile per definire il clock dell'elicottero. Ogni elicottero è infatti dotato di un timer utile per determinare i tempi di attesa. Tramite la variabile *isBlocked* è infatti possibile determinare dei tempi di attesa del velivolo mentre sta effettuando la determinata operazione. Per rendere la simulazione realistica è necessario conoscere le tempistiche per svolgere delle singole attività, ad esempio per spostare un elicottero dall'hangar al ponte di volo possono volerci 15 minuti (15 secondi simulati) l'elicottero deve restare quindi in blocco il periodo di tempo necessario non potendo compiere altre azioni.



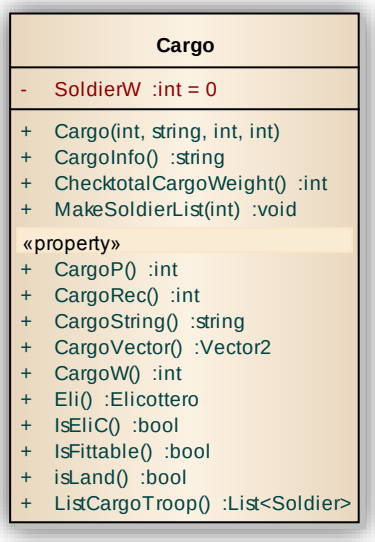
5.1.2 CLASSE SPOT



La classe *spot* serve per definire le caratteristiche dello spot da cui può decollare o su cui può appontare o un elicottero.
Deve essere in grado di contenere un elicottero pertanto tra i suoi attributi si trova *Eli,* che occupa lo spot e *eliWhoReserved () infatti u*n elicottero può prenotarsi ed ottenere priorità su uno spot anche se non lo occupa da subito, serve ad esempio agli elicotteri in rientro dalla *LZ*, oppure agli elicotteri ancora in Hangar che stanno per essere movimentati e che sono attualmente in fase di blocco temporale.

E' stata inoltre definita una struttura per definire l'accettabilità diurna o notturna di un una categoria eli: *DNCatAccept,* infatti per poter utilizzare un determinato spot oltre che la categoria deve coincidere anche l'utilizzabilità diurna o notturna.
Non tutte le categorie di elicottero sono impiegabili di giorno e di notte su un determinato Spot. E' possibile ad esempio che una categoria 1 di elicottero leggero possa appontare di giorno e di notte mentre una categoria 2 pesante possa appontare solo di notte.
La classe ha poi delle variabili di stato che indicano se l'elicottero è efficiente *SpotRunnable*() oppure se è occupato *SpotOccupied*().

5.1.3 CLASSE CARGO



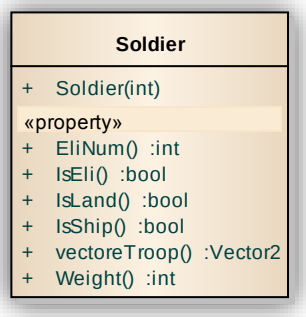
La classe *cargo* serve per definire le caratteristiche dello cargo che deve essere trasportato da un elicottero alla LZ.
Il Cargo è composto sia dal materiale che deve essere trasportato, sia dal personale (con il proprio peso) necessario al suo trasporto.
Il peso del personale viene calcolato in base al peso del singolo elemento inserito per le truppe.
Tra le variabili di stato disponibili sono state inserite quelle che indicano la posizione del cargo, in particolare il cargo può avere tre locazioni: nave, terra e a bordo dell'elicottero.
Viene inoltre determinato in fase iniziale se il cargo in questione può essere trasportato da almeno un elicottero.
Qualcosa non lo fosse il cargo non è gestibile e quindi risulta *isFittable=false* e non viene più considerato.

Il cargo ha inoltre una variabile *Eli* ad indicare quale elicottero viene impiegato per il suo trasporto, ed una lista di truppe corrispondenti ai soldati che andranno ad effettuare il trasporto dello stesso (nella realtà per muovere un cargo sono necessari alcuni uomini che per tale scopo viaggiano con il cargo stesso).

Il metodo *ChecktotalcargoWieght* () serve ad effettuare un calcolo del peso complessivo dell'oggetto creato, in quanto come accennato il peso viene definito dalla sommatoria del peso per il cargo ed il peso dei soldati che invece servono per il trasportano.

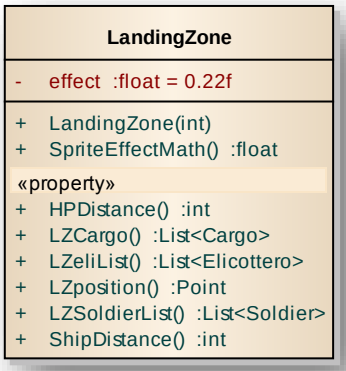
Esistono più cargo che possono imbarcare sullo stesso elicottero ma lo stesso cargo non può trovarsi su più elicotteri contemporaneamente, quindi la relazione è 1 solo elicottero per molti cargo.

5.1.4 CLASSE SOLDIER



La classe *soldier (soldato)* serve per definire le caratteristiche delle truppe che deve essere trasportato da un elicottero alla LZ.
La classe ha delle variabili di stato per indicare la posizione delle truppe ed una variabile peso generica.
Il peso di un soldato singolo, una volta definito un peso medio generale, non può essere modificato all'interno del programma.
Esistono più truppe che possono imbarcare sullo stesso elicottero ma lo stesso soldato non può trovarsi su più elicotteri contemporaneamente, quindi esiste una relazione 1 al molti tra elicottero e soldati.

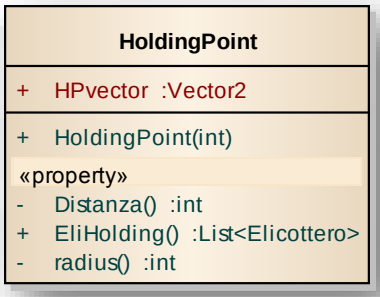
5.1.5 CLASSE LANDING ZONE



La classe LZ istanzia l'oggetto che serve ad identificare il punto di atterraggio degli elicotteri per il rilascio delle truppe e del cargo.

Non esistono variabili di stato in quanto non necessarie. La LZ può tuttavia contenere Cargo, Truppe ed Elicotteri sono state quindi previste delle liste di contenimento per gli oggetti che possono essere rilasciati.

5.1.6 CLASSE HOLDING POINT



La classe HP istanzia l'oggetto che serve ad identificare il punto di attesa per elicotteri, ossia quell'immaginario punto nello spazio attorno al quale gli elicotteri dovranno ruotare restando in attesa che si verifico altri eventi.

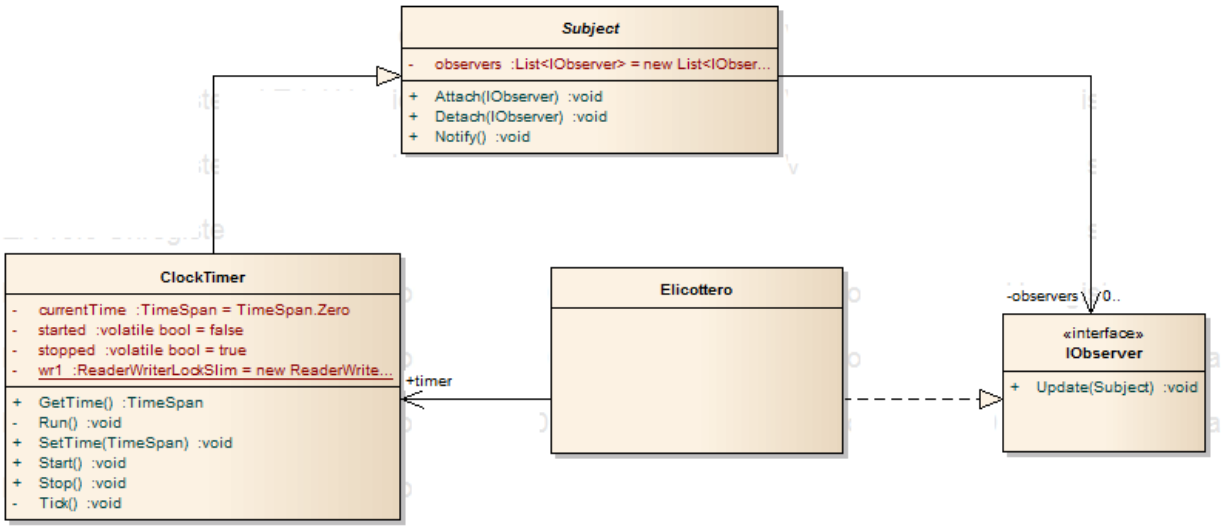
Ad esempio nello sbarco anfibio è necessario che le forze siano impiegate tutte insieme per massimizzare le capacità di attacco, in questo modo i primi elicotteri decollati dovranno restano in attesa del ricongiungimento degli altri per poi poter dirigere tutti insieme sulla LZ.

Altra funzione di attesa è per il rientro degli elicotteri dalla LZ, se ad esempio gli elicotteri che atterrano sulla LZ sono tre mentre esistono solo due spot disponibili, uno dei tre elicotteri dovrà restare in attesa fino a quando almeno uno degli spot non si sarà liberato da una situazione di stallo.

L'HP deve tenere traccia degli elicotteri che sono in attesa da e per la LZ, quindi conterrà una lista il cui controllo verrà verificato ciclicamente fino a quando non si determinino le condizioni di uscita degli elicotteri dalla condizione di attesa per dirigere o su uno spot o sulla LZ.

5.1.7 La classe Clock Tmer: pattern observer[1]

Ogni elicottero, come già accennato, dispone di un timer che serve per bloccarlo da altre funzioni critiche. La necessità di avere un mutamento sull'oggetto timer che venga propagata a tutti gli altri elementi istantaneamente, rende questo pattern di sicuro utilizzo per gli scopi del programma.



Il subject, contiene la lista degli observers, la lista è dinamica quindi qualunque numero di observers può far parte della lista. Gli elicotteri che hanno necessità di essere sincronizzati su un unico timer sono quindi gli osservatori concreti che entrano a far parte della lista, si connettono e disconnettono attraverso il metodo *Attach () e Detach ().*

Gli elicotteri ereditano dalla classe IObserver il metodo *update ()* (come viene mostrato nel paragrafo per la classe elicottero) per l'update del time generale, salvando il risultato nella variabile current_time. Il parametro subject della classe *IOBserver* permette all'observer di determinare chi è il soggetto mettendolo in gradi di gestire più subject alla volta.

Il clock timer fa uso di threading che permette pertanto di scorporare il calcolo del tempo basato sui tick dal resto dell'esecuzione del programma, si è tentato di migliorare l'aspetto *thrade safety* inserendo dei *lock* di lettura e scrittura.

La partenza del clock nel programma avviene in fase di inizializzazione della logica del programma, subito dopo l'inserimento dei dati iniziali. Quello che conta ai fini della simulazione è infatti avere un orario di start dal quale è necessario prendere le tempistiche per poter effettuare l'operazione. Se ad esempio inserisco 2 elicotteri in hangar con due spot liberi, il simulatore dovrà prendere in considerazione anche le tempistiche per movimentare gli elicotteri sul ponte e per caricare il cargo.

[1] Il pattern Observer compare nelle lezioni si Seng, vista l'utilizzabilità del pattern non sono state apportate modifiche basiche a quello riportato nelle lastrine.

5.2 CLASSI DI MANAGEMENT DELLE ISTANZE FISICHE: I MANAGER

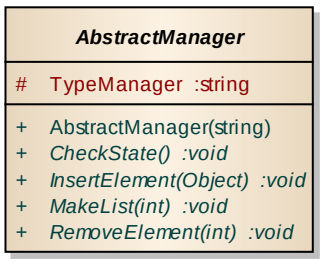
Per la gestione delle istanze sono state costituite delle classi il cui unico scopo è quello di controllo e coordinamento sulle istanze di classi per gli attori fisici. Le classi di gestione (manager) esistono per poter compiere delle azioni agli oggetti del mondo fisico (elicotteri, truppe, spot…).

Ad esempio le truppe dovranno scendere e salire dall'elicottero, dovranno sbarcare su una LZ, mentre gli elicotteri dovranno decollare, appontare, dovranno ad esempio essere effettuati dei controlli sul carburante e sulle quantità rimaste. Tutte queste azioni vengono coordinate tramite metodi delle classi di management.

Le classi di manager dovranno inoltre collaborare per potersi scambiare le informazioni necessarie e poter operare sinergicamente. Devono inoltre contenere un metodo *CheckState ()* in grado di verificare lo stato di ciascuna lista. Ad esempio per l'eli manager sarà necessario sapere quanti elicotteri sono disponibili in hangar, quanti di questi sono efficienti, quanti non servono per l'operazione di sbarco.

Tutte le classi di management sono uniche, è infatti possibile trovare un unico manager di gestione, pertanto sono tutte classi che istanziano un unico oggetto di tipo *singleton.*

5.2.1 Classe abstract manager

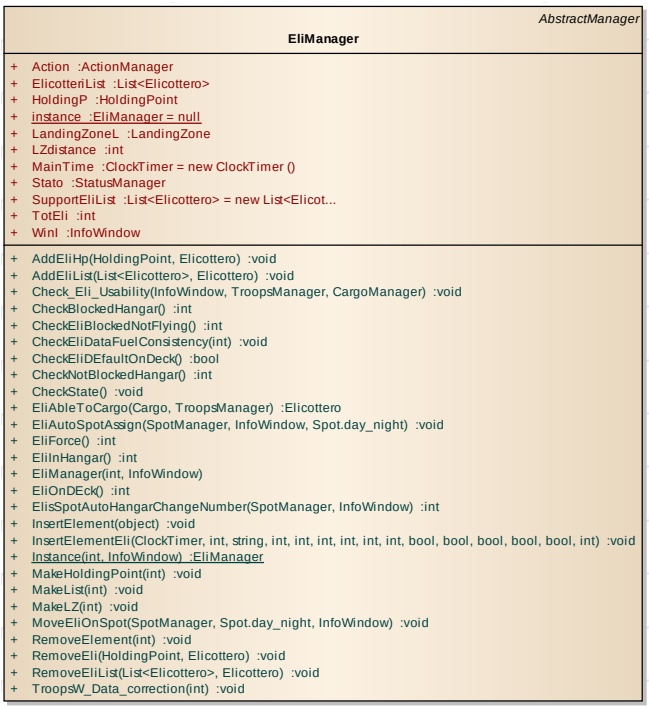


Tutte le classi manager derivano da un classe astratta, che pertanto non può essere direttamente istanziata e che contiene dei metodi comuni a tutti i manager. Principalmente tutti i metodo di management si occupano della gestione delle liste di spettanza quindi tutte le classi manager dovranno implementare almeno i metodi d'inserimento e rimozione e di controllo dello stato.

5.2.2 Classe factory manager

I manager vengono creati tramite un selettore dall'interfaccia grafica di competenza attivato dal metodo Make della classe statica Factory Manager, se ho la necessità di inserire le informazioni sugli elicotteri verrà creato il manager (e quindi anche la lista elicotteri) che si occupa della gestione per gli elicotteri. Questo sistema permette di ritardare la creazione del manager fino al momento del suo effettivo impiego, ossia ogni qualvolta l'utente mostri la sua volontà nell'inserimento effettivo delle informazioni necessarie alla simulazione, premendo il bottone di OPZIONI per ogni singola categoria da inserire. Se ad esempio si ha la necessità di inserire gli spot per poterlo fare mi serve costruire prima le strutture di contenimento degli stessi e le strutture di gestione degli stessi, quindi alla pressione del tasto OPZIONI per gli spot verrà creato anche lo SPOT MANAGER nonché una lista vuota di spot che poi dovrà essere riempita con l'inserimento dei dati necessari.

5.2.3 Classe Eli manager



La classe gestisce gli spostamenti degli elicotteri, è in gradi di tenere traccia di particolari gruppi di elicotteri in modo da determinarne lo status. Serve inoltre per effettuare delle correzioni basiche su input inseriti poco coerenti.

Se ad esempio in fase di immissione delle informazioni l'utente inserisce tre elicotteri dislocati di default sul ponte di volo ma poi all'immissione degli spot vengono inseirti solo due spot efficienti, i dati sono poco coerenti in quanto non esistono spot sufficienti ad ospitare tutti gli elicotteri predisposti di defualt sul ponte. Interviene a tal proposito un metodo di gestione *EliSpotAutoHangarChangeNumber()* che assegna al ponte in realtà solo due degli elicotteri mentre il terzo viene lasciato in hangar e verrà spostato poi successivamente durante la simulazione.

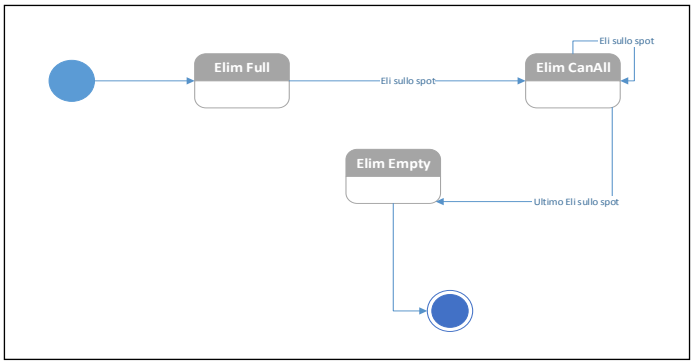
La classe per poter operare sugli elicotteri fa uso di una lista di gestione elicotteri, in realtà l'intero programma si basa su scambi di liste che vengono passati da un contenitore (eli manager) ad un altro contenitore (spot manager, LZ, HP).

La lista elicotteri, generata come proprietà del manager tramite il metodo *MakeList ()* in modo che il manager possa avere come proprietà anche gli elementi che deve gestire, viene incrementata per ogni elemento immesso. Sono stati quindi previsti i metodi d'inserimento e cancellazione degli elementi.

Oltre gli elicotteri che si dovranno spostare su una LZ o su un HP, una delle scelte progettuali è stata quella di implementare come attributo ogni elemento fisico che la classe dovrà poi gestire, acquisendo eventualmente le informazioni mancanti dagli altri manager.

Se ad esempio si ha la necessità di capire quale elicottero della lista ha le proprietà necessarie per trasportare un cargo, si farà uso del metodo *EliAbreToCargo* () che per poter operare ha necessità delle informazioni sul cargo, che gli verranno passate come input.

Le classi manager possono acquisire differenti stati durante l'esecuzione del programma. Lo stato è del manager è un elemento fondamentale che determinata l'andamento del ciclo di update della logica di funzionamento del programma.



Stato Full: indica che l'Hangar è pieno, da questo stato gli elicotteri possono solo diminuire per passare sul ponte, se necessari all'attività di volo.

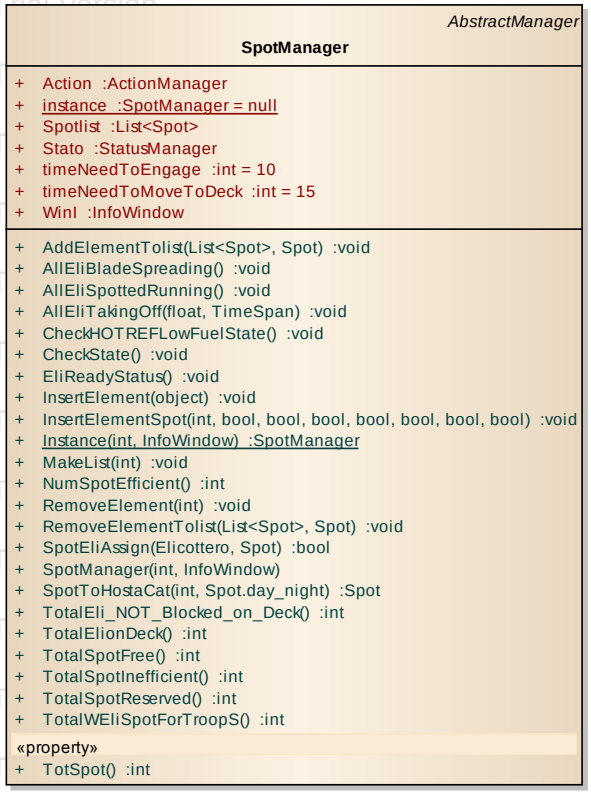
Stato CanAll: indica che l'Hangar ha ancora elicotteri da gestire ma che tale numero non corrisponde al massimo numero di elicotteri disponibili. Gli elicotteri possono solo

diminuire in questo caso venendo trasferiti sul ponte in quanto non è stato previsto un reintegro degli elicotteri in hangar.

Per le finalità dello sbarco anfibio non è previsto che gli elicotteri ritornino in hangar, quindi non è possibile passare da uno stato intermedio di *CanAll* ad uno stato *Full*, così come non è previsto che passino da uno stato *Empty* ad uno stato *Full*, in quanto gli elicotteri se non più necessari all'attività vengono semplicemente eliminati dalla gestione e non possono essere più usati.

Lo stato *Empty* è lo stato finale in cui tutti gli elicotteri sono stati trasferiti sul ponte che poi li deve gestire, passando il controllo allo Spot *Manager*.

5.2.4 Classe spot manager



Lo spot manager trova il suo impiego per la gestione degli spot sul ponte di volo.

Contiene la lista degli spot impiegabili che è stata inserita dall'utente ed è in grado di gestire le funzionalità per l'inserimento o la rimozione di elementi dalla *Spotlist*.

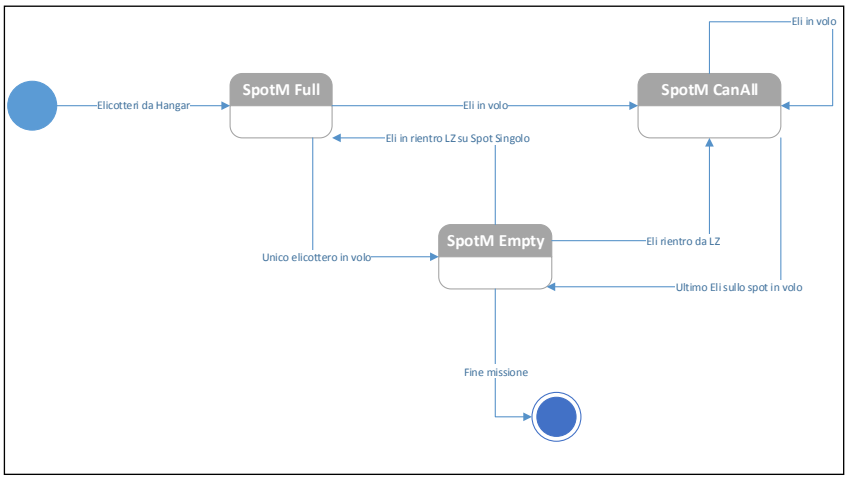
La classe fa uso di metodi di coordinamento per l'uso degli spot, nonché metodi di stato per determinare l'uso del ponte di volo.

Se ad esempio gli unici due spot presenti sul ponte di volo sono occupati, il metodo di stato riporterà uno stato del ponte *full,* qualora invece non dovessimo avere nessuno spot libero il ponte sarà in uno stato *empty.*

I metodi di gestione della classe riguardano principalmente i segnali che devono essere dati agli elicotteri per poter mettere in moto (*AllEliSpottedRunning*), aprire le pale (*AllEliSpottedBladeSpreding*) e per poter decollare (*AllEliTakingOff*) o di controllo sul carburante dell'elicottero prima del decollo (*CheckHotREDLowFuelState*).

Di particolare rilevanza è il metodo *SpotToHostaCat ()* in grado di determinare il primo spot libero e funzionante che può ospitare la categoria dell'elicottero che vogliamo spostare sul ponte di volo.

Lo spot manager possiede degli stati rappresentati con il seguente diagramma:

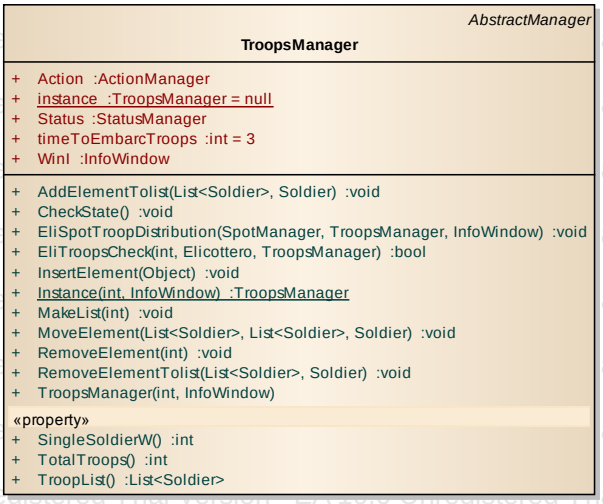


SpotM full: è lo stato in cui il ponte di volo è pieno di elicotteri e non può più acquisirne altri. E' possibile solo diminuire tale numero quando gli elicotteri vanno in volo, oppure eliminando gli elicotteri dalla gestione del manager se non più necessari all'attività.

SpotM CanAll: questo stato è ibrido in quanto il ponte può ancora ospitare altri elicotteri e può tuttavia mandare in volo quelli già presenti.

SpotM Empty: lo stato del ponte è vuoto, in attesa che rientrino gli elicotteri dalla LZ. Qualora il numero di elicotteri sia zero, in quanto le truppe ed il cargo da trasportare sono terminati e tutti gli elicotteri sono stati eliminati dalla lista si transiterà nello stato finale di simulazione terminata.

5.2.5 Classe TroopsManager

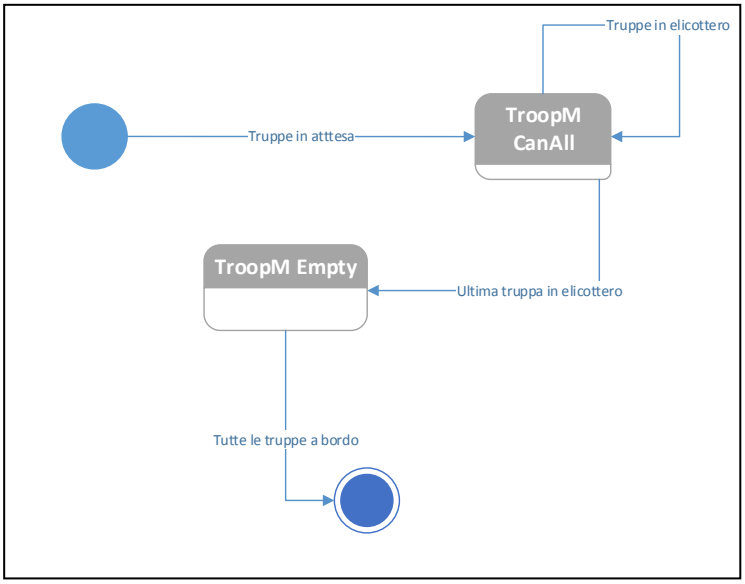


La classe *Troops manager* viene impiegata per la gestione delle truppe. Come anche le altri classe manager, è una classe di tipo *singleton*.

Contiene la lista degli elementi soldato impiegabili che è stata inserita dall'utente ed è in grado di gestire le funzionalità di rimozione, inserimento o spostamento da una lista all'altra.

La classe fa inoltre uso di metodi di gestione per lo stato delle truppe in modo che in ogni momento sia possibile sapere se ho ancora elementi da prelevare e da spostare sulla LZ. Per la distribuzione delle truppe viene impiegato il metodo *EliSpotTroopsDistribution ()*, tale metodo permette la distribuzione uniforme delle truppe su tutti gli elicotteri presenti sul ponte di volo.

Una volta definito se un elicottero è necessario oppure no al trasporto (viene infatti eliminato dalle liste qualora non lo fosse) la scelta progettuale è stata quella di distribuire ciclicamente ogni singolo individuo sugli elicotteri utilizzabili sul ponte di volo anziché effettuare uno spostamento di un blocco di elementi alla volta pari allo spazio presente a bordo.

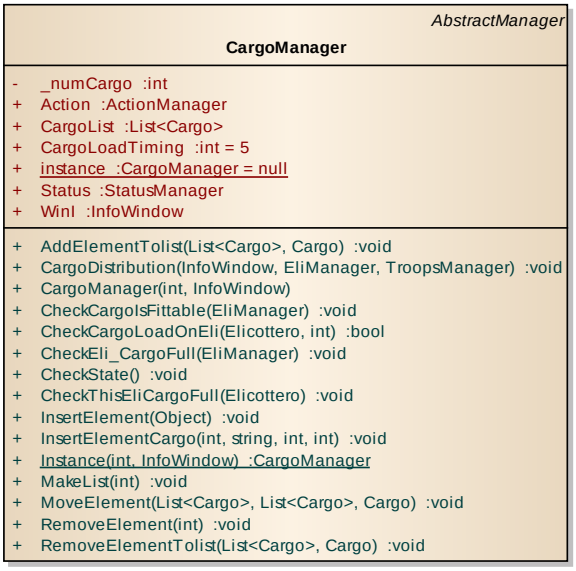


Il manager per le truppe offre solo due stati. Lo stato *CanAll* in cui le truppe possono essere caricate a bordo degli elicotteri oppure lo stato di *Empty* in cui le truppe hanno terminato la fase di boarding e sono tutte a bordo degli elicotteri oppure sulla LZ.

Lo stato di TroopM Empty è uno dei due presupposti per la riuscita della missione in quanto sarà necessario che tutte le truppe siano state scaricate sulla LZ.

In realtà se un elicottero cade in mare per mancanza di carburante e le truppe muoiono la questione è poco influente dal punto di vista termine missione, infatti lo scopo finale che vogliamo raggiungere è sempre quello di aver trasportato tutte le truppe disponibili.

5.2.6 Classe Cargo Manager



La classe Cargo Manager, *singleton*, a similitudine della classe Truppe Manager si occupa della gestione del cargo.

Contiene la lista degli elementi cargo impiegabili che è stata inserita dall'utente ed è in grado di gestire le funzionalità di rimozione, inserimento o spostamento da una lista all'altra.

In particolare per ogni cargo ho la necessità di sapere se ha le caratteristiche per poter essere contenuto da un elicottero, qualora il cargo non fosse trasportabile su nessun elicottero verrebbe generato il segnale di missione fallita in quanto non è stato possibile portare a termine in totale lo spostamento di tutto il materiale previsto per la buona riuscita della missione.

Il manager del cargo, come gli altri manager, implementa i metodi di gestione del cargo.

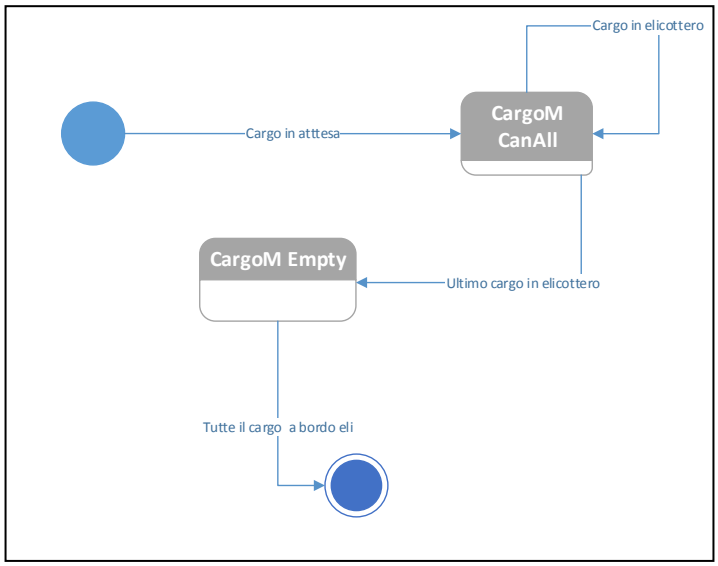
In particolare questi i metodi rilevanti:

CargoDistribution (): serve per distribuire il cargo a bordo degli elicotteri che sono effettuare il trasporto.

CheckCargoIsFittable(): controlla che ogni cargo sia trasportabile da almeno un elicottero.

CheckEli_CargoFull(): controlla che esiste ancora almeno un cargo che può essere inserito ancora in elicottero in modo che possa essere occupato tutto lo spazio disponibile.

Per quanto riguarda gli stati del Cargo Manager



Il manager per il cargo offre solo due stati disponibli. Lo stato di *CanAll* in cui può essere caricato a bordo degli elicotteri oppure lo stato di *Empty* in cui è terminata la fase di boarding e si trova a bordo degli elicotteri oppure sulla LZ.

Lo stato di CargoM empty è uno dei due presupposti per la riuscita della missione.

Le perdite di Cargo causate da un elicottero che ha terminato il carburante ed è caduto in mare non inficiano la riuscita finale della missione.

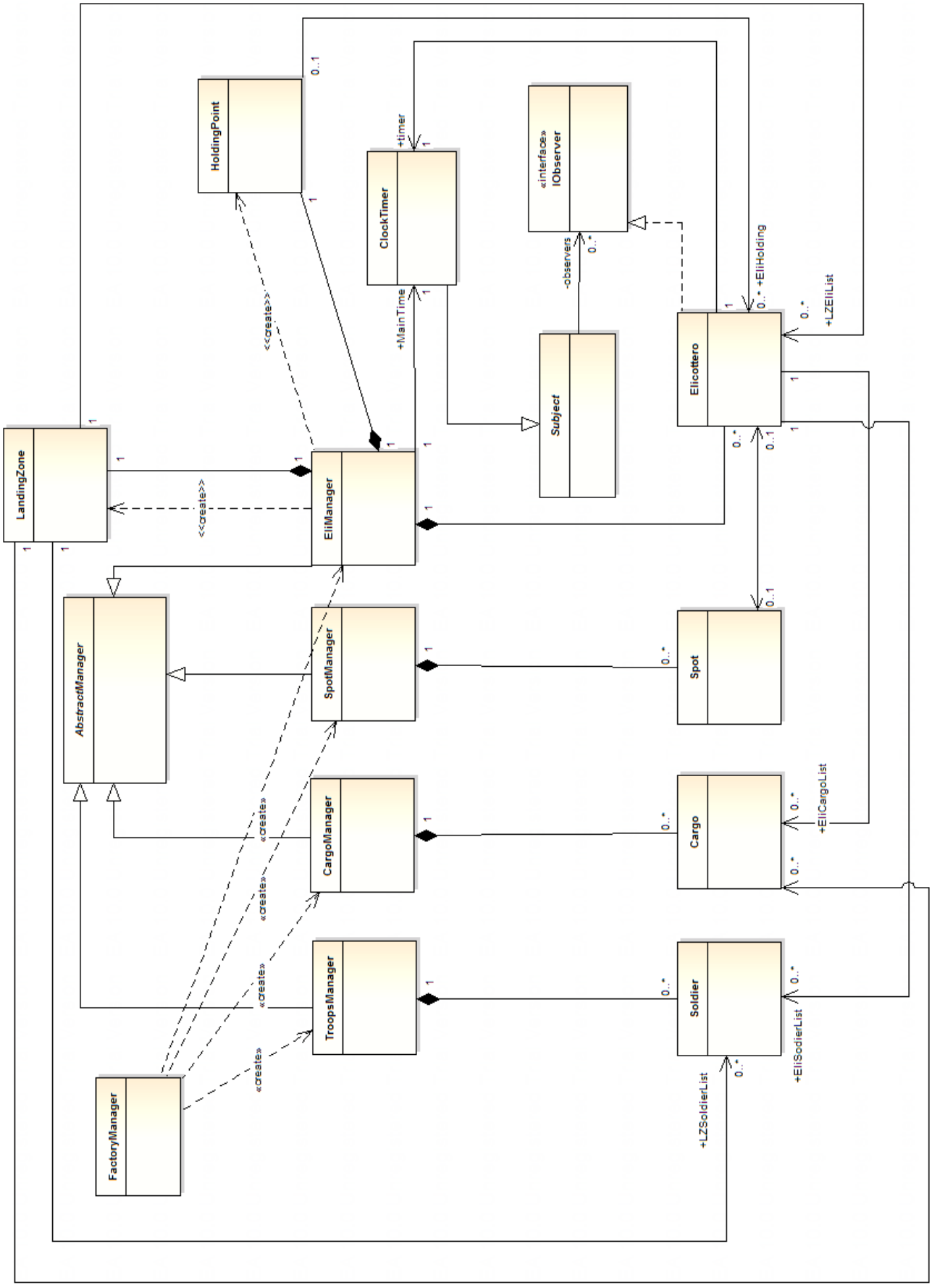
5.2.7 Diagramma delle classi parziale: gli attori FISICI

Nella pagina seguente viene rappresentato un diagramma parziale, che descrive le sole classi fisiche e di management, tali classi vengono espresse senza attributi in un formato ridotto per mancanza di spazio.

Nel diagramma vengono mostrate in particolare:

Le composizioni: ogni manager contiene fisicamente la lista di oggetti che poi deve gestire.

Le generalizzazioni: derivanti dalle classi


HoldingPoint
ClockTimer
«interface»
IObserver
Subject
Elicottero
LandingZone
EliManager
AbstractManager
SpotManager
Spot
CargoManager
Cargo
TroopsManager
Soldier
FactoryManager
+timer
+MainTime
-observers
+EliHolding
+LZEliList
+EliCargoList
+EliSoldierList
+LZSoldierList
«create»

5.3 CLASSI INSERIMENTO DATI E VISUALIZZAZIONE INFORMAZIONI

Alcune delle classi progettate hanno come unica funzione quella di gestire l'inserimento delle informazioni ed il controllo parziale dei dati inseriti. Si è preferito effettuare un inserimento dati tramite l'uso di barre, che da una parte snelliscono il processo di immissione per dati che non necessitano di un'elevata precisione (ad esempio i valori di peso possono essere approssimati, nel mondo reale avere una pesata precisa al kilogrammo non è fattibile) e dall'altra impongono dei limiti strutturali ai dati stessi che devono essere inseriti, le barre offrono infatti dei valori di minimo e massimo dai quali non possiamo uscire, questa funzionalità limita tutta una serie di problematiche che potrebbero essere rilevanti per il controllo dei singoli dati inseriti.

Tutte le finestre d'interfaccia grafica sono create grazie all'uso delle librerie Gtk. La costruzione grafica delle finestre è stata effettuata tramite l'uso del **designer interno di Xamarin studio** e le classi parziali sono state salvate nella directory *Interfaccia utente* dalla stessa IDE.

La parte controllo è stata invece progettata in classi parziali separate che hanno il compito appunto di gestire i dati inseriti.

Il modello MVC (Model View Controller) è spesso implementato con l'utilizzo del pattern observer in modo da poter avere una presentazione delle informazioni costantemente aggiornato. La problematica di mantenere informazioni, controllo e visualizzazione delle GUI separate in classi differenti è stata in questo caso determinata senza l'uso di un pattern observer specifico in quanto non ritenuto necessario.

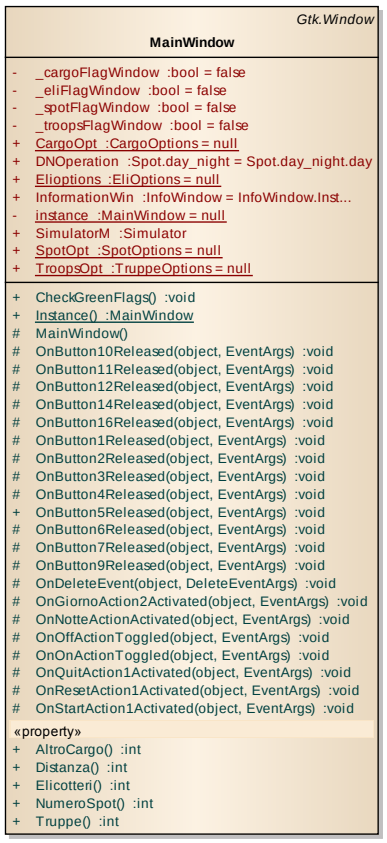
Viene proposta la seguente soluzione:

La View è la classe generata in automatico dal modellatore di GUI di Xamarin e si occupa solo della visualizzazione delle informazioni

Il controller è la classe generata (EliOptions, CargoOptions, TruppeOptions, SpotOptions) che viene attivato dalla View ogni volta che viene premuto un pulsante o modificata una barra accettando il dato inserito. Il controller aggrega una serie di azioni che l'utente può adottare sul modello quali inserimento dati in una lista, cancellazione dati da un alista, visualizzazione del record ecc.

Il model sono i dati e le strutture che devono essere modificati contenute nelle classi Manager.

5.3.1 Classe MainWindow



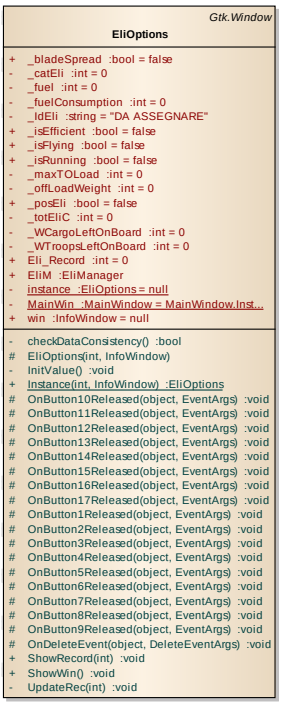
Classe singleton di controllo che si occupa dell'inserimento generale delle informazioni necessarie al programma quali:

- Numero di elicotteri
- Numero di truppe
- Numero di elementi cargo da trasportare
- Numero di spot disponibili
- Distanza della landing zone dalla nave

Una delle scelte progettuali adottate è stata quella di bloccare ogni dato inserito proteggendolo da scritture successive, per cui una volta che il numero di elicotteri, truppe, spot o cargo viene definito questo non può più essere più modificato.

In figura sottostante viene rappresentata la view che si occupa della visualizzazione dell'interfaccia. (Il file non è stato allegato ma viene mostrata solo la sua rappresentazione grafica)

5.3.2 Classe EliOptions



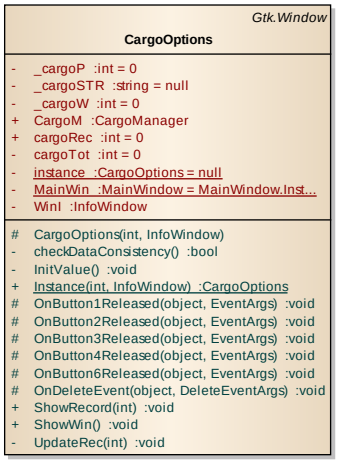
La classe nasce per l'inserimento delle informazioni di ogni elicottero, è una classe singleton, in quanto è necessaria una sola ed unica istanza.

Ogni dato inserito deve essere accettato singolarmente, quando il record risulta correttamente inserito è sufficiente salvare il record. Una volta salvato i campi verranno azzerati per il successivo inserimento.

I dati una volta inseriti vengono controllati tramite il metodo *checkDataconsistency ()* che è in grado di determinare la correttezza dei pesi inseriti per l'elicottero.

In figura è possibile vedere l'interfaccia per l'inserimento dati (view). (Il file non è stato allegato ma viene mostrata solo la sua rappresentazione grafica)

5.3.3 Classe CargoOptions



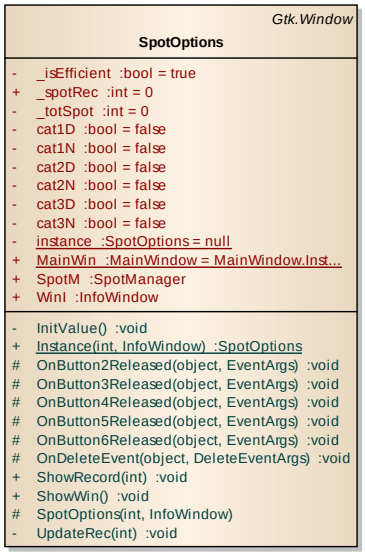
La classe nasce per l'inserimento delle informazioni di ogni cargo, è una classe singleton in quanto è necessaria una sola ed unica istanza.

Il metodo *checkDataconsistency ()* effettua un controllo sui dati inseriti per verificarne la congruenza.

In figura è possibile vedere l'interfaccia per l'inserimento dati (view).

(Il file non è stato allegato ma viene mostrata solo la sua rappresentazione grafica).

5.3.4 Classe SpotOptions



La classe viene definita per il controllo dell'inserimento delle informazioni su ogni spot disponibile, è una classe singleton in quanto è necessaria una sola ed unica istanza.

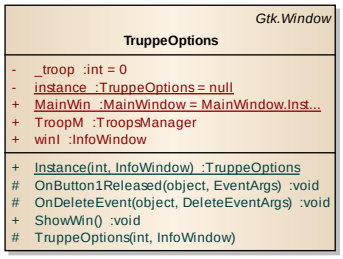
Il controllo sui dati inseriti è effettuato alla pressione del tasto SALVA RECORD, non sarà infatti possibile accettare valori di cargo di peso 0.

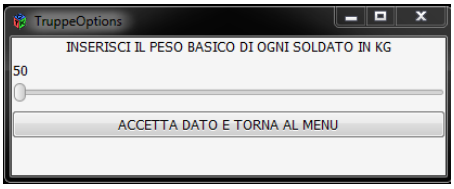
In figura è possibile vedere l'interfaccia per l'inserimento dati (view).

(Il file non è stato allegato ma viene mostrata solo la sua rappresentazione grafica)

5.3.5 Classe TruppeOptions

La classe viene definita per il controllo dell'inserimento delle informazioni su ogni spot disponibile, è una classe singleton in quanto è necessaria una sola ed unica istanza.



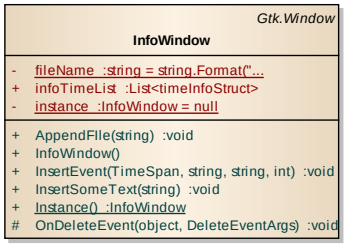


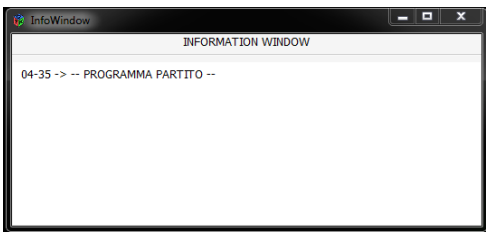
In figura è possibile vedere l'interfaccia per l'inserimento dati (view).

(Il file non è stato allegato ma solo la sua rappresentazione grafica)

5.3.6 Classe InfoWindow

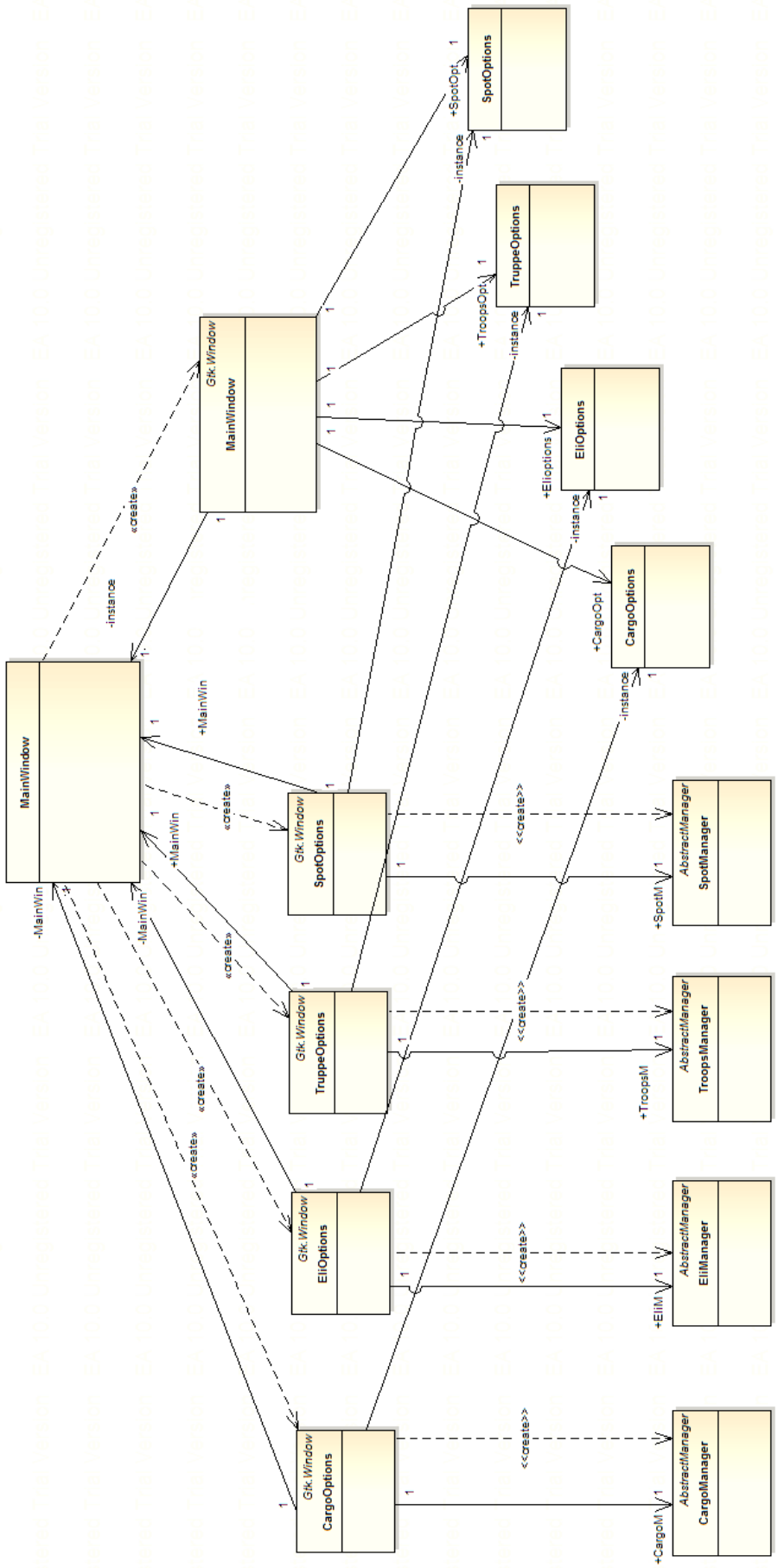
La classe singleton viene definita per visualizzare le informazioni principali e per dare un feedback all'utente durante l'inserimento dei dati o in caso di errori. La classe inoltre determina il salvataggio in un file di Log di tutte le informazioni che sono state visualizzate.





(Il file della view non è stato allegato ma viene mostrata solo la sua rappresentazione grafica)

5.3.7 Diagramma delle classi parziale: le classi GUI

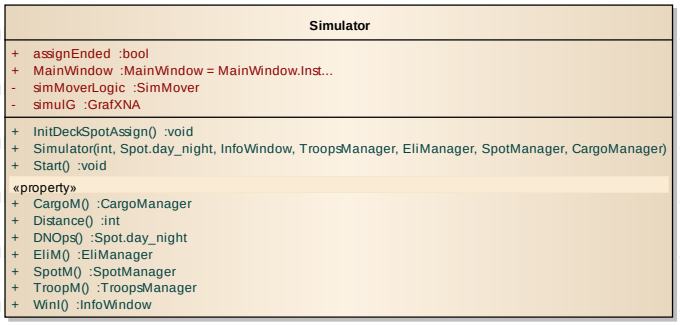


5.4 CLASSI GESTIONE SIMULAZIONE E VISUALIZZAZIONE GRAFICA

Sono tutte quelle classi necessarie alla gestione della simulazione, che quindi si occupano della gestione logica di tutti gli elicotteri delle truppe e del cargo da sbarcare e che si occupano di tradurre gli stati e le posizioni in una visualizzazione grafica.

Per quanto riguarda la gestione grafica, come vedremo, sono state impiegate le librerie di MonoGame (porting open source di XNA).

5.4.1 Classe Simulator

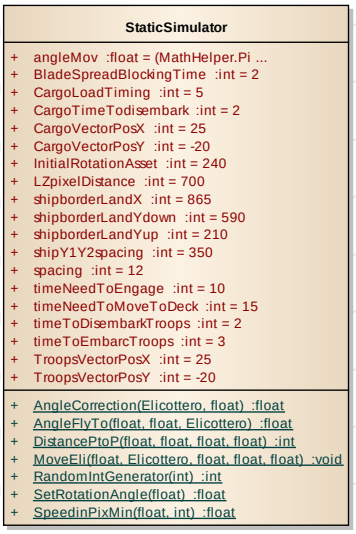


La Classe Simulatore permette l'inizializzazione ed il controllo dei dati inseriti ma in maniera più approfondita e inter relazionali.

Sebbene singolarmente i dati di un elicottero possano essere stati inseriti correttamente deve essere verificata anche l'interazione tra tutte le informazioni disponibili. E' necessario ad esempio che il carburante inserito sia in grado quantomeno di ricoprire la distanza voluta, oppure è necessario che la quantità di cargo disponibile sia quantomeno superiore al peso minimo tra i valori cargo inseriti.

Una volta effettuato il controllo iniziale, il simulatore richiama la classe di gestione grafica GrafXNA eseguendola.

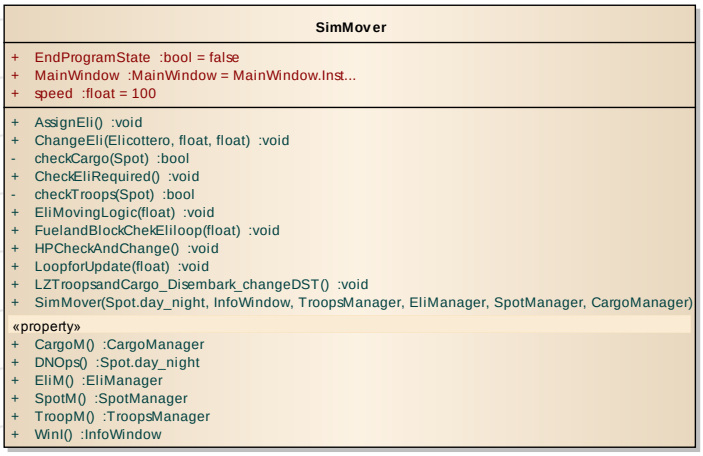
5.4.2 Classe StaticSimulator



E' stata inserita anche una classe statica, principalmente per assolvere i calcoli matematici utili per la parte grafica riguardanti il volo degli elicotteri, direzione, distanza dell'obiettivo da raggiungere.

Sono state inoltre raggruppate nella classe tutte le costanti utili per la rappresentazione di oggetti sullo schermo che servono ad identificare delle posizioni statiche. Per ora infatti, come già accennato la simulazione avviene su una grafica costante di 1024x768 punti, quindi alcuni parametri restano definibili sullo schermo tramite dei valori unici.

5.4.3 Classe SimMover



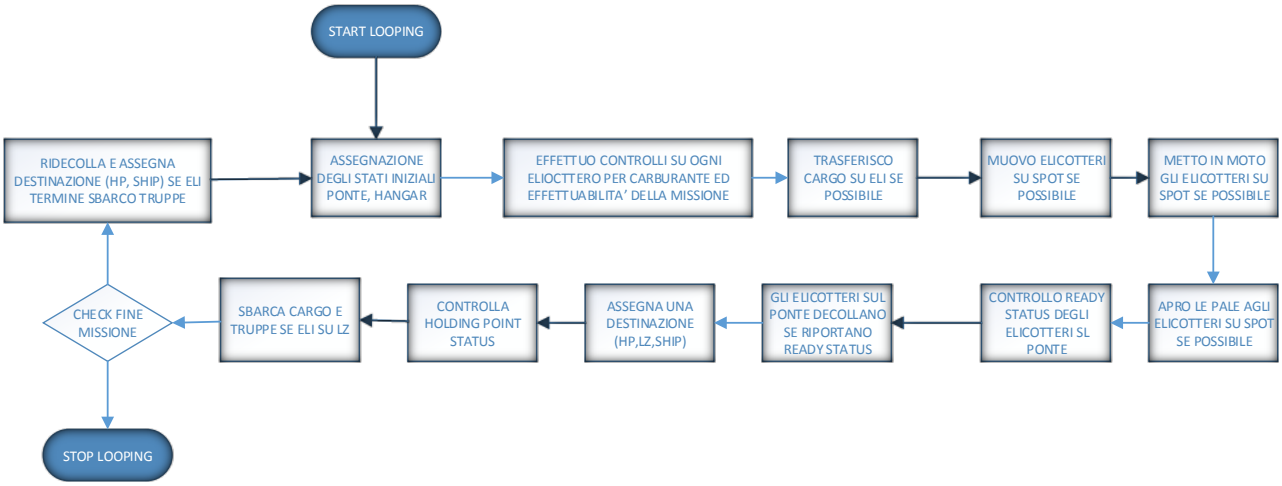
E' il cuore logico del sistema. Permette l'attuazione del flusso logico che deve essere adottato per poter gestire gli eventi che regolano la simulazione.

Il flusso logico dell'intera attività viene definito all'interno del metodo *LoopForUpdate(float)* e viene poi richiamato dalla classe di visualizzazione grafica *GrafXNA ().*

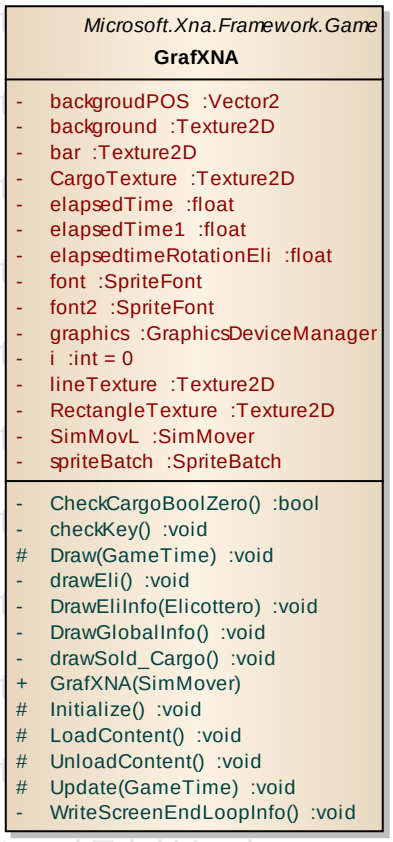
Viene separati due aspetti: logica di funzionamento del programma e visualizzazione grafica della simulazione. Le due tipologie di aspetti vengono trattati da classi separate.

La classe SimMover oltre a svolgere un ruolo fondamentale per le operazioni logiche, raggruppa anche un numero di metodi utili per il controllo dei dati. Ad esempio nel ciclo ho la necessità di controllare se gli elicotteri sul ponte hanno il carburante necessario per proseguire la missione (operazione di controllo) ed eventualmente devo effettuare un rifornimento (operazione logica).

In figura viene mostrato il loop delle operazioni che vengono compiute nell'esecuzione logica del programma, l'uscita dal loop avviene quando tutte le truppe ed il cargo (che è trasportabile negli elicotteri) hanno raggiunto destinazione oppure se gli elicotteri dichiarano tutti di non poter effettuare la missione azzerando la propria lista di contenimento e determinando in quest'ultimo caso un'uscita dal programma per missione fallita.



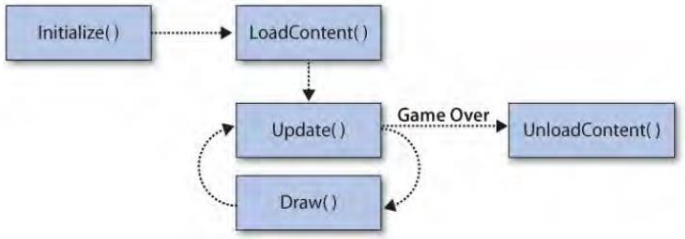
5.4.4 Classe GrafXNA



La classe GrafXNA costituisce il motore grafico per la visualizzazione della simulazione.

Eredita dalla classe Microsoft.Xna.Framework.Game alcuni metodi necessari per:

- Inizializzazione: vengono inserite le informazioni per inizializzare i dati e le strutture prima di poter iniziare il ciclo
- Caricamento del content: immagini, video, suono.
- Looping logic (Update): update della parte logica che viene separata dalla parte di visualizzazione grafica.
- Looping grafico (Draw): update della parte grafica che viene separata dalla parte logica.



L'update logico è stato scorporato dall'update grafico con l'impiego della classe SimMover sopra descritta che si occupa prettamente della parte logica.

La parte grafica è stata invece suddivisa in differenti metodi in modo da scorporare in differenti blocchi ogni tipologia di oggetto che poi viene visualizzato:

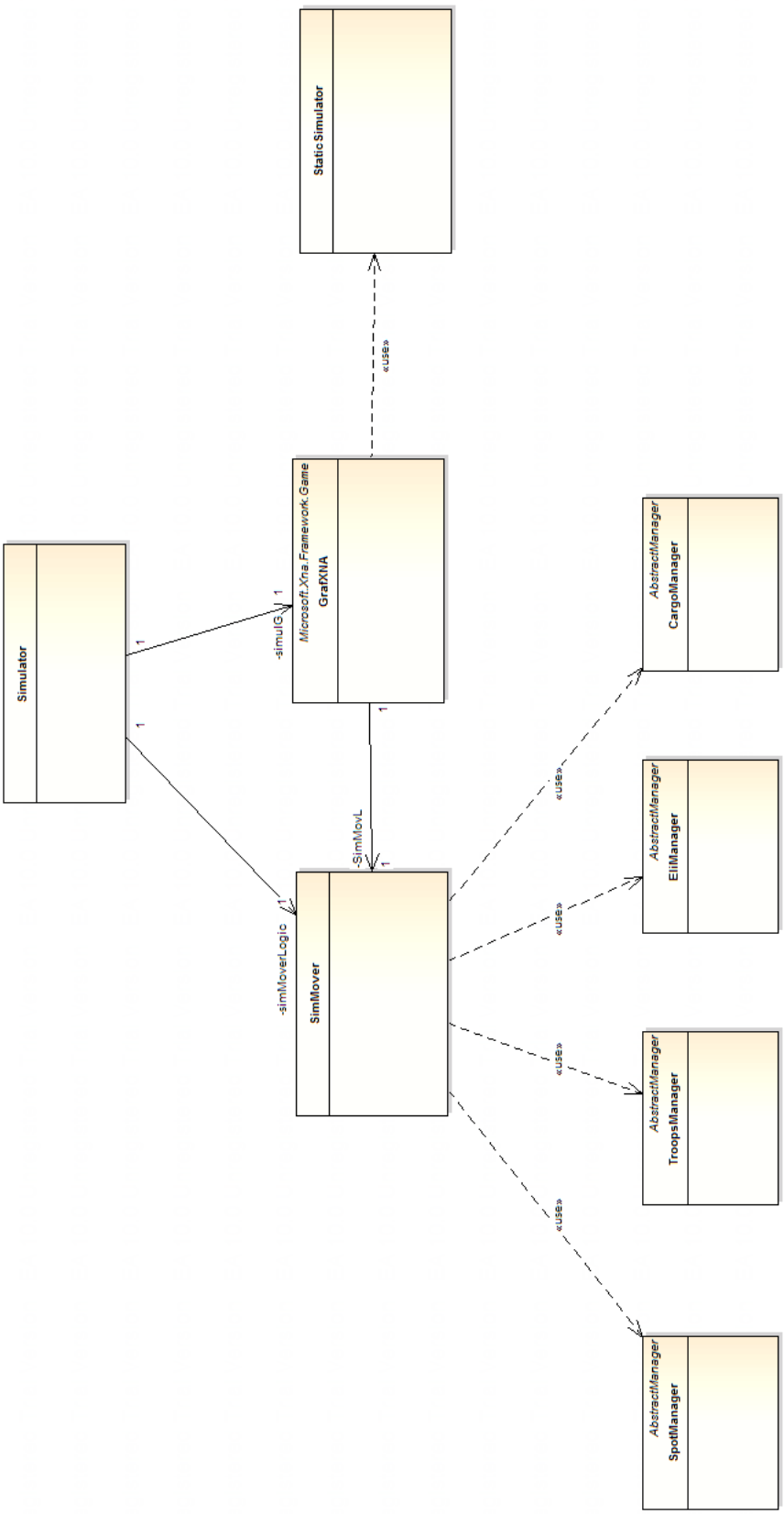
- *Draweli():* si occupa della rappresentazione dei soli elicotteri.
- *DrawSold_Cargo():* si occupa della rappresentazione del cargo e dei soldati.
- *DraweliInfo():* si occupa della scrittura delle info sugli elicotteri ottenibili premendo il tasto numpad corrispondente al record.
- *WriteScreenEndLoop():* si occupa della rappresentazione finale dei dati estrapolati dalla struttura *timeInfoStruct* creata dalla classe *InfoWindows* in cui vengono inseriti le principali informazioni utili per una rappresentazione finale.

5.4.5 Diagramma parziale: simulazione e rappresentazione grafica

Nella pagina seguente viene riportato un diagramma parziale per le classi di rappresentazione grafica che completa, assieme agli altri diagrammi parziali la figura d'insieme dei diagrammi.

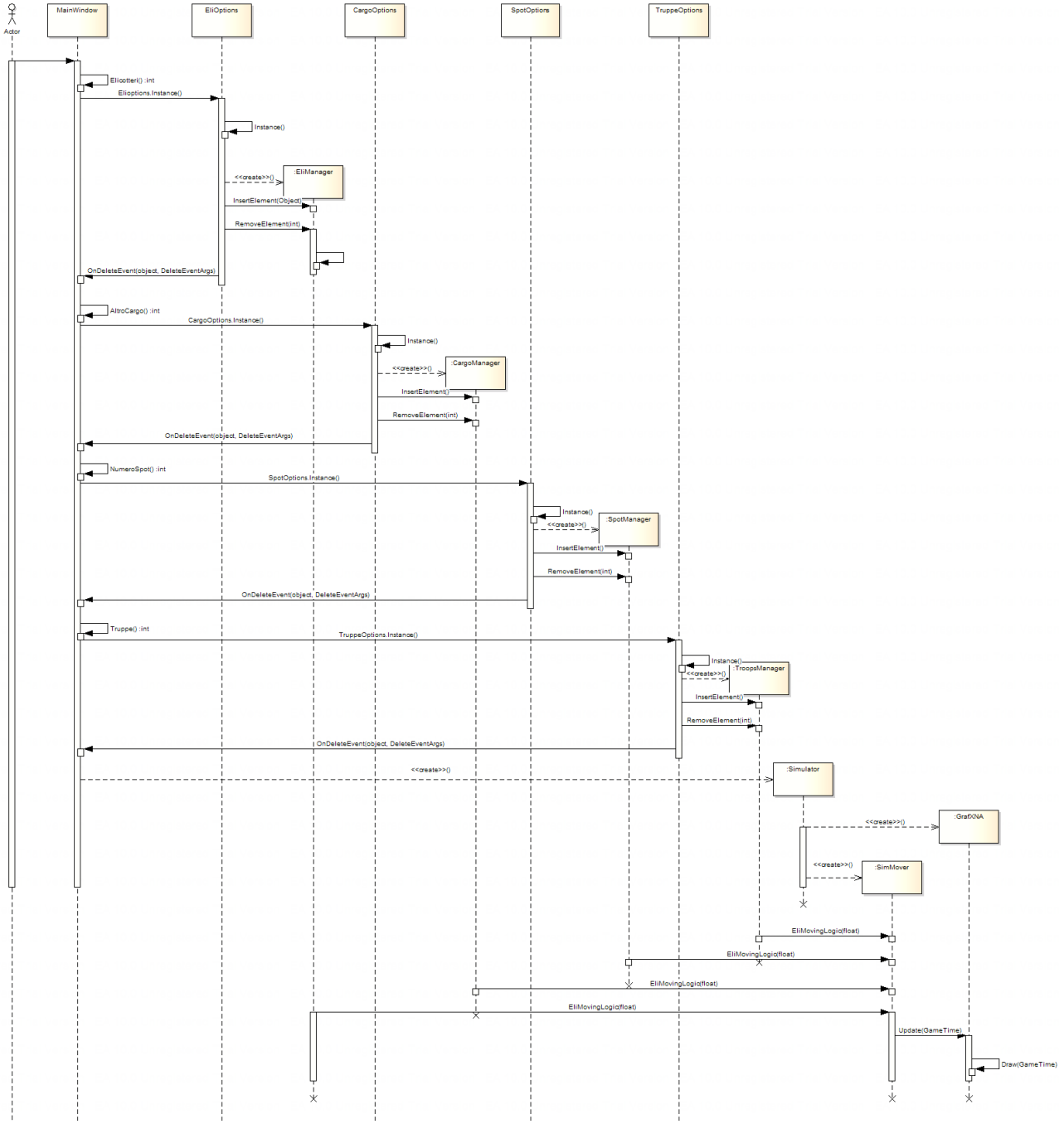
Per quanto possibile, in quanto MonoGame utilizza un proprio sistema di rappresentazione e gestione della grafica già descritto, si è cercato di scorporare rappresentazione grafica (la view) dal modello e dal controllo. In questo caso la classe SimMover si occupa del controllo, ossia della sezione logica di funzionamento che a sua volta determina una rimodulazione della view con la classe GrafXNA. Il controller è poi in grado di gestire i dati forniti dal modello e aggregati dalle classi Manager.

Il tutto viene modulato con il loop di azioni sopra descritto con cui è strutturato MonoGame.


StaticSimulator
«use»
Simulator
-simulG
1
1
1
Microsoft.Xna.Framework.Game
GrafXNA
1
-SimMovL
1
-simMoverLogic
1
SimMover
«use»
«use»
«use»
«use»
AbstractManager
CargoManager
AbstractManager
EliManager
AbstractManager
TroopsManager
AbstractManager
SpotlManager

5.5 DIAGRAMMA DI SEQUENZA

Viene descritto sommariamente il digramma di sequenza con la creazione dei manager attraverso le finestre d'interfaccia grafica. Dalla finestra MainWindow è possibile richiamare la creazione della simulazione dopo l'inserimento di tutte le informazioni necessarie.



6 IMPLEMENTAZIONE

Sono stati esclusi dai file di implementazione tutte le view per la gestione delle interfacce grafiche, in quanto le classi parziali sono state autogenerate da Xamarin Studio, è stato inoltre escluso il Makefile in quanto generato grazie al sistema di Xamarin che ne permette la creazione.

Vengono pertanto riportati solo i file di interesse creati manualmente per la funzionalità dell'applicazione. La simulazione prevede inoltre differenti file grafici in formato png salvati nella directory Content.

6.1 FILE MAINPROGRAM

```
//++++++++++++++++++++++++++++++++++++++
// punto iniziale di start del programma
//++++++++++++++++++++++++++++++++++++++

using System;
using Gtk;

namespace EliAssaltoAnfibio
{
    class MainClass
    {

        public static void Main (string[] args)
        {
            Application.Init ();
            MainWindow win = MainWindow.Instance();

            win.Show ();
            win.InformationWin.InsertSomeText( "-- PROGRAMMA PARTITO --");
            Application.Run ();
        }
    }
}
```

6.2 FILE MAINWINDOW

```
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// La classe di occupa del controllo gli eventi e delle attivazioni
// dei bottoni della schermata principale.
// La classe non è ripetibile quindi viene trattata come singleton
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

using System;
using Gtk;
using System.Collections.Generic;
using System.Windows.Forms;

namespace EliAssaltoAnfibio
{
    public partial class MainWindow: Gtk.Window
    {
        private static MainWindow instance = null;
        public Spot.day_night DNOperation = Spot.day_night.day;
        // setto le operazioni a day di default
        public int Elicotteri { set; get; }
        // indica il numero di elicotteri partecipanti
        public int Truppe { set; get; }
        // indica le truppe partecipanti
        public int Distanza { set; get; }
        // indica la distanza dall'obiettivo
        public int AltroCargo { set; get; }
        // indica se esiste dell'altro cargo
        public int NumeroSpot { set; get; }
        // indica il numero di spot disponibili
        //all'apertura delle finestre opzioni viene creato anche il Manager di ogni classe che poi verra' passato
        //come parametro allo Start della simulazione, quindi mi serve una flag per indicare l'apertura di ogni finestra
        //e quindi la definizione dei dati necessari per l'inizio della simulazione.
        private bool _eliFlagWindow = false;
        // se vero indica l'apertura della finestra EliOptions la prima volta
        private bool _troopsFlagWindow = false;
        // se vero indica l'apertura della finestra TruppeOptions per la prima volta
        private bool _spotFlagWindow = false;
        // indica l'apertura della finestra opzioni spot per la prima volta
        private bool _cargoFlagWindow = false;
        // indica l'apertura della finestra cargo
        public InfoWindow InformationWin = InfoWindow.Instance ();
        // viene creata la schermata di informazioni generali.
        static public EliOptions Elioptions = null;
        // finestra opzioni spot elicottero
        static public TruppeOptions TroopsOpt = null;
        // finestra opzioni truppe
        static public SpotOptions SpotOpt = null;
        // finestra opzioni spot
        static public CargoOptions CargoOpt = null;
        // finestra opzioni cargo
```

```
public Simulator SimulatorM;
// COSTRUTTORE DELLA CLASSE
protected  MainWindow () : base (Gtk.WindowType.Toplevel) // costruttore singleton richiamato da istance
{
    // singleton il costruttore è richiamato dall'istanza
    this.Build ();// disegna la grafica del menu
    this.Elicotteri = 0;
    this.Truppe = 0;
    this.Distanza = 0;
    this.AltroCargo = 0;
    this.NumeroSpot = 0;
    this.InformationWin.Show ();// finestra informazioni visibile
    // inizializzo i manager a null


}
// creazione di un metodo pubblico di accesso al costruttore SINGLETON
public static MainWindow Instance ()
{
    if (instance == null)
        instance = new MainWindow ();
    return instance;

}
// gestione degli eventi
// uscita per pressione tasto X di uscita dal programma
protected void OnDeleteEvent (object sender, DeleteEventArgs a)
{
    InformationWin.Destroy ();
    if (Elioptions != null) {
        Elioptions.EliM.MainTime.Stop (); // stop timer
        Gtk.Application.Quit ();
    }

    Gtk.Application.Quit ();// quit application
}
// MENU REGION
// MENU quit, uscita per selezione quit
protected void OnQuitAction1Activated (object sender, EventArgs e)
{
    InformationWin.Destroy ();
    if (Elioptions != null) {
        Elioptions.EliM.MainTime.Stop (); // stop timer
        Gtk.Application.Quit ();
    }
    Gtk.Application.Quit ();// quit application

}
// MENU RESET BUTTON, resetting dell'applicazione.
protected void OnResetAction1Activated (object sender, EventArgs e)
{
    if (Elioptions != null)
        Elioptions.EliM.MainTime.Stop (); // stop timer
    System.Windows.Forms.Application.Restart ();
    Gtk.Application.Quit ();

}
// MENU visualizza information window ON - rende la schermata d'informazioni visibile
protected void OnOnActionToggled (object sender, EventArgs e)
{
    InformationWin.Show ();
}
// menu nascondi information window - rende la schermata informazioni invisibile
protected void OnOffActionToggled (object sender, EventArgs e)
{
    InformationWin.Hide ();
}
//menu - opzioni simulazione - operazioni diurne
protected void OnGiornoAction2Activated (object sender, EventArgs e)
{
    DNOperation = Spot.day_night.day; // setto lo sbarco diurno
    InformationWin.InsertSomeText ("MAIN WINDOW: selezionata operazione DIURNA");

}
// menu - opzioni simulazione - operazioni notturne
protected void OnNotteActionActivated (object sender, EventArgs e)
{
    DNOperation = Spot.day_night.night; // setto lo sbarco notturne
    InformationWin.InsertSomeText ("MAIN WINDOW: selezionata operazione NOTTURNA");

}
// bottone: ACCETTA NUMERO ELICOTTERI
protected void OnButton1Released (object sender, EventArgs e)
{
    if (this.Elicotteri == 0) {
        this.Elicotteri = (int)hscale1.Value;
        label5.Text = "ELICOTTERI -> " + this.Elicotteri + " UNITA'";
        InformationWin.InsertSomeText ("MAIN WINDOW: Effettuato inserimento del NUMERO di elicotteri");
    } else {
        InformationWin.InsertSomeText ("MAIN WINDOW: numero di elicotteri già inserito");
        hscale1.Value =    this.Elicotteri;
    }

}
// bottone: ACCETTA NUMERO TRUPPE
protected void OnButton2Released (object sender, EventArgs e)
{
    if (this.Truppe == 0) {
        this.Truppe = (int)hscale2.Value; // assegna il valore alla variabile truppe
        label6.Text = "TRUPPE -> " + this.Truppe + " UNITA'";
```

```
            InformationWin.InsertSomeText ("MAIN WINDOW: Effettuato inserimento del NUMERO di truppe");

        } else {
            InformationWin.InsertSomeText ("MAIN WINDOW: numero di truppe già inserito");
            hscale1.Value =    this.Elicotteri;
        }
    }
    // bottone: ACCETTA ALTRO CARGO
    protected void OnButton3Released (object sender, EventArgs e)
    {
        if (this.AltroCargo == 0) {
            this.AltroCargo = (int)hscale3.Value;// assegna il valore alla variabile del cargo
            label7.Text = "ALTRO CARGO -> " + this.AltroCargo + " UNITA'";
            InformationWin.InsertSomeText ("MAIN WINDOW: Effettuato inserimento di ALTRO CARGO");
            this.label7.ModifyFg (StateType.Normal, new Gdk.Color (120, 120, 1));
        } else {
            InformationWin.InsertSomeText ("MAIN WINDOW: numero cargo già inserito");
            hscale1.Value =    this.Elicotteri;
        }
    }
    // bottone: ACCETTA DISTANZA
    protected void OnButton4Released (object sender, EventArgs e)
    {
        if (this.Distanza == 0) {

            this.Distanza = (int)hscale4.Value;// assegna il valore alla variabile della distanza
            label8.Text = "DISTANZA -> " + this.Distanza + " MILES";
            InformationWin.InsertSomeText ("MAIN WINDOW: inserita la distanza della landing zone dall'unità navale");
            this.label8.ModifyFg (StateType.Normal, new Gdk.Color (1, 120, 1)); // DISTANZA GREEN FLAG
        } else {
            InformationWin.InsertSomeText ("MAIN WINDOW: distanza già inserita");
            hscale1.Value =    this.Elicotteri;

        }
    }
    // bottone: ACCETTA NUMERO SPOT
    protected void OnButton12Released (object sender, EventArgs e)
    {

        if (this.NumeroSpot == 0) {

            this.NumeroSpot = (int)hscale5.Value;
            label12.Text = "SPOT -> " + this.NumeroSpot + " UTILIZZABILI";
            InformationWin.InsertSomeText ("MAIN WINDOW: Effettuato inserimento del NUMERO di SPOT");

        } else {
            InformationWin.InsertSomeText ("MAIN WINDOW: numero di spot già inserito");
            hscale1.Value =    this.Elicotteri;

        }
    }
    // bottone: OPZIONE  ELICOTTERI - determina l'accesso all'inserimento dati per ogni singolo elicotteri
    public void OnButton5Released (object sender, EventArgs e)
    {
        if (this.Elicotteri == 0) { //controllo se esistono elicotteri da inserire
            InformationWin.InsertSomeText ("MAIN WINDOW: WARNING !!Nessun elicottero inserito...");
        } else {
            // creazione della finestra inserimento dati elicotteri
            // alla classe grafica viene passato il numero di record per la creazione della lista
            if (this._eliFlagWindow == false) { // permette di effettuare la selezione solo 1 volta
                Elioptions = EliOptions.Instance (this.Elicotteri, this.InformationWin); // richiama l'istanza singleton
                this._eliFlagWindow = true; // imposta la flag di apertura finestra a VERO

                Elioptions.ShowWin ();// mostra la finestra opzioni per gli elicotteri

            } else
                Elioptions.ShowWin ();// mostra la finestra opzioni per gli elicotteri

            if (Elioptions.EliM.ElicotteriList != null && Elioptions.EliM.ElicotteriList.Count > 0)
                Elioptions.ShowRecord (Elioptions.Eli_Record);
        }
    }
    // BOTTONE: visualizzo informazioni sugli ELICOTTERI
    protected void OnButton10Released (object sender, EventArgs e)
    {

        if (Elioptions == null)
            InformationWin.InsertSomeText ("MAIN WINDOW: Nessun elicottero è stato creato al momento...");
        else {
            // stampa il nume eli di ogni elicottero presente nella lista elicotteri
            InformationWin.InsertSomeText ("MAIN WINDOW: E' stata inserita la seguente LISTA elicotteri: ");
            foreach (Elicottero eli in Elioptions.EliM.ElicotteriList) {
                InformationWin.InsertSomeText (eli.EliInfo ()); // scrittura informazioni per ogni elicottero3
            }
        }
    }
    // BOTTONE: opzione gestione truppe
    protected void OnButton6Released (object sender, EventArgs e)
    {
        if (this.Truppe == 0) {

            InformationWin.InsertSomeText ("MAIN WINDOW: WARNING !! INSERIRE UN NUMERO DI TRUPPE MAGGIORE DI 0");
        } else {
            // creazione della finestra inserimento dati Truppe
            TroopsOpt = TruppeOptions.Instance (this.Truppe, InformationWin); // richiama l'istanza singleton per la finestra truppe
            InformationWin.InsertSomeText ("MAIN WINDOW: VISUALIZZUATA SCHERMATA OPZIONI TRUPPE");
            this._troopsFlagWindow = true; // schermata segnalata aperta
            this.Sensitive = false;
            TroopsOpt.ShowWin (); //viene visualizzata la schermata elicotteri e nascosta la schermata principale
```

```
            }

        }
        // bottone: opzioni gestione SPOT per l'iserimento delle informazioni sugli spot
        protected void OnButton11Released (object sender, EventArgs e)
        {
            if (this.NumeroSpot == 0) {

                InformationWin.InsertSomeText ("MAIN WINDOW: WARNING !! INSERIRE UN NUMERO DI SPOT MAGGIORE DI 0");
            } else {
                // creazione della finestra inserimento del cago
                if (this._spotFlagWindow == false) {

                    // creazione della finestra inserimento dati Spot
                    SpotOpt = SpotOptions.Instance (this.NumeroSpot, InformationWin); // richiama l'istanza singleton
                    this._spotFlagWindow = true; // schermata segnalata aperta

                    SpotOpt.ShowWin (); //viene visualizzata la schermata elicotteri e nascosta la schermata principale

                } else
                    SpotOpt.ShowWin ();
                if (SpotOpt.SpotM.Spotlist != null && SpotOpt.SpotM.Spotlist.Count > 0)
                    SpotOpt.ShowRecord (SpotOpt._spotRec); // se la finestra gia' esiste mostra il record
            }
        }
// termine metodo opzioni SPOT
        // BOTTONE: opzioni altro cargo per l'inserimento delle infomazioni
        protected void OnButton7Released (object sender, EventArgs e)
        {

            if (this.AltroCargo == 0) {
                InformationWin.InsertSomeText ("MAIN WINDOW: WARNING !! INSERIRE UN NUMERO DI ELEMENTI CARGO MAGGIORE DI 0");
            } else {
                if (_cargoFlagWindow == false) {
                    CargoOpt = CargoOptions.Instance (this.AltroCargo, this.InformationWin); // richiama l'istanza singleton
                    this._cargoFlagWindow = true; // schermata segnalata aperta
                    this.label7.ModifyFg (StateType.Normal, new Gdk.Color (1, 120, 1));
                    CargoOpt.ShowWin ();
                } else
                    CargoOpt.ShowWin (); //viene visualizzata la schermata elicotteri e nascosta la schermata principale

                if (CargoOpt.CargoM.CargoList != null && CargoOpt.CargoM.CargoList.Count > 0)
                    CargoOpt.ShowRecord (CargoOpt.cargoRec);
               }

        }
        //BOTTONE : visualizza l'elenco  lista ALTRO CARGO
        protected void OnButton14Released (object sender, EventArgs e)
        {
            if (CargoOpt == null)
                InformationWin.InsertSomeText ("MAIN WINDOW: nessun cargo definito al momento...");
            else {
                if (CargoOpt.CargoM.CargoList.Count == 0)
                    InformationWin.InsertSomeText ("MAIN WINDOW: Nessun ALTRO CARGO è stato creato al momento...");
                else {
                    InformationWin.InsertSomeText ("MAIN WINDOW: E' stata inserita la seguente LISTA Altro Cargo: ");
                    foreach (Cargo carg in CargoOpt.CargoM.CargoList) {
                        InformationWin.InsertSomeText (carg.CargoInfo ());// scrivi l'informazioni per ogni cargo
                    }
                }
            }
        }
        //BOTTONE: visualizza l'elenco degli SPOT disponibili
        protected void OnButton16Released (object sender, EventArgs e)
        {
            if (SpotOpt == null)
                InformationWin.InsertSomeText ("MAIN WINDOW: nessuno spot è stato definito al momento...");
            else {
                if (SpotOpt.SpotM.Spotlist.Count == 0) // controllo se la spot list ha almeno un elemento
                InformationWin.InsertSomeText ("MAIN WINDOW: Nessuna LISTA SPOT è stato creata al momento...");
                else {
                    InformationWin.InsertSomeText ("MAIN WINDOW: E' stata inserita la seguente LISTA SPOT: ");
                    foreach (Spot spotL in SpotOpt.SpotM.Spotlist) {
                        InformationWin.InsertSomeText (spotL.SpotInfo ());
                    }
                }
            }
        }
        //BOTTONE: start - simulazione
        protected void OnButton9Released (object sender, EventArgs e)
        {

            if ((this.Distanza > 0)// la distanza e' maggiore di 0
                && _eliFlagWindow
                && !(Elioptions.EliM.ElicotteriList.Count == 0)// sono stati inseriti degli elicotteri
                && _spotFlagWindow
                && !(SpotOpt.SpotM.Spotlist.Count == 0)// sonno stati inseriti degli spot
                && _troopsFlagWindow
                && !(TroopsOpt.TroopM.TroopList.Count == 0)) { // controllo che siano state aperte le finestre per l'inserimento dati
                Elioptions.EliM.LZdistance = this.Distanza; // viene passata la distanza della nave al manager

                // verificare per il cargo che sia esistente <<<<< il cargo puo' anche non essere un parametro necessario alla simulazione
                // i parametri fondamentali sono elicotteri spot , truppe e distanza LZ
                if (CargoOpt == null)
                    SimulatorM = new Simulator (this.Distanza, this.DNOperation, this.InformationWin, TroopsOpt.TroopM, Elioptions.EliM, SpotOpt.
SpotM, null);
                else
                    SimulatorM = new Simulator (this.Distanza, this.DNOperation, this.InformationWin, TroopsOpt.TroopM, Elioptions.EliM, SpotOpt.
SpotM, CargoOpt.CargoM);
```

```
                SimulatorM.InitDeckSpotAssign (); // spot assegnazione

            } else
                InformationWin.InsertSomeText ("MAIN WINDOW: NON E' POSSIBILE INIZIALIZZARE IL SIMULATORE. INSERIRE I DATI MANCANTI...");
        }

        // attivazione dello start simulation attraverso il menu principale
        protected void OnStartAction1Activated (object sender, EventArgs e)
        {

            this.OnButton9Released (sender, e);
        }

        public void CheckGreenFlags ()
        {
            //elicotteri GREEN FLAG
            if (Elioptions != null && Elioptions.EliM != null && Elioptions.EliM.ElicotteriList != null && Elioptions.EliM.ElicotteriList.Count !
= 0)
                this.label5.ModifyFg (StateType.Normal, new Gdk.Color (1, 120, 1));

            // truppe GREEN FLAG
            if (TroopsOpt != null && TroopsOpt.TroopM != null && TroopsOpt.TroopM.TroopList != null && TroopsOpt.TroopM.TroopList.Count != 0)
                this.label6.ModifyFg (StateType.Normal, new Gdk.Color (1, 120, 1));

            // spot GREEN FLAG
            if (SpotOpt != null && SpotOpt.SpotM != null && SpotOpt.SpotM.Spotlist != null && SpotOpt.SpotM.Spotlist.Count != 0)
                this.label12.ModifyFg (StateType.Normal, new Gdk.Color (1, 120, 1));

            if (CargoOpt != null && CargoOpt.CargoM != null && CargoOpt.CargoM.CargoList.Count != 0)
                this.label7.ModifyFg (StateType.Normal, new Gdk.Color (1, 120, 1));

        }
    }
    // fine classe
}
 // fine namespa
```

6.3 FILE ELIOPTIONS

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// CLASSE DI CONTROLLO PER L'INSERIMENTO DATI ELICOTTERI
// SI OCCUPA DELL'INSERIMENTO E DEL CONTROLLO DELLE INFORMAZIONI
// SUGLI ELICOTTERI DA IMPIEGARE
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

using System;
using System.Windows.Forms;
using Gtk;
using System.Drawing;

namespace EliAssaltoAnfibio
{
    public partial class EliOptions : Gtk.Window
    {
        private static EliOptions instance = null;
        // istanza singleton
        private static MainWindow MainWin = MainWindow.Instance ();
        // identifica la finestra principale in modo che possa essere ritrovata alla chiusura di questa
        public int Eli_Record = 0;
        // indica il record trattato, È IL NUMERO DI SEQUENZA DELL'ELICOTTERO
        private int _totEliC = 0;
        // indica il numero voluto di elicotteri inizile
        public EliManager EliM;
        // definizione dell'eli manager
        // VARIABILI DA INSERIRE NEL RECORD ELICOTTERO
        //-----------------------------------------------
        private string _IdEli = "DA ASSEGNARE";
        // nome o matricola che identifica l'elicottero in volo ES. King-01
        private int _catEli = 0;
        // classe di elicottero 1=pesante 2=leggero 3=altro
        private float _fuel = 0;
        // indica il livello di carburante dell'elicottero
        private int _WTroopsLeftOnBoard = 0;
        // indica il numero di truppe imbarcate sull'elicottero
        private int _WCargoLeftOnBoard = 0;
        // indica altri componenti di cargo general purpouse
        private int _maxTOLoad = 0;
        // max Take Off load indica il peso massimo al decollo del velivolo
        private int _offLoadWeight = 0;
        // indica il peso del velivolo scarico di carburante, truppe o altro carico
        private float _fuelConsumption = 0;
        // indica la quantita' oraria di carburante consumato
        public bool _isRunning = false;
        // indica se i motori sono accesi
        public bool _bladeSpread = false;
        // l'elicottero è a pale aperte o chiuse
        public bool _isEfficient = false;
        // indica lo stato di efficienza del velivolo 1 efficiente 0 non efficiente
        public bool _posEli = false;
        // 0 se eli in hangar - 1 se eli sul ponte
        public bool _isFlying = false;
        // 0 se eli non in volo - 1 se eli in volo
        //-----------------------------------------------
        public InfoWindow win = null;
        // finestra informazioni
```

```
// viene definita la lista per la generazione degli elicotteri
// da aggiungere dinamicamente all'elenco
// COSTRUTTORE alla classe viene passato il numero di record iniziali
protected EliOptions (int totEli, InfoWindow _win) : base (Gtk.WindowType.Toplevel)
{
    this.Build (); // disegna fisicamente la finestra
    this._totEliC = totEli;// passo la variabile con il numero di elicotteri
    this.win = _win; // finestra informazioni
    this.EliM = (EliManager)FactoryManager.Make (0, totEli, win); // il manager crea l'elimanager
    EliM.MakeList (0); // viene definita l'istanza della lista di elementi
    this.InitValue (); // azzera valori dopo la creazione

}
// istance per la classe singleton - viene creata una sola istanza della cWinlasse
public static EliOptions Instance (int TotEli, InfoWindow winI)
{
    if (instance == null) { // se l'instanza non esiste la crea
        instance = new EliOptions (TotEli, winI);
        return instance;
    } else {  // riprende dal rec 0

        return instance; // ritorno il costruttore
    }
}
//---- METODI DI SUPPORTO-----------------------------------------------------------------------------
// INIZIALIZZA i campi prima interni dell'inserimento di un nuovo record
private void InitValue ()
{
    _IdEli = "NUOVO RECORD DA ASSEGNARE";
    this.label4.Text = this._IdEli;
    this.entry1.Text = _IdEli;
    _catEli = 0;
    this.label5.Text = "INSERIRE VALORE";
    this.hscale1.Value = 0;
    _fuel = 0;
    this.label6.Text = "INSERIRE VALORE";
    this.hscale2.Value = 0;
    _WTroopsLeftOnBoard = 0;
    this.label8.Text = "INSERIRE VALORE";
    this.hscale4.Value = 0;
    _WCargoLeftOnBoard = 0;
    this.label3.Text = "INSERIRE VALORE";
    this.hscale5.Value = 0;
    _maxTOLoad = 0;
    this.label11.Text = "INSERIRE VALORE";
    this.hscale6.Value = 0;
    _offLoadWeight = 0;
    this.label9.Text = "INSERIRE VALORE";
    this.hscale7.Value = 0;
    _fuelConsumption = 0;
    this.label10.Text = "INSERIRE VALORE";
    this.hscale3.Value = 0;
    _isFlying = false;
    this.button17.Label = "EFFICIENTE";
    _isEfficient = true;
    this.button16.Label = "SPENTO";
    _isRunning = false;
    this.button10.Label = "IN HANGAR";
    _posEli = false;
    this.button15.Label = "PALE CHIUSE";
    _bladeSpread = false;
    _isFlying = false;
    this.label1.Text = "Record # " + (Eli_Record + 1) + " Record inseriti # "
                     + EliM.ElicotteriList.Count + " su: " + this._totEliC;
}
// MOSTRA il record I sull'interfaccia grafica
public void ShowRecord (int i)
{
    this.entry1.Text = EliM.ElicotteriList [i].IdEli; // id eli
    this._IdEli=EliM.ElicotteriList [i].IdEli; // id eli
    this.label4.Text = "VALORE INSERITO: " + this.entry1.Text;
    this.hscale1.Value = EliM.ElicotteriList [i].CatEli; // cat eli
    this._catEli=EliM.ElicotteriList [i].CatEli; // cat eli
    switch ((int)this.hscale1.Value) {
    case 1:
        this.label5.Text = "ELICOTTERO TIPO LEGGERO";
        break;
    case 2:
        this.label5.Text = "ELICOTTERO TIPO PESANTE";
        break;
    case 3:
        this.label5.Text = "ALTRO MODELLO";
        break;
    }
    this.hscale2.Value = EliM.ElicotteriList [i].Fuel; // carburante
    this._fuel= EliM.ElicotteriList [i].Fuel;
    this.label6.Text = "ELICOTTERO RIFORNITO: " + this.hscale2.Value + "KG DI CARBURANTE";
    this.hscale4.Value = EliM.ElicotteriList [i].WTroopsLeftOnBoard;// truppe
    this._WTroopsLeftOnBoard=EliM.ElicotteriList [i].WTroopsLeftOnBoard;// truppe
    this.label8.Text = "PESO TRUPPE IMBARCABILI: " + this.hscale4.Value + " KG";
    this.hscale5.Value = EliM.ElicotteriList [i].WCargoLeftOnBoard; // peso cargo
    this._WCargoLeftOnBoard=EliM.ElicotteriList [i].WCargoLeftOnBoard; // peso cargo
    this.label3.Text = "PESO DEL CARGO IMBARCABILE: " + this.hscale5.Value + " KG";
    this.hscale6.Value = EliM.ElicotteriList [i].MaxTOLoad;// MAX TO
    this._maxTOLoad= EliM.ElicotteriList [i].MaxTOLoad;// MAX TO
    this.label11.Text = " PESO MASSIMO AL DECOLLO: " + this.hscale6.Value + " KG";
    this.hscale7.Value = EliM.ElicotteriList [i].OffLoadWeight;// off load W
    this._offLoadWeight= EliM.ElicotteriList [i].OffLoadWeight;// off load W
    this.label9.Text = " PESO BASICO ELICOTTERO SCARICO: " + this.hscale7.Value;
    this.hscale3.Value = EliM.ElicotteriList [i].FuelC;// consumi di carburante
```

```
    this._fuelConsumption=EliM.ElicotteriList [i].FuelC;// consumi di carburante
    this.label10.Text = " CONSUMO ELICOTTERO: " + this.hscale3.Value;
    if (EliM.ElicotteriList [i].PosEli) {
        this.button10.Label = "SUL PONTE";
        this._posEli = true;
    } else {
        this.button10.Label = "IN HANGAR";
        this._posEli = false;
    }

    if (EliM.ElicotteriList [i].IsBladeSpread) {
        this.button15.Label = "PALE APERTE";
        this._bladeSpread = true;
    } else {
        this.button15.Label = "PALE CHIUSE";
        this._bladeSpread = false;
    }

    if (EliM.ElicotteriList [i].IsRunning) {
        this.button16.Label = "ACCESO";
        this._isRunning = true;
    } else {
        this.button16.Label = "SPENTO";
        this._isRunning = false;
    }

    if (!EliM.ElicotteriList [i].IsEfficient) {
        this.button17.Label = "INEFFICIENTE";
    this._isEfficient = false;
    } else {
        this.button17.Label = "EFFICIENTE";
        this._isEfficient = true;
    }

    this.label1.Text = "Record # " + (Eli_Record + 1) + " Record inseriti # " + EliM.ElicotteriList.Count + " su: " + this._totEliC;
}
// effettua l'update del record corrente sostituiendo i nuovi valori rilevati a quelli vecchi in memoria
private void UpdateRec (int i)
{
    this._IdEli = this.entry1.Text;
    EliM.ElicotteriList [i].IdEli = this.entry1.Text; // id eli
    this._catEli = (int)this.hscale1.Value;
    EliM.ElicotteriList [i].CatEli = (int)this.hscale1.Value; // cat eli
    this._fuel = (int)this.hscale2.Value;
    EliM.ElicotteriList [i].Fuel = (int)this.hscale2.Value; // carburante
    this._WTroopsLeftOnBoard=(int)this.hscale4.Value;
    EliM.ElicotteriList [i].WTroopsLeftOnBoard = (int)this.hscale4.Value;// truppe
    this._WCargoLeftOnBoard=(int)this.hscale5.Value;
    EliM.ElicotteriList [i].WCargoLeftOnBoard = (int)this.hscale5.Value; // peso cargo
    this._maxTOLoad=(int)this.hscale6.Value;
    EliM.ElicotteriList [i].MaxTOLoad = (int)this.hscale6.Value;// MAX TO
    this._offLoadWeight = (int)this.hscale7.Value;
    EliM.ElicotteriList [i].OffLoadWeight = (int)this.hscale7.Value;// off load W
    this._fuelConsumption=(int)this.hscale3.Value;
    EliM.ElicotteriList [i].FuelC = (int)this.hscale3.Value;// consumi di carburante

    if (this.button10.Label == "IN HANGAR") {
        this._posEli = false;
        EliM.ElicotteriList [i].PosEli = false;
    } else {
        this._posEli = true;
        EliM.ElicotteriList [i].PosEli = true; // posizione eli hangar o ponte
    }

    if (this.button15.Label == "PALE APERTE")
     {
        this._bladeSpread = true;
        EliM.ElicotteriList [i].IsBladeSpread = true;
    } else
    {
        this._bladeSpread = false;
        EliM.ElicotteriList [i].IsBladeSpread = false;// pale aperte o chiuse
    }

    if (this.button16.Label == "ACCESO") {
        this._isRunning = true;
        EliM.ElicotteriList [i].IsRunning = true;
    } else
    {
        this._isRunning = false;
        EliM.ElicotteriList [i].IsRunning = false;// accesso spento
    }

    if (this.button17.Label == "EFFICIENTE")
    {
        this._isEfficient = true;
        EliM.ElicotteriList [i].IsEfficient = true;
    }
    else
    {
        this._isEfficient = false;
        EliM.ElicotteriList [i].IsEfficient = false; // is efficient - inefficiente
    }

    win.InsertSomeText (" OPZIONI ELICOTTERO: record # " + (i + 1) + " AGGIORNATO..");
} // termine update record


// ShowWin la finestra corrente e nasconde la MAIN WINDOW in modo da evitare inserimenti errati di dati
public void ShowWin ()
```

```
        {
            MainWin.Sensitive = false;
            this.Visible = true;
        }
        // effettuo il controllo dei dati inseriti per ogni elicottero
        // i pesi devono essere congruenti con il sistema
        private bool checkDataConsistency ()
        {
            // peso totale
            float allUpW = _fuel + _WTroopsLeftOnBoard + _WCargoLeftOnBoard + _offLoadWeight;

            if ((allUpW > _maxTOLoad) || (_fuel == 0) || ((_WTroopsLeftOnBoard + _WCargoLeftOnBoard) == 0)) { // se viene superato il peso massim
o al decollo le pesate devono essere corrette

                // crea il messaggio
                MessageBox.Show ("Sono stati inseriti dei dati incoerenti", "WARNING MESSAGE", MessageBoxButtons.OK);

                return false; // ritorna la flag dei dati errati

            }
            return true; // ritorno la flag dei dati corretti

        }
        //-----------------------------------------------------------------------------------------------------------
        // CONTROLS AREA-----------------------------------------------------------------------------
        // VENGONO GESTITI I CONTROLLI DELL'UI
        // bottone chiusura applicazione
        protected void OnDeleteEvent (object sender, DeleteEventArgs e)
        {
            if ((Eli_Record >= EliM.ElicotteriList.Count) && (EliM.ElicotteriList.Count > 0))// effettuo il check di out bound
                Eli_Record--;

            if (EliM.ElicotteriList.Count == 0)
                Eli_Record = 0;

            this.Hide (); // nascondo la finestra

            MainWin.Sensitive = true; // resetto la sensibilità della finestra principale
            MainWin.CheckGreenFlags ();// check green flags
            e.RetVal = true;
        }
        // BOTTONE ACCETTA DATI E TORNA AL MENU PRINCIPALE
        protected void OnButton9Released (object sender, EventArgs e)
        {

            if ((Eli_Record >= EliM.ElicotteriList.Count) && (EliM.ElicotteriList.Count > 0))// effettuo il check di out bound
                Eli_Record--;

            if (EliM.ElicotteriList.Count == 0)
                Eli_Record = 0;


            this.Hide (); // nascondo la finestra

            MainWin.Sensitive = true; // resetto la sensibilità della finestra principale
            MainWin.CheckGreenFlags ();// check green flags
        }
        // ACCETTAZIONE dell'identificativo dell'elicottero
        protected void OnButton1Released (object sender, EventArgs e)
        {
            _IdEli = this.entry1.Text;
            this.label4.Text = "VALORE INSERITO: " + this._IdEli;

        }
        // accettazione categoria elicottero
        protected void OnButton2Released (object sender, EventArgs e)
        {

            _catEli = (int)this.hscale1.Value;

            switch (_catEli) {
            case 1:
                this.label5.Text = "ELICOTTERO TIPO LEGGERO";
                break;
            case 2:
                this.label5.Text = "ELICOTTERO TIPO PESANTE";
                break;
            case 3:
                this.label5.Text = "ALTRO MODELLO";
                break;
            }
        }
        // accettazione peso carburante
        protected void OnButton3Released (object sender, EventArgs e)
        {
            _fuel = (int)this.hscale2.Value;
            this.label6.Text = "ELICOTTERO RIFORNITO: " + _fuel + "KG DI CARBURANTE";
        }
        // accettazione PESO TRUPPE
        protected void OnButton5Released (object sender, EventArgs e)
        {

            _WTroopsLeftOnBoard = (int)this.hscale4.Value;
            this.label8.Text = "PESO TRUPPE IMBARCABILI: " + _WTroopsLeftOnBoard + " KG";

        }
        // accettazione del PESO DEL CARGO
        protected void OnButton13Released (object sender, EventArgs e)
        {
            _WCargoLeftOnBoard = (int)this.hscale5.Value;
```

```
    this.label3.Text = "PESO DEL CARGO IMBARCABILE: " + _WCargoLeftOnBoard + " KG";

}
// accettazione del Valore di MAX TAKE OFF
protected void OnButton14Released (object sender, EventArgs e)
{
    _maxTOLoad = (int)this.hscale6.Value;
    this.label11.Text = " PESO MASSIMO AL DECOLLO: " + _maxTOLoad + " KG";

}
// accettazione del Valore di PESO dell'elicottero scarico
protected void OnButton6Released (object sender, EventArgs e)
{
    _offLoadWeight = (int)this.hscale7.Value;
    this.label9.Text = " PESO BASICO ELICOTTERO SCARICO: " + _offLoadWeight;
}

protected void OnButton4Released (object sender, EventArgs e)
{
    _fuelConsumption = (int)this.hscale3.Value;
    this.label10.Text = " CONSUMO ELICOTTERO: " + _fuelConsumption;
}
//BUTTON: PONTE HANGAR --- vengono applicate le restrizioni al caso
protected void OnButton10Released (object sender, EventArgs e)
{
    if (this._posEli == false) {
        _posEli = true;// switch nella posizione sul ponte
        this.button10.Label = "SUL PONTE";

        // constrain sul ponte deve essere efficiente
        this._isEfficient = true;
        this.button17.Label = "EFFICIENTE";


    } else {
        this._posEli = false; // switch nella posizione in hangar
        this.button10.Label = "IN HANGAR";

        // constrain in hangar - necessariamente a pale chiuse
        _bladeSpread = false; // switch nella posizione in hangar
        this.button15.Label = "PALE CHIUSE";

        // constrin in hangar - necessariamente spento
        _isRunning = false; // switch nella posizione in hangar
        this.button16.Label = "SPENTO";

    }
}
// BUTTON: pale CHIUSE O pale APERTE
protected void OnButton15Released (object sender, EventArgs e)
{
    if (this._bladeSpread == false) {
        this._bladeSpread = true;
        this.button15.Label = "PALE APERTE";

        // pale aperte necessita di elicottero sul ponte
        this._posEli = true;
        this.button10.Label = "SUL PONTE";

        // con le pale aperte deve essere efficiente
        _isEfficient = true;
        this.button17.Label = "EFFICIENTE";

        // a pale aperte deve essere acceso
        _isRunning = true;
        this.button16.Label = "ACCESO";

    } else {
        _bladeSpread = false; // switch nella posizione in hangar
        this.button15.Label = ("PALE CHIUSE");
    }
}
// BUTTON: indica se l'elicottero ha i motori accesi oppure spenti
protected void OnButton16Released (object sender, EventArgs e)
{
    if (!_isRunning) {
        _isRunning = true;
        this.button16.Label = "ACCESO";

        // constrain
        //se è acceso deve stare per forza sul ponte
        _posEli = true;
        this.button10.Label = "SUL PONTE";

        // se è acceso deve essere efficiente
        _isEfficient = true;
        this.button17.Label = "EFFICIENTE";

    } else {
        _isRunning = false; // switch nella posizione in hangar
        this.button16.Label = "SPENTO";

        // constrain elicottero spento -- pale chiuse
        _bladeSpread = false;
        this.button15.Label = "PALE CHIUSE";

    }
}
// indicatore di elicottero efficiente o inefficiente
protected void OnButton17Released (object sender, EventArgs e)
```

```
        {
            if (_isEfficient) {
                _isEfficient = false;
                this.button17.Label = "INEFFICIENTE";

                // constrain se inefficiente deve essere a pale chiuse
                _bladeSpread = false; // switch nella posizione in hangar
                this.button15.Label = "PALE CHIUSE";

                //constrain se inefficiente deve essere in hangar
                _posEli = false;
                this.button10.Label = "IN HANGAR";

                // contrain se inefficiente deve essere spento
                _isRunning = false;
                this.button16.Label = "SPENTO";
            } else {
                _isEfficient = true;
                this.button17.Label = "EFFICIENTE";
            }
        }
        //BOTTONE: SALVA RECORD E RESETTA
        protected void OnButton11Released (object sender, EventArgs e)
        {
            if (EliM.ElicotteriList.Count <= this._totEliC) { // se il numero di elicotteri è eccessivo


                if (checkDataConsistency ()) { // CHECK CONSISTENZA DATI// controllo prima la consistenza dei dati inseriti

                    //-------------------------------------------------------------------------------------------------------
                    if (EliM.ElicotteriList.Count <= this._totEliC // se il numero di record inseriti è minore del totale inseribile
                        && Eli_Record < _totEliC  // se il record che dobbiamo inserire è minore del totale inseribile
                        && Eli_Record >= EliM.ElicotteriList.Count) // se il record che dobbiamo inserire è maggiore del totale inserito
                    { // SE IL RECORD NON ESISTE: INSERIMENTO NUOVO RECORD-----------------------------------------------------
                        try
                        { // provo a inserire il dato cargo
                            EliM.InsertElementEli (EliM.MainTime, Eli_Record, _IdEli, _catEli, _fuel,
                                _WTroopsLeftOnBoard, _WCargoLeftOnBoard, _maxTOLoad, _offLoadWeight,
                                _isRunning, _bladeSpread, _isEfficient, _posEli, _isFlying, _fuelConsumption);
                        }
                        catch
                        {
                            System.Console.WriteLine ("ERRORE NELL'INSERIMENTO DEL RECORD ELICOTTERO"); // ERRORE INSERIMENTO
                        }
                        // visualizzo e salvo le informazioni sul record-----------------------------------------
                        win.InsertSomeText ("OPZIONI ELICOTTERO: inserto nuovo record#__" + (this.Eli_Record + 1) + " Elementi totali presenti: "
+ EliM.ElicotteriList.Count);
                        win.InsertSomeText (EliM.ElicotteriList [Eli_Record].EliInfo ()); // visualizzo le info nella finestra informazioni

                        // controllo se ultimo record e chiudo oppure se ci sono altri record allora inizializzo
                        if ((Eli_Record == EliM.ElicotteriList.Count - 1) && (EliM.ElicotteriList.Count == _totEliC)) {
                            this.Hide (); // nascondo la finestra
                            MainWin.Sensitive = true; // resetto la sensibilità della finestra principale
                            MainWin.CheckGreenFlags ();// check green flags
                        } else
                            // se non è l'ultimo inizializzo i valori per il prossimo inserimento
                        {
                            this.Eli_Record++;// incrementa il record
                            this.InitValue ();// reinizializza valori a 0 per il nuovo inserimento
                        }

                    }// fine inserimento nuovo record

                    else // controllo se si tratta di un UPDATE
                        if (EliM.ElicotteriList.Count <= _totEliC // se il numero di valori inseriti è minore al totale inseribile
                            && Eli_Record < _totEliC // se il numero di record da trattare è inferiore al totale
                            && Eli_Record < EliM.ElicotteriList.Count // se il numero di record da trattare è inferiore al totale inserito
                        ) {
                            // SE IL RECORD ESISTE E' UN UPDATE
                            //effettuo l'update del record
                            this.UpdateRec (Eli_Record);
                        }

                } // FINE CHECK CONSISTENZA DATI

            }// ed if NUMERO ECCESSIVO DI DATI
        }
        // end fine metodo
        // BOTTONE: elimina record
        protected void OnButton12Released (object sender, EventArgs e)
        {
            if (EliM.ElicotteriList.Count > 0) {


                bool flag = false;
                if (this.Eli_Record < EliM.ElicotteriList.Count) {
                    EliM.RemoveElement (Eli_Record); // elimina il record
                    // se dopo l'eliminazione il record è 0
                    if ((EliM.ElicotteriList.Count) == 0) { // se dopo la rimozione non ci sono piu' record
                        Eli_Record = 0; // setta il rec a 0
                        this.InitValue ();// inizializza valori
                        flag = true; // flag di elemento eliminato
                    } else {
                        if ((EliM.ElicotteriList.Count == Eli_Record)) { // se il mio record è l'ultimo della lista
                            Eli_Record = EliM.ElicotteriList.Count - 1;
                        }

                        this.ShowRecord (Eli_Record); // mostra il record
                        flag = true;
                    }
```

```
                if (flag)
                    win.InsertSomeText ("OPZIONI ELICOTTERO: rimosso record # " + (Eli_Record + 1));
                else
                    this.InitValue ();
            }
        }
    }
    // termine elimina record
    //---------------------------------------------------------------------------
    // BOTTONE: visualizza record precedente
    protected void OnButton8Released (object sender, EventArgs e)
    {

        if (EliM.ElicotteriList.Count > 0) { // controlla se esistono dati da leggere
            if (this.Eli_Record > 0) {
                this.Eli_Record--; // decrementa il numero di record per la visualizzazione
                this.ShowRecord (Eli_Record); // ShowWin il record
            }
            if (this.Eli_Record == 0) {
                win.InsertSomeText ("OPZIONI ELICOTTERO: raggiunto record iniziale.");
                this.ShowRecord (Eli_Record);
            }
        }
    }
    // fine metodo record precedente

    //BOTTONE: record successivo
    protected void OnButton7Released (object sender, EventArgs e)
    {
        // effettuo i controlli
        if ((EliM.ElicotteriList.Count > 0)// record presenti
            && (Eli_Record < EliM.ElicotteriList.Count)// il numero di record minore del massimo recordinseriti
            && (Eli_Record < (_totEliC))) { //il numero di record minore del massimo di rec inseribili

            if (this.Eli_Record == (EliM.ElicotteriList.Count - 1) && (EliM.ElicotteriList.Count == _totEliC)) {  // raggiunto ultimo record

                win.InsertSomeText ("OPZIONI ELICOTTERO: raggiunto record finale");

            } else if (Eli_Record==EliM.ElicotteriList.Count-1
                       && EliM.ElicotteriList.Count < _totEliC)
            {// controlla se esistono dati da leggere
                this.Eli_Record++;
                this.InitValue ();
            } else {
                this.Eli_Record++; // incrementa il record
                try {
                this.ShowRecord (Eli_Record); // mostra il record
                } catch{
                    System.Console.WriteLine ("ERRORE DI VISUALIZZAZIONE RECORD");
                }
            }
        }
    } // termine bottone record successivo
} // fine classe
} // fine namespace
```

6.4 FILE CARGOOPTIONS

```
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// CARGO OPTIONS
// la classe si occupa dell'inserimento dei dati cargo
// all'interno della lista definita nel cargo manager
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


using System;
using Gtk;
using System.Windows.Forms;

namespace EliAssaltoAnfibio
{
    public partial class CargoOptions : Gtk.Window
    {
        private static  CargoOptions instance= null;
        private InfoWindow WinI;//link alla finestra informazioni
        private int cargoTot=0; // cargo totale
        private string _cargoSTR=null; // nome del cargo
        private int _cargoW=0; // peso totale del cargo
        private int _cargoP=0; // personale necessario al trasporto del cargo
        public int cargoRec=0; // numero di record del CARGO visualizzato o inserito
        public CargoManager CargoM;// definizione per la creazione del cargomanager
        static private MainWindow MainWin = MainWindow.Instance();// identifica la finestra principale in modo che possa essere ritrovata alla ch
iusura di questa


        protected CargoOptions (int _cargo,  InfoWindow _win) :  base(Gtk.WindowType.Toplevel)
        {

            this.WinI=_win; // gli passo la finestra informazioni
            this.cargoTot=_cargo;
            WinI.InsertSomeText("OPZIONI CARGO: --> EFFETTUATA APERTURA FINESTRA OPZIONI CARGO <--");
            CargoM = (CargoManager) FactoryManager.Make(3,this.cargoTot, this.WinI); // viene creato lo Spot Manager tramite factory method
            CargoM.MakeList(cargoTot); // crea la  lista iniziale di un numero di spot impostato
            this.Build ();
```

```
            this.label1.Text= "Numero di Record cargo #  "+ (this.cargoRec+1) +" Record inseriti: "+ CargoM.CargoList.Count + " Record disponibil
i: "+this.cargoTot;
            this.InitValue();
        }

        // istanza singleton è prevista l'esistenza di un'unica finestra
        public static CargoOptions Instance(int TotCargo , InfoWindow winI)
        {
            if (instance == null) instance= new CargoOptions( TotCargo ,  winI );
            return instance; // ritorno il costruttore
        }

        // ShowWin la finestra corrente e nasconde la principlae
        public void ShowWin()
        {
            MainWin.Sensitive=false;
            this.Visible=true;;
            this.label1.Text= "Numero di Record cargo #  "+ (this.cargoRec+1) +" Record inseriti: "+ CargoM.CargoList.Count + " Record disponibil
i: "+this.cargoTot;

        }

        // ShowWin la finestra corrente e nasconde la principale DELETE EVENT ( richiamato dalla chiusura con la X )
        protected void OnDeleteEvent (object sender, DeleteEventArgs e)
        {

            if ((cargoRec >= CargoM.CargoList.Count) && ( CargoM.CargoList.Count>0)) // effettuo il check di out bound
                cargoRec--;
            if (CargoM.CargoList.Count == 0)
                cargoRec = 0;
            this.Hide (); // nascondo la finestra
            MainWin.Sensitive=true; // resetto la sensibilità della finestra principale
            MainWin.CheckGreenFlags ();// check green flags
            e.RetVal=true;
        }

        // Metodo di inizializzazione valori cargo a 0
        private void InitValue ()
        {
            this.label1.Text= "Numero di Record cargo #  "+ (this.cargoRec+1) +" Record inseriti: "+ CargoM.CargoList.Count + " Record disponibil
i: "+this.cargoTot;
            this.entry1.Text="";
            this.hscale1.Value=0;
            this.hscale2.Value=0;
            this._cargoSTR=" DA INSERIRE ";
            this._cargoW=0;
            this._cargoP=0;
        }

        // update record
        private void UpdateRec (int i)
        {
            this._cargoSTR= this.entry1.Text;
            CargoM.CargoList[i].CargoString = this._cargoSTR ;

            this._cargoW = (int)this.hscale1.Value;
            CargoM.CargoList[i].CargoW =this._cargoW ;

            this._cargoP = (int)this.hscale2.Value;
            CargoM.CargoList[i].CargoP =this._cargoP ;

            WinI.InsertSomeText(" OPZIONI CARGO: record # "+(i+1)+ " AGGIORNATO..");
        }

        // BOTTONE: accetta dati - torna al menu principale
        protected void OnButton2Released (object sender, EventArgs e)
        {
            if ((cargoRec >= CargoM.CargoList.Count) && (CargoM.CargoList.Count > 0)) // effettuo il check di out bound
                cargoRec--;
            if (CargoM.CargoList.Count == 0)
                cargoRec = 0;

            this.Hide (); // nascondo la finestra CARGO

            MainWin.Sensitive=true; // resetto la sensibilità della finestra principale
            MainWin.CheckGreenFlags ();// check green flags della finestra principale

        }

        // BOTTONE: salva record e resetta
        protected void OnButton3Released (object sender, EventArgs e)
        {

            this._cargoSTR = this.entry1.Text; // valore stringa cargo NOME cargo
            this._cargoW = (int)this.hscale1.Value; // peso cargo
            this._cargoP = (int)this.hscale2.Value; // personale per cargo
            if (CargoM.CargoList.Count <= this.cargoTot) { // se il numero di record cargo è dentro il range
                                                           // effettua inserimento o update

                if (checkDataConsistency ()) { // CHECK CONSISTENZA DATI// controllo prima la consistenza dei dati inseriti

                    //-------------------------------------------------------------
                    if (CargoM.CargoList.Count <= cargoTot // se il numero di record inseriti è minore del totale inseribile
                        && cargoRec < cargoTot  // se il record che dobbiamo inserire è minore del totale inseribile
                        && cargoRec >= CargoM.CargoList.Count) // se il record che dobbiamo inserire è maggiore del totale inserito
                    { // SE IL RECORD NON ESISTE: INSERIMENTO NUOVO RECORD
                        try
                        { // provo a inserire il dato cargo
                        CargoM.InsertElementCargo (cargoRec, this._cargoSTR, this._cargoW, this._cargoP); // inserisci il nuovo elemento all'inte
rno della lista cargo
```

```
                    }
                    catch
                    {
                        System.Console.WriteLine ("ERRORE NELL'INSERIMENTO DEL RECORD CARGO"); // ERRORE INSERIMENTO
                    }
                    // visualizzo e salvo le informazioni sul record----------------------------------------
                    WinI.InsertSomeText ("OPZIONI CARGO: inserto nuovo record#__" + (this.cargoRec + 1) + " Elementi totali presenti: " + Car
goM.CargoList.Count);
                    WinI.InsertSomeText (CargoM.CargoList [cargoRec].CargoInfo ()); // visualizzo le info nella finestra informazioni

                    // controllo se ultimo record e chiudo oppure se ci sono altri record allora inizializzo
                    if ((cargoRec == CargoM.CargoList.Count - 1) && (CargoM.CargoList.Count == cargoTot)) {
                        this.Hide (); // nascondo la finestra
                        MainWin.Sensitive = true; // resetto la sensibilità della finestra principale
                        MainWin.CheckGreenFlags ();// check green flags
                    } else
                        // se non è l'ultimo inizializzo i valori per il prossimo inserimento
                    {
                        this.cargoRec++;// incrementa il record
                        this.InitValue ();// reinizializza valori a 0 per il nuovo inserimento
                    }

                }// fine inserimento nuovo record

                else // controllo se si tratta di un UPDATE
                    if (CargoM.CargoList.Count <= cargoTot // se il numero di valori inseriti è minore al totale inseribile
                        && cargoRec < cargoTot // se il numero di record da trattare è inferiore al totale
                        && cargoRec < CargoM.CargoList.Count // se il numero di record da trattare è inferiore al totale inserito
                    )
                    {
                        // SE IL RECORD ESISTE E' UN UPDATE
                        //effettuo l'update del record
                        this.UpdateRec (cargoRec);
                    }
                } // FINE CHECK CONSISTENZA DATI
            }// ed if NUMERO ECCESSIVO DI DATI
        }  // end fine metodo


    // BOTTONE: elimina record corrente
    protected void OnButton4Released (object sender, EventArgs e)
    {
        if (CargoM.CargoList.Count > 0) { // esegui se il conteggio dati inseriti è positivo
            bool flag = false;
            if (this.cargoRec < CargoM.CargoList.Count) { // se il record attuale è minore del massimo di record inseirti
                try{
                CargoM.RemoveElement (cargoRec); // elimina il record
                }
                catch{
                    System.Console.WriteLine ("ERRORE DI RIMOZIONE RECORD CARGO");
                }
                // se dopo l'eliminazione il record è 0
                if ((CargoM.CargoList.Count) == 0) { // se dopo la rimozione non ci sono piu' record
                    cargoRec = 0; // setta il rec a 0
                    this.InitValue ();// inizializza valori
                    flag = true; // flag di elemento eliminato
                } else {
                    if ((CargoM.CargoList.Count == cargoRec)) { // se il mio record è l'ultimo della lista
                        cargoRec = CargoM.CargoList.Count - 1;// setto il mio record ultimo
                    }
                    try{
                    this.ShowRecord (cargoRec);
                    } catch {
                        System.Console.WriteLine ("ERRORE DI VISUALIZZAZIONE RECORD");
                    }
                    flag = true; // imposto la flag di rimozione se è riuscita
                }
                if (flag)
                    WinI.InsertSomeText ("OPZIONI ELICOTTERO: rimosso record # " + (cargoRec + 1));
                else // altrimenti
                    this.InitValue ();  //inizializzo i valori a 0
            }
        }
    }    // termine elimina record


    // BOTTONE: record precedente
    protected void OnButton1Released (object sender, EventArgs e)
    {
        if (CargoM.CargoList.Count > 0) // controlla se esistono dati da leggere
        {
            if (this.cargoRec > 0)
            {
                this.cargoRec--; // decrementa il numero di record per la visualizzazione
                this.ShowRecord(cargoRec); // ShowWin il record
            }
            if (this.cargoRec==0)
            {
                WinI.InsertSomeText ("OPZIONI CARGO: raggiunto record iniziale.");
                try{
                    this.ShowRecord(cargoRec); // visualizzo il record precedente
                } catch {
                    System.Console.WriteLine ("ERRORE DI VISUALIZZAZIONE RECORD");
                }
            }
        }
    } // fine metodo record precedente


    // BOTTONE: record successivo
```

```
        protected void OnButton6Released (object sender, EventArgs e)
        {
            // effettuo i controlli
            if ((CargoM.CargoList.Count > 0)// record presenti
                && (cargoRec < CargoM.CargoList.Count)// il numero di record minore del massimo recordinseriti
                && (cargoRec < (cargoTot))) //il numero di record minore del massimo di rec inseribili)
                 {// invece abbiamo ancora record
                if (this.cargoRec == (CargoM.CargoList.Count - 1) && (CargoM.CargoList.Count == cargoTot)) {  // raggiunto ultimo record

                    WinI.InsertSomeText ("OPZIONI ELICOTTERO: raggiunto record finale");

                } else if (cargoRec==CargoM.CargoList.Count-1
                           && CargoM.CargoList.Count < cargoTot)
                 {// controlla se esistono dati da leggere
                    this.cargoRec++;
                    this.InitValue ();
                } else {
                    this.cargoRec++; // incrementa il record
                    try{
                        this.ShowRecord(cargoRec); // visualizzo il record precedente
                    } catch {
                        System.Console.WriteLine ("ERRORE DI VISUALIZZAZIONE RECORD");
                    } } }

                }    // termine bottone record successivo


        // mostra il record in input sull'interfaccia grafica
        public void ShowRecord(int i)
        {
            this.entry1.Text=CargoM.CargoList[i].CargoString;
            this.hscale1.Value=CargoM.CargoList[i].CargoW;
            this.hscale2.Value=CargoM.CargoList[i].CargoP;
            this.label1.Text= "Numero di Record cargo #  "+ (this.cargoRec+1) +" Record inseriti: "+ CargoM.CargoList.Count + " Record disponibil
i: "+this.cargoTot;
        }



        // controllo consistenza dati
        // non è possibile inserire cargo con peso 0
            private bool checkDataConsistency()
            {

            if (_cargoW==0) // se viene superato il peso massimo al decollo le pesate devono essere corrette
            {
                // crea il messaggio
                MessageBox.Show ("Sono stati inseriti dei dati incoerenti", "WARNING MESSAGE",MessageBoxButtons.OK);
                return false; // ritorna la flag dei dati errati
            }
            return true; // ritorno la flag dei dati corretti
            }

    } // fine classe
} // fine name space
```

6.5 FILE TRUPPEOPTIONS

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// CLASSE DI CONTROLLO PER L'INSERIMENTO DATI TRUPPE
// SI OCCUPA DELL'INSERIMENTO E DEL CONTROLLO DELLE INFORMAZIONI
// SULLE TRUPPE DA IMPIEGARE
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

using System;
using Gtk;
using System.Windows.Forms;

namespace EliAssaltoAnfibio
{
    public partial class TruppeOptions : Gtk.Window // CLASSE parziale eredita da gtk.windows
    {

        private static TruppeOptions instance = null; // singleton
        static public MainWindow MainWin = MainWindow.Instance();// identifica la finestra principale in
                                                                 // modo che possa essere ritrovata alla chiusura di questa
        public TroopsManager TroopM;  // crea il manager delle truppe
        private int _troop=0;    //numero truppe
        public InfoWindow winI; // finestra informazioni
        // al costruttore viene passato il numero di truppe, la lista è ancora vuota
        // e viene allocato lo spazio in memoria necessario
        protected TruppeOptions (int Troop, InfoWindow _winI) : base(Gtk.WindowType.Toplevel)
        {
            this.winI=_winI;
            this._troop = Troop;
            this.Build ();
            // viene creato il manager delle truppe
            // con il factory method
            // l'opzione 1 indica che si tratta di un troops manager
            TroopM = (TroopsManager) FactoryManager.Make(1,Troop, winI); // creazione manager
            this.winI.InsertSomeText("TROOPS MANAGER: effettuata creazione manager per truppe");
            // viene definita l'istanza del numero di elementi richiesti
            TroopM.MakeList (Troop); // creazione della lista truppe

        }

        public static TruppeOptions Instance(int TotEli, InfoWindow WinI) // singleton
        {
```

```
            if (instance == null)
            instance= new TruppeOptions(TotEli, WinI);
            return instance; // ritorno il costruttore
        }

        // metodo per mostrare la finestra truppe
        public void ShowWin()
        {
            this.Visible=true; // finestra visibile
        }

        // metodo di chiusura finestra truppe tramite X
         protected void OnDeleteEvent (object sender, DeleteEventArgs e)
        {
            this.Visible=false; // finestra invisibile
            MainWin.Sensitive=true; // la main window puo' essere clikkata
            MainWin.CheckGreenFlags ();// check green flags
            e.RetVal=true; // ret value true evita il destroy  delle informazioni
        }

        // BOTTONE: viene accettato il peso delle truppe
        // viene creata la lista inserendo in essa le truppe richieste.
        // La lista verra poi gestita in modo da assegnare truppe ed elicotteri
        protected void OnButton1Released (object sender, EventArgs e)
        {
            if (TroopM.TroopList.Count == 0) { // se la lista è vuota  creala
                //inserimento soldati nella lista
                int i;
                for (i=0; i<_troop; i++)
                {
                    Soldier _soldier = new Soldier ((int)this.hscale1.Value);
                    try{
                        TroopM.InsertElement (_soldier);    // inserisco il dato truppa
                    }
                    catch {
                        System.Console.WriteLine ("ERRORE DI INSERIMENTO RECORD");
                    }
                }
                TroopM.SingleSoldierW=(int)this.hscale1.Value; // salvo il peso del soldato singolo
                winI.InsertSomeText("OPZIONI TRUPPE: effettuato l'inserimento di "+TroopM.TroopList.Count+ " soldati del peso singolo di: "+ Troo
pM.SingleSoldierW+"kg");
        }

            else
            {
                winI.InsertSomeText("OPZIONI TRUPPE: lista truppe già creata, sostituisco con i nuovi dati");

                foreach (Soldier sold in TroopM.TroopList)
                sold.Weight =(int)this.hscale1.Value;// varia il perso di ogni soldato nella lista
                winI.InsertSomeText("OPZIONI TRUPPE: nuovo peso inserito: "+ this.hscale1.Value);
            }

            this.Hide();
            MainWin.Sensitive=true;
            MainWin.CheckGreenFlags ();// check green flags
            }

    } // fine classe
} // FINE name space
```

6.6 FILE SPOTOPTIONS

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// FINESTRA VISUALIZZAZIONE OPZIONI SULLO SPOT
// PERMETTE L'INSERIMENTO DEI VALORI RICHIESTI PER OGNI SPOT
// I VALORI SONO DEFINITI IN BASE ALL'EFFICIENZA DELLO SPOT
// ED ALLA CLASSE DI UTILIZZO DELLO SPOT QUALI:
// LEGGERA - PESANTE - ALTRO.
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
using System;
using Gtk;
using System.Windows.Forms;

namespace EliAssaltoAnfibio
{
    public partial class SpotOptions : Gtk.Window
    {

        private static SpotOptions instance = null;
        static public MainWindow MainWin = MainWindow.Instance();// identifica la finestra principale in modo che
                                                                 //possa essere ritrovata alla chiusura di questa

        public SpotManager SpotM; // definizione lo spot manager
        public int _spotRec=0; // numero di record per gli spot
        private bool _isEfficient=true;
        private int _totSpot=0;

        public InfoWindow WinI;
        private bool cat1D=false;
        private bool cat2D=false;
        private bool cat3D=false;
        private bool cat1N=false;
        private bool cat2N=false;
        private bool cat3N=false;


        // COSTRUTTORE DELLA CLASSE SPOT OPTION
        protected SpotOptions (int spot , InfoWindow _win) :  base(Gtk.WindowType.Toplevel)
```

```
{
    this.WinI = InfoWindow.Instance (); // gli passo la finestra informazioni
    _totSpot=spot;
    this._spotRec = 0;
    Build ();
    WinI.InsertSomeText("OPZIONI SPOT: EFFETTUATA APERTURA FINESTRA OPZIONI SPOT");
    SpotM = (SpotManager) FactoryManager.Make(2,spot, WinI); // viene creato lo Spot Manager tramite factory method
    SpotM.MakeList(spot); // crea la  lista iniziale di un numero di spot impostato
    this.label1.Text=("Numero di Record SPOT #  "+ this._spotRec);
    this.InitValue();
    this.label1.Text = "Record # " + _spotRec + " Record inseriti: "+SpotM.Spotlist.Count+
                       " Record disponibli: "+ this._totSpot;
}

public static SpotOptions Instance(int TotSpot, InfoWindow winI)

{
    // crea il costruttore se non esiste
    if (instance == null) instance= new SpotOptions(TotSpot, winI);
    return instance; // ritorno il costruttore
}


private void InitValue ()
{
    // Inizializza variabili e checkbox per un nuovo inserimento
    _isEfficient=true;
     cat1D=false;
     cat2D=false;
     cat3D=false;
    cat1N=false;
     cat2N=false;
     cat3N=false;
    this.checkbutton7.Active = false;
    this.checkbutton1.Active =false;
    this.checkbutton2.Active=false;
    this.checkbutton3.Active=false;
    this.checkbutton4.Active=false;
    this.checkbutton5.Active=false;
    this.checkbutton6.Active=false;
    this.label1.Text = "Record # " + (_spotRec+1) + " Record inseriti: "+SpotM.Spotlist.Count+
                       " Record disponibli: "+ this._totSpot;
}

public void ShowRecord (int i)
{
    this.checkbutton7.Active =SpotM.Spotlist[i].SpotRunnable ;
    this.checkbutton1.Active =SpotM.Spotlist[i].DayCanAccept.Cat1;
    this.checkbutton2.Active=SpotM.Spotlist[i].DayCanAccept.Cat2 ;
    this.checkbutton3.Active=SpotM.Spotlist[i].DayCanAccept.Cat3 ;
    this.checkbutton4.Active=SpotM.Spotlist[i].NightCanAccept.Cat1;
    this.checkbutton5.Active=SpotM.Spotlist[i].NightCanAccept.Cat2 ;
    this.checkbutton6.Active=SpotM.Spotlist[i].NightCanAccept.Cat3;
    this.label1.Text = "Record # " + (_spotRec+1) + " Record inseriti: "+
                       SpotM.Spotlist.Count+ " Record disponibli: "+ this._totSpot;
}


private void UpdateRec (int i)
{

    SpotM.Spotlist[i].SpotRunnable = this.checkbutton7.Active ;
    SpotM.Spotlist[i].DayCanAccept.Cat1 =this.checkbutton1.Active;
    SpotM.Spotlist[i].DayCanAccept.Cat2 = this.checkbutton2.Active;
    SpotM.Spotlist[i].DayCanAccept.Cat3 =    this.checkbutton3.Active;
    SpotM.Spotlist[i].NightCanAccept.Cat1 = this.checkbutton4.Active;
    SpotM.Spotlist[i].NightCanAccept.Cat2 =    this.checkbutton5.Active;
    SpotM.Spotlist[i].NightCanAccept.Cat3 =this.checkbutton6.Active;
    WinI.InsertSomeText(" OPZIONI SPOT: record # "+(i+1)+ " AGGIORNATO..");
}


// Mostra la finestra secondaria e nascondi il main
public void ShowWin()
{
    MainWin.Sensitive=false;
    this.Visible=true;
}

// tasto di chiusura finestra
protected void OnDeleteEvent (object sender, DeleteEventArgs e)
{
    if ((_spotRec >= SpotM.Spotlist.Count) && (SpotM.Spotlist.Count>0))// effettuo il check di out bound
        _spotRec--;

    if (SpotM.Spotlist.Count==0) _spotRec=0;

    this.Hide (); // nascondo la finestra

    MainWin.Sensitive=true; // resetto la sensibilità della finestra principale
    MainWin.CheckGreenFlags ();// check green flags
    e.RetVal=true;
}


// bottone: accetta dati e torna al menu principale
protected void OnButton4Released (object sender, EventArgs e)
{
    if ((_spotRec >= SpotM.Spotlist.Count) && (SpotM.Spotlist.Count>0)) // effettuo il check di out bound
        _spotRec--;
```

```
    if (SpotM.Spotlist.Count==0) _spotRec=0;

    this.Hide (); // nascondo la finestra

    MainWin.Sensitive=true; // resetto la sensibilità della finestra principale
    MainWin.CheckGreenFlags ();// check green flags
}

// bottone : record precedente
protected void OnButton2Released (object sender, EventArgs e)
{
    if (SpotM.Spotlist.Count > 0) // controlla se esistono dati da leggere
    {
        if (this._spotRec > 0)
        {
            this._spotRec--; // decrementa il numero di record per la visualizzazione
            this.ShowRecord(_spotRec); // ShowWin il record
        }
        if (this._spotRec==0)
        {
            WinI.InsertSomeText ("OPZIONI ELICOTTERO: raggiunto record iniziale.");
            this.ShowRecord(_spotRec);
        }

    } if (SpotM.Spotlist.Count == 0)
        _spotRec = 0;
}

// bottone : record successivo
protected void OnButton3Released (object sender, EventArgs e)
{
    // effettuo i controlli
    if ((SpotM.Spotlist.Count > 0) // record presenti
        && (_spotRec<SpotM.Spotlist.Count) // il numero di record minore del massimo recordinseriti
        && (_spotRec<(_totSpot))) //il numero di record minore del massimo di rec inseribili
    {
        if (this._spotRec == (SpotM.Spotlist.Count - 1) && (SpotM.Spotlist.Count==_totSpot)) {  // raggiunto ultimo record

            WinI.InsertSomeText ("OPZIONI ELICOTTERO: raggiunto record finale");

        } else if (_spotRec==SpotM.Spotlist.Count-1
                   && SpotM.Spotlist.Count < _totSpot)
        {// controlla se esistono dati da leggere
            this._spotRec++;
            this.InitValue ();// se non esistono mostra dati vuoti
        }
        else{
            this._spotRec++; // incrementa il record
            this.ShowRecord (_spotRec); // mostra il record
        }
    }
} // fine metodo


    // bottone:  salva record corrente
        protected void OnButton5Released (object sender, EventArgs e)
        {
            // elenco dati da salvare nella lista
            cat1D = this.checkbutton1.Active;
            cat2D = this.checkbutton2.Active;
            cat3D = this.checkbutton3.Active;
            cat1N = this.checkbutton4.Active;
            cat2N = this.checkbutton5.Active;
            cat3N = this.checkbutton6.Active;
            _isEfficient = this.checkbutton7.Active;
            //inserisco il nuovo spot nella lista e stampo l'elenco
        //-----------------------------------------------------------------
    if (SpotM.Spotlist.Count <= _totSpot // se il numero di record inseriti è minore del totale inseribile
        && _spotRec < _totSpot  // se il record che dobbiamo inserire è minore del totale inseribile
        && _spotRec >= SpotM.Spotlist.Count) // se il record che dobbiamo inserire è maggiore del totale inserito
        { // SE IL RECORD NON ESISTE: INSERIMENTO NUOVO RECORD
            try
        { // provo a inserire il dato spot
            SpotM.InsertElementSpot (_spotRec, _isEfficient, cat1D, cat2D, cat3D, cat1N, cat2N, cat3N);
            }
            catch
            {
                System.Console.WriteLine ("ERRORE NELL'INSERIMENTO DEL RECORD CARGO"); // ERRORE INSERIMENTO
            }
            // visualizzo e salvo le informazioni sul record----------------------------------------
        WinI.InsertSomeText ("OPZIONI SPOT: inserto nuovo record#__" + (this._spotRec + 1) +
   " Elementi totali presenti: " + SpotM.Spotlist.Count);
        WinI.InsertSomeText (SpotM.Spotlist [_spotRec].SpotInfo ()); // visualizzo le info nella finestra informazioni

            // controllo se ultimo record e chiudo oppure se ci sono altri record allora inizializzo
        if ((_spotRec == SpotM.Spotlist.Count - 1) && (SpotM.Spotlist.Count == _totSpot)) {
                this.Hide (); // nascondo la finestra
                MainWin.Sensitive = true; // resetto la sensibilità della finestra principale
                MainWin.CheckGreenFlags ();// check green flags
            } else
                // se non è l'ultimo inizializzo i valori per il prossimo inserimento
            {
            this._spotRec++;// incrementa il record
                this.InitValue ();// reinizializza valori a 0 per il nuovo inserimento
            }

        }// fine inserimento nuovo record

        else // controllo se si tratta di un UPDATE
```

```
            if (SpotM.Spotlist.Count <= _totSpot // se il numero di valori inseriti è minore al totale inseribile
                && _spotRec < _totSpot // se il numero di record da trattare è inferiore al totale
                && _spotRec < SpotM.Spotlist.Count // se il numero di record da trattare è inferiore al totale inserito
                ) {
                    // SE IL RECORD ESISTE E' UN UPDATE
                    //effettuo l'update del record
                this.UpdateRec (_spotRec);
                }

        } // termine inserimento record o update

        //bottone: elimina record
        protected void OnButton6Released (object sender, EventArgs e)
        {
            if (SpotM.Spotlist.Count > 0) {

                bool flag = false;
                if (this._spotRec < SpotM.Spotlist.Count) {
                    SpotM.RemoveElement (_spotRec); // elimina il record
                    // se dopo l'eliminazione il record è 0
                    if ((SpotM.Spotlist.Count) == 0) { // se dopo la rimozione non ci sono piu' record
                        _spotRec = 0; // setta il rec a 0
                        this.InitValue ();// inizializza valori
                        flag = true; // flag di elemento eliminato
                    } else {
                        if ((SpotM.Spotlist.Count == _spotRec)) { // se il mio record è l'ultimo della lista
                            _spotRec = SpotM.Spotlist.Count - 1;
                        }
                        this.ShowRecord (_spotRec);
                        flag = true;
                    }
                    if (flag)
                        WinI.InsertSomeText ("OPZIONI SPOT: rimosso record # " + (_spotRec + 1));
                    else
                        this.InitValue ();
                }
            }
        } // fine metodo
    } // fine classe
}// fine namespace
```

6.7 File InfoWindow

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// definizione di una finestra per le informazioni sullo
// stato di avanzamento del programma
// lo scopo della finestra è quello di visualizzare
// le indicazione e le informazioni
// durante la prosecuzione del programma.
// le informaizoni vengono salvate in un file che funge
// da logger in modo da poter esaminare i dati
// al termine della simulazione
// parte delle informazioni rilevanti viene inoltre salvato
// all'interno di una struttura e servono per la visualizzazione
// finale delle informazioni rilevanti a schermo
// CLASSE SINGLETON
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++

using System;
using System.Collections.Generic;
using Gtk;
using System.Windows.Forms;
using System.IO;

namespace EliAssaltoAnfibio
{

    public partial class InfoWindow : Gtk.Window
    {

        private static InfoWindow instance = null;// istanza singleton

        public static InfoWindow Instance()
        {

            if (instance == null) { // se l'instanza non esiste la crea
                instance = new InfoWindow ();
                return instance;
            }
            else
            {  // riprende dal rec 0
                return instance; // ritorno il costruttore
            }
        }

        public InfoWindow () :  base(Gtk.WindowType.Toplevel)
        {
            using (StreamWriter filew = new StreamWriter (fileName)) { filew.WriteLine(" - INIZIO LOG IN DATA : "+DateTime.Now.ToString());};
            infoTimeList = new List<timeInfoStruct>(0);

            this.Build ();
        }

        // struttura di contenimento tempi rilevanti, evento rilevante
        public struct timeInfoStruct

        {
```

```
            public TimeSpan timer; // tempo di accadimento
            public string info;     // cosa accade
            public string EliName; // nome lei
            public int Elirec; // record eli che ha effettuato l'inserimento

            // costruttore
            public timeInfoStruct (TimeSpan time, string infoS, string eli, int rec): this()
            {
                this.timer = new TimeSpan();
                timer=time;
                this.info = infoS;
                this.Elirec=rec;
                this.EliName=eli;
            }
        }

        public  List<timeInfoStruct> infoTimeList;

        // salvo le informazioni nel file - uso come nome file la data e l'ora del salvataggio
        private static string fileName = string.Format("LogEliAssault_{0:hh-mm-ss}.txt", DateTime.Now);
        // file di testo


        // COSTRUTTORE classe
        public  void InsertSomeText(string s1)
        {
            string date_time = string.Format(" {0:hh-mm} ", DateTime.Now); // assegno nome file
            var iter = this.textview1.Buffer.StartIter;
            this.textview1.Buffer.Insert(ref iter, date_time +"-> "+ s1+"\n");
            this.AppendFIle ( date_time +"-> "+ s1+"\n");
        }


        // inserisci l'informazione nel file
        public  void AppendFIle (string text)
        {
            // scrivi l'informazione
            using (StreamWriter filew = new StreamWriter(fileName, true))
            {
                filew.WriteLine(text);
            }
        }

        // inserimento dell'evento nella struttura timeInfoStruct
        public void InsertEvent(TimeSpan timer, string str, string eliname, int rec)
        {
            this.infoTimeList.Add (new timeInfoStruct (timer, str,eliname, rec));
        }

        // chiusura finestra da X
        protected void OnDeleteEvent (object sender, DeleteEventArgs e)
        {
            this.InsertSomeText(" WARNING! Puoi nascondere la finestra informazioni dal menu principale");
            e.RetVal=true;
        }

    } // fine classe
} // fine namespace
```

6.8 FILE ELICOTTERO

```
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// CLASSE FISICA DI GESTIONE ELICOTTERO
// CONTIENE TUTTI GLI ATTRIBUTI NECESSARI
// ALLA DEFINIZIONE DI UN ELICOTTERO
// EREDITA DA IOBSERVER IN QUANTO OGNI ELICOTTERO
// E' DOTATO DI TIMER E NECESSITA UNA SINCRONIZZAZIONE
// PER IL CALCOLO DEL CARBURANTE E PER GESTIRE LE PAUSE
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++

using System;
using System.Collections.Generic;
using System.Threading;
using Microsoft.Xna;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;


namespace EliAssaltoAnfibio
{
    public class Elicottero : IObserver
    {
        //---internal varibiales----------------------------
        public ClockTimer timer;  // timer variabili si uso generico e MASTER TIMER
        public int NumEli {set; get;}// indica il numero di sequenza assegnato all'elicottero ( e' un indice )
        public string IdEli {set; get;}// nome o matricola che identifica l'elicottero in volo ES. King-01
        public int CatEli {set; get;} // classe di elicottero 1=pesante 2=Leggero 3=altro
        public float Fuel {set; get;} // indica il livello di carburante dell'elicottero
        public float FuelC {set;get;}// indica il consumo di carburante orario
        public int WTroopsLeftOnBoard {set; get;} // indica il numero di truppe imbarcate sull'elicottero
        public int WCargoLeftOnBoard {set;get;} // indica altri componenti di cargo general purpouse
        public int MaxTOLoad {set;get;} // max Take Off load indica il peso massimo al decollo del velivolo
        public int OffLoadWeight {set;get;}// indica il peso del velivolo scarico di carburante, truppe o altro carico
        public int TotalWeightCalculated {set;get;} // indica il peso del velivolo allo stato attuale
        public float EliSpeed {set; get;}// velocità del velivolo
        public int WaitingTime { set; get;} // waiting time per il blocco
        public int DistancetoRun { set;get;} // distanza obiettivo LZ in NM
        public int DIstanceToPnT { set; get;} // distanza obiettivo
```

```
        public float AngleToFly { set; get;} // angolo di volo per destinazione
        public float quota {set; get;} // quota iniziale , serve per mostrare graficamente la dimensione del texture
        // variando il parametro è possibile dare l'impressione ottica  di una variazione di quota
        private TimeSpan time;
        private int blockReason;
        public Spot Spotrequested{ set; get;}
        public LandingZone LZ { set; get;}
        // ---------------------------------------------

        // status flags---------------------------------------
        public bool IsRunning {set;get;} // indica se i motori sono accesi
        public bool IsRefueling {set;get;}// indica lo stato di refueling ( richiede 10 minuti )
        public bool IsBladeSpread {set;get;}// l'elicottero è a pale aperte o chiuse
        public bool IsEfficient {set;get;} // indica lo stato di efficienza del velivolo 1 efficiente 0 non efficiente
        public bool PosEli{set;get;} // indica se eli in hangar(false) o se eli sul ponte(true)
        public bool IsFlying {set;get;} // 0 se eli non in volo - 1 se eli in volo
        public bool IsHolding { set; get;} // flag indicante lo stato di holding dell'elicopttero
        public bool isLZ { set; get;} // elicottero sulla LZ
        public bool LowFuel {set;get;} // flag che indica se l'elicottero sta esaurendo il carburante
        public bool EliCrash {set;get;}// flag che indica se l'elicottero ha terminato il carburante ed è caduto
        public bool IsBlocked {set;get;} // indica lo stato di blocco momentaneo dell'elicottero ( la variabile è
        public bool IstroopsOnBoard { set; get;} // l'elicottero ha le truppe a bordo
        public bool IsCargoOnBoard { set; get;} // l'elicttero ha del cargo a bordo                              o
        public bool isOnDestination { set; get;} // l'elicottero sta occupando un a LZ
        public bool isREADYstatus { set; get;} // l'elicottero è sul ponte, truppe a bordo , carico a bordo pronto per partire
        public bool isRequiredForOP { set; get;} // elicottero richiesto per l'operazione
        public bool Go_goback { set; get;} // vai verso la lanzig zone (true ) torna dalla lanzing zone (false)
        public bool IsFull { set; get;} // indica che non è piu' possibile caricare altro materiale
        public bool IsCargoFull { set; get;} // cargo full
        public bool isTroopsFull { set; get;} // truppe full
        public bool isApproaching { set; get;} // approach to destination
        public bool isHotRef { set; get;} // flag di hotref
        public bool hasSpotReserved { set; get;} // ha riservato lo spot
        public bool IsBoarding { set; get;} // se l'elicottero sta caricando truppe
        public bool IsBoardingCargo { set; get; } // boarding del cargo
        // end status flags-----------------------------------

        //  settaggi per il movimento nello spazio grafico-----
        public Vector2 elivector2d { set; get;} // vettore pos e movimento

        public Point destination { set; get;}
        public float rotation {set;get;}    // indica la rotazione del vettore rispetto ad un punto misurata in radianti
        public Texture2D eliTexture{set; get;} // eli texture TODO cambia a seconda dell'elicottero

        public enum Direction: int {lz,hp,ship};// enumerazione giorno e notte
        public Direction DirToGo { set; get;}// direzione di movimento
        Random random = new Random(); // RANDOM GENERATOR utile per variabili casuali
        //-------------------------------------------------------

        //-------------------------------------------------------


        // liste di gestione ogni elicottero puo' essere carico con truppe e/o cargo
        public List<Soldier> EliSoldierList{set;get;} // lista soldati a bordo dell'elicottero
        public List<Cargo> EliCargoList{set;get;} //lista del cargo presente a bordo dell'elicottero - il cargo è formato da blocchi di materiale

                                    // o armamento che hanno un peso ( ogni cargo necessita di persone a bordo per il controllo e lo scarico
- carico)

        public Spot SpotAssigned{set;get;} // indica lo spot di assegnazione attribuito all'elicottero
                                    // il valore è nullo se l'elicottero non è sullo spot.



        public TimeSpan current_time; // rappresenta l'update del timer corrente


        // costruttore della classe elicottero
        public  Elicottero (ClockTimer timerCon, int numEli, string Id, int Cat, float Fuel ,  int Troops , int Cargo , int MaxToLoad , int OffLo
adWG, bool Running, bool BladeSprd , bool IsEff, bool InitialPos, bool IsFly , float FuelC)
        {
            // passo le variabili inizializzate all'oggetto elicottero

            this.NumEli=numEli;
            this.IdEli= Id;
            this.CatEli=Cat;
            this.Fuel=Fuel;
            this.WTroopsLeftOnBoard=Troops;
            this.WCargoLeftOnBoard=Cargo;
            this.MaxTOLoad=MaxToLoad;
            this.OffLoadWeight=OffLoadWG;
            this.IsRunning=Running;
            this.IsRefueling=false; // l'elicottero non parte mai in fase di refueling
            this.IsBladeSpread=BladeSprd;
            this.IsEfficient=IsEff;
            this.IsFlying=IsFly;
            this.PosEli=InitialPos; // posizione iniziale Hangar o Ponte
            this.FuelC=FuelC;// consumo di carburante all'ora
            this.SpotAssigned=null; // assegnazione spot inizializzata a null
            this.LowFuel=false;// setto il lowfuel state a falso, alla messa in moto viene ricontrollato.
            this.EliCrash=false;// setta la flagh di crash a falso
            this.IsBlocked = false; // l'elicottero non parte bloccato
            this.isREADYstatus = false; // l'elicottero inizializzato non pronto
            this.IsHolding = false; // l'elicottero è inizializzato non in holding
            this.isLZ = false; // elicottero sulla LZ
            this.hasSpotReserved = false;
            this.rotation = 0; // rotazione iniziale
            this.quota = 0.6f; // quota iniziale
            this.isRequiredForOP = true;// elicottero richiesto per l'operazione di default
            this.IstroopsOnBoard = false; // truppe a bordo
```

```
    this.IsCargoOnBoard = false; // l'elicottero parte senza cargo
    this.Go_goback = true; // setto l'andata
    this.IsFull = false; // elicottero parte vuoto
    this.IsCargoFull = false; // cargo full
    this.isTroopsFull =false;  // truppe full
    this.isApproaching = false; // approach to destinaiton
    this.isHotRef = false; // flas hot ref su falso
    this.IsBoardingCargo = false; // sett iniziale imbarco cargo falso
    this.EliSoldierList = new List<Soldier>(0); // ogni elicottero detiene la sua lista di soldati
    this.EliCargoList = new List<Cargo>(0);// ogni elicottero detiene la lista del proprio cargo
    this.DirToGo = Direction.ship; // inizalizzato per direzione holding point
    this.IsBoarding = false; // l'elcittero è iizializzato a caricamento truppe falso


    // il peso totale è dato dal peso dell'elicottero scarico + il peso del carburante + il peso delle truppe + il peso del cargo
    this.TotalWeightCalculated=this.ToTWGCalculus();

    //timer attachment: ogni elicottero e' dotato di un timer
    // serve per sincronizzare le operazioni
    this.timer = timerCon; // definizione del timer
    this.timer.Attach(this); // attachment del timer secondo il pattern Observer
                        // l'elicottero entra a far parte della lista di observers
                        // il notify() del timer provvederà all'update dei clock di tutti gli observer


    this.elivector2d = new Vector2 (0f,0f); // creazione del vettore di movimento verrà usato dal framework grafico monogame
                                            // indica la posizione attuale del velivolo sullo schermo
    this.destination = new Point (0,0); // destinazione dell'elicottero espressa con una struttura point X e Y

    this.EliSpeed = 0f; // viene settato il parametro velocita' iniziale dell'eli

}

//OBSERVER PATTERN
public void Update (Subject subj) // observe pattern, la variabile curren_time viene
                                  // ridefinita ogni volta che si effettua l'update del timer
{
    if (subj != null)
    {
        // UPDATE RICHIAMATO OGNI TICK
        this.current_time=timer.GetTime(); // copia il timer nella variabile timeSpan current_time
    }
}


// metodo di generazione di quota randomico
// l'elicottero simula una piccola variazione di quota
// generata graficamente con la variazione dello zoom dello sprite
public void  RandomQuoteGenrator ()
{
    if (random.Next (2) == 0) {
        if (quota < 0.8f)
            this.quota = this.quota + 0.005f;
    } else if (quota > 0.55f)
        this.quota = quota - 0.005f;
}



// setta la flag in blocco per il tempo necessario all'operazione
public void EliBlock (int ticks, int k)
{

    bool passFlag = false;

    if (!this.IsBlocked && k!=-1) { // se l'elicottero non è bloccato resetto il timer

        time = new TimeSpan (0, 0, ticks); // BLOCCO PER UN NUMERO DI TICKS
        time = this.timer.GetTime() + time; // tempo massimo di blocco
        this.blockReason = k; // salvo la motivazione di blocco NON POSSO RESETTARE LE MOTIVAZIONI DI
                              // BLOCCO SE L'ELICOTTERO NON E' SBLOCCATO
        this.IsBlocked = true;
        }

    if (time <= timer.GetTime() && this.IsBlocked )
        {
        passFlag = true; // se supero il tempo necessario la variabile passa diventa true
        }

    if (passFlag) // la passflag sblocchera l'accesso alle casistiche
        {
        // definizione dei blocchi temporali con il segnale che le ha generate


        if (this.blockReason == 1)  this.IsRunning = true; // messa in moto

        if (this.blockReason  == 2) this.IsBladeSpread = true; // pale aperte

        if (this.blockReason  == 3) // blocco per riposizionamento
            {
                this.PosEli = true;// posizione sullo spot
                this.SpotAssigned = this.Spotrequested;
                this.SpotAssigned.SpotOccupied = true; // spot occupato
                this.SpotAssigned.Eli = this; // assegna lo spot all'elicottero
                this.SpotAssigned.IsReservd = false; // lo spot è occupato non è piu' riservato
                this.hasSpotReserved = false;
                this.Spotrequested = null;
            }

        // SEGNALE 4 ATTESA PER APERTURA PALE
```

```
            if (this.blockReason  == 4) this.IsBladeSpread = true;

            // SEGNALE 5 ATTESA PER INSERIMENTO CARGO
            if (this.blockReason == 5) {
                this.IsBoardingCargo = false; // termine del boarding cargo
                this.IsCargoOnBoard = true; // cargo a bordo flag
            }
            // SEGNALE 6 ATTESA PER SCARICO DEL CARGO
            if (this.blockReason == 6)
            {
                this.IsCargoOnBoard = false; // cargo unloading time
                foreach (Cargo cargo in this.EliCargoList) {

                    cargo.IsEliC = false; // il cargo e' a terra
                    cargo.Eli = null;
                    cargo.isLand = true;// setto la flag per cargo a terra
                    // vettore cargo a terra
                    cargo.CargoVector = this.elivector2d + (new Vector2 (StaticSimulator.CargoVectorPosX, StaticSimulator.CargoVectorPosY));
                    this.WCargoLeftOnBoard = this.WCargoLeftOnBoard + cargo.ChecktotalCargoWeight ();  // aumento il peso totale destinato al
cargo
                    this.LZ.LZCargo.Add(cargo);// aggiungo elementi alla lista cargo presente sulla LZ


                }
                this.EliCargoList.Clear();//
                this.IsCargoFull = false;
                this.IsCargoOnBoard = false;

            }

            // SEGNALE 7 ATTESA PER HOT REF
            // due volte la distanza di andata e ritorno dalla LZ + 20 minuti di carburante
            if (this.blockReason  == 7)
            {

                this.Fuel = this.Fuelrequired (MainWindow.Elioptions.EliM.ElicotteriList.Count);
                this.isHotRef = false; // resetto la variabile hot ref e ripristino il
                                       // normale afflusso dicarburante
            }

            // SEGNALE 8 ATTESA PER IMBARCO TRUPPE
            if (this.blockReason == 8) this.IsBoarding = false; // CARICO TRUPPE - alla fine resetto la variabile il boarding a falso

            // SEGNALE 9 ATTESA PER SCARICO TRUPPE
            if (this.blockReason == 9)
            {
                foreach (Soldier troop in this.EliSoldierList)
                {
                    troop.IsEli = false; // le truppe sono a terra
                    troop.IsLand = true;// setto le flags per eli a terra
                    // vettore truppa a terra
                    troop.vectoreTroop = this.elivector2d + (new Vector2
                        (StaticSimulator.TroopsVectorPosX + StaticSimulator.RandomIntGenerator (10),
                        StaticSimulator.TroopsVectorPosY + StaticSimulator.RandomIntGenerator (10)));

                    this.LZ.LZSoldierList.Add(troop); // aggiungo elementi alla lista della LZ
                    this.WTroopsLeftOnBoard = this.WTroopsLeftOnBoard + troop.Weight; // aumento il peso totale destinato alle truppe

                }

                this.EliSoldierList.Clear (); // libera la lista sull' elicottero
                this.IstroopsOnBoard = false; // SCARICO TRUPPE
                 this.isTroopsFull = false; // l'elicottero essendo scarico resetta anche la variabile FULL
            }

            // FINE BLOCCO PER TUTTI I SEGNALI

            this.IsBlocked = false; // sblocco la variabile blocco
        }

    } // termine time
    // calcolo peso truppe a bordo
    private int totTroopOnBoard ()
    {
        if (this.EliSoldierList != null && this.EliSoldierList.Count != 0)
            return (this.EliSoldierList.Count * this.EliSoldierList [0].Weight);
        else return 0;
    }


    // calcolo peso cargo a bordo
    private int totCargoOnBoard ()
    {
        int part = 0;
        if (this.EliCargoList != null && this.EliCargoList.Count != 0) {
            foreach (Cargo cargo in this.EliCargoList) {
                part = part + cargo.ChecktotalCargoWeight ();
            }
            return part;
            }
        else return 0;
    }


    //calcolo il peso totale dell'elicottero a pieno carico con pieno di truppe e pieno di cargo
    public int ToTWMaxFull ()
    {
        return this.OffLoadWeight + this.WTroopsLeftOnBoard+this.WCargoLeftOnBoard+ (int)this.Fuel;
    }
```

```
        // calcolo del peso totale del velivolo con truppe e carburante a bordo
        public int ToTWGCalculus ()
        {
            return this.OffLoadWeight+this.totTroopOnBoard()+this.totCargoOnBoard ()+ (int)this.Fuel;
        }




        // scrivi le informazioni disponibili sull elicottero corrente
        public string EliInfo ()
        {
            return "Record_eli: "+ this.NumEli +" ID: "+this.IdEli+" PosEli:"+ this.PosEli+ " Cat: "+ this.CatEli+" Fuel: "+this.Fuel+ " Spazio
truppe: "+this.WTroopsLeftOnBoard+" Spazio Cargo: "+this.WCargoLeftOnBoard+" MaxT-
O weight: "+this.MaxTOLoad+" OffLoad Weight: "+this.OffLoadWeight+" Consumo: "+this.FuelC;
        }

        // CALCOLA IL CARBURANTE NECESSARIO
        // 2 volte la distanza andata e ritorno per raggiungere la LZ
        // calcolata alla minima velocità
        // viene sommato il tempo di attesa ipotetico in holding pari a 1h di volo
        // se il numero di elicotteri aumenta i tempi di attesa potrebbero aumentare
        // quindi viene sommato un fattore correttivo
        public float Fuelrequired (int _toteliNum)
        {
            int fuelKcorrection = 0;
            if (_toteliNum > 6)
                fuelKcorrection = 100 * (_toteliNum - 6);
            return (((((2 * this.DistancetoRun) / 70) * this.FuelC) + (this.FuelC))+fuelKcorrection);
        }
    }// end class
}// end namespace
```

6.9 File Soldier

```
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// classe soldato : descrive gli attributi di ogni soldato
// ogni singolo soldato che deve essere caricato
// a bordo dell'elicottero per effettuare lo sbarco
// la missione è completata quando truppe e cargo sono sulla LZ
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

using System;
using Microsoft.Xna;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;

namespace EliAssaltoAnfibio
{
    public class Soldier
    {

        public int Weight{ set; get;} // peso dell'unità
        public bool IsShip{ set; get;} // truppa a bordo nave, di default la truppa parte a bordo
        public Vector2 vectoreTroop { set; get;} // indica la posizione dell'ellemento truppa a terra


        public bool IsLand{ set; get;}  // truppa a terra
        public bool IsEli{ set; get;} // truppa in elicottero
        public int EliNum{ set; get;}  // indica il record # del numero di elicottero a cui viene assegnata la struppa

        // contrutto della classe soldato
        public Soldier (int weight)
        {
            this.IsShip = true; // i soldati partono sulla nave
            this.Weight= weight; // peso del singolo soldato
            this.IsLand = false;
            this.IsEli = true;
            this.vectoreTroop = new Vector2 (0, 0);

        }

    }// fine classe
}//fine name space
```

6.10 File Cargo

```
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// classe cargo : descrive gli attributi di ogni cargo
// ogni singolo cargo che deve essere caricato
// a bordo dell'elicottero per effettuare lo sbarco
// La missione è completata quando truppe e cargo sono sulla LZ
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


using System;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using System.Collections.Generic;

namespace EliAssaltoAnfibio
{
    public class Cargo
```

```
    {
        public int CargoRec {set;get;} // record cargo
        public int CargoW {set;get;} // cargo peso totale
        public int CargoP {set;get;}// personale necessario al trasporto
        public string CargoString {set;get;}// descrizione del cargo
        public Vector2 CargoVector { set; get;} // vettore cargo
        public Elicottero Eli{set;get;} // al cargo puo' essere assegnato un elicottero
        public bool IsEliC {set;get;}// flag indicante che il cargo è a bordo
        public bool isLand { set; get;} // flag indicante che il cargo è a terra
        public bool IsFittable { set; get;}// flag indicante che un cargo non è adattabile
        public List<Soldier> ListCargoTroop { set; get;}
        private int SoldierW =0;

        // costruttore della classe Cargo
        public Cargo (int cargoRec ,string cargoType, int cargoW , int cargoP)


        {
            this.CargoRec=cargoRec; // record del cargo
            this.CargoString=cargoType; // stringa che definisce il cargo ( es armi , munizioni...)
            this.CargoW=cargoW; // pero del cargo SOLO MATERIALI
            this.CargoP=cargoP; // truppe ( le truppe hanno un peso derivante dal record inserimento per le truppe )
            this.IsEliC = false; // il cargo non parte in elicottero
            this.isLand = false; // il cargo non parte a terra
            this.IsFittable = true; // il cargo viene generato inseribile in elicottero
            CargoVector = new Vector2 (0, 0); // inizializzo un vettore vuoto
            ListCargoTroop = new List<Soldier> (0); // inizializzo una lista vuota
        }


        // stringa informazioni per il cargo
        public string CargoInfo ()
        {
            return "Record_cargo: "+ this.CargoRec + " Tipo Cargo: "+ this.CargoString + " Peso Cargo: "+ this.CargoW + " Utenti necessari: "+thi
s.CargoP;
        }


        // calcola il peso totale del cargo e delle truppe necessarie al cargo
        public int ChecktotalCargoWeight ()
        {
            return  (this.CargoW + this.ListCargoTroop.Count * this.SoldierW);
        }

    } // fine classe
} // fine name space
```

6.11 File Spot

```
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// classe spot : descrive gli attributi di ogni spot.
// Ogni spot può avere caratteristiche che lo rendono
// gestibile di giorno e notte da ogni classe di elicotteri
// Un elicottero può appontare su uno spot di giorno o di notte
// in relazione alla classe di appartenenza leggero o pesante o altro
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


using System;

namespace EliAssaltoAnfibio
{
    public class Spot
    {
        public enum day_night: int {day, night};// enumerazione giorno e notte

        // struttura per definire l'accettabilità diurna/notturna di un una categoria eli
        // Non tutte le categorie di elicottero sono impiegabili di giorno e di notte
        // su un determinato Spot. E' possiible ad esempio che una categoria di elicottero leggero
        // possa appontare di giorno e di notte mentre una CAT pesante possa appontare solo di notte.
        public struct DNCatAccept

        {
            public day_night DN {get;set;} // accettazione day night
            public bool Cat1 {get; set;} // accettazione per la cat 1
            public bool Cat2 { get; set;} // accettazione per la cat 2
            public bool Cat3 {get; set;} // accettazione per la cat 3

        public  DNCatAccept(  day_night DN , bool cat1 , bool cat2, bool cat3): this()
        {
                this.DN = DN;
                this.Cat1=cat1;
                this.Cat2=cat2;
                this.Cat3=cat3;

        }}

        public DNCatAccept DayCanAccept = new DNCatAccept(day_night.day,false,false,false); // struttura operazioni diurne
        public DNCatAccept NightCanAccept = new DNCatAccept(day_night.night,false,false,false);// struttura operazioni notturne



        public int SpotPos{set;get;} // la posizione viene contrassegnata da prora verso poppa dal numero 1 al numero X
                            // ad esempio lo spot1 indica lo spot piu' a prora.
                            // viene usato come numero di indice per l'assegnazione

        // status flags ----
        public bool SpotOccupied {set;get;} // indica se lo spot è ingombro o libero da elicotteri
```

```
        public bool SpotRunnable{set;get;} // indica lo stato di efficienza dello spot
        public bool IsReservd { set; get;} // indica lo stato di prenotato dello spot
        // end status
        public Elicottero eliWhoReservd { set; get;} // indica l'elicottero che ha effettuato la prenotazione
        public Elicottero Eli {set;get;} // quando non nullo indica l'elicottero presente sullo spot



        // costruttore della classe con l'inserimento dei dati
        public Spot (int spotPos, bool spotRunnable, bool cat1D, bool cat2D , bool cat3D , bool cat1N , bool cat2N , bool cat3N )
        {
            this.SpotPos = spotPos; // viene assegnato un numero di pos allo spot
            this.SpotRunnable = spotRunnable; // lo spot è creato funzionante
            this.SpotOccupied=false; // lo spot non viene creato occupato
            this.IsReservd = false; // lo spot non nasce riservato ad alcun elicottero
            this.DayCanAccept.Cat1=cat1D; // classe
            this.DayCanAccept.Cat2=cat2D;
            this.DayCanAccept.Cat3=cat3D;
            this.NightCanAccept.Cat1=cat1N;
            this.NightCanAccept.Cat2=cat2N;
            this.NightCanAccept.Cat3=cat3N;

            // elicotteri
            this.Eli = null; // nessun elicottero sul ponte
            this.eliWhoReservd = null; // nessun elicottero riservante lo spot
        }

        // stringa informazioni per la classe spot
        public string SpotInfo ()
        {
            string _idEli="null"; // impostazione iniziale stringa eli nulla
            if (Eli!=null) _idEli=Eli.IdEli; // se esiste un elicottero sullo spot indica quale
        return "Record_Spot: "+ SpotPos+ " Efficienza: "+SpotRunnable+" Spot occupato: "+SpotOccupied +" Da Eli: "+_idEli+ " GIORNO Cat1: "+DayCa
nAccept.Cat1 + " GIORNO Cat2: "+DayCanAccept.Cat2 + " GIORNO Cat3: "+ DayCanAccept.Cat3 +" NOTTE Cat1: " + NightCanAccept.Cat1 + " NOTTE Cat2: "+
 NightCanAccept.Cat2 + " NOTTE Cat3: "+ NightCanAccept.Cat3;
        }
    } // fine classe
} // fine namespace
```

6.12 File LandingZone

```
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// la classe definisce le caratteristiche della landing zone
// la LZ puo' contenere truppe, cargo , elicotteri.
// le liste vengono create ed inizializzate a 0
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;

namespace EliAssaltoAnfibio
{

    public class LandingZone
    {

        public int ShipDistance{set;get;} // indica la distanza dall'unità
        public int HPDistance { set; get;} // indica la distanza dall'holding point
        public List<Soldier> LZSoldierList{set;get;} // definizione di una lista soldati di contenimento
        public List<Cargo> LZCargo { set; get; } // definizione di una lista cargo di contenimento
        public List<Elicottero> LZeliList { set; get;} // elicotteri sulla LZ
        public Point LZposition{ set; get;} // posizione LZ
        public Texture2D LZtexture; // texture LZ
        public Vector2 LZvector;  // vettore LZ
        private float effect=0.22f; // effetto per flikering

        public LandingZone (int dist)
        {
            this.ShipDistance = dist; // viene inizializzata la distanza LZ-Ship
            // la landing zone viene inizializzata in una posizione grafica permissiva
            LZposition = new Point (StaticSimulator.LZposX, StaticSimulator.LZposY);
            LZSoldierList = new List<Soldier>(0); // la landing è inizializzata a 0 elementi soldato
            LZCargo = new List<Cargo> (0); // la landing zone è inizializzata a 0 elementi cargo
            LZeliList = new List<Elicottero> (0); // crea un alista vuota di elicotteri
        }

        // effetto flikering LZ
        public float SpriteEffectMath(){

            if (effect > 0.24f) effect = 0.22f;
                else effect= effect+0.01f;

            return effect;
        }

    } // fine classe
} // fine namespace
```

6.13 File HoldingPoint

```
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// indica il punto di attesa in volo per gli elicotteri
// il punto di attesa è una rotta circolare in cui far attendere
```

```
// gli elicotteri in attesa che si verifichino alcuni eventi
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
using System;
using System.Collections.Generic;
using Microsoft.Xna;
using Microsoft.Xna.Framework;
namespace EliAssaltoAnfibio
{
    public class HoldingPoint
    {

        int radius {set;get;} // raggio di evoluzione dell'holding point
        int Distanza {set;get;} // indica la distanza dell'holding point dalla nave
        public List<Elicottero> EliHolding{set;get;}// l'holding zone contiene gli elicotteri
        public Vector2 HPvector; // la lz ha un vettore

        public HoldingPoint (int dist) // la lz è inizializzata con una distanza dalla nave

        {
            this.Distanza = dist; // viene passala la distanza dal costruttore
            EliHolding = new List<Elicottero>(0); // viene creata la lista di elicotteri preseti e settata a 0
            HPvector = new Vector2 (StaticSimulator.HPposX, StaticSimulator.HPposY); // nuovo vettore per l'Holding point
        }
    } // fine classe
}// fine namespace
```

6.14 File AbstractManager

```
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// ABSTRACT MANAGER
// nel abstract manager viene definita una classe astratta dalla quale
// derivare le altre tipologie di classe.
// nella fattispecie adottiamo differenti tipologie di manager
// in funzione degli oggetti che devono essere controllati.
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

using System;
using System.Collections.Generic;

namespace EliAssaltoAnfibio
{
    public abstract class AbstractManager
    {

        protected string TypeManager; // indica la tipologia di Manager che verra' implementato

        // stato FULL - la lista è piena e non puo' ospitare altri elementi
        // stato EMPTY - la lista è vuota ma non è detto che possa accettare altri elementi
        // stato CanAll - puo' fare tutto - caricare o scaricare
        public enum StatusManager: int
        {
            Full,
            Empty,
            CanAll
        } // stato lista piena , vuota , lista con alcuni dati

        // LO STATUS E' STATO PREVISTO PER SUCCESSIVE IMPLEMENTAZIONI MA MAI USATO
        // ALL'INTERNO DEL PROGRAMMA DETERMINA LO STATO DEL MANAGER DI GESTIONE
        // stato ReadyToSend - pronto per inviare elementi ad un altro manager
        // stato ReadyToReceive - pronto per ricevere elementi da un altro manager
        // stato ReadyToSendAndReceive - pronto per inivare o ricevere
        // stato wait - stato attesa
        public enum ActionManager: int
        {
            ReadyToSend ,
            ReadyToReceive ,
            ReadyToSendAndReceive,
            Wait
        } // lo stato di azione del manager

        public AbstractManager (string tmanager)
        {
            TypeManager = tmanager;
        }
        // definizione dei metodi comuni alle classi

        public abstract void MakeList (int i) ;

        // inserisci un elemento nella lista
        public abstract void InsertElement (Object i);

        //rimuovi un elemento dalla lista REMOVEAT INDEX
        public abstract  void RemoveElement (int i);

        //controlla lo stato della lista
        public abstract void CheckState ();
    } // end class
} // end name space
```

6.15 File FactoryManager

```
//++++++++++++++++++++++++++++++++++++++++++++
// classe provvisoria si tenta di creare un class
// maker per le differenti tipologie
// di manager degli elicotteri
//++++++++++++++++++++++++++++++++++++++++++++
```

```
using System;

namespace EliAssaltoAnfibio
{
    // classe statica del factory manager, serve per la creazione
    // di un oggetto di tipo manager necessario.

    static public class FactoryManager
    {
        // le classi statiche non hanno un costruttore

        // con la variabile elementi viene indicato il numero iniziale di elementi che dovra' vestire il manager.
        public static AbstractManager Make(int id, int elementi, InfoWindow winI)
        {

            switch(id)
            {
            case 0: default: return EliManager.Instance(elementi, winI);// manager per elicotteri

            case 1: return TroopsManager.Instance(elementi, winI);// manager per truppe

            case 2: return SpotManager.Instance(elementi, winI);// manager per spot

            case 3: return CargoManager.Instance(elementi, winI);//manager per cargo

            }
        }
    } // fine classe
} // fine name space
```

6.16 File EliManager

```
//************************************************************
// classe manager per la creazione elicotteri ed il controllo degli stati
// detiene la lista elicotteri e le operazioni su di essi
// classe singleton
//************************************************************

using System.Collections.Generic;
using System;
using System.Linq;
using Microsoft.Xna;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;


namespace EliAssaltoAnfibio
{
    public class EliManager : AbstractManager
    {
        public static  EliManager instance= null;
        public ClockTimer MainTime = new ClockTimer (); // definizione del main timer per gli elicotteri
        public int TotEli{ set; get;}    // indica il numero di elicotteri da creare
        public int LZdistance{ set; get;} // è la distanza della LZ dalla nave
        public List<Elicottero> ElicotteriList{ set; get;} // definizione della lista di elicotteri
        public HoldingPoint HoldingP{ set; get;}    // holding point per elicotteri
        public LandingZone LandingZoneL{ set; get;}// landing zone
        public StatusManager Stato{ set; get;}// variabili di stato
        public List<Elicottero> SupportEliList = new List<Elicottero> (0); // lista supporto
        public List<Elicottero> CrashList = new List<Elicottero> (0); //eli crash list
        public ActionManager Action{ set; get;}// definizioni dello stato del manager
        public InfoWindow WinI;
        public int initEliNum { set;get;}
        //implementare la lista di elicotteri e di vettori-----
        // si è preferito implementare la lista di frame per la visualizzazione dell'eli
        // con le pale in rotazione nella classe manager per eli
        public Texture2D[] EliTexture = new Texture2D[7]; // frames eli volo
        public Texture2D CrashedEli; // frame crash

        // eli list disposer specs
        // costruttore della classe, Eredita il tipo di manager dalla classe base
        public EliManager (int numeroEli, InfoWindow winI) : base ("Manager_per_Elicottero")
        {
            this.WinI = winI; // finestra informazioni
            this.TotEli = numeroEli; // acuisisce il numero di elicotteri iniziali da gestire
            this.Stato = StatusManager.Empty;  // setta lo stato come vuoto in quanto non ci sono ancora elicotteri inseriti
            this.Action = ActionManager.Wait; // le azioni sono in attesa di informazioni aggiuntive
            this.MainTime.SetTime (new TimeSpan (0, 0, 0));// setting del timer iniziale a 0
        }

        // istanza singleton è prevista l'esistenza di un'unica finestra
        public static EliManager Instance(int numeroEli, InfoWindow winI)
        {
            if (instance == null) instance= new EliManager( numeroEli ,  winI );
            return instance; // ritorno il costruttore
        }

        // crea la lista con un numero di elementi specifico
        // la lista in relata' resta vuota, ma viene assegnato lo spazio
        // iniziale per poter ospitare un numero di oggetti voluto
        // il manager puo' gestire differenti tipi di liste a seconda dell'override del metodo
        public override void MakeList (int elementi)
        {
            this.ElicotteriList = new List<Elicottero> (elementi);
```

```
        }

        // inserisci un elemento nella lista, viene usata per inseire nella lista un elicotteo
        // che è già stato creato
        public override void InsertElement (object eli)
        {
            this.ElicotteriList.Add ((Elicottero)eli);
        }

        //rimuovi un elemento dalla lista REMOVEAT INDEX
        public override  void RemoveElement (int i)
        {
            this.ElicotteriList.RemoveAt (i);
        }

        // aggiungi l'elicottero alla lista di holding
        public void AddEliHp (HoldingPoint hp, Elicottero eli)
        {

            hp.EliHolding.Add (eli);
        }
        //rimuovi l'elicottero dalla lista degli hoding
        public void RemoveEli (HoldingPoint hp, Elicottero eli)
        {

            hp.EliHolding.Remove (eli);

        }
        // aggiungi l'elicottero alla lista
        public void AddEliList (List<Elicottero> obj, Elicottero eli)
        {

            obj.Add (eli);
        }

        //metodo di creazione della landing zone
        public  void MakeLZ (int dist)
        {
            LandingZoneL = new LandingZone (dist); // viene creata la landing zone
        }
        // viene creato l'holding point quando necessario e dove necessario
        public void MakeHoldingPoint (int dist) // crea l'hoding point alla distanza definita
        {

            HoldingP = new HoldingPoint (dist);
        }

        // determina l'inserimento di un nuovo elicottero nella lista
        public  void InsertElementEli (ClockTimer time, int numEli, string id, int cat, float fuel, int troops, int cargo,
            int maxToLoad, int offLoadWg, bool running, bool bladeSprd, bool isEff, bool initialPos, bool isFly, float fuelC)
        {
            // creazione del nuovo elicottero con valori inizializzati
            Elicottero eli = new Elicottero (time, numEli, id, cat, fuel, troops, cargo,
                maxToLoad, offLoadWg, running, bladeSprd, isEff, initialPos, isFly, fuelC); // creazione del nuovo elicottero con valori iniziali
zzati
            // inserisco l'elemento creato nella lista
            ElicotteriList.Insert ((numEli), eli);
        }
        // controlla gli elicotteri bloccati e non volanti
        // controlla quanti elicotteri attualmente bloccati sono in hangar
        // l'elicottero inoltre deve essere efficiente e richiesto per l'operazione
        public int CheckEliBlockedNotFlying ()
        {
            int n = 0;
            foreach (Elicottero eli in this.ElicotteriList) {
                if (eli.IsBlocked && !eli.PosEli && eli.IsEfficient && eli.isRequiredForOP)
                    n++;
            }
            return n;
        }

        // restituisce il numero di elicotteri pianificati o presenti sul ponte di volo
        public int EliOnDEck ()
        {
            int n = 0;
            foreach (Elicottero eli in this.ElicotteriList) {
                if (eli.PosEli)
                    n++;
            }
            return n;
        }
        // restituisce il numero di elicotteri efficienti ed impiegabili per l'operazione
        // gli elicotteri sono la forza effettiva
        public int EliForce ()
        {
            int n = 0;
            foreach (Elicottero eli in this.ElicotteriList) {
                if (eli.IsEfficient
                    && eli.isRequiredForOP  // se gli eli sono required
                    && eli.Go_goback  // vanno in direzione lz
                    && !(eli.DirToGo==Elicottero.Direction.lz)) // hanno la destinazione LZ
                    n++; // incremento il contatore
            }
            return n;
        }
        // controllo gli elicotteri bloccato e richiesti per l'operazione attualmente in hangar
        public int CheckBlockedHangar ()
        {
            int n = 0;
```

```
    foreach (Elicottero eli in this.ElicotteriList)
    {
        if (eli.IsBlocked // se l'eli è bloccato
            && !eli.PosEli  // l'eli è in hangar
            && eli.hasSpotReserved) // ed ha uno spot riservato
            n++; // incremento il contatore
    }

    return n;
}

// restituisce il numero di elicotteri in hangar che possono essere spostati
// gli elicotteri devono essere efficienti non bloccati e richiesti per l'attività
public int EliInHangar ()
{


    int n = 0;
    foreach (Elicottero eli in this.ElicotteriList) {
        if (!eli.PosEli  // posizione in hangar
            && !eli.IsFlying // l'eli non vola
            && !eli.hasSpotReserved // l'eli ha uno spot riservato
            && eli.SpotAssigned == null // non
            )
            n++; // posizione in hangar posEli=false
    }
    return n;
}


// restituisce l'elicottero efficiente in grado di contenere il cargo specificato
// con le truppe necessarie
// viene effettuato anche il controllo sugli operatori necessari per la movimentazione
// del cargo stesso, Viene impiegato prima l'elicottero con il maggior peso dsponibile
public Elicottero  EliAbleToCargo (Cargo cargo, TroopsManager troopM)
{
    int minTWeight = 0;// peso totale minimo tra cargo e truppe a bordo trovato su elicotteri disponibili

    Elicottero eliMinW = null;// elicottero efficiente che ha il maggior carico disponibile

    foreach (Elicottero eli in this.ElicotteriList) { // per ogni elicottero in lista

        // controllo che lo spot sia occupato da un elicottero efficiente, e che abbia lo spazio per cargo e truppe richieste
        if ((eli.isRequiredForOP && eli.Go_goback && !eli.IsBlocked && eli.IsEfficient && !eli.IsCargoFull && !eli.IsFlying
            && eli.WCargoLeftOnBoard >= cargo.ChecktotalCargoWeight ()))

            // controllo che l'elicottero sia quello ceh offre la maggior disponibilità di cargo
        if ((eli.WCargoLeftOnBoard) > minTWeight) {
            minTWeight = eli.WCargoLeftOnBoard;
            eliMinW = eli;    // assegno l'elicottero
        }
    }
    return eliMinW; // ritorno l'elicottero trovato se non lo trova rotorna null
}

// LIST DISPOSER METODI - metodi di movimentazione degli elementi in relazione agli altri manager
#region list disposer
// sposta gli elicotteri sugli spot se questi possono contenerli
public  void MoveEliOnSpot (SpotManager SpotM ,Spot.day_night DN, InfoWindow WinI)
{
    bool elidispoSuccess; // flag disposizione avvenuta con successo
    // controllo se esistono elicotteri che possono essere ospitati dagli spot per le caratteristiche
    // che hanno elicottero e spot
    foreach (Elicottero eli in this.ElicotteriList) {
        if (!eli.IsBlocked // controllo se l'elico non è bloccato ed è in hangar
            && !eli.PosEli // eli è in hangar
            && !eli.hasSpotReserved // eli non ha riservato uno spot
        ) {

            Spot _spot = SpotM.SpotToHostaCat (eli.CatEli, DN);//ritorno il primo spot in grado di ospitare l'elicottero

            if (_spot != null) { // se lo spot è stato trovato
                WinI.InsertSomeText ("Spot trovato per assegnazione: " + _spot.SpotPos); // scrittura info

                // viene se l'elicottero è efficiente, la posizione è in hangar
                //e lo spot esiste e non è riservato ad altri elicotteri
                // inoltre possono essere movimentati solo due elicotteri alla volta

                    if (elidispoSuccess = SpotM.SpotEliAssign (eli, _spot)) {// prova ad assegnare l'elicottero
                    WinI.InsertSomeText ("DISPOSER: movimentato elicottero : "
                    + eli.IdEli + " su spot: " + eli.Spotrequested.SpotPos);
                    }
                } else
                    WinI.InsertSomeText ("DISPOSER: movimentazione elicottero " + eli.IdEli + " non effettuabile");
        } //termine IF
    } // termin foreach elicottero ciclo
}

//-----------------------------------------------------------------------------
//-- controllo eli in hangar e moviementazione sul ponte ------------------//
// è necessario controllare se gli eli assegnati sugli spot sono superiori al
// numero di spot effettivamente disponibile, in tal caso gli eli devono essere riassegnati

public int ElisSpotAutoHangarChangeNumber (SpotManager SpotM, InfoWindow WinI)
{

    int n = this.EliOnDEck (); // numero di elicotteri pianificati sul ponte
    int k = SpotM.NumSpotEfficient (); // numero di spot efficienti presente
    int diff = (n - k); // qundo positivo indica il numero di elicotteri da ridurre
```

```
    int z = 0; // variabile indice
    int i = 0; // variabile indice
    // se il numero di elicotteri è superiore al nuemro di spot efficienti
    // allora serve ridurre il numero di elicottero sullo spot
    if (diff > 0) { // se la diff è positiva il numero di eli assegnati al ponte supera il numero degli spot disponibili
    WinI.InsertSomeText (" SIMULATORE: auto correzione del numero di elicotteri sul ponte. Decremento di " + diff + " unità");

        // correzione del valore di posizione sui primi i elicotteri trovati
        while (i < diff) {
            if (this.ElicotteriList [z].PosEli == true) {
                this.ElicotteriList [z].PosEli = false; // cambio a false
                i++; // valore trovato incremento indice
            }
            z++; // incremento l'indice
        }
        return k; // restituisce il numero di elicotteri ga gestire manualmente
    }
    return n; // restituisce il numero di elicotteri da gestire manualmente
}
//-------------------------------------------------------------------------------------

//-------------------------------------------------------------------------------------
// assegnazione  degli elicotteri assegnati agli spot
// è necessario assegnare agli spot gli elicotteri che di default sono stati posizionati sugli spot
public void EliAutoSpotAssign (SpotManager SpotM, InfoWindow winI, Spot.day_night DN)
{
    // cambia automanticamente il numero di elicotteri assegnati sul ponte
    // la funzione assicura che il nuemero di eli è inferiore o uguale agli spot disponibili
    // e restituisce il numero di elicotteri da gestire  per l'assegnazione automatica degli spot
    int manualAssign = this.ElisSpotAutoHangarChangeNumber (SpotM, winI);

    // una volta determinato il numero di elicotteri da assegnare procedo con l'assegnazione
    if (manualAssign > 0) { // se il numero di assegnazioni è maggiore di 0 creo la finestra per le assegnazioni

        foreach (Elicottero eli in this.ElicotteriList) {
            if (eli.PosEli) { // se l'elicottero è posizionato sul ponte di default cerco di assegnarlo ad uno spot
                Spot spot = SpotM.SpotToHostaCat (eli.CatEli, DN);
                if (spot != null) {
                    winI.InsertSomeText ("SIMULATOR: Spot trovato per assegnazione AUTOMATICA: " + spot.SpotPos);
                    eli.DirToGo = Elicottero.Direction.hp; // direzione assegnata all'eli sul ponte holding point
                    eli.SpotAssigned = spot; // spot assegnato
                    eli.PosEli = true;   // posizione elicottero sul ponte
                    spot.IsReservd = true; // lo spot viene riservato
                    spot.eliWhoReservd = eli; // l'elicottero che ha richiesto il reserve
                    spot.Eli = eli; // assegno l'elicottero
                    spot.SpotOccupied = true; // lo spot e' occupato
                }
            }
        }
    }
} // fine metodo
//-------------------------------------------------------------------------------------

// metodo di supporto alla classe serve per attuare l'inizializzazione delle macchine,
// controllo se gli elicotteri proposti sono tutti necessari per l'operazione
// se ad esempio ho 3 elicotteriche possono trasportarte 10 truppe per un totate di
// 30 persone mentre ho solo 15 persone disponibili gli elciotteri necessari saranno
// solo 2. il terzo quindi risulterà non impiegabile
// tuttavia va controllato anche che l'elicottero risulti non necessario anche per
// il trasporto del cargo
public void Check_Eli_Usability (InfoWindow WinI,TroopsManager TroopM, CargoManager CargoM)
{

    WinI.InsertSomeText ("SIMULATOR : controllo effettiva NECESSITA' degli elicotteri inseriti per la missione");
    int TroopTotW =    TroopM.TroopList.Count * TroopM.SingleSoldierW; // peso totale delle truppe
    int EliTotW = this.ElicotteriList.Sum (x => x.WTroopsLeftOnBoard); // peso totale disponibile sugli elicotteri
    int diffT_E = EliTotW- TroopTotW; // calcolo la differenza tra il peso totale trsportabile e il peso totale da trasportare
    List<Cargo> SupportCargoList = new List<Cargo> (); // lista cargo di supporto
    Elicottero eliMinW = this.ElicotteriList.Find (y => y.WTroopsLeftOnBoard == (this.ElicotteriList.Min (x => x.WTroopsLeftOnBoard)));
    // elicottero con peso minimo disponibile nella lista elicotteri

    // l'algoritmo elimina tutti gli elicotteri che offrono meno peso per il trasporto
    // dopo che viene definita la quantità di peso totale necessaria
    while (diffT_E > 0 && diffT_E> eliMinW.WTroopsLeftOnBoard) {

        bool found = false; // bool di supporto al metodo
        int supportCounter = 0; // bool var di supporto

        if (SupportCargoList.Count > 0)
            SupportCargoList.Clear (); // reset list

        if (diffT_E > eliMinW.WTroopsLeftOnBoard) { //se la differenza tra il peso totale delle truppe
            // trasportabile ed il peso totale disponibile sugli elicotteri
            // è maggiore del peso disponibile sull'elicottero con peso trasportabile minimo allora ho trovato un
            // elicottero candidato

            this.SupportEliList.Add (eliMinW); // aggiungo l'elciottero alla lista
                                               //di supporto come candidato all'eliminazione

            // devo ora controllare che l'elicottero sia anche indispensabile per l'attività di trasporto cargo
            if (CargoM != null && CargoM.CargoList != null) {

                // estrapolo elenco cargo che puo' ospitare l'elicottero candidato
                foreach (Cargo cargo in CargoM.CargoList.FindAll(x => (x.ChecktotalCargoWeight() <= eliMinW.WCargoLeftOnBoard))) {
                    SupportCargoList.Add (cargo); // inserisco i valori trovati all'interno della lista di supporto
                }

                // DEVO CONTROLLARE
                // 1 CHE NON POSSA TRASPORTARE NESSUN CARGO
                // 2 CHE QUALORA POSSA TRASPORTARE DEL CARGO QUESTO NON SIA TRASPORTABILE DA ALTRI ELICOTTERI
```

```
                    // SE UNA O L'ALTRA CONDIZIONE SONO VERE L'ELICOTTERO PUO' ESSERE ELIMINATO DALL'ELENCO
                    if (SupportCargoList.Count > 0) { // controllo che non possa trasportare nessun cargo

                        // conrtollo tutti i cargo presenti nell'elenco in quanto devono essere tutti
                        // trasportabili da altri elicotteri
                        foreach (Cargo cargo in SupportCargoList) {
                            found = false; // resetto la variabile bool di supporto

                            // per ogni altro elicotteo NON MINIMO PESO effettuo il confronto con il cargo
                            foreach (Elicottero eli in this.ElicotteriList.FindAll(x=> x != eliMinW)) {
                                if (eli.WCargoLeftOnBoard >= cargo.ChecktotalCargoWeight ()) { // se esiste un elicottero che puo' imbarcare
il cargo

                                    //incremento il contatore, anche gli altri cargo nell'elenco devono essere inclusi nella condizione
                                    found = true;
                                }

                            } // end loop su eli

                            // se ho trovato almeno un elemento in grado di accettare il cargo la var bool found è settata su vero
                            // quindi posso incrementare il counter
                            if (found)
                                supportCounter++; // se lo trova il cargo nella lista cargo di supporto
                            //che poteva trasportare l'eliminW ora lo puo' trasportare qualun altro

                        } // end loop su cargo

                        // se il numero degli elicotteri che possono ospitare il cargo eguaglia il numero del cargo da ospitare
                        //l'elicottero non è necessario
                        if (supportCounter == SupportCargoList.Count) {

                            // posso eliminare l'elicottero dall'elenco elicotteri
                            this.ElicotteriList.Remove (eliMinW);
                            WinI.InsertSomeText ("SIMULATOR : rimosso elicottero: "+ eliMinW.IdEli + " l'eli risulta non necessario");
                        } else
                            // altrimenti devo rimuoverlo dall'elenco dei candidati
                            this.SupportEliList.Remove (eliMinW);
                    }// end if cargo count >0 ELICOTTERO INUTILE PER L'OPERAZIONE ELIMINARE ELICOTTERO DALL'ELENCO

                    else {
                        // elicottero inutile per l'operazione
                        this.ElicotteriList.Remove (eliMinW);
                        // l'elicottero viene cosi lasciato nell'elenco
                        // degli elicotteri di supporto non utilizzabili
                        WinI.InsertSomeText ("SIMULATOR : rimosso elicottero: "+ eliMinW.IdEli + " l'eli risulta non necessario");
                    }

                } // fine if CARGO NULL INESISTENTE

                else {
                    // elicottero inutile per l'operazione
                    this.ElicotteriList.Remove (eliMinW);
                    WinI.InsertSomeText ("SIMULATOR : rimosso elicottero: "+ eliMinW.IdEli + " l'eli risulta non necessario");
                }
            } // fine if differenza l'elicottero è utile per l'operazione

            // update delle variabili
            EliTotW = this.ElicotteriList.Sum (x => x.WTroopsLeftOnBoard); // peso totale disponibile sugli elicotteri
            diffT_E = EliTotW- TroopTotW; // calcolo la differenza tra il peso totale trsportabile e il peso totale da trasportare
            eliMinW = this.ElicotteriList.Find (y => y.WTroopsLeftOnBoard ==
                (this.ElicotteriList.Min (x => x.WTroopsLeftOnBoard))); //determino il nuovo min
        }// fine while
    } // fine metodo Check_Eli_Usability

    // effettuo il controllo carburante per ogni elicottero
    // IList carburante deve essere necessario alla missione
    // altrimenti devo aumentare il carburante e ridurre peso per le truppe
    public void CheckEliDataFuelConsistency(int dist)
    {
        float totalCarbRequired; // il carburante richiesto
        foreach (Elicottero eli in this.ElicotteriList) {
            // controllo il carburante necessario alla missione
            // il valore viene calcolato con una velocità media di 100 nodi
            // e viene sommato al risultato 20 minuti di volo.

            eli.DistancetoRun = dist; // resetto la run distance dell'elicottero
            totalCarbRequired =  eli.Fuelrequired(this.initEliNum); //calcolo il carburante necessario all'operazione
            totalCarbRequired = totalCarbRequired + 50; // assegno 50 Kg in piu' per lo start iniziale
            if (eli.Fuel < totalCarbRequired) { // se il carburante all'interno dell'elicottero NON
                // è sufficiente devo incrementasre il carburante levando peso alle truppe e al cargo

                float diffF = totalCarbRequired - eli.Fuel;// differenza di carburante che devo ancora inserire
                float freeW = eli.MaxTOLoad-eli.ToTWMaxFull (); // peso libero PESO MASSIMO AL DECOLLO - MESO A PIENO CARICO IMPOSTO UTENTE
                //  devo prima verificare che il carburante possa essere inferito in elicottero senza
                // diminuzione del carico di truppe o cargo
                if (freeW >= diffF) {
                    float fuelN = eli.Fuel + diffF; // correggo il fuel prendendolo tutto dal peso libero non occupato a bordo
                    eli.Fuel = (fuelN);
                } else {

                    float fuelK = eli.Fuel + freeW;// correggo il fuel prendnolo in parte dal peso libero non occupato a bordo
                    eli.Fuel= (fuelK); // nuovo fuel = vecchio fuel + la differenza di peso libero ancora imbarcabile
                    diffF = diffF - freeW; // differenza in peso di carburante+ ancora da imbarcaree

             if (diffF <= (eli.WTroopsLeftOnBoard + eli.WCargoLeftOnBoard)) //se posso inserire il resto del carburante al posto delle truppe
                    {    // applico la correzione
                    // elimino il peso dalle truppe e dal cargo proporzionalmente

                float p1 = ((float)(eli.WTroopsLeftOnBoard) /(float) (eli.WCargoLeftOnBoard + eli.WTroopsLeftOnBoard)); // percentuale peso truppe

                float p2 = ((float)(eli.WCargoLeftOnBoard) /(float) (eli.WCargoLeftOnBoard + eli.WTroopsLeftOnBoard)); // percentuale cargo
```

```
                        // sottraggo peso all'eli in percentuale
                        eli.WTroopsLeftOnBoard = eli.WTroopsLeftOnBoard - (int) (p1 * (int)diffF);
                        eli.WCargoLeftOnBoard = eli.WCargoLeftOnBoard - (int)(p2 * (int)diffF);

                        float fuelZ = eli.Fuel + diffF;
                        eli.Fuel= (fuelZ); // rieffettuo la distribuzione dei pesi di carburante
                    } else
                    {
                        // la missione non è effettuabile per l'elicottero
                        // setto le flag mettere l'elicottero fuori uso e lasciarlo in hangar
                        eli.IsEfficient = false;
                        eli.isRequiredForOP = false;
                        eli.PosEli = false;
                        eli.IsRunning = false;
                        eli.IsBladeSpread = false;
                    }}
                }
            }
        } // end method CheckEliDataFuelConsistency


        // correggo i pesi degli elicotteri per evitare spezzoni
        // se il peso di una singola truppa è 80 kg ma il mio elicottero puo' trasportare 90 K di truppe
        // in realtà il peso trasportabile è sempre 80 l'eli necessita di una correzione dei multipli
        public void TroopsW_Data_correction (int singleW)
        {
            foreach (Elicottero eli in this.ElicotteriList)
            {
                int W;

                W = eli.WTroopsLeftOnBoard / singleW; // numero di soldati
                                                      // l'arrotondamento è sempre all'intero minore
                eli.WTroopsLeftOnBoard = W * singleW;

            }
        }  // fine metodo TroopsW_Data_correction


        // controllo DEGLI STATI  lista elicotteri
        // controllo LO STATO DELL'HANGAR
        // CHE PUO' ESSERE PIENO, VUOTO O IN UNO STATO INTERMEDIO
        public override void CheckState ()
        {

            // rimuovo gli leementi inefficienti o non richiesti
            ElicotteriList.RemoveAll (x => x.IsEfficient == false); // rimuovo gli elementi inefficienti
            ElicotteriList.RemoveAll (x => x.isRequiredForOP == false); // rimuovo gli elementi non required

            int eliHangar = this.EliInHangar (); // elicotteri flaggati sul ponte richiesti per op efficienti
            // che non hanno riservato uno spot, che non sia voltante, che non abbia uno spot assegnato o riservato


            int eliHasSpotorFly = this.ElicotteriList.FindAll (x =>( x.PosEli==true || x.IsFlying==true || x.hasSpotReserved==true)).Count;
            int toteli = this.ElicotteriList.Count;
            int eliHangarUsable = toteli - eliHasSpotorFly;


            if (eliHangar == 0) {    // se gli elicotteri in hangar sono 0
                this.Stato = StatusManager.Empty;    // lo stato è vuoto, l'hangar non contiene elicotteri efficienti
                this.Action = ActionManager.ReadyToReceive; // l'hangar è pronto a ricevere
            }

            if ((eliHangar> 0) && (eliHasSpotorFly==0)) {//tutti gli eli sono in hangar se il numero di quelli che vola o è sullo spot è 0
                this.Stato = StatusManager.Full;        // lo stato risulta pieno
                this.Action = ActionManager.ReadyToSend;// l'hangar puo' solo inviare elicotteri
            }

            if ((eliHangar > 0 ) && (eliHasSpotorFly>0)) {// se eli in hangar non sono tutto ma piu' di 0
                this.Stato = StatusManager.CanAll;    // lo stato è promiscuo, l'hangar non è pieno o vuoto
                this.Action = ActionManager.ReadyToSendAndReceive; // è possibile qualcunque azione di invio o ricezione sul ponte
            }

        }
        #endregion
    } //---------------------------------------------------- fine classe
}
//---------------------------------------------------- fine namespace
```

6.17 FILE CARGOMANAGER

```
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// classe manager per la creazione del cargo ed il controllo degli stati
// detiene la lista cargo e le operazioni su di essi
// classe singleton
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


using System;
using System.Collections.Generic;

namespace EliAssaltoAnfibio
{
    public class CargoManager : AbstractManager
    {

        public static CargoManager instance = null;
        private int _numCargo;// indica il numero di record iniziali
        public List<Cargo> CargoList; // definizione della lista di elicotteri ( pubblica )
        public StatusManager Status;// definizioni dello stato del manager
        public ActionManager Action; //definizioni delle azioni svolte
        public const int CargoLoadTiming = 5; // tempo per operazioni di cargo load
        public const int CargoTimeTodisembark = 4; // tempo richiesto per disimbarcare il cargo
        public InfoWindow WinI;


        public CargoManager(int numCargo, InfoWindow winI) : base("Manager_per_Cargo")
        {
            this.WinI = winI;
            this._numCargo=numCargo;
            this.Status=StatusManager.Empty;  // setta lo stato come vuoto
            this.Action=ActionManager.Wait; // le azioni sono in attesa di informazioni aggiuntive

        }

        // istanza singleton è prevista l'esistenza di un'unica finestra
        public static CargoManager Instance(int numCargo, InfoWindow winI)
        {
            if (instance == null) instance= new CargoManager( numCargo ,  winI );
            return instance; // ritorno il costruttore
        }

        // definizione dei metodi comuni alle classi in OVERRIDING

        public override void MakeList(int i )
        {
            this.CargoList = new List<Cargo>(i);
        }

        // METODI INSERIMENTO - RIMOZIONE LISTA
        // inserisci un elemento i nella lista
        public override void InsertElement(Object i)
        {
                this.CargoList.Add ((Cargo)i);
        }


        //rimuovi un elemento dalla lista REMOVEAT INDEX
        public override  void RemoveElement(int i)
        {
            this.CargoList.RemoveAt (i);
        }


        // inserisci elemento nella lista
        public  void AddElementTolist (List<Cargo> lstobj, Cargo obj)
        {

            lstobj.Add (obj);

        }
        // elimina elemento dalla lista
        public  void RemoveElementTolist (List<Cargo> lstobj, Cargo obj)
        {
            lstobj.Remove (obj);

        }


        public  void InsertElementCargo (int cargoRec ,string cargoType, int cargoW , int cargoP )
        {

                 Cargo cargoN = new Cargo( cargoRec ,cargoType, cargoW , cargoP ); // creazione del nuovo record spot
                CargoList.Insert ( (cargoRec), cargoN);
        }

        // movimentazione del cargo ( sorgente , destinazione , cargo)
        public  void MoveElement (List<Cargo> list1src, List<Cargo> list2dst, Cargo cargo)
        {
            list2dst.Add (cargo);// aggiungo l'elicottero alla seconda lista
            list1src.Remove(cargo); // sottraggo l'elicottero alla lista iniziale
        }
        // FINE METODI INSERIMENTO RIMOZIONE



        // ------------------------------------------------------------------------------------------
        // list disposer methods
```

```
// distribuzione del cargo sull'elicottero che lo puo' ospitare
// il metodo serve per distribuire il cargo a bordo degli elicotteri
// che possono effettuare il trasporto
// blocca temporalmente l'elicottero che effettua il trasferimento di cargo
public void CargoDistribution (InfoWindow winI , EliManager EliM , TroopsManager TroopM)
{
    // variabili d'impiego locale
    int i = 0; // index
    int maxCargoNum = this.CargoList.Count; // quanti pacchi di cargo devo distribuire
    Cargo cargo = null;
    Elicottero eli = null;

    if (this.CargoList != null && this.CargoList.Count!=0) {// controlla che la lista cargo contenga elementi


        // è stato usato un indice per l'impossibilità di operare sulla lista dinamicamente con un foreach
        // cancellando i dati e leggendoli insieme.
        while ((i < this.CargoList.Count)) {// continua fino a quando l'indice raggiunge il valore massimo di elementi
            cargo = this.CargoList [i];// cargo da assegnare



            if (!cargo.IsEliC   // il cargo vede essere caricabile su uno degli elicotteri
                && cargo.IsFittable)  // controlla se il cargo non è ancora stato assegnato
            {
                eli = EliM.EliAbleToCargo (cargo, TroopM);// definisce l'elicottero in grado di ospitare il cargo
                //l'elicottero per ospitare il cargo non deve essere necessariamente sul ponte di volo


                // controlla che l'elicottero assegnato esista e se esistono le condizioni per poter accettare il cargo
                if (eli != null && !eli.IsCargoFull // elicottero non vede essere cargo full
                    && !eli.IsFlying // non deve essere in volo
                    && !eli.IsBlocked // no deve essere time blocked
                    && eli.IsEfficient  // deve essere efficiente
                    && eli.isRequiredForOP)  // deve essere richiesto per l'operazione
                {

  eli.IsBoardingCargo = true; // eli inizia ad imbarcare il cargo

  eli.WCargoLeftOnBoard = (eli.WCargoLeftOnBoard - cargo.ChecktotalCargoWeight());// sottraggo il peso del cargo al carico utile dell'eli
                    // tale peso comprende le truppe che fanno parte del cargo

  this.MoveElement (this.CargoList, eli.EliCargoList, cargo); // effettua lo spostamento necessario setto la flag del cargo a bordo

  this.CheckThisEliCargoFull (eli); // controlla se esistono altri cargo che possono essere
                    //inseriti altrimenti CARGO FULL set flag

  cargo.IsEliC = true; // la flag del cargo viene settata in elicottero
  cargo.Eli = eli;// assegno al cargo l'elicottero su cui viaggia

  eli.EliBlock (CargoManager.CargoLoadTiming,5 ); // blocco l'elicottero N ticks per operazioni di cargo
                    // al termine dell'attesa IsCargoOnBoard variable e' TRUE

  winI.InsertSomeText ("CARGO DISPOSER: è stato assegnato: " + cargo.CargoString +
                       " , restano ancora " + this.CargoList.Count + " elementi CARGO da imbarcare");
                    i++; // passo al rec ord successivo
                }
                else i++;// incremento il contatore del cargo, cambio cargo

            }
        }// fine while
    } else
        winI.InsertSomeText ("TROOPS DISPOSER: nessuna lista di CARGO è stata definita");

}// NotFiniteNumberException metodo distribuzione cargo

// il metodo controlla che esista ancora un cargo che possa essere inserto in elicottero
// il metodo va riproposto dopo ogni inserimento di cargo
// nonchè dopo ogni scarico
public void CheckEli_CargoFull (EliManager EliM)
{
    foreach (Elicottero eli in EliM.ElicotteriList)
    {

        eli.IsCargoFull = true;

        if (eli.isRequiredForOP && eli.IsEfficient ) // se l'elicottero è efficiente e richiesto per l'op
            foreach (Cargo cargo in this.CargoList)  // controlla che esista ancora un cargo utile
            {
                if ( (cargo.ChecktotalCargoWeight() < eli.WCargoLeftOnBoard ) && !cargo.IsEliC )
                    eli.IsCargoFull = false;
            }
    }
    this.CheckState (); // aggiorno lo stato
}

// controllo il singolo Eli per verificare se è possibile impiegarlo per altri carghi
public void CheckThisEliCargoFull (Elicottero eli)
{
    eli.IsCargoFull = true;

    foreach (Cargo cargo in this.CargoList)
    {
        if ( (cargo.ChecktotalCargoWeight() < eli.WCargoLeftOnBoard ) && !cargo.IsEliC )
            eli.IsCargoFull = false;
    }

}
```

```
        // controllo che ogni cargo sia trasportabile da almeno 1 elicottero
        // altrimenti il cargo è unfittable
        public void CheckCargoIsFittable (EliManager EliM)
        {
            foreach (Cargo cargo in this.CargoList) // per ogni cargo
            {
                foreach (Elicottero eli in EliM.ElicotteriList) {  // per ogni elicottero

                    // nota le truppe sono inserite come CARGO e quindi fanno parte della pesata totale del cargo
                    // la classe cargo è infatti composta da una lista di truppe assegnate aventi come peso
                    // il peso basico del soldato inserio per i soldati
                    if (eli.WCargoLeftOnBoard > cargo.ChecktotalCargoWeight ())
                    {
                        cargo.IsFittable = true; // switch cargo flag
                        return;
                    }
                }
                cargo.IsFittable = false;
            }
        }


        // stato lista elicotteri
        public override void CheckState ()
        {
        int cargoNotFill = this.CargoList.FindAll (cargo => cargo.IsFittable == false).Count; // conta il numero di cargo che non puo' essere imbarcato
            int cargoONboard = this.CargoList.FindAll (cargo => cargo.IsEliC == true).Count; // conta il numero di cargo gia' imbarcato
            int totalCargo = this.CargoList.Count;

            // se ho N carghi di cui K sono gia' a bordo e I sono inutlizzabili
            // se N==0 l'hangar è vuoto di cargo
            if ((totalCargo-cargoONboard-cargoNotFill)== 0) { // se il cargo in hangar è 0
                    this.Status = StatusManager.Empty;// lo stato è vuoto
                    this.Action = ActionManager.ReadyToReceive;// l'hangar è pronto a ricevere
                }
            // se N è maggiore di 0 l'hangar ha ancora del Cargo da movimentare
            if ((totalCargo-cargoONboard-cargoNotFill)> 0) {// se il cargo in hangar è maggiore di 0
                    this.Status = StatusManager.CanAll;// lo stato è promiscuo, il restante cargo deve ancora essere distribuito
                    this.Action = ActionManager.ReadyToSendAndReceive; // è possibile qualcunque azione di invio o ricezione sul ponte
                }

        }

        // eliste almeno un cargo in grado di trasportare le truppe sull'elicottero
        // restituisce vero se esiste almeno un cargo che è possibile caricare sull'elicottero
        public bool CheckCargoLoadOnEli (Elicottero eli, int singleSoldW)
        {
            foreach (Cargo cargo in this.CargoList)
                if ((cargo.CargoW <= eli.WCargoLeftOnBoard ) && ( cargo.CargoP*singleSoldW <= eli.WTroopsLeftOnBoard))
                {
                    return true ;
                }
                return false;
        }

    }// fine classe
}// fine name space
```

6.18 File SpotManager

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// classe manager per la creazione spot ed il controllo degli stati
// detiene la lista spot e le operazioni su di essi
// classe singleton
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


using System;
using System.Collections.Generic;

namespace EliAssaltoAnfibio
{
    public class SpotManager: AbstractManager
    {
        public static SpotManager instance = null; // classe singleton
        public int TotSpot {set;get;} // numero totale di spot
        public List<Spot> Spotlist; // definizione della lista di elicotteri ( pubblica )
        public StatusManager Stato;// definizioni dello stato del manager
        public ActionManager Action; //definizioni delle azioni svolte
        public InfoWindow WinI; // finestra informativa
        public const int timeNeedToEngage = 10; //  minuti per la messa in moto dell'elicottero
        public const int timeNeedToMoveToDeck = 15; // 15 minuti per muove l'elicottero sul ponte dall'hangar

        public SpotManager (int SpotNum, InfoWindow winI): base("Manager_per_Spot")
        {
            this.WinI = InfoWindow.Instance();
            this.TotSpot = SpotNum;
            this.Stato = StatusManager.Empty;  // setta lo stato come vuoto
            this.Action = ActionManager.Wait; // le azioni sono in attesa di informazioni aggiuntive

        }


        //istanza singleton è prevista l'esistenza di un'unica finestra
        public static SpotManager Instance(int SpotNum, InfoWindow winI)
        {
            if (instance == null) instance= new SpotManager( SpotNum ,  winI );
            return instance; // ritorno il costruttore
```

```
}


// override crea la lista di spot per la gestione
public override void MakeList (int elementi)
{
    this.Spotlist = new List<Spot> (elementi);
}

// inserisci un elemento nella lista
public override void InsertElement (object spot)
{
    this.Spotlist.Add ((Spot)spot);
}

//rimuovi un elemento dalla lista REMOVEAT INDEX
public override  void RemoveElement (int i)
{
    this.Spotlist.RemoveAt (i);
}


// inserisci lo spot nell'elenco degli spot
public  void InsertElementSpot (int spotPos, bool spotRunnable, bool cat1D, bool cat2D, bool cat3D, bool cat1N, bool cat2N, bool cat3N)
{
    // creazione del nuovo elicottero con valori inizializzati
    Spot spotN = new Spot (spotPos, spotRunnable, cat1D, cat2D, cat3D, cat1N, cat2N, cat3N); // creazione del nuovo record spot
    // inserisco l'elemento creato nella lista
    Spotlist.Insert ((spotPos), spotN);
}

// restituisce il primo spot efficiente in grado di ospitare una determinata categoria di elicottero
// se non ci sono spot efficienti allora viene restituito il valore nullo
public Spot SpotToHostaCat (int catEli, Spot.day_night _dn)
{

    if (this.Spotlist.Count > 0) { // controllo se il numero di spot presenti è maggiore di 0
        foreach (Spot _spot in this.Spotlist) {
            if (_spot.SpotRunnable
                && (!_spot.SpotOccupied)
                && !_spot.IsReservd) { // controllo se lo spot trovat è efficiente e non è gia occupato

                if (_dn == Spot.day_night.day) { // controllo le cateogire accettate di giorno
                    if (((catEli == 1) && (_spot.DayCanAccept.Cat1 == true))
                        || ((catEli == 2) && (_spot.DayCanAccept.Cat2 == true))
                        || ((catEli == 3) && (_spot.DayCanAccept.Cat3 == true)))
                        return _spot;
                } else {// qua la categoria deve essere per forza notturna
                    if (((catEli == 1) && (_spot.NightCanAccept.Cat1 == true))
                        || ((catEli == 2) && (_spot.NightCanAccept.Cat2 == true))
                        || ((catEli == 3) && (_spot.NightCanAccept.Cat3 == true)))
                        return _spot;
                }
            }
        }
    }
    return null;
}// fine metodo SpotToHostaCat


// resittuisce il numero di spot attualmente efficienti
public int  NumSpotEfficient ()
{
    int n = 0;
    foreach (Spot _spot in this.Spotlist)
        if (_spot.SpotRunnable == true)
            n++;
    return n;
}

// assegno un elicottero allo spot ed effettua il blocco
public bool SpotEliAssign (Elicottero eli, Spot spot)
{

    if (spot != null
        && eli != null
        && !spot.IsReservd
        && !eli.IsBlocked) {    // spot ed eli devono essere diversi da null

        spot.IsReservd = true;// spot riservato
        spot.eliWhoReservd = eli; // elicottero riservante
        eli.Spotrequested = spot;// assegna lo spot all'elicottero
        eli.hasSpotReserved = true; // l'elicottero riserva un posto

        eli.EliBlock(SpotManager.timeNeedToMoveToDeck, 3); // elicottero bloccato per attività di movimentazione per 15 primi
        // al termine del blocco eli.PosEli = true indica che l'elicottero ha assunto la posizione sul ponte
        return true;// assegnazione avvenuta con successo
    }
    else return false;// assegnazione non avvenuta
}

// restituisce il peso totale disponibile per imbarcare le truppe
// calcolato con gli elciotteri attualmente presenti sullo spot
public int TotalWEliSpotForTroopS ()
{
    int totalW = 0;

    foreach (Spot spot in this.Spotlist) {
        // controllo che lo spot sia occupato da un elicottero efficiente
        // che abbia una configurazione di pale aperte
```

```
        // che sia con i motori in moto
        // che non sia in uno stato di pieno e non possa piu' imbarcare altro
        if (spot.SpotOccupied
            && !spot.Eli.IsBlocked
            && spot.Eli.IsBladeSpread
            && spot.Eli.IsRunning
             && !spot.Eli.IsFull)

            totalW = totalW + spot.Eli.WTroopsLeftOnBoard;
    }
    return totalW;

}


// fornisce il numero di elicotteri attualmente presenti sul ponte
public int TotalElionDeck ()
{
    int n = 0;// indicatore numero di elicotteri

    foreach (Spot spot in this.Spotlist) {
        if (spot.SpotOccupied && spot.Eli.IsEfficient)
            n++; // incrementa se lo spot e occupato da un eli efficiente
    }
    return n;

}

// tutti gli elicotteri presenti sul ponte mettono in moto i motori
// la messa in moto è propedeutica per l'apertura delle pale
public void AllEliSpottedRunning ()
{
    // metto in moto gli elicotteri sul ponte che
    foreach (Spot spot in this.Spotlist)
        if (spot.Eli!=null
            && spot.Eli.PosEli  //sono sul ponte
            && spot.SpotOccupied // lo spot è occupato
            && !spot.Eli.IsBlocked // eli non bloccato
            && !spot.Eli.IsRunning // eli non running
            && !spot.Eli.IsBladeSpread   // eli non a pale paerte
            && !spot.Eli.IsFlying // eli non volante
        )
        { // controlla che l'elicottero sullo spot sia efficiente e non blocked status, che non sia già in moto
            // e che sia required per l'operazione.
            spot.Eli.EliBlock (SpotManager.timeNeedToEngage, 1); // metto l'elicottero in blocco per la messa in moto
                                                    // il thread cambierà la variabile running alla fine dell'esecuzione

        }
}


//  gli elicotteri presenti sul ponte aprono le pale
public void AllEliBladeSpreading ()
{
    foreach (Spot spot in this.Spotlist)
        // controlla che l'elicottero non sia in stato di blocco, che sia efficiente e che le pale non siano gia aperte
        if ( spot.Eli!=null && !spot.Eli.IsBlocked &&  !spot.Eli.IsBladeSpread && spot.Eli.IsRunning && !spot.Eli.IsFlying )
        {
            spot.Eli.EliBlock (StaticSimulator.BladeSpreadBlockingTime, 4); // blocco l'eli segnale 10

        }
}


// gli elicotteri decollano ed assumono la velocità di crociera indicata
public void AllEliTakingOff(float speed)
{
    foreach (Spot spot in this.Spotlist)
        // controllo che lo spot abbia un elicottero funzionanete a pale aperte e a motore acceso
        if (spot.Eli!=null
            && !spot.Eli.IsBlocked
            && spot.Eli.isREADYstatus
            && !spot.Eli.IsFlying)
        {
            // setto le flags
            spot.Eli.isApproaching = false; // setto l'elicottero non in apporch alla dest
            spot.Eli.IsFlying=true;// viene settata la flag di elicottero in volo
            spot.eliWhoReservd = null; // eli che ha effettuato la riserva
            spot.SpotOccupied = false; // lo spost non è piu' occupato
            spot.IsReservd = false; // fine riserva
            spot.Eli.SpotAssigned = null; // fine spot assegnato
            spot.Eli.hasSpotReserved = false; // l'eli ha lo spot riservato
            spot.Eli.isOnDestination = false; // resetto la var di destinazione
            spot.Eli.isREADYstatus = false; // reimposta lo stato di pronto al decollo su falso in quanto è deocllato
            spot.Eli.DirToGo = Elicottero.Direction.hp; // l'elicottero assume destinazione HP
            spot.Eli.EliSpeed = speed; // impost la velocità dell'elicottero a 1

            // salvo le informazioni prima di liberare lo spot
            WinI.InsertEvent ( spot.Eli.current_time , " T/O SHIP ", spot.Eli.IdEli, spot.Eli.NumEli);
            WinI.InsertSomeText( " ELI " +spot.Eli.IdEli + " DECOLLA time: "+ spot.Eli.current_time.ToString() );
            //-----------------------------------------------------
            spot.Eli = null; // libera l'elicottero elicottero assegnato spot
        }
}

// indica il numero di spot liberi e funzionali
public int TotalSpotFree ()
{
    int n = 0;                                      // indicatore numero di elicotteri
    foreach (Spot spot in this.Spotlist) {
```

```
        if (!spot.SpotOccupied && spot.SpotRunnable)
            n++; }
    return n;                                       // restituisce il numero di spot

}// fine metodo

// restituisce gli spot inefficienti
public int TotalSpotInefficient ()
{
    int n = 0;

    foreach (Spot spot in this.Spotlist) {
        if (!spot.SpotRunnable)
            n++; }
    return n;
} // fine metodo


// restituisce il numero di elicotteri attualmente bloccati temporalmente sul ponte
// gli elicotteri devono essere pronti per l'imbarco truppe
public int TotalEli_NOT_Blocked_on_Deck ()
{
    int n = 0;

    foreach (Spot spot in this.Spotlist) {
        if (spot.Eli!=null
            && !spot.Eli.IsBlocked // se l'elicottero non è bloccato
            && !spot.Eli.hasSpotReserved // se l'eli non ha spot riservato
            && spot.Eli.IsRunning // eli running
            && spot.Eli.IsBladeSpread  // eli blade spread
            && spot.Eli.isRequiredForOP) // eli required
            n++; }
    return n;
} // fine metodo

// serve per calcolare quanti elicotteri ho pronti sul ponte per il decollo
public void EliReadyStatus()
{
    foreach (Spot spot in this.Spotlist)
        if (spot.Eli != null && !spot.Eli.IsBlocked
            && spot.Eli.IsBladeSpread && spot.Eli.IsRunning
            && !spot.Eli.IsFlying// se l'elicottero è pronto
            && !spot.Eli.isHotRef// eli non hot ref
            && ((spot.Eli.EliSoldierList.Count > 0) // eli con truppe
                || (spot.Eli.EliCargoList.Count > 0)) // oppure cargo
            && spot.Eli.Go_goback// elicottero in direzione LZ
            && !spot.Eli.LowFuel// controllo che non l'elicottero non abbia lowfuel prima di decollare
        ) // ed ha truppe o cargo
        {
            spot.Eli.isREADYstatus = true;// stato di pronti al decollo viene assunto solo se l'elicottero è
        }
}// fine metodo

// metodo di supporto serve a controllare che gli elicotteri sul ponte
// non siano in uno stato di fuel prima della partenza che richiedere rifornimento
// se lo sono devo effettuare rifornimento
public void CheckHOTREFLowFuelState (int numEli)
{
    foreach (Spot spot in this.Spotlist)
        if (spot.Eli != null
            && spot.Eli.PosEli
            && spot.SpotOccupied
            && spot.Eli.EliSoldierList.Count==0 // le truppe devono essere 0 a bordo per l'hot ref
            && !spot.Eli.IsBlocked // elicottero non deve essere bloccato
            && spot.Eli.IsBladeSpread// elicottrero a pale apertr
            //    && (spot.Eli.LowFuel )
            && (spot.Eli.Fuel<spot.Eli.Fuelrequired(numEli))
            && spot.Eli.IsRunning
            && spot.Eli.isRequiredForOP// l'elicottero è running
            && !spot.Eli.IsFlying)

         { // l'elicottero non deve volare

            spot.Eli.isHotRef = true;// eli hot ref status
            spot.Eli.EliBlock (6, 7); // blocco per 6 minuti per carburante
            // al temrine del blocco l'elicottero è rifornito
            // e la flag FuelWarning viene resettata dal fuel_thread
        }
}//fine metodo


//controllo lo status degli spot
// metodo di controllo stato per spot
// FULL = ponte pieno
// EMPTY = ponte vuoto
// CanAll = ponte con elicotteri ma che puo' ancora ospitare
public override void CheckState ()
{
    int spotListCount = this.Spotlist.Count; // spot totali di qualunque tipo
    int ineff = this.TotalSpotInefficient (); // Spot inefficienti
    int used = Spotlist.FindAll (x => x.SpotOccupied == true).Count;// List count spot occupati
    int reserved = Spotlist.FindAll (x => x.SpotOccupied==false && x.IsReservd==true ).Count; // list count spot riservati
    int totaSpotFree=spotListCount-ineff-used-reserved;

    if (used==0 && reserved==0 && (spotListCount>ineff)) { // se gli spot liberi corrispondono a tutti gli spot disponibili allora
            this.Stato = StatusManager.Empty;               // lo stato di ponte è vuoto, non ci sono elicotteri sul ponte
            this.Action = ActionManager.ReadyToReceive;        // il ponte è pronto a ricevere elicotteri
        }
    if ((totaSpotFree > 0) &&  (used>0 || reserved>0)) {                    // se eli in hangar non sono tutto ma piu' di 0
            this.Stato = StatusManager.CanAll;                                      // lo stato è promiscuo il ponte non è nè vuoto nè
```

```
pieno
                this.Action = ActionManager.ReadyToSendAndReceive;                    // è possibile qualcunque azione di invio o ricezi
one sul ponte
            }
            if (totaSpotFree == 0) {                                                  //tutti gli spot sono occupati
                this.Stato = StatusManager.Full;                                      // lo stato risulta pieno
               this.Action = ActionManager.ReadyToSend;                             // l'hangar puo' solo inviare elicotteri
               }
        }// fine metodo
    }// fine classe
}// fine name space
```

6.19 File TroopsManager

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// classe manager per la creazione truppe ed il controllo degli stati
// detiene la lista truppe e le operazioni su di essi
// classe singleton
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

using System;
using System.Collections.Generic;
using Microsoft.Xna;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;

namespace EliAssaltoAnfibio
{
    public class TroopsManager : AbstractManager
    {
        public static  TroopsManager instance= null;

        public List<Soldier> TroopList{set;get;} // definizione della lista di elicotteri ( pubblica )
        public StatusManager Status;// definizioni dello stato del manager
        public int TotalTroops{set;get;}// truppe totali
        public ActionManager Action; //definizioni delle azioni svolte
        public int SingleSoldierW {set;get;} // indica il peso del soldato singolo
        public InfoWindow WinI; // finestra informativa
        public Texture2D Troopstexture; // truppe shape

        // contruttore
        public TroopsManager (int Element, InfoWindow winI):base("Manager_per_Truppe")
        {
            this.WinI = winI;
            this.TotalTroops=Element; // inizializzo gli elementi

        }


        // istanza singleton è prevista l'esistenza di un'unica finestra
        public static TroopsManager Instance(int Element, InfoWindow winI)
        {
            if (instance == null) instance= new TroopsManager( Element ,  winI );
            return instance; // ritorno il costruttore
        }


        // crea la lista fatta di N elementi
        public override void MakeList(int element)
        {
            this.TroopList =new List<Soldier>(element);
        }

        // inserisci un elemento nella lista
        public override void InsertElement (Object troop)

        {
            this.TroopList.Add ((Soldier) troop);
        }

        //rimuovi un elemento dalla lista REMOVEAT INDEX
        public override  void RemoveElement (int i)

        {
            this.TroopList.RemoveAt (i);
        }


        // il meteodo derve per spostare un elemento di truppa su un elicottero
        // ogni elicottero deve quindi possedere la propria lista di soldati
        // ogni elicottero deve possedere la propria lista cargo
        public  void MoveElement (List<Soldier> list1src, List<Soldier> list2dst, Soldier troop)
        {
            list2dst.Add (troop);// aggiungo l'elicottero alla seconda lista
            list1src.Remove(troop); // sottraggo l'elicottero alla lista iniziale
        }


        // controllo stato manager override
        // EMPTY non ci sono piu' truppe
        // CanAll ci sono ancora truppe da sbarcare
        public override void CheckState ()// controllo stato lista manager
        {

            if (this.TroopList.Count== 0) { // se le truppe in hangar sono 0
                    this.Status = StatusManager.Empty;// lo stato è vuoto
                    this.Action = ActionManager.ReadyToReceive;// l'hangar è pronto a ricevere
```

```
                }

            if (this.TroopList.Count > 0) {// se le truppe in hangar sono maggiori di 0
                    this.Status = StatusManager.CanAll;// lo stato è promiscuo
                    this.Action = ActionManager.ReadyToSendAndReceive; // è possibile qualcunque azione di invio o ricezione sul ponte
                }
        }

        // LIST DISPOSER MODULES
        // - controllo il numero di spot occupati
        // - calcolo lo spazio totate sugli eli
        // - distribuisco le truppe equamente sugli elicotteri
        public void EliSpotTroopDistribution (SpotManager SpotM , TroopsManager TroopsM , InfoWindow winI)

        {
            int totWEli=SpotM.TotalWEliSpotForTroopS(); // calcolo il peso totale disponibile dagli elicotteri sugli spot
            int totWTroops = TroopsM.TroopList.Count*TroopsM.SingleSoldierW; // calcolo il peso totale delle truppe da imbarcare
            int w_Embarcable=0;// peso imbarcabile corrispondente al minore tra il peso totale disponibile ed il peso totale truppe
            int troopsE=0;// truppe imbarcabili
            int i=0; // variabile indice

            if (totWTroops<totWEli) w_Embarcable=totWTroops; else w_Embarcable=totWEli; // se il peso totale dell truppe da imbarcare è
            // minore rispetto al peso totale disponibile sugli elicotteri
            // allora le truppe imbarcabili saranno il minimo tra i due valori
            troopsE=w_Embarcable/TroopsM.SingleSoldierW;//trasformo il peso imbarcabile in uomini da imbarcare
            winI.InsertSomeText("TROOPS DISPOSER: ponte di volo occupato da: "+SpotM.TotalElionDeck()
                 + " Elicotteri sul ponte pronti per l'imbarco di "+troopsE +" truppe");

            //trasferisco ogno uomo su ogni elicottero disponibile con una distribuzione alternata
            //su tutti gli elicotteri disponibili, la distribuzione di reitera fino a  a quando tutti gli uomini
            // sono stati spostati sugli elicotteri.
            if (troopsE>0) // se esistono truppe da imbarcare
                while ( i<troopsE) // continua fino a quando l'indice è inferiore alle truppe
                {
                    foreach (Spot spot in SpotM.Spotlist) // per ogni spot
                    {    // se spot occupato da un elicottero in moto e pale aperte e l'elicottero puo' ospitare le truppe, allora effettuo il ca
rico
                        if (spot != null && spot.Eli != null
                            && !spot.Eli.IsBlocked  // eli non bloccato
                            &&!spot.Eli.IsFull  // eli non pieno di cargo e truppe
                            && !spot.Eli.isTroopsFull // eli non pieno di truppe
                            && spot.Eli.IsRunning  // eli in moto
                            && spot.Eli.IsBladeSpread  // eli a pale a perte
                            && (TroopsM.TroopList.Count>0) // eli con truppe a bordo
                            && (spot.Eli.WTroopsLeftOnBoard >= TroopsM.SingleSoldierW)) {// truppe accettabili superano il peso del singolo


                            // scambio di lista,  simulo che la truppa imbarca sull'elicottero sistemato sul ponte,
                            //La lista verra poi passata alla landing zone
                            // per permettere lo sbarco della truppa

                            TroopsM.TroopList [TroopsM.TroopList.Count - 1].IsEli = true; // imposto la truppa in elicottero
                            spot.Eli.IstroopsOnBoard = true; // elicottero ha truppe a bordo
                            try{
                            // sposto la truppa dalla lista truppe a bordo
                            TroopsM.MoveElement (TroopsM.TroopList, spot.Eli.EliSoldierList, TroopsM.TroopList [(TroopsM.TroopList.Count) - 1]);
                            }
                            catch{
                                System.Console.WriteLine ("ERRORE SPOSTAMENTO TRUPPE");
                            }

                            spot.Eli.IsBoarding = true; // l'elicottero sta imbarcando truppe
                            // diminuisco il peso totale disponibile sull'eli
                            spot.Eli.WTroopsLeftOnBoard = (spot.Eli.WTroopsLeftOnBoard - TroopsM.SingleSoldierW);
                            if (spot.Eli.WTroopsLeftOnBoard < TroopsM.SingleSoldierW)
                                spot.Eli.isTroopsFull = true; // modifico la flag di full truppe
                            if (spot.Eli.isTroopsFull && spot.Eli.IsCargoFull)
                                spot.Eli.IsFull = true;; // modifico la flag di full cargo e truppe

                            i++; // VAR INDICE,indica il numero di truppe spostate
                        }}}// fine metodo

            // inserisco attesa elicotteri imbarco truppe se la variaibile is boarding is TRUE
            foreach (Spot spot in SpotM.Spotlist) { // per ogni spot

                if (spot.Eli != null && spot.Eli.IsBoarding) // gestione tempi attesa
                    spot.Eli.EliBlock (StaticSimulator.timeToEmbarcTroops, 8); // blocco l'eli e resetto la variabile isBoarding to false
                }
        } // fine metodo

        // controlla se l'elicottero puo' ospitare il numero di truppe indicato
        // restituisce vero se l'operazione è fattibile falso se invece l'operazione non è fattibile
        public   bool EliTroopsCheck (int troopsN, Elicottero eli , TroopsManager TroopsM)
        {
            int k; // var indice
            int totW = 0;//peso totale delle truppe

            if (TroopsM.TroopList.Count >= troopsN) {// controllo se ho le truppe che servono
                for (k=0; k< troopsN; k++)
                    totW = totW + TroopsM.SingleSoldierW;// controllo il peso totale necessario
                if (totW < eli.WTroopsLeftOnBoard)    return true; // risultato
            }return false;
        } // fine metodo


    }// fine classe
}// fine name space
```

6.20 File ClockTimer

```
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// CLOCK TIMER
// sistema di gestione del timer per la sincronizzazione delle
// attività. ogni elicottero deve essere dotato di un timer
// opportunamente fasato su un master ( PATTERN OBSERVER)
// che permette la sincronizzazione degli eventi.
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

using System;
using System.Threading;

namespace EliAssaltoAnfibio
{
    public class ClockTimer : Subject // LA CLASSE EREDITA DA SUBJECT
   {

            private static ReaderWriterLockSlim wr1 = new ReaderWriterLockSlim();
            private TimeSpan currentTime = TimeSpan.Zero; // default value 00:00:00
            private volatile bool started = false, stopped = true;
            public void SetTime(TimeSpan newTime) { currentTime = newTime; }
            public TimeSpan GetTime()     {

            try {  wr1.EnterReadLock();
                return currentTime;  }
                finally{  wr1.ExitReadLock(); }}

            // lista di eventi - REGISTRAZIONE TIMER
        public void Start ()
        {
            if (!started) {
                started = true;
                new Thread (Run).Start (); // carica il thread del metodo run
            }  }
            public void Stop() { stopped = true; }

            private void Run() {
                stopped = false;
                    while (!stopped) {
                    Thread.Sleep(1000); // pausa 1 secondo
                                // posso modificare il moltiplicatore di sleep per variare
                                // la velocità di simulazione dell'intero sistema SPERIMENTALE
                        Tick();  }
            started = false; stopped = true; }
            private void Tick() {
            wr1.EnterWriteLock ();
            currentTime += new TimeSpan(0, 0, 1); // increment: 1 sec
            wr1.ExitWriteLock ();
            Notify(); //METODO DELLA CLASSE SUBJECT: permette l'update di tutti i timer degli elicotteri
            } }}
```

6.21 File Subject

```
// ++++++++++++++++++++++++++++++++++++++++++++++
// pattern observer per il timer degli elicotteri
// contiene gli attach e detach per l'osservatore
//++++++++++++++++++++++++++++++++++++++++++++++

using System;
using System.Collections.Generic;

namespace EliAssaltoAnfibio{
    public abstract class Subject
    {
        private List<IObserver> observers = new List<IObserver>();
        public void Attach(IObserver obs) { observers.Add(obs); }
        public void Detach(IObserver obs) { observers.Remove(obs); }
        public void Notify()
            {
            foreach (IObserver obs in observers) // viene essettuato l'udate di ogni elemento
                                                 // nella lista di Observers
            obs.Update(this);

        }  } }
```

6.22 File IObserver

```
//++++++++++++++++++++++++++++++++++++++++++++++++++
// interfaccia Observer
// definita per il design patter Observer
// serve un timer di sincronizzazione per le attivita'
//++++++++++++++++++++++++++++++++++++++++++++++++++
using System;

namespace EliAssaltoAnfibio
{
    public interface IObserver
    {
        void Update(Subject subj);
    }

} // fine name space
```

6.23 File SimMover

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// la classe contiene la logica di funzionamento del programma
// la gestione dei movimenti degli elicotteri
// la gestione dei movimenti delle truppe
// la gestione dei movimenti del cargo
// vengono effettuate all'interno della classe
// la gestione logica delle informazioni
// verrà poi utilizzata dalla classe di visualizzazione
// delle informazioni
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

using System;
using Microsoft.Xna;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using System.Collections.Generic;

namespace EliAssaltoAnfibio
{
    public class SimMover
    {


        public Spot.day_night DNOps { set; get; } // spot day night struct
        // gestione delle liste
        public TroopsManager TroopM { set; get; } // gestore truppe

        public EliManager EliM { set; get; } // gestore elicotteri

        public SpotManager SpotM { set; get; } // gestore spot

        public CargoManager CargoM { set; get; } // gestore cargo

        public InfoWindow WinI { set; get; } // finestra informazioni

        public float Speed{ set; get;} // set iniziale della velocità pari a 100 nodi

        public bool EndProgramState { set; get;}// flag di fine simulazione

        // costruttore della classe
        public SimMover (Spot.day_night _dnOps, InfoWindow _winI, TroopsManager _troopM, EliManager _eliM, SpotManager _spotM, CargoManager _carg
oM)
        {
            this.DNOps = _dnOps; // passo la struttua giorno notte
            this.WinI = _winI; // passo la finestra informazioni
            this.TroopM = _troopM; // passo il manager truppe
            this.EliM = _eliM; // passo il manager elicotteri
            this.SpotM = _spotM; // passo il manager spot
            this.CargoM = _cargoM; // passo il manager cargo
            this.Speed = 100; // set iniziale della velocità pari a 100 knots
            this.EndProgramState = false; // inizializzo l'end state a falso
        }
        //-----------------------------------------------------------------
        // METODI DI SUPPORTO  ALLA GESTIONE DELLA CLASSE
        // controlla che tutti gli elicotteri disponibili siano in volo
        // cambia la destinazione dell'elicottero
        public void ChangeEli (Elicottero eli, float x, float y)
        {

            eli.destination = new Point ((int)x, (int)y);
            //    eli.destination.X = (int)x;
            //eli.destination.Y = (int)y;
        }

        // -----------------------------------------------------------------
        // assegna una destinazione in coordinate all'elicottero
        // viene trasformata la destinazione enum in coordinate
        public void AssignEli ()
        {
            foreach (Elicottero eli in EliM.ElicotteriList) {
                // direzione holding point assegno le coordinate HP
                if (!eli.isOnDestination && eli.DirToGo == Elicottero.Direction.hp) {
                    ChangeEli (eli, EliM.HoldingP.HPvector.X, EliM.HoldingP.HPvector.Y);
                }

                // direzione landing zone assegno le coordinate LZ
                if (eli.DirToGo == Elicottero.Direction.lz) {
                    ChangeEli (eli, (EliM.LandingZoneL.LZposition.X - (eli.NumEli) * 5), (EliM.LandingZoneL.LZposition.Y - (eli.NumEli) * 5));
                    eli.LZ = EliM.LandingZoneL; // assegno la LZ

                }

                // direzione nave assegna le coordinate nave agli elicotteri
                if (eli.DirToGo == Elicottero.Direction.ship) {
                    ChangeEli (eli, StaticSimulator.shipborderLandX,
                        (StaticSimulator.shipborderLandYdown-20 -
                        ((eli.NumEli) * ((StaticSimulator.shipborderLandYdown - StaticSimulator.shipborderLandYup) / EliM.initEliNum))));
                }
            }
        }
        // restituisce vero se l'elicottero sullo spot puo' caricare un cargo qualunque ancora da prelevare
        // restituisce falso altrimenti
        private bool checkCargo (Spot spot)
        {

            if ((CargoM != null) && (CargoM.CargoList.Count > 0) && // se il cargo non è nullo
                (CargoM.CheckCargoLoadOnEli (spot.Eli, TroopM.SingleSoldierW))) // ed esiste un cargo che possa andare sull'elicottero
```

```
            return true; // ritorna vero
        return false; // altrimenti ritorna falso

    }
    // ------------------------------------------------------------------
    // restituisce vero se l'elicottero sullo spot puo' caricare ancora delle truppe rimaste
    private bool checkTroops (Spot spot)
    {
        if ((TroopM.TroopList.Count == 0)
            || ((TroopM.TroopList.Count > 0)
            && (spot.Eli.WTroopsLeftOnBoard < TroopM.SingleSoldierW)))
            return false;
        return true;
    }
    //-------------------------------------------------------------------------------------
    // ho necessità di controllare che gli elicotteri sul ponte siano carichi di soldati o truppe
    // se un elicottero è scarico non h atruppe o cargo a bordo e l'hangar è vuoto
    // l'elicottero va dichiarato non impiegabile e rimesso in hangar
    public void CheckEliRequired ()
    {
        foreach (Spot spot in SpotM.Spotlist) {
            // la condizione si verifica quando l'elicottero è scarico
            // e non esistono truppe o cargo che possono essere caricate
            if (spot.Eli != null && (spot.Eli.EliSoldierList.Count == 0)  // se l'elicottero è scarico di truppe
                &&(spot.Eli.EliCargoList.Count == 0)  // l'elicottero è scarico di cargo
                &&!checkTroops (spot) // non ci sono piu' truppe da caricare
                && !this.checkCargo (spot) // non ce' piu' cargo da caricare
                && !spot.Eli.IsFlying // e l'elicottero non è in volo ma è atterrato

            ) { // non ci sono truppe e cargo che possono essere prelevate
                // se la condzione si verifica all'ora l'elicottero non è piu' necessario all'attività
                // e pertanto viene rimesso in hangar
                spot.Eli.isRequiredForOP = false; // elicottero non richiesto
                spot.Eli.PosEli = false; // eli posizionato in hangar
                spot.Eli.IsEfficient = false; // eli impostato non efficiente
                spot.Eli.hasSpotReserved = false; //  eli senza uno spot
                spot.Eli.SpotAssigned = null; // eli senza uno spot
                spot.Eli.IsBladeSpread = false; // eli pale chiuse
                spot.Eli.IsFlying = false; // eli non vola
                spot.Eli.IsRunning = false; // eli not running
                spot.Eli.isApproaching = false; // eli not apprch
                spot.SpotOccupied = false;// eli non spot
                spot.IsReservd = false; // eli non spot
                spot.eliWhoReservd = null; // eli non sopot
                spot.Eli = null; // eli non spot
            }
        }
    } // fine classe
    //-------------------------------------------------------------------------

    // metodo per il disimbarco delle truppe sulla LZ
    public void LZTroopsandCargo_Disembark_changeDST ()
    {
        foreach (Elicottero eli in EliM.LandingZoneL.LZeliList) {

            if (eli.isLZ  // Se l'elicottero è sulla LZ
                && !eli.IsBlocked // non è bloccato
                // ha cargo o truppe a bordo
                && (eli.EliSoldierList.Count>0 || eli.EliCargoList.Count>0)
                ) {

                // scarico le truppe e modifico le variaibli
                if (eli.IstroopsOnBoard
                    && !eli.IsBlocked) // se l'elicottero ha la flag di truppe a bordo le scarico

                eli.EliBlock (StaticSimulator.timeToDisembarkTroops, 9); // blocco elicottero 2 minuti per scarico truppe
                // qui si setta elitroopsisonboard == false FLAG , viene inoltre cancellata la lista truppe da bordo eli


                // scarico il cargo e modifico le variaibili ( la cargo list essite ed è 0 se non cè cargo)
                if (!eli.IstroopsOnBoard
                    && eli.IsCargoOnBoard
                    && !eli.IsBlocked) // se l'elicottero ha scaricato le truppe
                                    // ma ancora non ha scaricato il cargo , allora deve scaricare anche il cargo

                    // il blocco degli eli per scaricare il cargo avviene in modo incrementale per il numero di cargo presente a bordo
                    // ogni cargo ha in media una necessita' di scarico di 3 minuti
                    eli.EliBlock (CargoManager.CargoTimeTodisembark*eli.EliCargoList.Count, 6);

                if (CargoM!=null)    CargoM.CheckThisEliCargoFull (eli); // controllo l'eli
                }

                if (CargoM != null) CargoM.CheckEli_CargoFull (EliM);// il metodo controlla che esista ancora un cargo
                                                                     //che possa essere inserto in elicottero

                // se l'elicottero non è bloccato
                // non ha cargo a bordo
                // non ha truppe a bordo
                // procedi all'assegnazione del rientro
                if (eli.isLZ
                    &&!eli.IsBlocked
                    && !eli.IsCargoOnBoard
                    && !eli.IstroopsOnBoard)
            {
                    //---------------------CARGO E PERSONALE SCARICATO
                    //-------------------- ASSEGNAZIONE SPOT PER IL RIENTRO
                    eli.Go_goback = false; // setta l'eli per il rientro
                    eli.IsFull = false; // resetto la variabile di carico completo

                    // controllo l'esistenza di uno spot in grado di ospitare l'elicottero al rientro dalla LZ
```

```
                    Spot spot = SpotM.SpotToHostaCat (eli.CatEli, DNOps);

                    if (spot != null)
                    { // se lo spot non è nullo procedo verso ship
                        eli.DirToGo = Elicottero.Direction.ship; // assegno direzione Ship
                        SpotM.Spotlist.Find(x => x==spot).IsReservd = true; // spot riservato per l'elicottero
                        eli.SpotAssigned = spot; // spot assegnato all'elicottero
                        SpotM.Spotlist.Find(x=> x==spot).eliWhoReservd = eli; // l'elicottero che ha richiesto il reserve
                        SpotM.CheckState (); // assegnazione dello spot cambio di stato del manager
                    } else
                        eli.DirToGo = Elicottero.Direction.hp; // se il ponte è occupato vado
                    // resto in atesa nell'holding point


                    //-----------------------------------------
                    // reset variabili di stato eli per il rientro
                    eli.isApproaching = false;
                    eli.EliSpeed = this.Speed;// resetto la velocità dell'eli
                    eli.isOnDestination = false; // resetto la destinazione
                    eli.isLZ = false; // resetto la flag LZ
                    this.AssignEli (); // assegno il vettore della nuova destinazione

                    WinI.InsertSomeText ("LANDING ZONE : elicottero : " + eli.IdEli + " lascia LZ time: " + eli.current_time.ToString ());
                    WinI.InsertEvent (eli.current_time, "T/O LZ", eli.IdEli, eli.NumEli);
                }

            } // termine scansione elioctteri sulla LZ
            // cancellare l'eli dalla LZ
            if (EliM.LandingZoneL.LZeliList.Count>0)
                EliM.LandingZoneL.LZeliList.RemoveAll (x => x.isLZ==false);// rimuovi gli elicotteri dalla LZ che sono stati rilasciati



        }// termine metodo
        //-------------------------------------------------------------------

        // metodo di controllo elicotteri in holding
        // vengono controllati tutti gli elicotteri in holding sia in direzione per sbarco truppe
        // sia in direzione per rientro a bordo
        // al termine del controllo o viene mantenuto l'holding attesa
        // oppure viene assegnata una nuova direzione di movimento
        public void HPCheckAndChange ()
        {
            //int eliHolding;
            int eliForces;
            //controlla lo stato della lista degli elicotteri in holdin

            int dirHoldGO = EliM.HoldingP.EliHolding.FindAll (eli => eli.Go_goback == true).Count; // conta gli elicotteri all'interno della LZ d
ir to GO
            int dirHoldBACK = EliM.HoldingP.EliHolding.FindAll (eli => eli.Go_goback == false).Count; // eli con direzione nave in attesa


            //eliHolding = EliM.HoldingP.EliHolding.Count- dirHoldGO; // numero  elicotteri in holding attualmente
            eliForces = EliM.EliForce (); // tutte le forze elicottero utilizzabili
            // per l'assalto mi serve che ci siano tutti gli elicotteri impiegabili in una sola ondata

            if (dirHoldGO == eliForces) { // se gli elicotteri sono tutti presenti allora inizia l'assalto
                foreach (Elicottero eli in EliM.ElicotteriList) {
                    if (eli.IsHolding && eli.Go_goback) { // se l'eliocottero è in holding e ha direzione ANDATA
                        eli.DirToGo = Elicottero.Direction.lz; // gli elicotteri assumono tutti la direzione della LZ
                        eli.isApproaching = false; // resetta l'approaching
                        eli.EliSpeed = this.Speed; // NECESSARIA?
                        eli.isOnDestination = false; // l'elicottero non è piu' a destinazione
                        eli.IsHolding = false; // l'elicottero non è piu' in holding
                        EliM.RemoveEli (EliM.HoldingP, eli); // rimuovi gli elicotteri dall'HP
                        WinI.InsertSomeText ("HOLDING POINT : elicottero : " + eli.IdEli + " lascia l'HP time: "+ eli.current_time.ToString());
                        WinI.InsertEvent ( eli.current_time, "OUT HP ",eli.IdEli,eli.NumEli);
                    }
                }
            }


            // controllo che ci sia uno spot libero per l'elicottero in holding e con direzione Nave
            // se lo spot esiste lo assegnao all'elicottero
            if (dirHoldBACK > 0) {
                foreach (Elicottero eli in EliM.ElicotteriList) {
                    if (eli.IsHolding && !eli.Go_goback) {
                        Spot spot = SpotM.SpotToHostaCat (eli.CatEli, DNOps);
                        if (spot != null) { // se lo spot non è nullo procedo verso ship
                            eli.isOnDestination = false; // resett flag
                            eli.isApproaching = false; // resetta l'approaching
                            eli.SpotAssigned = spot; // assegnazione spot
                            eli.IsHolding = false;// reset flag
                            eli.DirToGo = Elicottero.Direction.ship; // assegno nuova direzione
                            eli.EliSpeed = Speed;
                            spot.IsReservd = true; // lo spot è riservato
                            spot.eliWhoReservd = eli; // l'elicottero che ha richiesto il reserve
                            EliM.RemoveEli (EliM.HoldingP, eli); // rimuovi gli elicotteri dall'HP
                            WinI.InsertSomeText ("HOLDING POINT : elicottero : "
                            + eli.IdEli + " lascia l'HP time: "+ eli.current_time.ToString());
                            WinI.InsertEvent ( eli.current_time, "OUT HP",eli.IdEli, eli.NumEli);
                        }
                    }
                }
            }
            this.AssignEli (); // assegna la nuova destinazione vettoriale in base alla LZ
            SpotM.CheckState (); // ricontrollo stato ponte
        }
```

```
// FUEL DATA , BLOKING QUOTE GENERATORS PER ELICOTTERI
// LA FUNZIONE VIENE RICHIAMATA DAL LOOP PRINCIPALE
// ------------------------------------------------------------------------------

// controllo del carburante: per ogni elicottero
// viene effettuato un update del carburante
public void FuelandBlockChekEliloop (float elapsedtime)
{
    foreach (Elicottero eli in EliM.ElicotteriList) {
        if (eli.IsRunning) {
            if (eli.Fuel > 0) {
                if (!eli.isHotRef)
                    eli.Fuel = eli.Fuel - ((eli.FuelC / 60) * elapsedtime); // sottraggo il consumo del carburante di 1 minuto simulato

            } // decremento il carburante al tick del secondo
            //-----------------------------------------------

            if (eli.Fuel < (eli.FuelC / 3)) {
                eli.LowFuel = true; //se mancano 20 minuti di carburante dichiaro LOW FUEL STATE
            } else {
                eli.LowFuel = false; // la variaible puo' RICAMBIARE dopo il rifornimento a caldo
            }
            //-----------------------------------------------
            if (eli.Fuel <= 0 && !eli.EliCrash) { // setta le variaibl
                eli.IsRunning = false; // se il carburante finisce ferma i motori
                if (eli.IsFlying == true) {
                    eli.EliCrash = true;// se l'elicottero era in moto setta il crash
                    eli.IsFlying = false;
                    eli.IsEfficient = false; // eli inefficiente
                    eli.isRequiredForOP = false; // eli non piu' richiesto

                    if (eli.IsHolding) {
                        EliM.HoldingP.EliHolding.Remove (eli);
                        eli.IsHolding = false;
                    }


                    if (eli.SpotAssigned != null) {
                        eli.Spotrequested = null; // spot richiesto dall'eli
                        // resetto lo spot assegnao all'elicottero abbattuto
                        int k = eli.SpotAssigned.SpotPos;
                        SpotM.Spotlist.Find(x => x == eli.SpotAssigned).Eli = null;
                        SpotM.Spotlist.Find(x => x == eli.SpotAssigned).eliWhoReservd = null;
                        SpotM.Spotlist.Find(x => x == eli.SpotAssigned).IsReservd = false;
                        SpotM.Spotlist.Find(x => x == eli.SpotAssigned).SpotOccupied = false;
                        eli.SpotAssigned = null; // resetto lo spot assegnato
                        SpotM.CheckState (); // ricontrollo lo stato del ponte
                    }

                        EliM.CrashList.Add (eli); // eli crashed
                        WinI.InsertSomeText ("SIMULATOR: crash eli (0 fuel): " + eli.IdEli
                        + " truppe perse: " + eli.EliSoldierList.Count
                        + " cargo perso: " + eli.EliCargoList.Count);
                        WinI.InsertEvent (eli.current_time, "ELI CRASH", eli.IdEli, eli.NumEli);
                    }
            }
            //-----------------------------------------------------------

            eli.TotalWeightCalculated = eli.ToTWGCalculus ();// update del peso totale

            //-----------------------------------------------------------
        }
        eli.RandomQuoteGenrator (); // generatore di quota randomica in volo
        eli.EliBlock (0, -1); // EFFETTUO UN CONTROLLO SUL BLOCCO ELICOTTERI

    } // termine for each
} // termine FuelandBlockChekEliloop-------------------------------------

// ritorna vero se il cargo esiste ed è 0 count
private bool CheckCargoBoolZero()
{
    if (this.CargoM != null && this.CargoM.CargoList != null)
    {
        if (this.CargoM.CargoList.Count == 0) return true;
        else return false;
    }
    else  return true;
} // termine CheckCargoBoolZero-----------------------------------------------------------------

//-------------------------------------------------------------------------------------------------
#region [LOGICA DI FUNZIONAMENTO DEL PROGRAMMA ]
// LOOP LOGICO PRINCIPALE, QUI SI SVOLGE LA LOGICA DI FUNZIONAMENTO DEL PROGRAMMA //
//-------------------------------------------------------------------------------------------------
// looping delle attività di controllo effettuate dalla logica del programma
// le informazioni vengono passate al manager grafico per la visualizzazione
// AL TERMINE DELL'AGGIORNAMENTO LOGICO VIENE RICHIAMATO IL LOOP
// CHE SI OCCUPA DELL'AGGIORNAMENTO DELLA GRAFICA E DELLA VISUALIZZAZIONE
public void LoopforUpdate (float elapsed)
{

    this.FuelandBlockChekEliloop (elapsed); // CONTROLLO E AGGIORNO FUEL A BORDO ELI

    // ----- inizio il loop con aggiorno gli STATI dei manager
    EliM.CheckState ();
    SpotM.CheckState ();
    TroopM.CheckState ();
    //------termine controllo stati

    // CARGO-------------------------------------------------------------------------
```

```
// controllo se esiste del cargo da prelevare
// il cargo non deve essere inserito necessariamente negli elicotteri sul ponte
// puo' essere inserito anche se gli elicotteri sono in hangar
if (CargoM != null) {

    CargoM.CheckState ();
    if (CargoM.Status != AbstractManager.StatusManager.Empty) {
        // carica il cargo se disponibile sugli elicotteri disponibili
        if (this.CargoM != null)
            CargoM.CargoDistribution (WinI, EliM, TroopM); // distribuisci prima il cargo delle truppe,
    }                                                      // il cargo potrebbe non essere presente

    this.CheckEliRequired (); // se gli elicotteri non sono required vengono rimessi in hangar
    CargoM.CheckState (); // aggiorno lo status del cargo
    TroopM.CheckState (); // aggiorno lo statu delle truppe

}
//-------------------------------------------------------------------------------


// SPOSTO ELI SUL PONTE ---------------------------------------------------------
// controllo se l'hangar ha ancora elicotteri da movimentare
// se l'hangar è vuoto non ho possibilità di movimentare altri elicotteri
// se il ponte è pieno non ho possibilità di movimentare gli elicotteri
// inoltre se esistono piu' di due elicotteri bloccati non ne posso movimentare altri,
// il ponte dispone solo di due elevatori per elicottero


if (((EliM.Stato != AbstractManager.StatusManager.Empty) &&
    (SpotM.Stato != AbstractManager.StatusManager.Full)) /*&& (EliM.CheckBlockedHangar()<=2)*/) {
    // sposta gli elicotteri sul ponte se ci sono spost liberi
    EliM.MoveEliOnSpot (SpotM, DNOps, WinI); // muovi, se possibile, gli elicotteri dall'hangar sugli spot
    EliM.CheckState (); // check elim state
    SpotM.CheckState (); // check spot state
}


//-----------------------------------------------------------------------------------------
//  RUNNING AND BLADE SPREADING
// controllo la presenza di elicotteri sul ponte
if (SpotM.Stato != AbstractManager.StatusManager.Empty) {
    // controlla che gli elicotteri sul ponte siano funzionali per l'attività altrimenti li rimette in hangar
    this.CheckEliRequired ();
    // metti in moto gli elicotteri sul ponte
    if ((EliM.CheckBlockedHangar () == 0)) { // se non ci sono piu' elicotteri in attesa in hangar

        // posso metter in moto tutti gli elicotteri sul ponte
        SpotM.AllEliSpottedRunning ();// elicotteri sul ponte mettono in moto INIZIO FUEL CHECK

        // apri le pale degli elicotteri sul ponte
        SpotM.AllEliBladeSpreading ();// elicotteri sul ponte aprono le pale pronti per l'imbarco

        // controllo lo STATO del carburante degli elicotteri sul ponte PER LA MISSIONE
        // se gli elciotteri hanno necessità di carburante devo effettuare il rifornimento
        // avvinado un ritardo al decollo di 6 primi
        SpotM.CheckHOTREFLowFuelState (EliM.initEliNum);
    }
}

//-----------------------------------------------------------------------------------------

// SPOSTO LE TRUPPE A BORDO -----------------------------------------------------------
// controllo che ci siano truppe ancora non assegnate, che il ponte non sia vuoto e che gli elicotteri presenti non siano bloccati
// le truppe devono essere assegnate agli elicotteri sul ponte running e a pale aperte

if ((TroopM.Status == AbstractManager.StatusManager.CanAll)// se il manager ha delle truppe
     && (SpotM.Stato != AbstractManager.StatusManager.Empty)// se il ponte ha degli elicotteri
     && (EliM.ElicotteriList.FindAll (x => (x.IsRunning// il numero di elicotteri running
     && !x.IsFlying// ma che non volano
     && !x.IsBlocked// elicottero non bloccato
        && !x.isTroopsFull)).Count > 0)) //  hanno spazio per le truppe
        // procedo con l'imbarco delle truppe se le condizioni sono corrette
    {
    //carica le truppe a bordo degli elicotteri disponibili sul ponte
    TroopM.EliSpotTroopDistribution (SpotM, TroopM, WinI);// muovi le truppe sul sugli spot dopo l'apertura

    TroopM.CheckState (); // aggiorno lo stato delle truppe dopo l'imbarco
    this.CheckEliRequired (); // se gli elicotteri non sono required vengono rimessi in hangare successivamente eliminati
    }

//-----------------------------------------------------------------------------------------

// HOT REF, READY STATUS, TAKE OFF PER TUTTI GLI ELICOTTERI SUL PONTE------------------------
// controllo che esistano elicotteri sul ponte per il decollo
if (SpotM.Stato != AbstractManager.StatusManager.Empty) {

    // se gli elicotteri sul ponte non bloccati assumono lo stato di pronti al decollo
    SpotM.EliReadyStatus (); // gli elicotteri in volo non entrano in questo loop
    // viene settata la velocità di crociera per gli elicotteri decollati

    this.SpotM.AllEliTakingOff (this.Speed); // decolla ed assume destinazione HP
    // gli eicotteri in volo non entrano in questo loop
    // assegna una destinazione agli elicotteri (HP , LZ , DECK )

    //RICONTROLLO GLI STATI -----
    EliM.CheckState (); // controlla lo stato hangat
    SpotM.CheckState (); // controlla lo stato ponte
    this.AssignEli (); // assegna una destinazione vettoriale
    //--------------------------
}
```

```
            //-----------------------------------------------------------------
            // se gli elicotteri disponibi sono tutti in HP lascia  HP
            // e assegna LZ come nuova destinazione
            if (EliM.HoldingP.EliHolding.Count > 0)
            {
                // controlla che tutti gli elicotteri disponibili siano in HP
                this.HPCheckAndChange ();
            }

            // se l'elicottero è sulla LZ sbarca le truppe
            if (EliM.LandingZoneL.LZeliList.Count > 0) {
                this.LZTroopsandCargo_Disembark_changeDST (); //TRUPPE E CARGO RILASCIO

                TroopM.CheckState (); // aggiorno lo statu delle truppe
                EliM.CheckState ();
            } // controllo gli spot assegnabili

            if (EliM.LandingZoneL.LZeliList.Count > 0)
                EliM.LandingZoneL.LZeliList.RemoveAll (x => x.isLZ == false); // rimuovo dalla LZ l'elicottero ridecollato


            // se il numero di elicotteri lista è 0
            // in quanto se non utili sono stati eliminati
            // allora fine simulazione raggiunto
            //controllo la fine della simulazione
            if (EliM.ElicotteriList.Count == 0) {
                this.EndProgramState = true;
                // loggo il tempo di termine operazioni
                WinI.InsertSomeText ("TERMINE OPERAZIONI " + EliM.MainTime.GetTime ().ToString ());
                WinI.InsertEvent (EliM.MainTime.GetTime (), "TERMINE OPERAZIONI ", "", -2);
                // loggo il tempo di termine operazioni

                // MOTIVAZIONI PROGRAM FAIL
                if (this.TroopM.TroopList.Count == 0 && CheckCargoBoolZero ())  // se il numero di truppe da trasportare ==0 e il cargo è == 0
{                       // simulazione terminata con successo altrimenti simulazione fallita
                    WinI.InsertSomeText ("SIMULAZIONE TERMINATA CON SUCCESSO");
                    WinI.InsertEvent (EliM.MainTime.GetTime (), "SIMULAZIONE TERMINATA CON SUCCESSO ", "", -3);

                } else {

                    WinI.InsertSomeText ("SIMULAZIONE FALLITA");
                    WinI.InsertEvent (EliM.MainTime.GetTime (), "SIMULAZIONE FALLITA ", "", -3);
                }
                    // stoppa i thread ------------------------
                    EliM.MainTime.Stop ();
                    //---------------------------------------
                }

                this.EliMovingLogic (elapsed); // logica movimento eli
            }


        #endregion [TERMINA IL MAIN LOOP ]
        //-------------------------------------------------------------------------

        //-------------------------------------------------------------------------
        #region [ MATEMATICA DEI MOVIMENTI GRAFICI]

            public void EliMovingLogic (float elaspedT)
        {

            //-----------------------------------------------------------------------------------------------
            // per ogni elicottero controlla la destinazione e permetti la rotazione per raggiungere la destinazione
            foreach (Elicottero eli in EliM.ElicotteriList) {

                eli.DIstanceToPnT=(StaticSimulator.DistancePtoP ((float)eli.destination.X, (float)eli.destination.Y, eli.elivector2d.X, eli.elive
ctor2d.Y));// distanza obiettivo

                // muove gli elicotteri alla destinazione
                if (!eli.IsBlocked && eli.IsFlying && !eli.isOnDestination) {
                    if (!eli.isApproaching)
                        eli.EliSpeed = this.Speed;
                    StaticSimulator.MoveEli (eli.EliSpeed, eli, (float)eli.destination.X, (float)eli.destination.Y, elaspedT);
                }

                eli.DIstanceToPnT=(StaticSimulator.DistancePtoP ((float)eli.destination.X, (float)eli.destination.Y, eli.elivector2d.X, eli.elive
ctor2d.Y));
                // controllo la distanza dal destinazione
                if ((eli.DirToGo == Elicottero.Direction.hp)// la direzione dell'eli è lhp
                    && (eli.DIstanceToPnT < 30)// la distanza è inferiore a 30px
                    && !eli.isOnDestination) { // e l'elicottero non è ancora flaggato a destinazione// se è arrivato a destinazione
                    eli.isApproaching = true; // approach to destination smette di prendere la velocità dalla formazione e la riduce per l'atterr
aggio
                    eli.isOnDestination = true; // sotto i 20 px l'elicotter è a destinazione nell'HP
                    eli.IsHolding = true; // flag holding!!
                    WinI.InsertEvent ( eli.current_time, "IN HP",eli.IdEli, eli.NumEli);
                    WinI.InsertSomeText( " ELI " + eli.IdEli + " ENTRA HP: "+ eli.current_time.ToString());
                    eli.EliSpeed = 0f; // setto la velocità
                    EliM.AddEliHp (EliM.HoldingP, eli); // aggiungi l'elicottero alla holding lista
                }


                // ELICOTTERI IN HOLDING MANTENGONO UNA ROTTTA CIRCOLARE
                //-----------------------------------------------------------------------------------
                if (eli.IsHolding && eli.isOnDestination) {  // se l'elicottero è in holdign ruota attorno ad un punto fisso
                    eli.rotation = eli.rotation + 0.9F*elaspedT; // viene impostata la velocità di rotazione
                }
```

```
                //-------------------------------------------------------------------------------------------
                // LANDING zone acquisita
                if ((eli.DirToGo == Elicottero.Direction.lz) && (eli.DIstanceToPnT < 25)) {
                    eli.isApproaching = true;
                    if (eli.EliSpeed > 0) {
                        eli.EliSpeed = eli.EliSpeed - 5f; // decremento della velocità
                        eli.quota = eli.quota - 0.01f;
                    }
                        else { // se l'elicottero ha raggiungo velcotà 0
                        if (!eli.isOnDestination) { // se la variabile non è settata a destinazione
                            eli.isOnDestination = true; // setta la destinazione come raggiunta
                            //    eli.Go_goback = false; // eli assume direzione rientro
                            eli.isLZ = true; // flag LZ
                            WinI.InsertEvent ( eli.current_time, "LAND LZ" ,eli.IdEli, eli.NumEli);
                            WinI.InsertSomeText( " ELI " + eli.IdEli + " SULLA LZ time: "+ eli.current_time.ToString());
                            EliM.AddEliList (EliM.LandingZoneL.LZeliList, eli); // aggiungo l'elicottero alla lista

                            // INSERIRE SBARCO TRUPPE E CARGO
                            // QUANDO TERMINATO CAMBIARE DESTINAZIONE TO SHIP
                        }
                    }
                }


                // DIREZIONE NAVE l'elicottero rientra dalla LZ PUO': RIFORNIRE, TERMINARE L'OPERAZIONE, CARICARE CARGO TRUPPE E DECOLALRE
                //-------------------------------------------------------------------------------------------------------------
                if (!eli.isApproaching && (eli.DirToGo == Elicottero.Direction.ship)
                    && eli.IsFlying && !eli.Go_goback
                    && eli.DIstanceToPnT < 25) {

                    eli.isApproaching = true; // elicottero approaching
                }
                // ship acquisita diminuzione di speed
                if (eli.isApproaching && (eli.DirToGo == Elicottero.Direction.ship)
                    && eli.IsFlying && !eli.Go_goback
                    && eli.DIstanceToPnT < 20) {
                    if (eli.EliSpeed > 0)
                        eli.EliSpeed = eli.EliSpeed - 5f; // decremento della velocità
                    else {
                        eli.isOnDestination = true; // elicottero arrivato a destinazione
                        eli.SpotAssigned = SpotM.Spotlist.Find (spot => spot.eliWhoReservd == eli);// assegno lo spot
                        // salvo dati
                        WinI.InsertSomeText ("ELICOTTERO APPONTA: " + eli.IdEli + " time: "+ eli.current_time.ToString());
                        WinI.InsertEvent ( eli.current_time, "LAND SHIP",eli.IdEli, eli.NumEli);
                        //--- salvo dati
                        eli.SpotAssigned.SpotOccupied = true; // lo spot è occupato
                        eli.SpotAssigned.Eli = eli; // as segno l'elicottero allo spot
                        eli.PosEli = true; // pos sul ponte
                        eli.Go_goback = true;
                        eli.IsFlying = false; // elicottero a terra non piu' in volo
                        eli.rotation = 230; // riporto l'angolo iniziale
                        eli.isREADYstatus = false; // deve riacquisire il ready status perridecollare
                        SpotM.CheckState (); // controllo lo stato del simulatore
                        eli.EliSpeed = 0; // resetto la speed a 0
                    }
                }
            }

        } // temine loop
        #endregion [ TERMINA LA LOGICA DEI MOVIMENTI ]
    }
    // fine classe
}
// fine namespace
```

6.24 File StaticSimulator

```
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// classe di gestione statica del simulatore
// contiene le costanti statiche di posizione degli oggetti
// sullo schermo, e contiene i metodi per il calcolo del volo
// degli elicotteri
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

using System;
using Microsoft.Xna;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using System.Collections.Generic;

namespace EliAssaltoAnfibio
{
    public static class StaticSimulator
    {
        // costanti utilizzate nella simulazione
        public const int LZpixelDistance = 700;// distanza in pixel ship lz
        public const float angleMov = (MathHelper.Pi / 360f)*2; // all'interno dei 2 gradi
        public const int CargoLoadTiming = 5; // tempo per operazioni di cargo load
        public const int CargoTimeTodisembark = 2; // tempo richiesto per disimbarcare il cargo
        public const int BladeSpreadBlockingTime=2; // tempo di blocco per l'apertura pale
        public const int timeNeedToEngage = 10; //  minuti per la messa in moto dell'elicottero
        public const int timeNeedToMoveToDeck = 15; // 15 minuti per muove l'elicottero sul ponte dall'hangar
        public const int timeToEmbarcTroops = 3; //3 minuti per l'imbarco truppe
        public const int timeToDisembarkTroops = 2;// 2 minuti scarico truppe
        public const int spacing = 12; // spaziatura caratteri
        public const int shipborderLandX = 865;// X di appontaggio degli elicotteri sulla nave 865
        public const int shipborderLandYup = 210;// min Y di appontaggio
        public const int shipborderLandYdown = 590;//max Y di appontaggio
```

```
public const int shipY1Y2spacing = 350; // max - min Y di appontaggio
public const int InitialRotationAsset = 240; // ni gradi la posizione iniziale degli elicotteri
public const int HPposX = 750;// posizione X holding point
public const int HPposY = 430;// posizione Y holding point
public const int LZposX = 155;// posizione X LZ point
public const int LZposY = 390;// posizione Y LZ point
public const int CargoVectorPosX = 25; // posizione relativa X cargo vector
public const int CargoVectorPosY = -20; // posizione relativa Y cargo vector
public const int TroopsVectorPosX = 25; // posizione relativa X truppe vector
public const int TroopsVectorPosY = -20; // posizione relativa Y truppe vector

// gruppo costanti utilizzate nella simulazione
//-------------------------------------------------------------------------


#region [FLY MATH]
// calcolo della velcità degli elicotti, movimento in pixel in base
// alla distanza dalla LZ
public static float SpeedinPixMin (float speed , int distance)
{
    float pixVal = 0;

    try{
        pixVal =    (StaticSimulator.LZpixelDistance /distance); // ogni viglio corrisponde a X pixel calcolati
    }
    catch
    {
        System.Console.WriteLine ("ERRORE CALCOLO DISTANZA"); // division by 0?
    }
    // il simulatore temporizza ogni MINUTO nel mondo reale come se fosse un secondo sul simulatore

        float speedVal = (speed / 60); // quante miglia faccio in un minuto
        return speedVal * pixVal; // pixel che devono essere proporzionalmente percorsi in 1 secondo

}

// generatore di interi randomici fino al max inserito
public static int RandomIntGenerator (int max )
{
    Random rnd = new Random ();
    int i= rnd.Next(max);
    return i;
}

//calcola la distanza tra due punti
public static int DistancePtoP (float x1pos, float y1pos, float x2pos, float y2pos)
{
    float Xdelta, Ydelta;
    Xdelta = Math.Abs (x1pos - x2pos);
    Ydelta = Math.Abs (y1pos - y2pos);

    return (int)Math.Sqrt (Xdelta * Xdelta + Ydelta * Ydelta);

}
// calcola l'angolo per la nuova posizione
public static float AngleFlyTo (float xpos, float ypos, Elicottero eli)
{
    float Xdelta, Ydelta, val;

    Xdelta = xpos - eli.elivector2d.X;
    Ydelta = ypos - eli.elivector2d.Y;
    // calcolo l'angolo tra l'elicottero ed il punto di arrivo

    val = ((float)(Math.Atan ((Ydelta) / (Xdelta))));

    //    if (Xdelta > 0 && Ydelta > 0)
    //    val = val;

    if (Xdelta < 0 && Ydelta <= 0)
        val = val + MathHelper.Pi;

    if (Xdelta < 0 && Ydelta > 0)
        val = val + MathHelper.Pi;

    //---------------------------
    //val = val + MathHelper.Pi;

    if (val < 0)
        val = val + MathHelper.TwoPi;

    if (val > MathHelper.TwoPi)
        val = val - MathHelper.TwoPi;

    return val;
}
//-------------------------------------------------------------------------
// setto la correzione dell'angolo del vettore eli
// con imput la direzione voluta
public static float AngleCorrection (Elicottero eli, float directionAngle)
{
    // rinormalizza gli angoli all'interno del range 0/2pi
    if (eli.rotation >= MathHelper.TwoPi)
        eli.rotation = eli.rotation - MathHelper.TwoPi;

    if (directionAngle >= MathHelper.TwoPi)
        directionAngle = directionAngle - MathHelper.TwoPi;

    if (eli.rotation < 0) eli.rotation = eli.rotation + MathHelper.TwoPi; // trasla di +2pi
    if (directionAngle < 0) directionAngle = directionAngle + MathHelper.TwoPi; //trasla di +2pi
```

```
            float delta = eli.rotation - directionAngle; // delta angle

            // normalizza il delta per valori 0 / 2pi
            if (delta < 0)
                delta = delta + MathHelper.TwoPi;
            if (delta > MathHelper.TwoPi)
                delta = delta - MathHelper.TwoPi;


            float dirPlusPi = eli.rotation + MathHelper.Pi;
            if (dirPlusPi < 0)
                dirPlusPi = dirPlusPi + MathHelper.TwoPi;
            if (dirPlusPi > MathHelper.TwoPi)
                dirPlusPi = dirPlusPi - MathHelper.TwoPi;


            if (delta <= MathHelper.Pi && delta > angleMov) {
                eli.rotation = eli.rotation - angleMov;
                return eli.rotation ;
            }

            if (delta > MathHelper.Pi && delta > angleMov) {
                eli.rotation  = eli.rotation  + angleMov;
                return eli.rotation ;

            }

            return eli.rotation;
        }
        //-------------------------------------------------------------------------
        // trasforma il valore da radianti a gradi
        public static float SetRotationAngle (float f )
        {
            return  MathHelper.ToRadians (f);
        }

        // effettua il calcolo per muovere l'elicottero
        // calcolo il vettore , lo normalizzo e gli assegno una velocità
        public static void MoveEli (float speed, Elicottero eli, float x, float y, float elapsed)
        {
            eli.AngleToFly = AngleFlyTo (x, y, eli);
            Vector2 direction = new Vector2 ((float)Math.Cos (AngleCorrection (eli,eli.AngleToFly)),
            (float)Math.Sin (AngleCorrection (eli, eli.AngleToFly))); // calcolo del vettore di movimento
            direction.Normalize (); // normalizzazione del vettore , 1 unità

            eli.elivector2d +=  direction * SpeedinPixMin (eli.EliSpeed, eli.DistancetoRun) * elapsed ;
        }
        //-------------------------------------------------------------------------
        #endregion
    } // fine classe
} // fine name space
```

6.25 FILE SIMULATOR

```
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Vengono raccolti tutti i valori inseriti
// viene effettuata l'inizializzazione del sistema
// e controllati i dati inseriti nel loro insieme.
// Una volta inseriti singolarmente i dati di ogni elicottero
// spot, truppa, cargo deve essere verificata anche che
// l'interazione tra tutte le informazioni inserite funzioni
// Una volta effettuato il controllo iniziale il simulatore
// può partire caricando le classi che gestiscono la parte grafica
// e la parte logica.
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

using System;
using System.Threading;
using System.Collections.Generic;

namespace EliAssaltoAnfibio
{
    public class Simulator
    {
        public Spot.day_night DNOps { set; get; }


        // gestione delle liste
        public TroopsManager TroopM { set; get; }

        public EliManager EliM { set; get; }

        public SpotManager SpotM { set; get; }

        public CargoManager CargoM { set; get; }

        public InfoWindow WinI { set; get; }

        public MainWindow MainWindow = MainWindow.Instance ();

        public int Distance { set; get;} // distanza LZ - ship
        //-----------------------------------------------
        // il simulatore è composto da un gestore grafico
        // non chè da un supporto logico di simulazione
        GrafXNA simulG;// supporto grafico

        SimMover simMoverLogic;// supporto logico
```

```
        //-----------------------------------------------

        public bool assignEnded;// se TRUE indica l'assegnazione iniziale degli spot dedicati

        // al simulatore vengono passate tutte le informazioni necessarie alla simulazione degli eventi
       public Simulator (int distance, Spot.day_night _dnOps, InfoWindow _winI, TroopsManager _troopM, EliManager _eliM, SpotManager _spotM, CargoManager _cargoM)
        {
            // passo al simulatore TUTTE le informazioni necessarie
            this.DNOps = _dnOps; // operazione day night
            this.TroopM = _troopM; // manager truppe  e lista truppe
            this.EliM = _eliM; // manager eli e lista eli
            this.SpotM = _spotM; // magaer spot e lista spot
            this.CargoM = _cargoM; // manager cargo e lista cargo
            this.WinI = _winI; // information window
            this.Distance = distance; // pass il paramentro distanz
            this.WinI.InsertSomeText ("SIMULATOR: simulatore creato. Effettuo l'inizializzazione....");

            // creazione della logica di gesitione e della grafica di funzionamento
            simMoverLogic = new SimMover (this.DNOps, this.WinI, this.TroopM, this.EliM, this.SpotM, this.CargoM);
            simulG = new GrafXNA (this.simMoverLogic); // costruzione del supporto grafico di riferimento

        }

        // disposizione di elicotteri e truppe
        public void InitDeckSpotAssign ()
        {
            //-------------------------------------------------------------
            // fase inizializzazione GENERALE e controllo dei dati inseriti
            //-------------------------------------------------------------

            WinI.infoTimeList.Add (new InfoWindow.timeInfoStruct(EliM.MainTime.GetTime()," - INIZIO MOVIMENTAZIONI PONTE- ", "", -1));
            this.WinI.InsertSomeText (" - SIMULATORE: INIZIO MOVIMENTAZIONI SUL PONTE");

            // crea la LZ e L'HP come membro di elimanager
            this.EliM.MakeHoldingPoint (3);// creazione dell'HP con la distanza standard dalla NAVE
            this.EliM.MakeLZ (this.Distance); // viene creata la LZ con la distanza dalla nave
            //-----------------------------------------------------------------

          this.EliM.initEliNum = this.EliM.ElicotteriList.Count; //salvo il numero iniziale di elicotteri poiche' la variabile tendera a mutare

          this.EliM.CheckEliDataFuelConsistency (this.Distance);// controllo il carburante a bordo degli elicotteri per poter effettuare la missione

          this.EliM.TroopsW_Data_correction (TroopM.SingleSoldierW); // corregge i pesi degli elicotteri diventando multipli


          this.EliM.Check_Eli_Usability (WinI, TroopM, CargoM); // controllo l'effettiva necessità iniziale degli elicotteri inseriti
                                                  // elimino gli elicotteri che non servono alla missione


            EliM.EliAutoSpotAssign (SpotM, WinI, DNOps); // assegnazione AUTOMATICA degli spot-possibile riduzione degli elicotteri
                                                    // se gli elicotteri definiti sul ponte superano il numero degli spot assegnabili

            if (CargoM != null) // se il cargo è stato inserito
            {
                CargoM.CheckCargoIsFittable (EliM); // controllo che il cargo sia imbarcabile su almeno 1 elicottero
                // se i dati del cargo non sono coerenti con i pesi trasportabili dall'elicottero il cargo viene taggato non trasportabile
            }

            EliM.CheckState (); // eli manager check state
            TroopM.CheckState (); // troops manager check state
            SpotM.CheckState (); // spot manager check state
            simMoverLogic.AssignEli (); // assegna le destinazioni iniziali
            // INIZIO SIMULAZIONE
            this.Start (); // passa il controllo a start()
            // INIZIO SIMULAZIONE
        }

        public void Start ()
        {
            // timer start
            WinI.InsertSomeText ("SIMULATORE: ciclo inserimento e disposizione iniziale ultimato");
            WinI.InsertSomeText ("SIMULATORE: INIZIO SIMULAZIONE, starting X timer\"");
            //SIMULATION WAITING SEQUNCE
            this.EliM.MainTime.Start ();// start main time
            simulG.Run (); // grafica del simulatore gestito tramite XNA
        }

}// fine classe
}// fine name space
```

6.26 FILE GRAFXNA

```
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// CLASSE DI GESTIONE GRAFICA
// SI OCCUPA DEL LOOP DI DISEGNO DEGLI SPRITE
// IN MODO CHE POSSANO ESSERE VISUALIZZATI A SCHERMO
// ELICOTTERI CARGO TRUPPE LZ , BACKGROUND E INFORMAZIONI
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

using System;
using System.Collections.Generic;
using Microsoft.Xna;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;

namespace EliAssaltoAnfibio
{
    public class GrafXNA: Microsoft.Xna.Framework.Game
```

```
{

    // spazio caratteri info eli
    GraphicsDeviceManager graphics;
    SpriteBatch spriteBatch;
    //------------------------------------------------------
    Texture2D background;
    Vector2 backgroudPOS;
    //    Texture2D Troopstexture; // truppe shape
    Texture2D RectangleTexture; // rettangolo shape
    Texture2D bar; // barra visualizzazione finale
    Texture2D CargoTexture; // cargo shape
    Texture2D lineTexture; // linea
    //------------------------------------------------------
    private SpriteFont font;
    private SpriteFont font2;

    int i = 0; // var indice eli sprite

    float elapsedTime; // elapsed timer 1
    float elapsedTime1; // elapsed timer 2
    float elapsedtimeRotationEli; // eli rotation frame

    SimMover SimMovL;

    // velocità di movimento rotazionale
    public GrafXNA (SimMover simM) // costruttore
    {
        graphics = new GraphicsDeviceManager (this); // costruzione del manager grafico

        Window.AllowUserResizing = true;            //costruisci con allow resizing viene resettata nell'inizializzazione
        Window.BeginScreenDeviceChange (true);

        this.SimMovL = simM; // contiene tutte le informazioni di gestione

        Content.RootDirectory = "Content";           // directory contente i multimedia
    }
    // METODO DI INIZIALIZZAZIONE: viene utilizzato per inizializzare i valori e gli oggetti grafici
    protected override void Initialize () // metodo eseguito all'inizio del programma
                                  // serve per inizializzare le variabili
    {
        //inizializzazione variabili grafiche per deifinizione shcermo
        graphics.PreferredBackBufferWidth = 1024;   // impostazione schermo
        graphics.PreferredBackBufferHeight = 768;   // impostazione schermo
        GraphicsDevice.PresentationParameters.BackBufferWidth = 1024; //IMPOSTAZIONI GRAFICHE
        GraphicsDevice.PresentationParameters.BackBufferHeight = 768; // IMPOSTAZIONI GRAFICHE

        spriteBatch = new SpriteBatch (GraphicsDevice); // nuovo oggetto spritebatch per la visualizzazione
        backgroudPOS = new Vector2 (0f, 0f); // posizione iniziale del background img
        // inizliazzazione posiz LZ
        SimMovL.EliM.LandingZoneL.LZvector = new Vector2 (SimMovL.EliM.LandingZoneL.LZposition.X, SimMovL.EliM.LandingZoneL.LZposition.Y);

        int disposerContent = (StaticSimulator.shipY1Y2spacing) / SimMovL.EliM.initEliNum; // calcolo degli interspazi sul ponte
        // per la disposizione grafica degli elicotteri
        // poisiziono elicotteri sul ponte
        foreach (Elicottero eli in SimMovL.EliM.ElicotteriList) {
            eli.rotation = StaticSimulator.SetRotationAngle (StaticSimulator.InitialRotationAsset); // prende gradi

            eli.elivector2d = new Vector2 (eli.elivector2d.X + StaticSimulator.shipborderLandX ,
                eli.elivector2d.Y + StaticSimulator.shipborderLandYdown- 20- (eli.NumEli) * disposerContent);
        }

        Window.AllowUserResizing = true; // L'untente pu' dimesionare la finestra
        graphics.IsFullScreen = true;      // simulazione windowed
        graphics.ApplyChanges ();        // applico i cambiamenti alla grafica
        base.Initialize ();
    }
    // METODO DI CONTENT: viene utilizzato per caricare le informazioni gfrafiche
    // immagini, suono, filmati...
    protected override void LoadContent () // solo grafica
    {
        // All'interno del metodo LoadContent() ci sono
        //tutte le chiamate ai file media da caricare.
        background = Content.Load<Texture2D> ("terra4"); // sfondo
        SimMovL.EliM.EliTexture[0] = Content.Load<Texture2D> ("Helicopter80_1");//elicottero pale aperte
        SimMovL.EliM.EliTexture[1] = Content.Load<Texture2D> ("Helicopter80_2");//elicottero pale aperte
        SimMovL.EliM.EliTexture[2] = Content.Load<Texture2D> ("Helicopter80_3");//elicottero pale aperte
        SimMovL.EliM.EliTexture[3] = Content.Load<Texture2D> ("Helicopter80_4");//elicottero pale aperte
        SimMovL.EliM.EliTexture[4] = Content.Load<Texture2D> ("Helicopter80_5");//elicottero pale aperte
        SimMovL.EliM.EliTexture[5] = Content.Load<Texture2D> ("Helicopter80x");//elicottero pale aperte
        SimMovL.EliM.EliTexture[6] = Content.Load<Texture2D> ("HeliFolded"); // elicottero pale piegate
        SimMovL.EliM.CrashedEli = Content.Load<Texture2D> ("skull"); // crash eli
        bar = Content.Load<Texture2D> ("barj"); // barra visualizzazione finale
        RectangleTexture = Content.Load<Texture2D> ("green-rectangle-hi");// rettangolo
        SimMovL.EliM.LandingZoneL.LZtexture = Content.Load<Texture2D> ("LZ1"); // landing zone
        SimMovL.TroopM.Troopstexture = Content.Load<Texture2D> ("troops.png"); // truppe
        font = Content.Load<SpriteFont> ("Sprite2"); // carico il font
        font2 = Content.Load<SpriteFont> ("Sprite5");//
        CargoTexture = Content.Load<Texture2D> ("pallet60");
        lineTexture = Content.Load<Texture2D> ("line");
    }
    // Quando si deve liberare la memoria di determinati
    //oggetti, va utilizzato questo metodo che viene esegui-
    //to nel momento in cui la classe che lo contiene viene
    //chiusa.
    protected override void UnloadContent ()
    {
        Content.Unload ();
    }


    #region [UPDATE LOGIC LOOP]
    //-----------------------------------------UPDATE LOGIC-----------------------------------------

    // logica di update per la grafica dle simulatore
    // LA PARTE DI UPDATE LOGICO NON GRAFICO VIENE DELEGATA ALLA CLASSE SimMover
    protected override void Update (GameTime gameTime)
    {
        if (!SimMovL.EndProgramState)
        {
            i++;    if (i == 5) i = 0; // sprite COUNTER , VARIA lo sprite degli elicotteri in volo per dare l'effetto pale rotanti

            SimMovL.LoopforUpdate (elapsedTime1); // LOOP UPDATER - e' il cuore della gestione logica
                                  // viene effettuato un update logico di truppe cargo elicotteri e posizioni
                                  //vengono determinate le posizioni e cosa fare
                                  // la gestione è lasciata al simumover, la classe di logica di movimento e simulazione
            base.Update (gameTime);
        }
    }
```

```
//-----------------------------------------------------------------------------------------------
#endregion [FINE UPDATE LOGIC]


// DRAWING-------------------------------------------------------------------------------------
// metodo di disegno contiene le immagini da disegnare
// l'ordine è importante in quanto definisce la visuale layerizzata
protected override void Draw (GameTime gameTime) // loop di disegno
{
    GraphicsDevice.Clear (Color.TransparentBlack);
    // logica di update per la grafica dle simulatore
                elapsedTime = (float)gameTime.ElapsedGameTime.TotalSeconds; // elapsed per speed
                elapsedTime1 =(float)gameTime.ElapsedGameTime.TotalSeconds; // lapsed fot eli
                elapsedtimeRotationEli =(float)gameTime.ElapsedGameTime.TotalSeconds;

    #region [INIZIO DRAWING LOOP]

    spriteBatch.Begin (); //inizio spritebatch
    // null indica valori di default


    // disegna LZ
    if(!SimMovL.EndProgramState)
    {
        //disegna sfondo
            spriteBatch.Draw (background, GraphicsDevice.Viewport.Bounds, Color.White);

        spriteBatch.Draw (SimMovL.EliM.LandingZoneL.LZtexture,SimMovL.EliM.LandingZoneL.LZvector,
                          null, Color.White, 0f, new Vector2 (SimMovL.EliM.LandingZoneL.LZtexture.Width / 2,
                          SimMovL.EliM.LandingZoneL.LZtexture.Height / 2),
                          SimMovL.EliM.LandingZoneL.SpriteEffectMath (), SpriteEffects.None, 0f);

        // L'ORDINE DI DISEGNO CONTA
        this.DrawGlobalInfo (); // disegna info GENERALI
        this.drawSold_Cargo (); // disegna info OPZIONALI ELI
        this.drawEli (); // disegna elicotteri

    }
        else  this.WriteScreenEndLoopInfo (); // se la simulazione è terminata mostra i risultati

        this.checkKey (); // keyboard posso regolare alcuni parametri
        spriteBatch.End (); // termine spritebatch
#endregion
    base.Draw (gameTime);

}
//-----------------------------------------------------------------------------------------------------


#region [CHECK CONTROL KEY SECTION]
// vivne usate per modificare alcuni parametri all'interno della simulazione
// regolo alcuni parametri tramite la tastiera
void checkKey ()
{

    if (Keyboard.GetState ().IsKeyDown (Keys.PageUp)) {
        if (SimMovL.Speed < 160) SimMovL.Speed++;

    }
    if (Keyboard.GetState ().IsKeyDown (Keys.PageDown))
    if (SimMovL.Speed > 60) SimMovL.Speed--;


    // premi escape per uscire dalla scermata grafica
    if (Keyboard.GetState ().IsKeyDown (Keys.Escape))
    {
        SimMovL.EliM.MainTime.Stop(); // stop timer
        Gtk.Application.Quit ();
        Exit ();
    }
}
#endregion

// DISEGNA GLI ELICOTTERI
void drawEli ()
{



    // draw crashed eli
    if (SimMovL.EliM.CrashList.Count>0)
        foreach (Elicottero eli in SimMovL.EliM.CrashList)
            spriteBatch.Draw ( SimMovL.EliM.CrashedEli, eli.elivector2d, null, Color.White, -MathHelper.PiOver2 ,
                new Vector2 (SimMovL.EliM.CrashedEli.Width / 2, SimMovL.EliM.CrashedEli.Height / 2),
                0.6f, SpriteEffects.None, 0f);//mostra l'immagine degli elicotteri su video

    // DISEGNO ELICOTTERI
    foreach (Elicottero eli in SimMovL.EliM.ElicotteriList) {

        if (eli.PosEli != false) {// se poseli == false helo in hangar NON VISIBILE
            //eli pale paerte
            if (eli.PosEli && !eli.IsFlying && eli.isRequiredForOP &&eli.IsBladeSpread) { // posizione sul ponte eli non vola
                spriteBatch.Draw (SimMovL.EliM.EliTexture[5], eli.elivector2d, null, Color.White, eli.rotation,
                    new Vector2 (SimMovL.EliM.EliTexture[5].Width / 2, SimMovL.EliM.EliTexture[5].Height / 2),
                    0.7f, SpriteEffects.None, 0f);//mostra l'immagine degli elicotteri su video
            }
            // eli pale piegate
            if (eli.PosEli && !eli.IsFlying && eli.isRequiredForOP &&!eli.IsBladeSpread) { // posizione sul ponte eli non vola
                spriteBatch.Draw (SimMovL.EliM.EliTexture[6], eli.elivector2d, null, Color.White, eli.rotation,
                    new Vector2 (SimMovL.EliM.EliTexture[6].Width / 2, SimMovL.EliM.EliTexture[6].Height / 2),
                    0.7f, SpriteEffects.None, 0f);//mostra l'immagine degli elicotteri su video
            }
            // se l'elicottero vola livellato e non è in holding
            if (eli.IsFlying && !eli.IsHolding && !eli.isOnDestination && !eli.isLZ) {

                // eli in volo
                spriteBatch.Draw (SimMovL.EliM.EliTexture[i], eli.elivector2d, null, Color.White, eli.rotation,
                    new Vector2 (SimMovL.EliM.EliTexture[i].Width / 2,SimMovL.EliM.EliTexture[i].Height / 2),
                     eli.quota, SpriteEffects.None, 0f);//mostra l'immagine degli elicotteri su video
            }

            // eli IS holding
            if (eli.IsHolding) { // il vettore di rotazione permette la rotazione sul centro dell'immagine
                spriteBatch.Draw (SimMovL.EliM.EliTexture[i], eli.elivector2d, null, Color.White, eli.rotation,
                    new Vector2 (0, 100), eli.quota, SpriteEffects.None, 0f);//mostra l'immagine degli elicotteri su video
            }

            // elicottero sulla LZ
            if (eli.isLZ) {
```

```
                    spriteBatch.Draw (SimMovL.EliM.EliTexture[i], eli.elivector2d, null, Color.White, eli.rotation,
                        new Vector2 (SimMovL.EliM.EliTexture[i].Width / 2, SimMovL.EliM.EliTexture[i].Height / 2),
                         0.5f, SpriteEffects.None, 0f);//mostra l'immagine degli elicotteri su video
                }
            } // termine if NON VISUALIZZA NULLA SE NON CI SONO ELICOTTERI SUL PONTE
            this.DrawEliInfo (eli);
        }
    }

    // DISEGNO CARGO SU LZ
    void drawSold_Cargo ()
    {

        // mostra il cargo sulla LZ
        foreach (Cargo cargo in SimMovL.EliM.LandingZoneL.LZCargo) {
            if (cargo.isLand)
                spriteBatch.Draw (CargoTexture, cargo.CargoVector, null, Color.White, 0f,
                    new Vector2 (CargoTexture.Width / 2, CargoTexture.Height / 2),
                     0.5f, SpriteEffects.None, 0f);//mostra l'immagine degli elicotteri su video

        }

        // mostra le truppe sulla LZ
        foreach (Soldier troop in SimMovL.EliM.LandingZoneL.LZSoldierList) {
            if (troop.IsLand)
                spriteBatch.Draw (SimMovL.TroopM.Troopstexture, troop.vectoreTroop, null, Color.White, 0f,
                    new Vector2 (SimMovL.TroopM.Troopstexture.Width / 2, SimMovL.TroopM.Troopstexture.Height / 2),
                    0.35f, SpriteEffects.None, 0f);//mostra l'immagine degli elicotteri su video
        }
    }
    // ritorna vero se il cargo esiste ed è 0 count
    // il metodo serve per i dati su mission failed
    private bool CheckCargoBoolZero()
    {
        if (SimMovL.CargoM != null && SimMovL.CargoM.CargoList != null)
        {
            if (SimMovL.CargoM.CargoList.Count == 0) return true;
            else return false;
        }
            else  return true;
            }

    // DRAW END LOOP DATA INFORMATIONS
    // IL METODO SERVE PER SCRIVERE E FORMATTARE I DATI VISIBILI A FINE SIMULAZIONE
    // I DATI VENGONO PRELEVATI DA UNA LISTA DI STRUTTURE IN CUI SONO STATI SALVATI
    // I DATI FONDAMENTALI DELLA SIMULAZIONE.
    void WriteScreenEndLoopInfo ()
    {

        int totInfo = SimMovL.WinI.infoTimeList.Count - 2; // numero totale di elementi info per eli senza start e fine
        int deY = 768 / (SimMovL.EliM.initEliNum+1); // delta Y per rappresentazione

        int timeD = ((int)SimMovL.WinI.infoTimeList.Find (x => x.Elirec == -2).timer.TotalSeconds / 20)+1; // blocchi di 20 minuti
        int deX = 900 / timeD; // lunghezza in px di un blocco da 20 min
        float minPix = (float)deX / 20; // lunghezza in pix di 1 min

        if (SimMovL.TroopM.TroopList.Count == 0 && CheckCargoBoolZero ()) {
             // se il numero di truppe da trasportare ==0 e il cargo è == 0
            // simulazione terminata con successo altrimenti simulazione fallita

            int y = 0; // indicatore per posizione
            int y1 = 0; // indicatore per posizione
            for (int v = 1; v <= timeD; v++) { // disegno le barre verticali che designano blocchi da 20 minuti
                spriteBatch.Draw (lineTexture, new Viewport (new Rectangle (v * deX +30 , 0, 3, 768)).Bounds, Color.White);
                spriteBatch.DrawString (font, "T: " + 20 * v, new Vector2 (v * deX+30, 10), Color.LightGreen);
            }

            for (int w = 0; w < SimMovL.EliM.initEliNum; w++)
             {  // disegno i rettangoli orizzonatali

                spriteBatch.Draw (bar, new Viewport (new Rectangle (0, deY * (w+1) - 85+30, 1024, 60)).Bounds, Color.White);
                spriteBatch.DrawString (font,"ELI#"+ SimMovL.WinI.infoTimeList.Find(x=>x.Elirec==w).Elirec ,
                    new Vector2 (10,deY *(w+1) - 65+ 30), Color.Red);
                // disegno le informazioni necessarie in ogni barra orizzonatale
                foreach (InfoWindow.timeInfoStruct timeStr in  ( SimMovL.WinI.infoTimeList.FindAll(x=>x.Elirec==w))) {
                    if (y >= 24)y = 0; else y=y+8; // correzione pos scritta
                    if (y1 >2) y1 = 0; else y1=y1+1; // correzione pos scritta
                    spriteBatch.DrawString (font2, timeStr.timer.TotalSeconds.ToString(),
                        new Vector2 ((float)timeStr.timer.TotalSeconds*minPix + 30, deY *(w+1)-10-y1*5),
                        Color.Blue,0f, new Vector2(0,0),0.8f,SpriteEffects.None,0f); // draw timers

                    spriteBatch.DrawString (font2, timeStr.info,
                        new Vector2 ((float)timeStr.timer.TotalSeconds*minPix + 30, deY * (w+1) - 80+y+25),
                        Color.Red,0f, new Vector2(0,0),0.8f,SpriteEffects.None,0f);// draw info
                }
            }

        } // end if
        // altrimenti la simulazione è FALLITA
        else spriteBatch.DrawString (font, " SIMULAZIONE FALLITA ", new Vector2 (50, 30), Color.GhostWhite);
}

    // SCRITTURA INFORMAZIONI GLOBALI IN ALTO A DESTRA
    void DrawGlobalInfo ()
    {
        spriteBatch.Draw (RectangleTexture, new Vector2(71,104), null, Color.White, MathHelper.PiOver2,
            new Vector2 (RectangleTexture.Width / 2, RectangleTexture.Height / 2),
             0.38f, SpriteEffects.None, 0f);//mostra l'immagine degli elicotteri su video
        spriteBatch.DrawString (font, "MINUTI TRASCORSI: "+(int)this.SimMovL.EliM.MainTime.GetTime ().TotalSeconds,
            new Vector2 (10, 10), Color.Black);
        spriteBatch.DrawString (font, "SPOT STATUS: " + SimMovL.SpotM.Stato.ToString (),
             new Vector2 (10, 30), Color.Black);
        spriteBatch.DrawString (font, "TROOPS STATUS: " + SimMovL.TroopM.Status.ToString (),
            new Vector2 (10, 50), Color.Black);
        spriteBatch.DrawString (font, "HANGAR STATUS: " + SimMovL.EliM.Stato.ToString (),
            new Vector2 (10, 70), Color.Black);
        spriteBatch.DrawString (font, "TRUPPE RESTANTI: " + SimMovL.TroopM.TroopList.Count,
            new Vector2 (10, 90), Color.Black);
        spriteBatch.DrawString (font, "TRUPPE SU LZ: " + SimMovL.EliM.LandingZoneL.LZSoldierList.Count,
            new Vector2 (10, 110), Color.Black);
        spriteBatch.DrawString (font, "FORMATION SPEED: " + SimMovL.Speed,
            new Vector2 (10, 130), Color.Black);
        spriteBatch.DrawString (font, "HOLDING: " + SimMovL.EliM.HoldingP.EliHolding.Count,
            new Vector2 (10, 150), Color.Black);
        if (SimMovL.CargoM != null)
            spriteBatch.DrawString (font, "CARGO SU LZ: " + SimMovL.EliM.LandingZoneL.LZCargo.Count,
                 new Vector2 (10, 170), Color.Black);
        else spriteBatch.DrawString (font, "CARGO SU LZ: null", new Vector2 (10, 170), Color.Black);

        if (SimMovL.CargoM != null)
```

```
            spriteBatch.DrawString (font, "CARGO RESTATE: " + SimMovL.CargoM.CargoList.Count,
                new Vector2 (10, 190), Color.Black);
        else spriteBatch.DrawString (font, "CARGO RESTANTE: null", new Vector2 (10, 190), Color.Black);

        // disegna le scritte per gli elicotteri crashati
        if (SimMovL.EliM.CrashList.Count > 0) {
            spriteBatch.DrawString (font, "ELI CADUTI: " + SimMovL.EliM.CrashList.Count, new Vector2 (150, 10), Color.Red);
            // uomini morti e cargo perso scrittura dati
            int i = 0; int k = 0;
            foreach (Elicottero eli in SimMovL.EliM.CrashList) {
                i = i + eli.EliSoldierList.Count;
                k = k + eli.EliCargoList.Count;
            }
            spriteBatch.DrawString (font, "UOMINI MORTI: " +i , new Vector2 (150, 30), Color.Red);
            spriteBatch.DrawString (font, "CARGO PERSO: " +k , new Vector2 (150, 50), Color.Red);
        }
    }

    // DRAW ELI INFORMATIONS -- PREMERE IL TASTO CORRISPONDENTE AL RECORD DELL'ELICOTTERO PER VISUALIZZA LE INFORMAZIONI

    void DrawEliInfo (Elicottero eli)
    {
        if (eli.LowFuel)
            spriteBatch.DrawString (font2, "LOW FUEL" , new Vector2 (eli.elivector2d.X -30, eli.elivector2d.Y), Color.Blue);
        if (eli.isHotRef)
            spriteBatch.DrawString (font2, "REFUELING", new Vector2 (eli.elivector2d.X +30, eli.elivector2d.Y), Color.Blue);
        if (eli.IsBoarding)
            spriteBatch.DrawString (font2, "BOARDING TRUPPE", new Vector2 (eli.elivector2d.X-15, eli.elivector2d.Y+30), Color.Blue);
        if (eli.IsBoardingCargo)
            spriteBatch.DrawString (font2, "BOARDING CARGO", new Vector2 (eli.elivector2d.X-15, eli.elivector2d.Y+30), Color.Blue);
        if (eli.hasSpotReserved && !eli.PosEli)
            spriteBatch.DrawString (font2, eli.IdEli+" IN MOVIMENTAZIONE", new Vector2 (eli.elivector2d.X-30, eli.elivector2d.Y-15), Color.Blue);
        if (!eli.IsRunning && eli.PosEli)
            spriteBatch.DrawString (font2, eli.IdEli+" PRE_START CHECKS", new Vector2 (eli.elivector2d.X-30, eli.elivector2d.Y+15), Color.Blue);
        if (eli.IsRunning && !eli.IsBladeSpread)
            spriteBatch.DrawString (font2, eli.IdEli+" BLADE SPREADING", new Vector2 (eli.elivector2d.X-30, eli.elivector2d.Y+15), Color.Blue);

        int keypress=10;

        Keys[] keyArray = (Keyboard.GetState().GetPressedKeys()); // pressione dei tasti numpad 1 - 9 per visualizzazione info elicottero
        if (keyArray.Length > 0)
         {
            if (keyArray [0] >= Keys.NumPad1 && keyArray [0] <= Keys.NumPad9) {
                switch (keyArray [0]) {
                case Keys.NumPad1:
                    keypress = 1; break;
                case Keys.NumPad2:
                    keypress = 2;break;

                case Keys.NumPad3:
                    keypress = 3;break;
                case Keys.NumPad4:
                    keypress = 4;break;
                case Keys.NumPad5:
                    keypress = 5;break;
                case Keys.NumPad6:
                    keypress = 6;break;
                case Keys.NumPad7:
                    keypress = 7;break;
                case Keys.NumPad8:
                    keypress = 8;break;
                case Keys.NumPad9:
                    keypress = 9;break;
                }
            }
        }

        if (eli.NumEli==keypress-1 )
        {
            spriteBatch.DrawString (font2, eli.IdEli, new Vector2 (eli.elivector2d.X -10, eli.elivector2d.Y), Color.Blue); // nome eli

            spriteBatch.DrawString (font2, "RUNNING: " + eli.IsRunning.ToString (), new Vector2
                (eli.elivector2d.X + 20, eli.elivector2d.Y - StaticSimulator.spacing), Color.Red);
            spriteBatch.DrawString (font2, "BLADE SPREAD: " + eli.IsBladeSpread.ToString (), new Vector2
                (eli.elivector2d.X + 20, eli.elivector2d.Y - StaticSimulator.spacing * 2), Color.Red);
            spriteBatch.DrawString (font2, "READY: " + eli.isREADYstatus.ToString (), new Vector2
                (eli.elivector2d.X + 20, eli.elivector2d.Y -  StaticSimulator.spacing * 3), Color.Red);
            spriteBatch.DrawString (font2, "REQUIRED: " + eli.isRequiredForOP, new Vector2
                (eli.elivector2d.X + 20, eli.elivector2d.Y -  StaticSimulator.spacing * 4), Color.Red);
            spriteBatch.DrawString (font2, "FUEL: " + eli.Fuel, new Vector2
                (eli.elivector2d.X +20, eli.elivector2d.Y -  StaticSimulator.spacing * 5), Color.Red);

            spriteBatch.DrawString (font2, "FLY: " + eli.IsFlying, new Vector2
                (eli.elivector2d.X - 110, eli.elivector2d.Y -  StaticSimulator.spacing), Color.Red);
            spriteBatch.DrawString (font2, "BLOCKED: " + eli.IsBlocked, new Vector2
                (eli.elivector2d.X - 110, eli.elivector2d.Y - StaticSimulator.spacing * 2), Color.Red);
            spriteBatch.DrawString (font2, "IS HOLDING: " + eli.IsHolding, new Vector2
                (eli.elivector2d.X - 110, eli.elivector2d.Y -  StaticSimulator.spacing* 3), Color.Red);
            spriteBatch.DrawString (font2, "TRUPPE : " + eli.EliSoldierList.Count, new Vector2
                (eli.elivector2d.X - 110, eli.elivector2d.Y - StaticSimulator.spacing*4), Color.Red);
            if (SimMovL.CargoM != null)
            spriteBatch.DrawString (font2, "CARGO : " + eli.EliCargoList.Count, new Vector2
                    (eli.elivector2d.X -110, eli.elivector2d.Y -  StaticSimulator.spacing * 5), Color.Red);


            spriteBatch.DrawString (font2, "DIREZIONE: " + eli.DirToGo.ToString (), new Vector2
                (eli.elivector2d.X - 110, eli.elivector2d.Y +  StaticSimulator.spacing), Color.Red);
            spriteBatch.DrawString (font2, "LOW FUEL: " + eli.LowFuel, new Vector2
                (eli.elivector2d.X - 110, eli.elivector2d.Y +  StaticSimulator.spacing * 2), Color.Red);
            spriteBatch.DrawString (font2, "ELI FULL: " + eli.IsFull, new Vector2
                (eli.elivector2d.X - 110, eli.elivector2d.Y +  StaticSimulator.spacing * 3), Color.Red);
            spriteBatch.DrawString (font2, "ANGLE toFLY: " + eli.AngleToFly, new Vector2
                 (eli.elivector2d.X - 110, eli.elivector2d.Y +  StaticSimulator.spacing * 4), Color.Red);
            spriteBatch.DrawString (font2, "IS SPEED: " + eli.EliSpeed, new Vector2
                (eli.elivector2d.X - 110, eli.elivector2d.Y +  StaticSimulator.spacing * 5), Color.Red);


            spriteBatch.DrawString (font2, "FREE W CARGO: " + eli.WCargoLeftOnBoard, new Vector2
                (eli.elivector2d.X + 20, eli.elivector2d.Y +  StaticSimulator.spacing), Color.Red);
            spriteBatch.DrawString (font2, "FREE W TROOPS: " + eli.WTroopsLeftOnBoard, new Vector2
                (eli.elivector2d.X + 20, eli.elivector2d.Y +  StaticSimulator.spacing * 2), Color.Red);
            spriteBatch.DrawString (font2, "ANGOLO ATTUALE: " + eli.rotation, new Vector2
                (eli.elivector2d.X + 20, eli.elivector2d.Y +  StaticSimulator.spacing * 3), Color.Red);
            spriteBatch.DrawString (font2, "DISTANZA AL PUNTO: " + eli.DIstanceToPnT, new Vector2
                (eli.elivector2d.X + 20, eli.elivector2d.Y +  StaticSimulator.spacing * 4), Color.Red);
        }
    } // termine metodo draw info
}
// termina classe XNA
```

```
}
// termina name space
```

7 TESTING

La fase di testing è stata effettuata tramite il supporto del log file risultato delle simulazioni proposte, come già descritto curante la simulazione di viene scritto un log con i principali dati dal quale si può determinare la linea degli eventi che se ne sviluppa ed il risultato finale. Il file log visualizza inoltre i dettagli su tutti i record inseriti.

La simulazione, qualora abbia successo, offre una visualizzazione grafica degli eventi proposti alla fine.

Si propongono alcuni test di verifica incollando il log del file. E' possibile visualizzare tutti gli orari di decollo e di atterraggio su nave o landing zone, di ingresso o uscita dall'holding point. L'ultimo record del file viene sempre identificato dall'orario di fine missione.

In figura viene mostrata la schermata grafica con cui viene effettuata la simulazione.

A sinistra è possibile visualizzare una serie di informazioni in tempo reale, è inoltre possibile osservare la posizione della LZ rappresentata con l'icona dell'elicottero cerchiato di rosso.

Gli elicotteri appariranno invece sul ponte di volo, sarà possibile visualizzare le informazioni per ogni elicottero con l'uso del NumPad premendo il tasto in corrispondenza al record dell'eli. Ad esempio per visualizzare il primo record basterà premere il tasto 1. Le info sugli elicotteri sono visibili anche quando l'elicottero non è visibile ma è posizionato in hangar.

La visualizzazione è stata testata e funziona correttamente correttamente.

7.1 TEST 1 : TESTO BASICO CON 1 ELICOTTERO, 2 CARGO E 25 TRUPPE

12-37 -> -- PROGRAMMA PARTITO --
12-44 -> MAIN WINDOW: Effettuato inserimento del NUMERO di elicotteri
12-45 -> OPZIONI ELICOTTERO: inserto nuovo record#__1 Elementi totali presenti: 1
12-45 -> Record_eli: 0 ID: eli 1 PosEli:False Cat: 1 Fuel: 1995 Spazio truppe: 1044 Spazio Cargo: 1062 MaxT-O weight: 8649 OffLoad Weight: 1000 Consumo: 892
12-45 -> MAIN WINDOW: Effettuato inserimento del NUMERO di truppe
12-45 -> TROOPS MANAGER: effettuata creazione manager per truppe
12-45 -> MAIN WINDOW: VISUALIZZUATA SCHERMATA OPZIONI TRUPPE
12-45 -> OPZIONI TRUPPE: effettuato l'inserimento di 25 soldati
12-45 -> MAIN WINDOW: Effettuato inserimento di ALTRO CARGO
12-45 -> OPZIONI CARGO: --> EFFETTUATA APERTURA FINESTRA OPZIONI CARGO <--
12-45 -> OPZIONI CARGO: inserto nuovo record#__1 Elementi totali presenti: 1
12-45 -> Record_cargo: 0 Tipo Cargo: cargo1 Peso Cargo: 792 Utenti necessari: 1
12-45 -> OPZIONI CARGO: inserto nuovo record#__2 Elementi totali presenti: 2
12-45 -> Record_cargo: 1 Tipo Cargo: cargo 2 Peso Cargo: 662 Utenti necessari: 1
12-45 -> MAIN WINDOW: inserita la distanza della landing zone dall'unità navale
12-45 -> MAIN WINDOW: Effettuato inserimento del NUMERO di SPOT
12-45 -> OPZIONI SPOT: EFFETTUATA APERTURA FINESTRA OPZIONI SPOT
12-45 -> OPZIONI SPOT: inserto nuovo record#__1 Elementi totali presenti: 1
12-45 -> Record_Spot: 0 Efficienza: True Spot occupato: False Da Eli: null GIORNO Cat1: True GIORNO Cat2: False GIORNO Cat3: False NOTTE Cat1: True NOTTE Cat2: False NOTTE Cat3: False
12-45 -> SIMULATOR: simulatore creato. Effettuo l'inizializzazione....
12-45 -> - SIMULATORE: INIZIO MOVIEMENTAZIONI PONTE
12-45 -> SIMULATOR : controllo effettiva NECESSITA' degli elicotteri inseriti per la missione
12-45 -> SIMULATORE: ciclo inserimento e disposizione iniziale ultimato
12-45 -> SIMULATORE: INIZIO SIMULAZION, starting X timer"
12-45 -> CARGO DISPOSER: è stato assegnato: cargo1 , restano ancora 1 elementi CARGO da imbarcare
12-46 -> Spot trovato per assegnazione: 0
12-46 -> DISPOSER: movimentato elicottero : eli 1 su spot: 0
12-46 -> TROOPS DISPOSER: ponte di volo occupato da: 1 Elicotteri sul ponte pronti per l'imbarco di 20 truppe
12-46 -> ELI eli 1 DECOLLA time: 00:00:36
12-46 -> ELI eli 1 ENTRA HP: 00:00:42
12-46 -> HOLDING POINT : elicottero : eli 1 lascia l'HP time: 00:00:42
12-47 -> ELI eli 1 SULLA LZ time: 00:01:08
12-47 -> LANDING ZONE : elicottero : eli 1 lascia LZ time: 00:01:14
12-47 -> ELICOTTERO APPONTA: eli 1 time: 00:01:44
12-47 -> CARGO DISPOSER: è stato assegnato: cargo 2 , restano ancora 0 elementi CARGO da imbarcare
12-47 -> TROOPS DISPOSER: ponte di volo occupato da: 1 Elicotteri sul ponte pronti per l'imbarco di 5 truppe
12-47 -> ELI eli 1 DECOLLA time: 00:01:58
12-47 -> ELI eli 1 ENTRA HP: 00:02:03
12-47 -> HOLDING POINT : elicottero : eli 1 lascia l'HP time: 00:02:03
12-48 -> ELI eli 1 SULLA LZ time: 00:02:29
12-48 -> LANDING ZONE : elicottero : eli 1 lascia LZ time: 00:02:35
12-49 -> ELICOTTERO APPONTA: eli 1 time: 00:03:05
12-49 -> TERMINE OPERAZIONI 00:03:05

7.2 TEST2: 4 ELICOTTERI, 3 SPOT, 3 CARGO, 100 TRUPPE

01-48 -> -- PROGRAMMA PARTITO --
01-48 -> MAIN WINDOW: Effettuato inserimento del NUMERO di elicotteri: 4
01-49 -> OPZIONI ELICOTTERO: inserto nuovo record#__1 Elementi totali presenti: 1
01-49 -> Record_eli: 0 ID: eli 1 PosEli:False Cat: 1 Fuel: 2005 Spazio truppe: 1339 Spazio Cargo: 0 MaxT-O weight: 8815 OffLoad Weight: 1000 Consumo: 914
01-49 -> OPZIONI ELICOTTERO: inserto nuovo record#__2 Elementi totali presenti: 2
01-49 -> Record_eli: 1 ID: eli 2 PosEli:False Cat: 1 Fuel: 2089 Spazio truppe: 1404 Spazio Cargo: 1155 MaxT-O weight: 12917 OffLoad Weight: 1000 Consumo: 852
01-49 -> OPZIONI ELICOTTERO: inserto nuovo record#__3 Elementi totali presenti: 3
01-49 -> Record_eli: 2 ID: eli 3 PosEli:False Cat: 1 Fuel: 2226 Spazio truppe: 1524 Spazio Cargo: 0 MaxT-O weight: 9370 OffLoad Weight: 1000 Consumo: 1076
01-50 -> OPZIONI ELICOTTERO: inserto nuovo record#__4 Elementi totali presenti: 4
01-50 -> Record_eli: 3 ID: eli 4 PosEli:False Cat: 1 Fuel: 2110 Spazio truppe: 1155 Spazio Cargo: 693 MaxT-O weight: 10090 OffLoad Weight: 1000 Consumo: 792
01-50 -> MAIN WINDOW: Effettuato inserimento del NUMERO di truppe: 100
01-50 -> TROOPS MANAGER: effettuata creazione manager per truppe
01-50 -> MAIN WINDOW: VISUALIZZUATA SCHERMATA OPZIONI TRUPPE
01-50 -> OPZIONI TRUPPE: effettuato l'inserimento di 100 soldati
01-50 -> MAIN WINDOW: Effettuato inserimento di ALTRO CARGO: 3
01-50 -> OPZIONI CARGO: --> EFFETTUATA APERTURA FINESTRA OPZIONI CARGO <--
01-50 -> OPZIONI CARGO: inserto nuovo record#__1 Elementi totali presenti: 1
01-50 -> Record_cargo: 0 Tipo Cargo: cargo 1 Peso Cargo: 792 Utenti necessari: 2
01-50 -> OPZIONI CARGO: inserto nuovo record#__2 Elementi totali presenti: 2
01-50 -> Record_cargo: 1 Tipo Cargo: cargo2 Peso Cargo: 879 Utenti necessari: 2
01-50 -> OPZIONI CARGO: inserto nuovo record#__3 Elementi totali presenti: 3
01-50 -> Record_cargo: 2 Tipo Cargo: cargo 3 Peso Cargo: 184 Utenti necessari: 2
01-51 -> MAIN WINDOW: inserita la distanza della landing zone dall'unità navale: 70 NM
01-51 -> MAIN WINDOW: Effettuato inserimento del NUMERO di SPOT: 3
01-51 -> OPZIONI SPOT: EFFETTUATA APERTURA FINESTRA OPZIONI SPOT
01-51 -> OPZIONI SPOT: inserto nuovo record#__1 Elementi totali presenti: 1
01-51 -> Record_Spot: 0 Efficienza: True Spot occupato: False Da Eli: null GIORNO Cat1: True GIORNO Cat2: False GIORNO Cat3: False NOTTE Cat1: True NOTTE Cat2: False NOTTE Cat3: False
01-51 -> OPZIONI SPOT: inserto nuovo record#__2 Elementi totali presenti: 2
01-51 -> Record_Spot: 1 Efficienza: True Spot occupato: False Da Eli: null GIORNO Cat1: True GIORNO Cat2: False GIORNO Cat3: False NOTTE Cat1: True NOTTE Cat2: False NOTTE Cat3: False
01-51 -> OPZIONI SPOT: inserto nuovo record#__3 Elementi totali presenti: 3

01-51 -> Record_Spot: 2 Efficienza: True Spot occupato: False Da Eli: null GIORNO Cat1: True GIORNO Cat2: False GIORNO Cat3: False NOTTE Cat1: True NOTTE Cat2: False NOTTE Cat3: False
01-51 -> SIMULATOR: simulatore creato. Effettuo l'inizializzazione....
01-51 -> - SIMULATORE: INIZIO MOVIEMENTAZIONI PONTE
01-51 -> SIMULATOR : controllo effettiva NECESSITA' degli elicotteri inseriti per la missione
01-51 -> SIMULATORE: ciclo inserimento e disposizione iniziale ultimato
01-51 -> SIMULATORE: INIZIO SIMULAZIONE, starting X timer"
01-51 -> CARGO DISPOSER: è stato assegnato: cargo 1 , restano ancora 2 elementi CARGO da imbarcare
01-51 -> CARGO DISPOSER: è stato assegnato: cargo 3 , restano ancora 1 elementi CARGO da imbarcare
01-51 -> Spot trovato per assegnazione: 0
01-51 -> DISPOSER: movimentato elicottero : eli 1 su spot: 0
01-51 -> Spot trovato per assegnazione: 1
01-51 -> DISPOSER: movimentato elicottero : eli 3 su spot: 1
01-51 -> Spot trovato per assegnazione: 2
01-51 -> DISPOSER: movimentato elicottero : eli 2 su spot: 2
01-51 -> DISPOSER: movimentazione elicottero eli 4 non effettuabile
01-52 -> TROOPS DISPOSER: ponte di volo occupato da: 3 Elicotteri sul ponte pronti per l'imbarco di 45 truppe
01-52 -> ELI eli 1 DECOLLA time: 00:00:42
01-52 -> ELI eli 3 DECOLLA time: 00:00:42
01-52 -> ELI eli 2 DECOLLA time: 00:00:42
01-52 -> Spot trovato per assegnazione: 0
01-52 -> DISPOSER: movimentato elicottero : eli 4 su spot: 0
01-52 -> ELI eli 3 ENTRA HP: 00:00:47
01-52 -> ELI eli 2 ENTRA HP: 00:00:47
01-52 -> ELI eli 1 ENTRA HP: 00:00:50
01-52 -> TROOPS DISPOSER: ponte di volo occupato da: 1 Elicotteri sul ponte pronti per l'imbarco di 12 truppe
01-52 -> ELI eli 4 DECOLLA time: 00:01:18
01-52 -> ELI eli 4 ENTRA HP: 00:01:26
01-52 -> HOLDING POINT : elicottero : eli 1 lascia l'HP time: 00:01:26
01-52 -> HOLDING POINT : elicottero : eli 2 lascia l'HP time: 00:01:26
01-52 -> HOLDING POINT : elicottero : eli 3 lascia l'HP time: 00:01:26
01-52 -> HOLDING POINT : elicottero : eli 4 lascia l'HP time: 00:01:26
01-53 -> ELI eli 1 SULLA LZ time: 00:02:02
01-53 -> ELI eli 4 SULLA LZ time: 00:02:03
01-53 -> LANDING ZONE : elicottero : eli 1 lascia LZ time: 00:02:04
01-53 -> ELI eli 2 SULLA LZ time: 00:02:04
01-53 -> ELI eli 3 SULLA LZ time: 00:02:05
01-53 -> LANDING ZONE : elicottero : eli 3 lascia LZ time: 00:02:07
01-53 -> LANDING ZONE : elicottero : eli 4 lascia LZ time: 00:02:09
01-53 -> LANDING ZONE : elicottero : eli 2 lascia LZ time: 00:02:10
01-54 -> ELI eli 2 ENTRA HP: 00:02:44
01-54 -> ELICOTTERO APPONTA: eli 1 time: 00:02:46
01-54 -> ELICOTTERO APPONTA: eli 3 time: 00:02:48
01-54 -> ELICOTTERO APPONTA: eli 4 time: 00:02:51
01-54 -> TROOPS DISPOSER: ponte di volo occupato da: 3 Elicotteri sul ponte pronti per l'imbarco di 14 truppe
01-54 -> TROOPS DISPOSER: ponte di volo occupato da: 3 Elicotteri sul ponte pronti per l'imbarco di 16 truppe
01-54 -> ELI eli 1 DECOLLA time: 00:02:55
01-54 -> HOLDING POINT : elicottero : eli 2 lascia l'HP time: 00:02:55
01-54 -> TROOPS DISPOSER: ponte di volo occupato da: 2 Elicotteri sul ponte pronti per l'imbarco di 12 truppe
01-54 -> ELI eli 3 DECOLLA time: 00:02:57
01-54 -> ELI eli 4 DECOLLA time: 00:03:00
01-54 -> ELI eli 3 ENTRA HP: 00:03:00
01-54 -> ELI eli 1 ENTRA HP: 00:03:02
01-54 -> ELICOTTERO APPONTA: eli 2 time: 00:03:03
01-54 -> CARGO DISPOSER: è stato assegnato: cargo2 , restano ancora 0 elementi CARGO da imbarcare
01-54 -> ELI eli 4 ENTRA HP: 00:03:07
01-54 -> TROOPS DISPOSER: ponte di volo occupato da: 1 Elicotteri sul ponte pronti per l'imbarco di 1 truppe
01-54 -> ELI eli 2 DECOLLA time: 00:03:17
01-54 -> ELI eli 2 ENTRA HP: 00:03:20
01-54 -> HOLDING POINT : elicottero : eli 1 lascia l'HP time: 00:03:20
01-54 -> HOLDING POINT : elicottero : eli 2 lascia l'HP time: 00:03:20
01-54 -> HOLDING POINT : elicottero : eli 3 lascia l'HP time: 00:03:20
01-54 -> HOLDING POINT : elicottero : eli 4 lascia l'HP time: 00:03:20
01-55 -> ELI eli 1 SULLA LZ time: 00:03:57
01-55 -> ELI eli 2 SULLA LZ time: 00:03:57
01-55 -> ELI eli 4 SULLA LZ time: 00:03:58
01-55 -> ELI eli 3 SULLA LZ time: 00:03:58
01-55 -> LANDING ZONE : elicottero : eli 1 lascia LZ time: 00:03:59
01-55 -> LANDING ZONE : elicottero : eli 4 lascia LZ time: 00:04:00
01-55 -> LANDING ZONE : elicottero : eli 3 lascia LZ time: 00:04:00
01-55 -> LANDING ZONE : elicottero : eli 2 lascia LZ time: 00:04:03
01-56 -> ELI eli 2 ENTRA HP: 00:04:37
01-56 -> ELICOTTERO APPONTA: eli 1 time: 00:04:41
01-56 -> HOLDING POINT : elicottero : eli 2 lascia l'HP time: 00:04:41
01-56 -> ELICOTTERO APPONTA: eli 3 time: 00:04:41
01-56 -> ELICOTTERO APPONTA: eli 4 time: 00:04:42
01-56 -> ELICOTTERO APPONTA: eli 2 time: 00:04:50
01-56 -> TERMINE OPERAZIONI 00:04:50

7.3 TEST3: 1 ELICOTTERO, 1 SPOT, 0 CARGO, 1 TRUPPA , CHECK SIMULAZIONE FALLITA

La simulazione fallisce in quanto l'elicottero non ha sufficiente capacità di carburante per raggiungere la destinazione tornare indietro in sicurezza. Il peso del carburante viene ricalcolato sottraendo peso alle truppe o al cargo, tuttavia le correzioni risultano non sufficienti per l'effettuabilità della missione.

05-31 -> -- PROGRAMMA PARTITO --
05-31 -> MAIN WINDOW: Effettuato inserimento del NUMERO di elicotteri: 1
05-32 -> OPZIONI ELICOTTERO: inserto nuovo record#__1 Elementi totali presenti: 1

05-32 -> Record_eli: 0 ID: 1 PosEli:False Cat: 1 Fuel: 1312 Spazio truppe: 203 Spazio Cargo: 0 MaxT-O weight: 2718 OffLoad Weight: 1000 Consumo: 1500
05-32 -> MAIN WINDOW: Effettuato inserimento del NUMERO di truppe: 17
05-32 -> TROOPS MANAGER: effettuata creazione manager per truppe
05-32 -> MAIN WINDOW: VISUALIZZUATA SCHERMATA OPZIONI TRUPPE
05-32 -> OPZIONI TRUPPE: effettuato l'inserimento di 17 soldati
05-32 -> MAIN WINDOW: Effettuato inserimento di ALTRO CARGO: 0
05-32 -> MAIN WINDOW: WARNING !! INSERIRE UN NUMERO DI ELEMENTI CARGO MAGGIORE DI 0
05-32 -> MAIN WINDOW: inserita la distanza della landing zone dall'unità navale: 50 NM
05-32 -> MAIN WINDOW: Effettuato inserimento del NUMERO di SPOT: 1
05-32 -> OPZIONI SPOT: EFFETTUATA APERTURA FINESTRA OPZIONI SPOT
05-32 -> OPZIONI SPOT: inserto nuovo record#__1 Elementi totali presenti: 1
05-32 -> Record_Spot: 0 Efficienza: True Spot occupato: False Da Eli: null GIORNO Cat1: True GIORNO Cat2: False GIORNO Cat3: False NOTTE Cat1: True NOTTE Cat2: False NOTTE Cat3: False
05-32 -> SIMULATOR: simulatore creato. Effettuo l'inizializzazione....
05-32 -> - SIMULATORE: INIZIO MOVIMENTAZIONI PONTE
05-32 -> SIMULATOR : controllo effettiva NECESSITA' degli elicotteri inseriti per la missione
05-32 -> SIMULATORE: ciclo inserimento e disposizione iniziale ultimato
05-32 -> SIMULATORE: INIZIO SIMULAZIONE, starting X timer"
05-32 -> TERMINE OPERAZIONI 00:00:01
05-32 -> SIMULAZIONE FALLITA

7.4 TEST4: STRESS TEST _ 9 ELICOTTERI, 8 SPOT, 3 CARGO , 500 TRUPPE

06-17 -> -- PROGRAMMA PARTITO --
06-17 -> MAIN WINDOW: Effettuato inserimento del NUMERO di elicotteri: 9
06-17 -> OPZIONI ELICOTTERO: inserto nuovo record#__1 Elementi totali presenti: 1
06-17 -> Record_eli: 0 ID: eli1 PosEli:False Cat: 1 Fuel: 1785 Spazio truppe: 1478 Spazio Cargo: 1376 MaxT-O weight: 11143 OffLoad Weight: 1000 Consumo: 823
06-18 -> OPZIONI ELICOTTERO: inserto nuovo record#__2 Elementi totali presenti: 2
06-18 -> Record_eli: 1 ID: eli2 PosEli:False Cat: 1 Fuel: 1837 Spazio truppe: 1210 Spazio Cargo: 1192 MaxT-O weight: 8261 OffLoad Weight: 1000 Consumo: 833
06-19 -> OPZIONI ELICOTTERO: inserto nuovo record#__3 Elementi totali presenti: 3
06-19 -> Record_eli: 2 ID: eli3 PosEli:False Cat: 1 Fuel: 2405 Spazio truppe: 1644 Spazio Cargo: 0 MaxT-O weight: 11587 OffLoad Weight: 7111 Consumo: 808
06-20 -> OPZIONI ELICOTTERO: inserto nuovo record#__4 Elementi totali presenti: 4
06-20 -> Record_eli: 3 ID: eli4 PosEli:False Cat: 1 Fuel: 2152 Spazio truppe: 1303 Spazio Cargo: 0 MaxT-O weight: 9203 OffLoad Weight: 1000 Consumo: 739
06-20 -> OPZIONI ELICOTTERO: inserto nuovo record#__5 Elementi totali presenti: 5
06-20 -> Record_eli: 4 ID: eli5 PosEli:False Cat: 1 Fuel: 2016 Spazio truppe: 1376 Spazio Cargo: 1339 MaxT-O weight: 14081 OffLoad Weight: 1000 Consumo: 954
06-21 -> OPZIONI ELICOTTERO: inserto nuovo record#__6 Elementi totali presenti: 6
06-21 -> Record_eli: 5 ID: eli6 PosEli:False Cat: 1 Fuel: 2573 Spazio truppe: 1626 Spazio Cargo: 1552 MaxT-O weight: 11032 OffLoad Weight: 1000 Consumo: 1403
06-21 -> OPZIONI ELICOTTERO: inserto nuovo record#__7 Elementi totali presenti: 7
06-21 -> Record_eli: 6 ID: eli7 PosEli:False Cat: 1 Fuel: 2268 Spazio truppe: 1460 Spazio Cargo: 0 MaxT-O weight: 8815 OffLoad Weight: 1000 Consumo: 911
06-22 -> OPZIONI ELICOTTERO: inserto nuovo record#__8 Elementi totali presenti: 8
06-22 -> Record_eli: 7 ID: eli8 PosEli:False Cat: 1 Fuel: 2426 Spazio truppe: 1238 Spazio Cargo: 0 MaxT-O weight: 9536 OffLoad Weight: 1000 Consumo: 677
06-22 -> OPZIONI ELICOTTERO: inserto nuovo record#__9 Elementi totali presenti: 9
06-22 -> Record_eli: 8 ID: eli9 PosEli:False Cat: 1 Fuel: 2373 Spazio truppe: 1681 Spazio Cargo: 0 MaxT-O weight: 9037 OffLoad Weight: 1000 Consumo: 714
06-22 -> MAIN WINDOW: Effettuato inserimento del NUMERO di truppe: 500
06-22 -> TROOPS MANAGER: effettuata creazione manager per truppe
06-22 -> MAIN WINDOW: VISUALIZZUATA SCHERMATA OPZIONI TRUPPE
06-22 -> OPZIONI TRUPPE: effettuato l'inserimento di 500 soldati
06-22 -> MAIN WINDOW: Effettuato inserimento di ALTRO CARGO: 3
06-22 -> OPZIONI CARGO: --> EFFETTUATA APERTURA FINESTRA OPZIONI CARGO <--
06-22 -> OPZIONI CARGO: inserto nuovo record#__1 Elementi totali presenti: 1
06-22 -> Record_cargo: 0 Tipo Cargo: 1 Peso Cargo: 792 Utenti necessari: 3
06-23 -> OPZIONI CARGO: inserto nuovo record#__2 Elementi totali presenti: 2
06-23 -> Record_cargo: 1 Tipo Cargo: 2 Peso Cargo: 662 Utenti necessari: 2
06-23 -> OPZIONI CARGO: inserto nuovo record#__3 Elementi totali presenti: 3
06-23 -> Record_cargo: 2 Tipo Cargo: 3 Peso Cargo: 868 Utenti necessari: 3
06-23 -> MAIN WINDOW: inserita la distanza della landing zone dall'unità navale: 67 NM
06-23 -> MAIN WINDOW: Effettuato inserimento del NUMERO di SPOT: 8
06-23 -> OPZIONI SPOT: EFFETTUATA APERTURA FINESTRA OPZIONI SPOT
06-23 -> OPZIONI SPOT: inserto nuovo record#__1 Elementi totali presenti: 1
06-23 -> Record_Spot: 0 Efficienza: True Spot occupato: False Da Eli: null GIORNO Cat1: True GIORNO Cat2: False GIORNO Cat3: False NOTTE Cat1: True NOTTE Cat2: False NOTTE Cat3: False
06-23 -> OPZIONI SPOT: inserto nuovo record#__2 Elementi totali presenti: 2
06-23 -> Record_Spot: 1 Efficienza: True Spot occupato: False Da Eli: null GIORNO Cat1: True GIORNO Cat2: False GIORNO Cat3: False NOTTE Cat1: True NOTTE Cat2: False NOTTE Cat3: False
06-23 -> OPZIONI SPOT: inserto nuovo record#__3 Elementi totali presenti: 3
06-23 -> Record_Spot: 2 Efficienza: True Spot occupato: False Da Eli: null GIORNO Cat1: True GIORNO Cat2: False GIORNO Cat3: False NOTTE Cat1: True NOTTE Cat2: False NOTTE Cat3: False
06-23 -> OPZIONI SPOT: inserto nuovo record#__4 Elementi totali presenti: 4
06-23 -> Record_Spot: 3 Efficienza: True Spot occupato: False Da Eli: null GIORNO Cat1: True GIORNO Cat2: False GIORNO Cat3: False NOTTE Cat1: True NOTTE Cat2: False NOTTE Cat3: False
06-23 -> OPZIONI SPOT: inserto nuovo record#__5 Elementi totali presenti: 5
06-23 -> Record_Spot: 4 Efficienza: True Spot occupato: False Da Eli: null GIORNO Cat1: True GIORNO Cat2: False GIORNO Cat3: False NOTTE Cat1: True NOTTE Cat2: False NOTTE Cat3: False
06-23 -> OPZIONI SPOT: inserto nuovo record#__6 Elementi totali presenti: 6
06-23 -> Record_Spot: 5 Efficienza: True Spot occupato: False Da Eli: null GIORNO Cat1: True GIORNO Cat2: False GIORNO Cat3: False NOTTE Cat1: True NOTTE Cat2: False NOTTE Cat3: False
06-23 -> OPZIONI SPOT: inserto nuovo record#__7 Elementi totali presenti: 7
06-23 -> Record_Spot: 6 Efficienza: True Spot occupato: False Da Eli: null GIORNO Cat1: True GIORNO Cat2: False GIORNO Cat3: False NOTTE Cat1: True NOTTE Cat2: False NOTTE Cat3: False
06-23 -> OPZIONI SPOT: inserto nuovo record#__8 Elementi totali presenti: 8
06-23 -> Record_Spot: 7 Efficienza: True Spot occupato: False Da Eli: null GIORNO Cat1: True GIORNO Cat2: False GIORNO Cat3: False NOTTE Cat1: True NOTTE Cat2: False NOTTE Cat3: False
06-23 -> SIMULATOR: simulatore creato. Effettuo l'inizializzazione....
06-23 -> - SIMULATORE: INIZIO MOVIMENTAZIONI SUL PONTE
06-23 -> SIMULATOR : controllo effettiva NECESSITA' degli elicotteri inseriti per la missione
06-23 -> SIMULATORE: ciclo inserimento e disposizione iniziale ultimato
06-23 -> SIMULATORE: INIZIO SIMULAZIONE, starting X timer"
06-23 -> CARGO DISPOSER: è stato assegnato: 1 , restano ancora 2 elementi CARGO da imbarcare
06-23 -> CARGO DISPOSER: è stato assegnato: 3 , restano ancora 1 elementi CARGO da imbarcare
06-23 -> Spot trovato per assegnazione: 0
06-23 -> DISPOSER: movimentato elicottero : eli22 su spot: 0

06-23 -> Spot trovato per assegnazione: 1
06-23 -> DISPOSER: movimentato elicottero : eli3 su spot: 1
06-23 -> Spot trovato per assegnazione: 2
06-23 -> DISPOSER: movimentato elicottero : eli4 su spot: 2
06-23 -> Spot trovato per assegnazione: 3
06-23 -> DISPOSER: movimentato elicottero : eli5 su spot: 3
06-23 -> Spot trovato per assegnazione: 4
06-23 -> DISPOSER: movimentato elicottero : eli7 su spot: 4
06-23 -> Spot trovato per assegnazione: 5
06-23 -> DISPOSER: movimentato elicottero : eli8 su spot: 5
06-23 -> Spot trovato per assegnazione: 6
06-23 -> DISPOSER: movimentato elicottero : eli9 su spot: 6
06-23 -> CARGO DISPOSER: è stato assegnato: 2 , restano ancora 0 elementi CARGO da imbarcare
06-23 -> Spot trovato per assegnazione: 7
06-23 -> DISPOSER: movimentato elicottero : eli1 su spot: 7
06-24 -> TROOPS DISPOSER: ponte di volo occupato da: 8 Elicotteri sul ponte pronti per l'imbarco di 125 truppe
06-24 -> ELI eli22 DECOLLA time: 00:00:36
06-24 -> ELI eli3 DECOLLA time: 00:00:36
06-24 -> ELI eli4 DECOLLA time: 00:00:36
06-24 -> ELI eli5 DECOLLA time: 00:00:36
06-24 -> ELI eli7 DECOLLA time: 00:00:36
06-24 -> ELI eli8 DECOLLA time: 00:00:36
06-24 -> ELI eli9 DECOLLA time: 00:00:36
06-24 -> ELI eli1 DECOLLA time: 00:00:36
06-24 -> Spot trovato per assegnazione: 0
06-24 -> DISPOSER: movimentato elicottero : eli6 su spot: 0
06-24 -> ELI eli5 ENTRA HP: 00:00:41
06-24 -> ELI eli4 ENTRA HP: 00:00:41
06-24 -> ELI eli3 ENTRA HP: 00:00:42
06-24 -> ELI eli7 ENTRA HP: 00:00:43
06-24 -> ELI eli22 ENTRA HP: 00:00:43
06-24 -> ELI eli8 ENTRA HP: 00:00:44
06-24 -> ELI eli1 ENTRA HP: 00:00:45
06-24 -> ELI eli9 ENTRA HP: 00:00:46
06-24 -> TROOPS DISPOSER: ponte di volo occupato da: 1 Elicotteri sul ponte pronti per l'imbarco di 18 truppe
06-24 -> ELI eli6 DECOLLA time: 00:01:06
06-24 -> ELI eli6 ENTRA HP: 00:01:11
06-24 -> HOLDING POINT : elicottero : eli1 lascia l'HP time: 00:01:11
06-24 -> HOLDING POINT : elicottero : eli2 lascia l'HP time: 00:01:11
06-24 -> HOLDING POINT : elicottero : eli3 lascia l'HP time: 00:01:11
06-24 -> HOLDING POINT : elicottero : eli4 lascia l'HP time: 00:01:11
06-24 -> HOLDING POINT : elicottero : eli5 lascia l'HP time: 00:01:11
06-24 -> HOLDING POINT : elicottero : eli6 lascia l'HP time: 00:01:11
06-24 -> HOLDING POINT : elicottero : eli7 lascia l'HP time: 00:01:11
06-24 -> HOLDING POINT : elicottero : eli8 lascia l'HP time: 00:01:11
06-24 -> HOLDING POINT : elicottero : eli9 lascia l'HP time: 00:01:11
06-25 -> ELI eli1 SULLA LZ time: 00:01:47
06-25 -> ELI eli2 SULLA LZ time: 00:01:48
06-25 -> ELI eli3 SULLA LZ time: 00:01:49
06-25 -> ELI eli6 SULLA LZ time: 00:01:49
06-25 -> ELI eli7 SULLA LZ time: 00:01:49
06-25 -> LANDING ZONE : elicottero : eli2 lascia LZ time: 00:01:50
06-25 -> ELI eli8 SULLA LZ time: 00:01:50
06-25 -> ELI eli4 SULLA LZ time: 00:01:50
06-25 -> ELI eli5 SULLA LZ time: 00:01:50
06-25 -> LANDING ZONE : elicottero : eli3 lascia LZ time: 00:01:51
06-25 -> LANDING ZONE : elicottero : eli7 lascia LZ time: 00:01:51
06-25 -> ELI eli9 SULLA LZ time: 00:01:51
06-25 -> LANDING ZONE : elicottero : eli8 lascia LZ time: 00:01:52
06-25 -> LANDING ZONE : elicottero : eli4 lascia LZ time: 00:01:52
06-25 -> LANDING ZONE : elicottero : eli5 lascia LZ time: 00:01:52
06-25 -> LANDING ZONE : elicottero : eli1 lascia LZ time: 00:01:53
06-25 -> LANDING ZONE : elicottero : eli9 lascia LZ time: 00:01:53
06-25 -> LANDING ZONE : elicottero : eli6 lascia LZ time: 00:01:59
06-26 -> ELICOTTERO APPONTA: eli2 time: 00:02:33
06-26 -> ELICOTTERO APPONTA: eli3 time: 00:02:34
06-26 -> ELICOTTERO APPONTA: eli7 time: 00:02:34
06-26 -> ELI eli6 ENTRA HP: 00:02:34
06-26 -> ELICOTTERO APPONTA: eli4 time: 00:02:35
06-26 -> ELICOTTERO APPONTA: eli5 time: 00:02:35
06-26 -> ELICOTTERO APPONTA: eli8 time: 00:02:36
06-26 -> ELICOTTERO APPONTA: eli1 time: 00:02:36
06-26 -> ELICOTTERO APPONTA: eli9 time: 00:02:37
06-26 -> TROOPS DISPOSER: ponte di volo occupato da: 8 Elicotteri sul ponte pronti per l'imbarco di 13 truppe
06-26 -> TROOPS DISPOSER: ponte di volo occupato da: 8 Elicotteri sul ponte pronti per l'imbarco di 34 truppe
06-26 -> TROOPS DISPOSER: ponte di volo occupato da: 8 Elicotteri sul ponte pronti per l'imbarco di 29 truppe
06-26 -> TROOPS DISPOSER: ponte di volo occupato da: 8 Elicotteri sul ponte pronti per l'imbarco di 30 truppe
06-26 -> ELI eli2 DECOLLA time: 00:02:42
06-26 -> HOLDING POINT : elicottero : eli6 lascia l'HP time: 00:02:42
06-26 -> TROOPS DISPOSER: ponte di volo occupato da: 7 Elicotteri sul ponte pronti per l'imbarco di 19 truppe
06-26 -> ELI eli3 DECOLLA time: 00:02:43
06-26 -> ELI eli7 DECOLLA time: 00:02:43
06-26 -> ELI eli4 DECOLLA time: 00:02:44
06-26 -> ELI eli5 DECOLLA time: 00:02:44
06-26 -> ELI eli8 DECOLLA time: 00:02:45
06-26 -> ELI eli1 DECOLLA time: 00:02:45
06-26 -> ELI eli9 DECOLLA time: 00:02:46
06-26 -> ELI eli3 ENTRA HP: 00:02:48
06-26 -> ELI eli4 ENTRA HP: 00:02:48
06-26 -> ELI eli2 ENTRA HP: 00:02:48
06-26 -> ELI eli5 ENTRA HP: 00:02:48
06-26 -> ELICOTTERO APPONTA: eli6 time: 00:02:50
06-26 -> ELI eli7 ENTRA HP: 00:02:51

06-26 -> ELI eli1 ENTRA HP: 00:02:53
06-26 -> ELI eli8 ENTRA HP: 00:02:55
06-26 -> TROOPS DISPOSER: ponte di volo occupato da: 1 Elicotteri sul ponte pronti per l'imbarco di 18 truppe
06-26 -> ELI eli9 ENTRA HP: 00:02:58
06-26 -> ELI eli6 DECOLLA time: 00:02:59
06-26 -> ELI eli6 ENTRA HP: 00:03:04
06-26 -> HOLDING POINT : elicottero : eli1 lascia l'HP time: 00:03:05
06-26 -> HOLDING POINT : elicottero : eli2 lascia l'HP time: 00:03:05
06-26 -> HOLDING POINT : elicottero : eli3 lascia l'HP time: 00:03:05
06-26 -> HOLDING POINT : elicottero : eli4 lascia l'HP time: 00:03:05
06-26 -> HOLDING POINT : elicottero : eli5 lascia l'HP time: 00:03:05
06-26 -> HOLDING POINT : elicottero : eli6 lascia l'HP time: 00:03:05
06-26 -> HOLDING POINT : elicottero : eli7 lascia l'HP time: 00:03:05
06-26 -> HOLDING POINT : elicottero : eli8 lascia l'HP time: 00:03:05
06-26 -> HOLDING POINT : elicottero : eli9 lascia l'HP time: 00:03:05
06-27 -> ELI eli1 SULLA LZ time: 00:03:40
06-27 -> LANDING ZONE : elicottero : eli1 lascia LZ time: 00:03:42
06-27 -> ELI eli6 SULLA LZ time: 00:03:42
06-27 -> ELI eli7 SULLA LZ time: 00:03:42
06-27 -> ELI eli2 SULLA LZ time: 00:03:42
06-27 -> ELI eli5 SULLA LZ time: 00:03:42
06-27 -> ELI eli8 SULLA LZ time: 00:03:42
06-27 -> ELI eli9 SULLA LZ time: 00:03:43
06-27 -> ELI eli4 SULLA LZ time: 00:03:43
06-27 -> ELI eli3 SULLA LZ time: 00:03:43
06-27 -> LANDING ZONE : elicottero : eli6 lascia LZ time: 00:03:44
06-27 -> LANDING ZONE : elicottero : eli7 lascia LZ time: 00:03:44
06-27 -> LANDING ZONE : elicottero : eli2 lascia LZ time: 00:03:44
06-27 -> LANDING ZONE : elicottero : eli5 lascia LZ time: 00:03:44
06-27 -> LANDING ZONE : elicottero : eli8 lascia LZ time: 00:03:44
06-27 -> LANDING ZONE : elicottero : eli9 lascia LZ time: 00:03:45
06-27 -> LANDING ZONE : elicottero : eli4 lascia LZ time: 00:03:45
06-27 -> LANDING ZONE : elicottero : eli3 lascia LZ time: 00:03:45
06-28 -> ELI eli3 ENTRA HP: 00:04:19
06-28 -> ELICOTTERO APPONTA: eli1 time: 00:04:24
06-28 -> ELICOTTERO APPONTA: eli2 time: 00:04:26
06-28 -> ELICOTTERO APPONTA: eli5 time: 00:04:26
06-28 -> ELICOTTERO APPONTA: eli6 time: 00:04:26
06-28 -> ELICOTTERO APPONTA: eli7 time: 00:04:27
06-28 -> ELICOTTERO APPONTA: eli4 time: 00:04:27
06-28 -> ELICOTTERO APPONTA: eli8 time: 00:04:27
06-28 -> ELICOTTERO APPONTA: eli9 time: 00:04:29
06-28 -> TROOPS DISPOSER: ponte di volo occupato da: 8 Elicotteri sul ponte pronti per l'imbarco di 16 truppe
06-28 -> TROOPS DISPOSER: ponte di volo occupato da: 8 Elicotteri sul ponte pronti per l'imbarco di 46 truppe
06-28 -> TROOPS DISPOSER: ponte di volo occupato da: 8 Elicotteri sul ponte pronti per l'imbarco di 44 truppe
06-28 -> ELI eli1 DECOLLA time: 00:04:33
06-28 -> HOLDING POINT : elicottero : eli3 lascia l'HP time: 00:04:33
06-28 -> TROOPS DISPOSER: ponte di volo occupato da: 7 Elicotteri sul ponte pronti per l'imbarco di 19 truppe
06-28 -> ELI eli6 DECOLLA time: 00:04:35
06-28 -> ELI eli2 DECOLLA time: 00:04:35
06-28 -> ELI eli5 DECOLLA time: 00:04:35
06-28 -> ELI eli7 DECOLLA time: 00:04:36
06-28 -> ELI eli8 DECOLLA time: 00:04:36
06-28 -> ELI eli4 DECOLLA time: 00:04:36
06-28 -> ELI eli9 DECOLLA time: 00:04:38
06-28 -> ELI eli5 ENTRA HP: 00:04:39
06-28 -> ELI eli4 ENTRA HP: 00:04:40
06-28 -> ELI eli6 ENTRA HP: 00:04:41
06-28 -> ELI eli1 ENTRA HP: 00:04:41
06-28 -> ELI eli2 ENTRA HP: 00:04:41
06-28 -> ELICOTTERO APPONTA: eli3 time: 00:04:41
06-28 -> ELI eli7 ENTRA HP: 00:04:43
06-28 -> ELI eli8 ENTRA HP: 00:04:45
06-28 -> TROOPS DISPOSER: ponte di volo occupato da: 1 Elicotteri sul ponte pronti per l'imbarco di 18 truppe
06-28 -> ELI eli3 DECOLLA time: 00:04:50
06-28 -> ELI eli9 ENTRA HP: 00:04:50
06-28 -> ELI eli3 ENTRA HP: 00:04:54
06-28 -> HOLDING POINT : elicottero : eli1 lascia l'HP time: 00:04:54
06-28 -> HOLDING POINT : elicottero : eli2 lascia l'HP time: 00:04:54
06-28 -> HOLDING POINT : elicottero : eli3 lascia l'HP time: 00:04:54
06-28 -> HOLDING POINT : elicottero : eli4 lascia l'HP time: 00:04:54
06-28 -> HOLDING POINT : elicottero : eli5 lascia l'HP time: 00:04:54
06-28 -> HOLDING POINT : elicottero : eli6 lascia l'HP time: 00:04:54
06-28 -> HOLDING POINT : elicottero : eli7 lascia l'HP time: 00:04:54
06-28 -> HOLDING POINT : elicottero : eli8 lascia l'HP time: 00:04:54
06-28 -> HOLDING POINT : elicottero : eli9 lascia l'HP time: 00:04:54
06-29 -> ELI eli1 SULLA LZ time: 00:05:30
06-29 -> ELI ELI2 SULLA LZ time: 00:05:30
06-29 -> ELI eli3 SULLA LZ time: 00:05:31
06-29 -> ELI eli4 SULLA LZ time: 00:05:31
06-29 -> LANDING ZONE : elicottero : eli1 lascia LZ time: 00:05:32
06-29 -> LANDING ZONE : elicottero : eli2 lascia LZ time: 00:05:32
06-29 -> ELI eli6 SULLA LZ time: 00:05:32
06-29 -> ELI eli5 SULLA LZ time: 00:05:32
06-29 -> ELI eli8 SULLA LZ time: 00:05:32
06-29 -> LANDING ZONE : elicottero : eli3 lascia LZ time: 00:05:33
06-29 -> ELI eli7 SULLA LZ time: 00:05:33
06-29 -> ELI eli9 SULLA LZ time: 00:05:33
06-29 -> LANDING ZONE : elicottero : eli4 lascia LZ time: 00:05:34
06-29 -> LANDING ZONE : elicottero : eli6 lascia LZ time: 00:05:34
06-29 -> LANDING ZONE : elicottero : eli5 lascia LZ time: 00:05:34
06-29 -> LANDING ZONE : elicottero : eli8 lascia LZ time: 00:05:34
06-29 -> LANDING ZONE : elicottero : eli7 lascia LZ time: 00:05:35

06-29 -> LANDING ZONE : elicottero : eli9 lascia LZ time: 00:05:35
06-29 -> ELI eli9 ENTRA HP: 00:06:11
06-29 -> ELICOTTERO APPONTA: eli2 time: 00:06:14
06-29 -> ELICOTTERO APPONTA: eli1 time: 00:06:14
06-29 -> ELICOTTERO APPONTA: eli3 time: 00:06:15
06-29 -> ELICOTTERO APPONTA: eli4 time: 00:06:16
06-29 -> ELICOTTERO APPONTA: eli5 time: 00:06:16
06-29 -> ELICOTTERO APPONTA: eli6 time: 00:06:16
06-29 -> ELICOTTERO APPONTA: eli8 time: 00:06:17
06-29 -> ELICOTTERO APPONTA: eli7 time: 00:06:18
06-30 -> TROOPS DISPOSER: ponte di volo occupato da: 8 Elicotteri sul ponte pronti per l'imbarco di 29 truppe
06-30 -> TROOPS DISPOSER: ponte di volo occupato da: 8 Elicotteri sul ponte pronti per l'imbarco di 18 truppe
06-30 -> TROOPS DISPOSER: ponte di volo occupato da: 8 Elicotteri sul ponte pronti per l'imbarco di 24 truppe
06-30 -> HOLDING POINT : elicottero : eli9 lascia l'HP time: 00:06:22
06-30 -> ELI eli1 DECOLLA time: 00:06:23
06-30 -> ELI eli2 DECOLLA time: 00:06:23
06-30 -> ELI eli3 DECOLLA time: 00:06:24
06-30 -> ELI eli4 DECOLLA time: 00:06:25
06-30 -> ELI eli6 DECOLLA time: 00:06:25
06-30 -> ELI eli5 DECOLLA time: 00:06:25
06-30 -> ELI eli3 ENTRA HP: 00:06:28
06-30 -> ELI eli4 ENTRA HP: 00:06:29
06-30 -> ELI eli2 ENTRA HP: 00:06:29
06-30 -> ELI eli5 ENTRA HP: 00:06:29
06-30 -> ELI eli6 ENTRA HP: 00:06:31
06-30 -> ELI eli1 ENTRA HP: 00:06:31
06-30 -> HOLDING POINT : elicottero : eli1 lascia l'HP time: 00:06:31
06-30 -> HOLDING POINT : elicottero : eli2 lascia l'HP time: 00:06:31
06-30 -> HOLDING POINT : elicottero : eli3 lascia l'HP time: 00:06:31
06-30 -> HOLDING POINT : elicottero : eli4 lascia l'HP time: 00:06:31
06-30 -> HOLDING POINT : elicottero : eli5 lascia l'HP time: 00:06:31
06-30 -> HOLDING POINT : elicottero : eli6 lascia l'HP time: 00:06:31
06-30 -> ELICOTTERO APPONTA: eli9 time: 00:06:35
06-30 -> ELI eli1 SULLA LZ time: 00:07:06
06-30 -> LANDING ZONE : elicottero : eli1 lascia LZ time: 00:07:08
06-30 -> ELI ELI2 SULLA LZ time: 00:07:08
06-30 -> ELI eli6 SULLA LZ time: 00:07:08
06-30 -> ELI eli5 SULLA LZ time: 00:07:08
06-30 -> ELI eli4 SULLA LZ time: 00:07:08
06-30 -> ELI eli3 SULLA LZ time: 00:07:08
06-30 -> LANDING ZONE : elicottero : eli2 lascia LZ time: 00:07:10
06-30 -> LANDING ZONE : elicottero : eli6 lascia LZ time: 00:07:1 0
06-30 -> LANDING ZONE : elicottero : eli5 lascia LZ time: 00:07:10
06-30 -> LANDING ZONE : elicottero : eli4 lascia LZ time: 00:07:10
06-30 -> LANDING ZONE : elicottero : eli3 lascia LZ time: 00:07:10
06-31 -> ELICOTTERO APPONTA: eli1 time: 00:07:51
06-31 -> ELICOTTERO APPONTA: eli3 time: 00:07:53
06-31 -> ELICOTTERO APPONTA: eli2 time: 00:07:53
06-31 -> ELICOTTERO APPONTA: eli4 time: 00:07:53
06-31 -> ELICOTTERO APPONTA: eli5 time: 00:07:53
06-31 -> ELICOTTERO APPONTA: eli6 time: 00:07:53
06-31 -> TERMINE OPERAZIONI 00:07:53
06-31 -> SIMULAZIONE TERMINATA CON SUCCESSO

7.5 TEST5: 4 ELICOTTERI, 4 SPOT, 0 CARGO , 44 TRUPPE

Sebbene gli elicotteri inseriti inizialmente siano 4, il simulatore determina che in base ai pesi solo 2 dei 4 elicotteri possono portare a termine la missione, inoltre il carburante degli elicotteri deve essere rifasato in base al peso massimo di decollo (il consumo degli elicotteri è stato massimizzato 1500 kg / h). L'eli 1 e l'eli4 hanno infatti lo spazio necessario al trasporto di tutte le truppe necessarie nonché il carburante necessario imbarcabile per poter terminare la missione.

06-59 -> -- PROGRAMMA PARTITO --
07-01 -> MAIN WINDOW: Effettuato inserimento del NUMERO di elicotteri: 4
07-02 -> OPZIONI ELICOTTERO: inserto nuovo record#__1 Elementi totali presenti: 1
07-02 -> Record_eli: 0 ID: eli1 PosEli:False Cat: 1 Fuel: 2184 Spazio truppe: 1497 Spazio Cargo: 0 MaxT-O weight: 12861 OffLoad Weight: 5236 Consumo: 1500
07-04 -> OPZIONI ELICOTTERO: inserto nuovo record#__2 Elementi totali presenti: 2
07-04 -> Record_eli: 1 ID: eli2 PosEli:False Cat: 1 Fuel: 2026 Spazio truppe: 1386 Spazio Cargo: 0 MaxT-O weight: 9647 OffLoad Weight: 5591 Consumo: 1500
07-04 -> OPZIONI ELICOTTERO: inserto nuovo record#__3 Elementi totali presenti: 3
07-04 -> Record_eli: 2 ID: eli3 PosEli:False Cat: 1 Fuel: 2447 Spazio truppe: 1072 Spazio Cargo: 0 MaxT-O weight: 10312 OffLoad Weight: 1000 Consumo: 1500
07-05 -> OPZIONI ELICOTTERO: inserto nuovo record#__4 Elementi totali presenti: 4
07-05 -> Record_eli: 3 ID: eli4 PosEli:False Cat: 1 Fuel: 1837 Spazio truppe: 1460 Spazio Cargo: 0 MaxT-O weight: 8372 OffLoad Weight: 1000 Consumo: 1500
07-05 -> MAIN WINDOW: Effettuato inserimento del NUMERO di truppe: 44
07-05 -> TROOPS MANAGER: effettuata creazione manager per truppe
07-05 -> MAIN WINDOW: VISUALIZZUATA SCHERMATA OPZIONI TRUPPE
07-05 -> OPZIONI TRUPPE: effettuato l'inserimento di 44 soldati del peso singolo di 50kg
07-05 -> MAIN WINDOW: Effettuato inserimento di ALTRO CARGO: 0
07-05 -> MAIN WINDOW: inserita la distanza della landing zone dall'unità navale: 100 NM
07-05 -> MAIN WINDOW: Effettuato inserimento del NUMERO di SPOT: 4
07-05 -> OPZIONI SPOT: EFFETTUATA APERTURA FINESTRA OPZIONI SPOT
07-05 -> OPZIONI SPOT: inserto nuovo record#__1 Elementi totali presenti: 1
07-05 -> Record_Spot: 0 Efficienza: True Spot occupato: False Da Eli: null GIORNO Cat1: True GIORNO Cat2: False GIORNO Cat3: False NOTTE Cat1: True NOTTE Cat2: False NOTTE Cat3: False
07-05 -> OPZIONI SPOT: inserto nuovo record#__2 Elementi totali presenti: 2
07-05 -> Record_Spot: 1 Efficienza: True Spot occupato: False Da Eli: null GIORNO Cat1: True GIORNO Cat2: False GIORNO Cat3: False NOTTE Cat1: True NOTTE Cat2: False NOTTE Cat3: False
07-05 -> OPZIONI SPOT: inserto nuovo record#__3 Elementi totali presenti: 3

07-05 -> Record_Spot: 2 Efficienza: True Spot occupato: False Da Eli: null GIORNO Cat1: True GIORNO Cat2: False GIORNO Cat3: False NOTTE Cat1: True NOTTE Cat2: False NOTTE Cat3: False
07-05 -> OPZIONI SPOT: inserto nuovo record#__4 Elementi totali presenti: 4
07-05 -> Record_Spot: 3 Efficienza: True Spot occupato: False Da Eli: null GIORNO Cat1: True GIORNO Cat2: False GIORNO Cat3: False NOTTE Cat1: True NOTTE Cat2: False NOTTE Cat3: False
07-05 -> SIMULATOR: simulatore creato. Effettuo l'inizializzazione....
07-05 -> - SIMULATORE: INIZIO MOVIMENTAZIONI SUL PONTE
07-05 -> SIMULATOR : controllo effettiva NECESSITA' degli elicotteri inseriti per la missione
07-05 -> SIMULATOR : rimosso elicottero: eli3 l'eli risulta non necessario
07-05 -> SIMULATOR : rimosso elicottero: eli2 l'eli risulta non necessario
07-05 -> SIMULATORE: ciclo inserimento e disposizione iniziale ultimato
07-05 -> SIMULATORE: INIZIO SIMULAZIONE, starting X timer"
07-05 -> Spot trovato per assegnazione: 0
07-05 -> DISPOSER: movimentato elicottero : eli1 su spot: 0
07-05 -> Spot trovato per assegnazione: 1
07-05 -> DISPOSER: movimentato elicottero : eli4 su spot: 1
07-06 -> TROOPS DISPOSER: ponte di volo occupato da: 2 Elicotteri sul ponte pronti per l'imbarco di 44 truppe
07-06 -> ELI eli1 DECOLLA time: 00:00:31
07-06 -> ELI eli4 DECOLLA time: 00:00:31
07-06 -> ELI eli4 ENTRA HP: 00:00:43
07-06 -> ELI eli1 ENTRA HP: 00:00:44
07-06 -> HOLDING POINT : elicottero : eli1 lascia l'HP time: 00:00:44
07-06 -> HOLDING POINT : elicottero : eli4 lascia l'HP time: 00:00:44
07-07 -> ELI eli1 SULLA LZ time: 00:01:35
07-07 -> ELI eli4 SULLA LZ time: 00:01:36
07-07 -> LANDING ZONE : elicottero : eli1 lascia LZ time: 00:01:37
07-07 -> LANDING ZONE : elicottero : eli4 lascia LZ time: 00:01:38
07-08 -> ELICOTTERO APPONTA: eli1 time: 00:02:38
07-08 -> ELICOTTERO APPONTA: eli4 time: 00:02:38
07-08 -> TERMINE OPERAZIONI 00:02:38
07-08 -> SIMULAZIONE TERMINATA CON SUCCESSO

7.6 TEST6: 1 ELICOTTERO, 1 SPOT, 1 CARGO , 1 TRUPPE

La simulazione viene tentata fino al verificarsi del fallimento, in quanto l'elicottero effettua il trasporto delle truppe, tuttavia al termine della simulazione viene verificata l'impossiibilità di trasportare il cargo per il peso eccessivo, la conseguenza è l'infattibilità di raggiungere lo scopo finale di trasporto.

07-34 -> -- PROGRAMMA PARTITO --
07-35 -> MAIN WINDOW: Effettuato inserimento del NUMERO di elicotteri: 1
07-35 -> OPZIONI ELICOTTERO: inserto nuovo record#__1 Elementi totali presenti: 1
07-35 -> Record_eli: 0 ID: eli1 PosEli:True Cat: 1 Fuel: 2268 Spazio truppe: 1691 Spazio Cargo: 804 MaxT-O weight: 9314 OffLoad Weight: 1000 Consumo: 1500
07-35 -> MAIN WINDOW: Effettuato inserimento del NUMERO di truppe: 1
07-35 -> TROOPS MANAGER: effettuata creazione manager per truppe
07-35 -> MAIN WINDOW: VISUALIZZUATA SCHERMATA OPZIONI TRUPPE
07-35 -> OPZIONI TRUPPE: effettuato l'inserimento di 1 soldati del peso singolo di: 50kg
07-35 -> MAIN WINDOW: Effettuato inserimento di ALTRO CARGO: 1
07-35 -> OPZIONI CARGO: --> EFFETTUATA APERTURA FINESTRA OPZIONI CARGO <--
07-36 -> OPZIONI CARGO: inserto nuovo record#__1 Elementi totali presenti: 1
07-36 -> Record_cargo: 0 Tipo Cargo: cargo1 Peso Cargo: 2202 Utenti necessari: 6
07-36 -> MAIN WINDOW: inserita la distanza della landing zone dall'unità navale: 50 NM
07-36 -> MAIN WINDOW: Effettuato inserimento del NUMERO di SPOT: 1
07-36 -> OPZIONI SPOT: EFFETTUATA APERTURA FINESTRA OPZIONI SPOT
07-36 -> OPZIONI SPOT: inserto nuovo record#__1 Elementi totali presenti: 1
07-36 -> Record_Spot: 0 Efficienza: True Spot occupato: False Da Eli: null GIORNO Cat1: True GIORNO Cat2: False GIORNO Cat3: False NOTTE Cat1: True NOTTE Cat2: False NOTTE Cat3: False
07-36 -> SIMULATOR: simulatore creato. Effettuo l'inizializzazione....
07-36 -> - SIMULATORE: INIZIO MOVIMENTAZIONI SUL PONTE
07-36 -> SIMULATOR : controllo effettiva NECESSITA' degli elicotteri inseriti per la missione
07-36 -> SIMULATOR: Spot trovato per assegnazione AUTOMATICA: 0
07-36 -> SIMULATORE: ciclo inserimento e disposizione iniziale ultimato
07-36 -> SIMULATORE: INIZIO SIMULAZIONE, starting X timer"
07-36 -> TROOPS DISPOSER: ponte di volo occupato da: 1 Elicotteri sul ponte pronti per l'imbarco di 1 truppe
07-36 -> ELI eli1 DECOLLA time: 00:00:04
07-36 -> ELI eli1 ENTRA HP: 00:00:10
07-36 -> HOLDING POINT : elicottero : eli1 lascia l'HP time: 00:00:10
07-36 -> ELI eli1 SULLA LZ time: 00:00:36
07-36 -> LANDING ZONE : elicottero : eli1 lascia LZ time: 00:00:38
07-37 -> ELICOTTERO APPONTA: eli1 time: 00:01:09
07-37 -> TERMINE OPERAZIONI 00:01:09
07-37 -> SIMULAZIONE FALLITA

7.7 TEST7: 1 ELICOTTERO, 1 SPOT ,3 CARGO ,23 TRUPPE

La simulazione viene effettuata con un trasporto di cargo massivo, tuttavia il peso disponibile a bordo dell'elicottero non permette il raggruppamento di differenti cargo, che quindi devono essere trasportati singolarmente sulla LZ.

09-41 -> -- PROGRAMMA PARTITO --
09-41 -> MAIN WINDOW: Effettuato inserimento del NUMERO di elicotteri: 1
09-51 -> OPZIONI ELICOTTERO: inserto nuovo record#__1 Elementi totali presenti: 1
09-51 -> Record_eli: 0 ID: eli1 PosEli:False Cat: 1 Fuel: 2000 Spazio truppe: 1404 Spazio Cargo: 1423 MaxT-O weight: 9314 OffLoad Weight: 1000 Consumo: 998
09-51 -> MAIN WINDOW: Effettuato inserimento del NUMERO di truppe: 23
09-51 -> TROOPS MANAGER: effettuata creazione manager per truppe
09-51 -> MAIN WINDOW: VISUALIZZUATA SCHERMATA OPZIONI TRUPPE
09-51 -> OPZIONI TRUPPE: effettuato l'inserimento di 23 soldati del peso singolo di: 71kg
09-51 -> MAIN WINDOW: Effettuato inserimento di ALTRO CARGO: 3
09-51 -> OPZIONI CARGO: --> EFFETTUATA APERTURA FINESTRA OPZIONI CARGO <--

09-52 -> OPZIONI CARGO: inserto nuovo record#__1 Elementi totali presenti: 1
09-52 -> Record_cargo: 0 Tipo Cargo: cargo 1 Peso Cargo: 1041 Utenti necessari: 1
09-52 -> OPZIONI CARGO: inserto nuovo record#__2 Elementi totali presenti: 2
09-52 -> Record_cargo: 1 Tipo Cargo: cargo 2 Peso Cargo: 1052 Utenti necessari: 1
09-52 -> OPZIONI CARGO: inserto nuovo record#__3 Elementi totali presenti: 3
09-52 -> Record_cargo: 2 Tipo Cargo: cargo 3 Peso Cargo: 1041 Utenti necessari: 2
09-52 -> MAIN WINDOW: inserita la distanza della landing zone dall'unità navale: 50 NM
09-52 -> MAIN WINDOW: Effettuato inserimento del NUMERO di SPOT: 1
09-52 -> OPZIONI SPOT: EFFETTUATA APERTURA FINESTRA OPZIONI SPOT
09-52 -> OPZIONI SPOT: inserto nuovo record#__1 Elementi totali presenti: 1
09-52 -> Record_Spot: 0 Efficienza: True Spot occupato: False Da Eli: null GIORNO Cat1: True GIORNO Cat2: False GIORNO Cat3: False NOTTE Cat1: True NOTTE Cat2: False NOTTE Cat3: False
09-52 -> SIMULATOR: simulatore creato. Effettuo l'inizializzazione....
09-52 -> - SIMULATORE: INIZIO MOVIMENTAZIONI SUL PONTE
09-52 -> SIMULATOR : controllo effettiva NECESSITA' degli elicotteri inseriti per la missione
09-52 -> SIMULATORE: ciclo inserimento e disposizione iniziale ultimato
09-52 -> SIMULATORE: INIZIO SIMULAZIONE, starting X timer"
09-52 -> CARGO DISPOSER: è stato assegnato: cargo 1 , restano ancora 2 elementi CARGO da imbarcare
09-52 -> Spot trovato per assegnazione: 0
09-52 -> DISPOSER: movimentato elicottero : eli1 su spot: 0
09-53 -> TROOPS DISPOSER: ponte di volo occupato da: 1 Elicotteri sul ponte pronti per l'imbarco di 19 truppe
09-53 -> ELI eli1 DECOLLA time: 00:00:36
09-53 -> ELI eli1 ENTRA HP: 00:00:42
09-53 -> HOLDING POINT : elicottero : eli1 lascia l'HP time: 00:00:42
09-53 -> ELI eli1 SULLA LZ time: 00:01:08
09-53 -> LANDING ZONE : elicottero : eli1 lascia LZ time: 00:01:14
09-54 -> ELICOTTERO APPONTA: eli1 time: 00:01:45
09-54 -> CARGO DISPOSER: è stato assegnato: cargo 2 , restano ancora 1 elementi CARGO da imbarcare
09-54 -> TROOPS DISPOSER: ponte di volo occupato da: 1 Elicotteri sul ponte pronti per l'imbarco di 4 truppe
09-54 -> ELI eli1 DECOLLA time: 00:01:59
09-54 -> ELI eli1 ENTRA HP: 00:02:04
09-54 -> HOLDING POINT : elicottero : eli1 lascia l'HP time: 00:02:05
09-55 -> ELI eli1 SULLA LZ time: 00:02:30
09-55 -> LANDING ZONE : elicottero : eli1 lascia LZ time: 00:02:36
09-55 -> ELICOTTERO APPONTA: eli1 time: 00:03:07
09-55 -> CARGO DISPOSER: è stato assegnato: cargo 3 , restano ancora 0 elementi CARGO da imbarcare
09-55 -> ELI eli1 DECOLLA time: 00:03:18
09-55 -> ELI eli1 ENTRA HP: 00:03:23
09-55 -> HOLDING POINT : elicottero : eli1 lascia l'HP time: 00:03:24
09-56 -> ELI eli1 SULLA LZ time: 00:03:49
09-56 -> LANDING ZONE : elicottero : eli1 lascia LZ time: 00:03:53
09-56 -> ELICOTTERO APPONTA: eli1 time: 00:04:24
09-56 -> TERMINE OPERAZIONI 00:04:24
09-56 -> SIMULAZIONE TERMINATA CON SUCCESSO

7.8 TEST8: 5 ELICOTTERI , 5 SPOT ,5 CARGO, 140 TRUPPE

Viene reinserta la peseta delle truppe aggiornando il record

10-04 -> -- PROGRAMMA PARTITO --
10-04 -> MAIN WINDOW: Effettuato inserimento del NUMERO di elicotteri: 5
10-05 -> OPZIONI ELICOTTERO: inserto nuovo record#__1 Elementi totali presenti: 1
10-05 -> Record_eli: 0 ID: eli 1 PosEli:False Cat: 1 Fuel: 2268 Spazio truppe: 1561 Spazio Cargo: 1589 MaxT-O weight: 9425 OffLoad Weight: 1711 Consumo: 1004
10-09 -> OPZIONI ELICOTTERO: inserto nuovo record#__2 Elementi totali presenti: 2
10-09 -> Record_eli: 1 ID: eli 2 PosEli:False Cat: 1 Fuel: 2310 Spazio truppe: 1469 Spazio Cargo: 1469 MaxT-O weight: 7984 OffLoad Weight: 1744 Consumo: 1007
10-10 -> OPZIONI ELICOTTERO: inserto nuovo record#__3 Elementi totali presenti: 3
10-10 -> Record_eli: 2 ID: eli3 PosEli:False Cat: 1 Fuel: 2236 Spazio truppe: 1487 Spazio Cargo: 1072 MaxT-O weight: 8926 OffLoad Weight: 1000 Consumo: 755
10-10 -> OPZIONI ELICOTTERO: inserto nuovo record#__4 Elementi totali presenti: 4
10-10 -> Record_eli: 3 ID: eli4 PosEli:False Cat: 1 Fuel: 2194 Spazio truppe: 1284 Spazio Cargo: 0 MaxT-O weight: 8095 OffLoad Weight: 1000 Consumo: 596
10-11 -> OPZIONI ELICOTTERO: inserto nuovo record#__5 Elementi totali presenti: 5
10-11 -> Record_eli: 4 ID: eli5 PosEli:False Cat: 1 Fuel: 2247 Spazio truppe: 1155 Spazio Cargo: 1284 MaxT-O weight: 9758 OffLoad Weight: 1000 Consumo: 923
10-11 -> MAIN WINDOW: Effettuato inserimento del NUMERO di truppe: 140
10-11 -> TROOPS MANAGER: effettuata creazione manager per truppe
10-11 -> MAIN WINDOW: VISUALIZZUATA SCHERMATA OPZIONI TRUPPE
10-11 -> OPZIONI TRUPPE: effettuato l'inserimento di 140 soldati del peso singolo di: 50kg
10-11 -> MAIN WINDOW: VISUALIZZUATA SCHERMATA OPZIONI TRUPPE
10-11 -> OPZIONI TRUPPE: lista truppe già creata, sostituisco con i nuovi dati
10-11 -> OPZIONI TRUPPE: nuovo peso inserito: 94
10-11 -> MAIN WINDOW: Effettuato inserimento di ALTRO CARGO: 5
10-11 -> OPZIONI CARGO: --> EFFETTUATA APERTURA FINESTRA OPZIONI CARGO <--
10-11 -> OPZIONI CARGO: inserto nuovo record#__1 Elementi totali presenti: 1
10-11 -> Record_cargo: 0 Tipo Cargo: cargo 1 Peso Cargo: 879 Utenti necessari: 7
10-11 -> OPZIONI CARGO: inserto nuovo record#__2 Elementi totali presenti: 2
10-11 -> Record_cargo: 1 Tipo Cargo: cargo 2 Peso Cargo: 1312 Utenti necessari: 7
10-12 -> OPZIONI CARGO: inserto nuovo record#__3 Elementi totali presenti: 3
10-12 -> Record_cargo: 2 Tipo Cargo: cargo 3 Peso Cargo: 770 Utenti necessari: 2
10-12 -> OPZIONI CARGO: inserto nuovo record#__4 Elementi totali presenti: 4
10-12 -> Record_cargo: 3 Tipo Cargo: cargo 4 Peso Cargo: 607 Utenti necessari: 4
10-12 -> OPZIONI CARGO: inserto nuovo record#__5 Elementi totali presenti: 5
10-12 -> Record_cargo: 4 Tipo Cargo: cargo 5 Peso Cargo: 879 Utenti necessari: 3
10-12 -> OPZIONI CARGO: inserto nuovo record#__2 Elementi totali presenti: 6
10-12 -> Record_cargo: 1 Tipo Cargo: cargo 2 Peso Cargo: 1009 Utenti necessari: 7
10-12 -> MAIN WINDOW: inserita la distanza della landing zone dall'unità navale: 50 NM
10-12 -> MAIN WINDOW: Effettuato inserimento del NUMERO di SPOT: 5
10-12 -> OPZIONI SPOT: EFFETTUATA APERTURA FINESTRA OPZIONI SPOT
10-12 -> Record_Spot: 0 Efficienza: True Spot occupato: False Da Eli: null GIORNO Cat1: True GIORNO Cat2: False GIORNO Cat3: False NOTTE Cat1: True NOTTE Cat2: False NOTTE Cat3: False
10-12 -> OPZIONI SPOT: inserto nuovo record#__2 Elementi totali presenti: 2
10-12 -> Record_Spot: 1 Efficienza: True Spot occupato: False Da Eli: null GIORNO Cat1: True GIORNO Cat2: False GIORNO Cat3: False NOTTE Cat1: True NOTTE Cat2: False NOTTE Cat3: False

10-12 -> OPZIONI SPOT: inserto nuovo record#__3 Elementi totali presenti: 3
10-12 -> Record_Spot: 2 Efficienza: True Spot occupato: False Da Eli: null GIORNO Cat1: True GIORNO Cat2: False GIORNO Cat3: False NOTTE Cat1: True NOTTE Cat2: False NOTTE Cat3: False
10-12 -> OPZIONI SPOT: inserto nuovo record#__4 Elementi totali presenti: 4
10-12 -> Record_Spot: 3 Efficienza: True Spot occupato: False Da Eli: null GIORNO Cat1: True GIORNO Cat2: False GIORNO Cat3: False NOTTE Cat1: True NOTTE Cat2: False NOTTE Cat3: False
10-12 -> OPZIONI SPOT: inserto nuovo record#__5 Elementi totali presenti: 5
10-12 -> Record_Spot: 4 Efficienza: True Spot occupato: False Da Eli: null GIORNO Cat1: True GIORNO Cat2: False GIORNO Cat3: False NOTTE Cat1: True NOTTE Cat2: False NOTTE Cat3: False
10-12 -> SIMULATOR: simulatore creato. Effettuo l'inizializzazione....
10-12 -> - SIMULATORE: INIZIO MOVIMENTAZIONI SUL PONTE
10-12 -> SIMULATOR : controllo effettiva NECESSITA' degli elicotteri inseriti per la missione
10-12 -> SIMULATORE: ciclo inserimento e disposizione iniziale ultimato
10-12 -> SIMULATORE: INIZIO SIMULAZIONE, starting X timer"
10-12 -> CARGO DISPOSER: è stato assegnato: cargo 1 , restano ancora 5 elementi CARGO da imbarcare
10-12 -> CARGO DISPOSER: è stato assegnato: cargo 2 , restano ancora 4 elementi CARGO da imbarcare
10-12 -> CARGO DISPOSER: è stato assegnato: cargo 4 , restano ancora 3 elementi CARGO da imbarcare
10-12 -> Spot trovato per assegnazione: 0
10-12 -> DISPOSER: movimentato elicottero : eli3 su spot: 0
10-12 -> Spot trovato per assegnazione: 1
10-12 -> DISPOSER: movimentato elicottero : eli4 su spot: 1
10-12 -> Spot trovato per assegnazione: 2
10-12 -> DISPOSER: movimentato elicottero : eli 1 su spot: 2
10-12 -> Spot trovato per assegnazione: 3
10-12 -> DISPOSER: movimentato elicottero : eli 2 su spot: 3
10-12 -> Spot trovato per assegnazione: 4
10-12 -> DISPOSER: movimentato elicottero : eli5 su spot: 4
10-13 -> CARGO DISPOSER: è stato assegnato: cargo 2 , restano ancora 2 elementi CARGO da imbarcare
10-13 -> TROOPS DISPOSER: ponte di volo occupato da: 5 Elicotteri sul ponte pronti per l'imbarco di 137 truppe
10-13 -> ELI eli3 DECOLLA time: 00:00:36
10-13 -> ELI eli4 DECOLLA time: 00:00:36
10-13 -> ELI eli 1 DECOLLA time: 00:00:36
10-13 -> ELI eli 2 DECOLLA time: 00:00:36
10-13 -> ELI eli5 DECOLLA time: 00:00:36
10-13 -> ELI eli3 ENTRA HP: 00:00:39
10-13 -> ELI eli 2 ENTRA HP: 00:00:40
10-13 -> ELI eli4 ENTRA HP: 00:00:40
10-13 -> ELI eli 1 ENTRA HP: 00:00:42
10-13 -> ELI eli5 ENTRA HP: 00:00:43
10-13 -> HOLDING POINT : elicottero : eli 1 lascia l'HP time: 00:00:43
10-13 -> HOLDING POINT : elicottero : eli 2 lascia l'HP time: 00:00:43
10-13 -> HOLDING POINT : elicottero : eli3 lascia l'HP time: 00:00:43
10-13 -> HOLDING POINT : elicottero : eli4 lascia l'HP time: 00:00:43
10-13 -> HOLDING POINT : elicottero : eli5 lascia l'HP time: 00:00:43
10-14 -> ELI eli 1 SULLA LZ time: 00:01:09
10-14 -> ELI eli5 SULLA LZ time: 00:01:09
10-14 -> ELI eli4 SULLA LZ time: 00:01:10
10-14 -> ELI eli 2 SULLA LZ time: 00:01:10
10-14 -> ELI eli3 SULLA LZ time: 00:01:11
10-14 -> LANDING ZONE : elicottero : eli4 lascia LZ time: 00:01:12
10-14 -> LANDING ZONE : elicottero : eli 1 lascia LZ time: 00:01:15
10-14 -> LANDING ZONE : elicottero : eli5 lascia LZ time: 00:01:15
10-14 -> LANDING ZONE : elicottero : eli 2 lascia LZ time: 00:01:16
10-14 -> LANDING ZONE : elicottero : eli3 lascia LZ time: 00:01:17
10-14 -> ELICOTTERO APPONTA: eli4 time: 00:01:43
10-14 -> TROOPS DISPOSER: ponte di volo occupato da: 1 Elicotteri sul ponte pronti per l'imbarco di 3 truppe
10-14 -> ELI eli4 DECOLLA time: 00:01:46
10-14 -> ELICOTTERO APPONTA: eli 1 time: 00:01:46
10-14 -> CARGO DISPOSER: è stato assegnato: cargo 3 , restano ancora 1 elementi CARGO da imbarcare
10-14 -> ELICOTTERO APPONTA: eli5 time: 00:01:46
10-14 -> CARGO DISPOSER: è stato assegnato: cargo 5 , restano ancora 0 elementi CARGO da imbarcare
10-14 -> ELICOTTERO APPONTA: eli 2 time: 00:01:47
10-14 -> ELICOTTERO APPONTA: eli3 time: 00:01:48
10-14 -> ELI eli4 ENTRA HP: 00:01:51
10-14 -> ELI eli 1 DECOLLA time: 00:01:57
10-14 -> ELI eli5 DECOLLA time: 00:01:57
10-14 -> ELI eli 1 ENTRA HP: 00:02:03
10-14 -> ELI eli5 ENTRA HP: 00:02:05
10-14 -> HOLDING POINT : elicottero : eli 1 lascia l'HP time: 00:02:05
10-14 -> HOLDING POINT : elicottero : eli4 lascia l'HP time: 00:02:05
10-14 -> HOLDING POINT : elicottero : eli5 lascia l'HP time: 00:02:05
10-15 -> ELI eli5 SULLA LZ time: 00:02:31
10-15 -> ELI eli4 SULLA LZ time: 00:02:31
10-15 -> ELI eli 1 SULLA LZ time: 00:02:32
10-15 -> LANDING ZONE : elicottero : eli4 lascia LZ time: 00:02:33
10-15 -> LANDING ZONE : elicottero : eli5 lascia LZ time: 00:02:35
10-15 -> LANDING ZONE : elicottero : eli 1 lascia LZ time: 00:02:36
10-15 -> ELICOTTERO APPONTA: eli4 time: 00:03:04
10-15 -> ELICOTTERO APPONTA: eli5 time: 00:03:06
10-16 -> ELICOTTERO APPONTA: eli 1 time: 00:03:07
10-16 -> TERMINE OPERAZIONI 00:03:07
10-16 -> SIMULAZIONE TERMINATA CON SUCCESSO

7.9 TEST 9 : HELO CRASH 3 ELICOTTERI , 1 SPOT ,1 CARGO, 151 TRUPPE

In questa simulazione, vengono usati 3 elicotteri ed 1 spot, in questo caso le tempistiche di holding degli elicotteri aumentano in quanto l'uso di un solo spot non permette più movimentazioni in parallelo. I calcoli del carburante necessario, anche se ricalibrati dal programma, risultano non sufficienti alla missione per eccessiva permanenza in

holding. La simulazione termina in quanto tute le truppe ed il cargo viene trasportato a destinazione, tuttavia con la perdita di un elicottero.

```
11-07 -> -- PROGRAMMA PARTITO --
 11-07 -> MAIN WINDOW: Effettuato inserimento del NUMERO di elicotteri: 3
 11-07 -> OPZIONI ELICOTTERO: inserto nuovo record#__1 Elementi totali presenti: 1
 11-07 -> Record_eli: 0 ID: 1 PosEli:False Cat: 1 Fuel: 2478 Spazio truppe: 1764 Spazio Cargo: 1016 MaxT-O weight: 10090 OffLoad Weight: 1000 Consumo: 1039
 11-08 -> OPZIONI ELICOTTERO: inserto nuovo record#__2 Elementi totali presenti: 2
 11-08 -> Record_eli: 1 ID: 2 PosEli:False Cat: 1 Fuel: 2257 Spazio truppe: 1469 Spazio Cargo: 0 MaxT-O weight: 10866 OffLoad Weight: 1000 Consumo: 876
 11-08 -> OPZIONI ELICOTTERO: inserto nuovo record#__3 Elementi totali presenti: 3
 11-08 -> Record_eli: 2 ID: 3 PosEli:False Cat: 1 Fuel: 2215 Spazio truppe: 1478 Spazio Cargo: 0 MaxT-O weight: 10811 OffLoad Weight: 1000 Consumo: 721
 11-08 -> MAIN WINDOW: Effettuato inserimento del NUMERO di truppe: 151
 11-08 -> TROOPS MANAGER: effettuata creazione manager per truppe
 11-08 -> MAIN WINDOW: VISUALIZZUATA SCHERMATA OPZIONI TRUPPE
 11-08 -> OPZIONI TRUPPE: effettuato l'inserimento di 151 soldati del peso singolo di: 50kg
 11-08 -> MAIN WINDOW: Effettuato inserimento di ALTRO CARGO: 1
 11-08 -> OPZIONI CARGO: --> EFFETTUATA APERTURA FINESTRA OPZIONI CARGO <--
 11-08 -> OPZIONI CARGO: inserto nuovo record#__1 Elementi totali presenti: 1
 11-08 -> Record_cargo: 0 Tipo Cargo: 1 Peso Cargo: 260 Utenti necessari: 1
 11-08 -> MAIN WINDOW: inserita la distanza della landing zone dall'unità navale: 50 NM
 11-08 -> MAIN WINDOW: Effettuato inserimento del NUMERO di SPOT: 1
 11-08 -> OPZIONI SPOT: EFFETTUATA APERTURA FINESTRA OPZIONI SPOT
 11-08 -> OPZIONI SPOT: inserto nuovo record#__1 Elementi totali presenti: 1
 11-08 -> Record_Spot: 0 Efficienza: True Spot occupato: False Da Eli: null GIORNO Cat1: True GIORNO Cat2: False GIORNO Cat3: False NOTTE Cat1: True NOTTE Cat2: False
NOTTE Cat3: False
 11-08 -> SIMULATOR: simulatore creato. Effettuo l'inizializzazione....
 11-08 ->  - SIMULATORE: INIZIO MOVIMENTAZIONI SUL PONTE
 11-08 -> SIMULATOR : controllo effettiva NECESSITA' degli elicotteri inseriti per la missione
 11-08 -> SIMULATORE: ciclo inserimento e disposizione iniziale ultimato
 11-08 -> SIMULATORE: INIZIO SIMULAZIONE, starting X timer"
 11-08 -> CARGO DISPOSER: è stato assegnato: 1 , restano ancora 0 elementi CARGO da imbarcare
 11-08 -> Spot trovato per assegnazione: 0
 11-08 -> DISPOSER: movimentato elicottero : 2 su spot: 0
 11-08 -> DISPOSER: movimentazione elicottero 3 non effettuabile
 11-09 -> TROOPS DISPOSER: ponte di volo occupato da: 1 Elicotteri sul ponte pronti per l'imbarco di 29 truppe
 11-09 ->  ELI 2 DECOLLA time: 00:00:31
 11-09 -> Spot trovato per assegnazione: 0
 11-09 -> DISPOSER: movimentato elicottero : 1 su spot: 0
 11-09 -> DISPOSER: movimentazione elicottero 3 non effettuabile
 11-09 ->  ELI 2 ENTRA HP: 00:00:34
 11-09 -> TROOPS DISPOSER: ponte di volo occupato da: 1 Elicotteri sul ponte pronti per l'imbarco di 35 truppe
 11-09 ->  ELI 1 DECOLLA time: 00:01:01
 11-09 -> Spot trovato per assegnazione: 0
 11-09 -> DISPOSER: movimentato elicottero : 3 su spot: 0
 11-09 ->  ELI 1 ENTRA HP: 00:01:07
 11-10 -> TROOPS DISPOSER: ponte di volo occupato da: 1 Elicotteri sul ponte pronti per l'imbarco di 29 truppe
 11-10 ->  ELI 3 DECOLLA time: 00:01:31
 11-10 ->  ELI 3 ENTRA HP: 00:01:36
 11-10 -> HOLDING POINT : elicottero : 1 lascia l'HP time: 00:01:36
 11-10 -> HOLDING POINT : elicottero : 2 lascia l'HP time: 00:01:36
 11-10 -> HOLDING POINT : elicottero : 3 lascia l'HP time: 00:01:36
 11-10 ->  ELI 1 SULLA LZ time: 00:02:02
 11-10 ->  ELI 3 SULLA LZ time: 00:02:02
 11-10 ->  ELI 2 SULLA LZ time: 00:02:03
 11-10 -> LANDING ZONE : elicottero : 3 lascia LZ time: 00:02:04
 11-10 -> LANDING ZONE : elicottero : 2 lascia LZ time: 00:02:05
 11-10 -> LANDING ZONE : elicottero : 1 lascia LZ time: 00:02:08
 11-11 ->  ELI 2 ENTRA HP: 00:02:39
 11-11 ->  ELI 1 ENTRA HP: 00:02:44
 11-11 -> ELICOTTERO APPONTA: 3 time: 00:02:47
 11-11 -> TROOPS DISPOSER: ponte di volo occupato da: 1 Elicotteri sul ponte pronti per l'imbarco di 29 truppe
 11-11 ->  ELI 3 DECOLLA time: 00:02:56
 11-11 -> HOLDING POINT : elicottero : 1 lascia l'HP time: 00:02:56
 11-11 -> SIMULATOR: crash eli (0 fuel): 2 truppe perse: 0 cargo perso: 0
 11-11 ->  ELI 3 ENTRA HP: 00:03:05
 11-11 -> HOLDING POINT : elicottero : 3 lascia l'HP time: 00:03:05
 11-11 -> ELICOTTERO APPONTA: 1 time: 00:03:10
 11-12 -> TROOPS DISPOSER: ponte di volo occupato da: 1 Elicotteri sul ponte pronti per l'imbarco di 29 truppe
 11-12 ->  ELI 1 DECOLLA time: 00:03:19
 11-12 ->  ELI 1 ENTRA HP: 00:03:28
 11-12 -> HOLDING POINT : elicottero : 1 lascia l'HP time: 00:03:28
 11-12 ->  ELI 3 SULLA LZ time: 00:03:37
 11-12 -> LANDING ZONE : elicottero : 3 lascia LZ time: 00:03:39
 11-12 ->  ELI 1 SULLA LZ time: 00:03:45
 11-12 -> LANDING ZONE : elicottero : 1 lascia LZ time: 00:03:47
 11-12 -> ELICOTTERO APPONTA: 3 time: 00:03:59
 11-12 ->  ELI 1 ENTRA HP: 00:04:03
 11-12 -> HOLDING POINT : elicottero : 1 lascia l'HP time: 00:04:03
 11-12 -> ELICOTTERO APPONTA: 1 time: 00:04:08
 11-12 -> TERMINE OPERAZIONI 00:04:08
 11-12 -> SIMULAZIONE TERMINATA CON SUCCESSO
```

7.10 TESTO 10: SIMULAZIONE FALLITA PER SPOT INCOMPATIBILI 2 ELICOTTERI, 2 SPOT

La categoria di elicottero inserita è di tipo 2 (elicottero pesante), gli spot inseriti sono tutti permissivi per elicotteri di categoria 1 (elicottero leggero). Non potendo moviemntare gli elicotteri sugli spot assegnabili la simulazione fallisce.

01-17 -> -- PROGRAMMA PARTITO -
01-17 -> MAIN WINDOW: Effettuato inserimento del NUMERO di elicotteri: 2
01-18 -> OPZIONI ELICOTTERO: inserto nuovo record#__1 Elementi totali presenti: 1
01-18 -> Record_eli: 0 ID: eli1 PosEli:False Cat: 2 Fuel: 1637 Spazio truppe: 1145 Spazio Cargo: 0 MaxT-O weight: 8594 OffLoad Weight: 1000 Consumo: 702
01-18 -> OPZIONI ELICOTTERO: inserto nuovo record#__2 Elementi totali presenti: 2
01-18 -> Record_eli: 1 ID: eli2 PosEli:False Cat: 2 Fuel: 2079 Spazio truppe: 1099 Spazio Cargo: 0 MaxT-O weight: 10589 OffLoad Weight: 1000 Consumo: 668
01-18 -> OPZIONI ELICOTTERO: raggiunto record iniziale.
01-18 -> MAIN WINDOW: Effettuato inserimento del NUMERO di truppe: 159
01-18 -> TROOPS MANAGER: effettuata creazione manager per truppe
01-18 -> MAIN WINDOW: VISUALIZZUATA SCHERMATA OPZIONI TRUPPE
01-18 -> OPZIONI TRUPPE: effettuato l'inserimento di 159 soldati del peso singolo di: 50kg
01-18 -> MAIN WINDOW: Effettuato inserimento di ALTRO CARGO: 0
01-18 -> MAIN WINDOW: inserita la distanza della landing zone dall'unità navale: 50 NM
01-18 -> MAIN WINDOW: Effettuato inserimento del NUMERO di SPOT: 2
01-18 -> OPZIONI SPOT: EFFETTUATA APERTURA FINESTRA OPZIONI SPOT
01-18 -> OPZIONI SPOT: inserto nuovo record#__1 Elementi totali presenti: 1
01-18 -> Record_Spot: 0 Efficienza: True Spot occupato: False Da Eli: null GIORNO Cat1: True GIORNO Cat2: False GIORNO Cat3: False NOTTE Cat1: True NOTTE Cat2: False NOTTE Cat3: False
01-18 -> OPZIONI SPOT: inserto nuovo record#__2 Elementi totali presenti: 2
01-18 -> Record_Spot: 1 Efficienza: True Spot occupato: False Da Eli: null GIORNO Cat1: True GIORNO Cat2: False GIORNO Cat3: False NOTTE Cat1: True NOTTE Cat2: False NOTTE Cat3: False
01-18 -> SIMULATOR: simulatore creato. Effettuo l'inizializzazione....
01-18 -> - SIMULATORE: INIZIO MOVIMENTAZIONI SUL PONTE
01-18 -> SIMULATOR: rimosso elicottero eli1 ,nessuno spot accessibile
01-18 -> SIMULATOR: rimosso elicottero eli2 ,nessuno spot accessibile
01-18 -> SIMULATOR : controllo effettiva NECESSITA' degli elicotteri inseriti per la missione
01-18 -> SIMULATORE: ciclo inserimento e disposizione iniziale ultimato
01-18 -> SIMULATORE: INIZIO SIMULAZIONE, starting X timer"
01-18 -> TERMINE OPERAZIONI 00:00:01
01-18 -> SIMULAZIONE FALLITA

8 COMPILAZIONE ED ESECUZIONE

Per lo sviluppo progettuale si è utilizzo l'ambiente di sviluppo **Xamarin Studio versione 4.4.2** (build 2). Il programma è stato testato efficacemente su sistemi con un SO Windows 7, in particolare sono stati effettuati i test su due macchine quali:

Desktop : processore Intel core i3 3.3Ghz, 8gb RAM

Portatile (samsung serie 5): processore Intel core i3 1.5 Ghz, 4gb RAM

Su entrambe le macchine la simulazione è avvenuta con successo senza rallentamenti.

Il progetto è stato definito per una risoluzione di schermo abbastanza comune 1024*768 che deve necessariamente essere accettata dalla macchina che lo esegue. Attualmente, per mantenere il progetto il più semplice possibile, non sono state previste altre risoluzioni che potrebbero comunque essere implementate in futuro.

Purtroppo alcune caratteristiche non ancora portate delle librerie open source di MONOGAME non hanno permesso il pieno utilizzo su piattaforme con SO linux a causa di alcune deficienze del porting.

La simulazione, come accennato, fa uso di MONOGAME per le librerie grafiche per la visualizzazione degli Sprite e delle immagini di elicotteri, truppe e cargo. MONOGAME è un clone open source di XNA (framework comunemente usato per lo sviluppo di Video Games).

MonoGame per funzionare correttamente sfrutta delle librerie Open Source OpenAL che è necessario installare prima di poter compilare il programma.

E' necessario inoltre installare il *Tao Framework* che include le librerie SDL (Simple DirectMedia Layer) una libreria multimediale multi piattaforma sviluppata in C che crea un livello astratto al di sopra delle altre piattaforme software grafiche e sonore.

Per iniziare la simulazione è necessario caricare il file di progetto di Xamarin: *EliassaltoAnfibio.sln*

Bisogna assicurarsi di aver installato tutti i riferimenti necessari sopradescritti (le dll necessarie sono già comunque presenti nella directory di *debug* del progetto).

Assicurarsi pertanto che nella directory di debug siano presenti: Monogame.Framework.dll, OpenTK.dll, sdl.dll, sdl_mixer.dll, Tao.OpenAl.dll, Tao.OpenGl.dll, Tao.Sdl. Tutte queste librerie vengono richiamate dal programma durante la sua esecuzione e pertanto sono necessarie alla sua compilazione.

Una volta che le librerie sono correttamente posizionate è possibile avviare la compilazione, inserire le informazioni richieste e verificare l'esito della simulazione. Al termine della simulazione, all'interno della directory di */bin/debug/* verrà salvato il file di Log *LogEliAssault_<orario>.txt* che sarà possibile visionare separatamente per avere riscontro sui dati sviluppati dal simulatore.

Per ottenere **un file eseguibile** è necessario impostare Xamarin su Release è copiare tutte le DLL sopra descritte nella directory di release, l'ambiente di sviluppo sarà in grado di generare un file eseguibile e stand alone.

Altri software utilizzati per lo sviluppo del programma e per la definizione dei content

- Enterprise Architect per lo sviluppo UML del progetto
- Modelio 3.1 per lo sviluppo degli USE CASES
- Microsoft Visio per la creazione di diagrammi di flusso/ di stato
- GIMP per il modelling delle immagini utilizzate nel content del progetto