ESAME DI FONDAMENTI DI INFORMATICA T-2 dell' 8/7/2019
Proff. E. Denti, R. Calegari, A. Molesini – Tempo: 4 ore

NOME PROGETTO ECLIPSE: CognomeNome-matricola (es. RossiMario-0000123456)
NOME CARTELLA PROGETTO: CognomeNome-matricola (es. RossiMario-0000123456)
NOME ZIP DA CONSEGNARE: CognomeNome-matricola.zip (es. RossiMario-0000123456.zip)

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

Si ricorda che compiti *non compilabili* o *palesemente lontani da 18/30* NON SARANNO CORRETTI e causeranno la verbalizzazione del giudizio "RESPINTO"

Poiché oggigiorno è possibile scaricare da appositi siti web un file che descrive l'intero percorso di un volo aereo in una certa data, è stato richiesto lo sviluppo di un'applicazione grafica in grado di mostrare il percorso di tale volo su un (semi-)planisfero.

DESCRIZIONE DEL DOMINIO DEL PROBLEMA

Appositi siti (es. FlightRadar24) consentono di seguire in tempo reale la posizione dei voli aerei, nonché di scaricare un file contenente la traccia completa di un volo, a scelta fra tutti quelli conclusi negli ultimi due o tre giorni. Il file descrive il volo dal momento in cui inizia a muoversi dal gate dell'aeroporto di partenza a quello in cui parcheggia al gate dell'aeroporto di arrivo: pertanto, è composto tipicamente da alcune centinaia di rilevazioni, disposte su altrettante righe, nel formato dettagliato più oltre. Ogni rilevazione è caratterizzata da:

- il timestamp, in formato UTC
- la posizione (latitudine e longitudine), in gradi, con 6 cifre decimali
- l'altitudine rispetto al terreno, in piedi (valore intero)
- la velocità, in nodi nautici (valore intero)
- la direzione, in gradi (nord = 0°) (valore intero)

La prima e l'ultima rilevazione hanno velocità e altitudine 0, a conferma che l'aereo è fermo a terra al gate.

Cose da ricordare

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

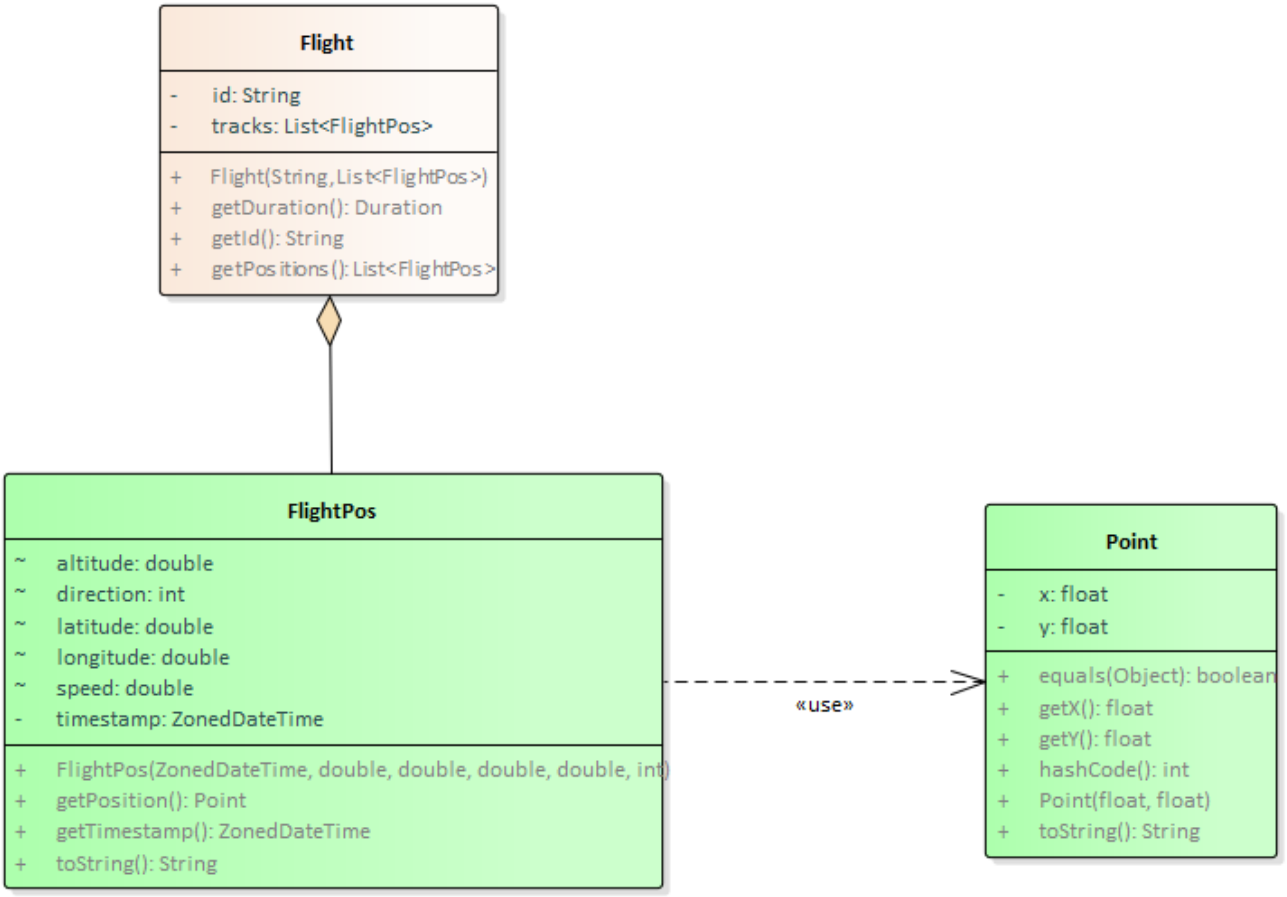
Checklist di consegna

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

Parte 1 (punti: 16)

Modello dei dati (package flightTracker.model) *(punti: 5)*

Il modello dei dati deve essere organizzato secondo il diagramma UML sotto riportato.



SEMANTICA:

a) La classe ***Point*** (fornita) rappresenta una posizione (latitudine, longitudine), con relativi metodi accessor

b) La classe ***FlightPos*** (fornita) rappresenta una rilevazione, caratterizzata dalle sei proprietà descritte nel *Dominio del Problema*, rese accessibili de appositi accessor

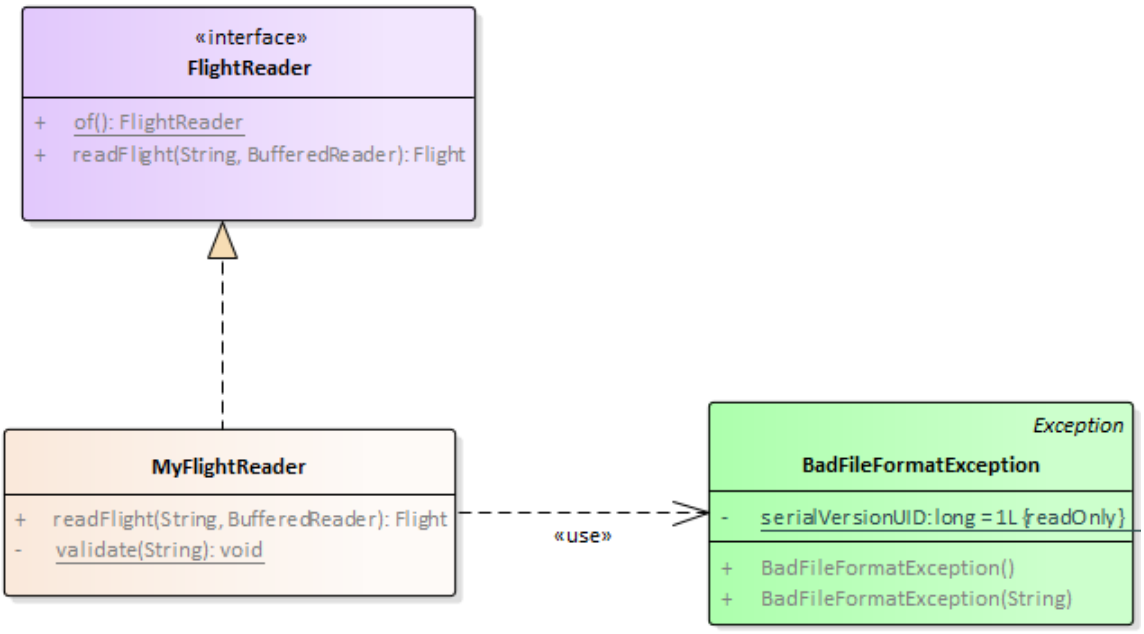
c) La classe ***Flight*** (da realizzare) rappresenta un volo, caratterizzato dal suo identificativo univoco e dalla lista delle rilevazioni (***FlightPos***): il costruttore deve effettuare opportuni controlli sulla validità degli argomenti ricevuti, lanciando ***IllegalArgumentException*** in caso contrario, con idonea messaggistica; la classe deve esporre ovvi metodi accessor, nonché il metodo `getDuration` che calcoli la durata del volo.

Persistenza (package flightTracker.persistence) *(punti 11)*

Sono forniti svariati file ".csv" nel formato fornito da FlightRadar24: il nome del file incapsula l'identificativo del volo e la data di effettuazione, nel formato *FlightID_AAAAMMGG* (ad esempio "AZ604_20190510.csv"). La prima riga contiene l'intestazione delle cinque colonne, separate da punti e virgola, mentre le successive contengono ciascuna un rilevazione, i cui campi sono anch'essi separati da punti e virgola. Da notare però che latitudine e longitudine costituiscono un unico campo, separato al suo interno da una virgola:

```
UTC;Position;Altitude;Speed;Direction
2019-05-10T10:54:39Z;45.661972,8.726303;1975;183;356
2019-05-10T10:54:49Z;45.67049,8.725822;2450;182;358
2019-05-10T10:55:11Z;45.689159,8.726234;3100;185;1
...
```

La struttura di questa parte dell'applicazione è illustrata nel diagramma UML sotto riportato.



SEMANTICA:

a) L'interfaccia ***FlightReader*** dichiara il metodo `readFlight` che legge – da un ***BufferedReader*** ricevuto come argomento – i dati di un volo e li incapsula in un'opportuna istanza di ***Flight***, lanciando le eccezioni ***IOException*** o ***BadFileFormatException*** rispettivamente nel caso si verifichino errori di IO o il formato del file differisca da quello atteso. È anche presente un metodo factory `of` che istanzia un ***MyFlightReader***.

b) La classe ***MyFlightReader*** (da realizzare) implementa tale interfaccia nel caso specifico del formato dei file .csv di questo caso concreto, fornendo specifici messaggi d'errore nelle eccezioni lanciate, così da distinguere le sorgenti di errore. In particolare occorre validare la riga d'intestazione, che deve contenere esattamente le cinque parole previste ("UTC","Position","Altitude","Speed","Direction") nel giusto ordine e formato.

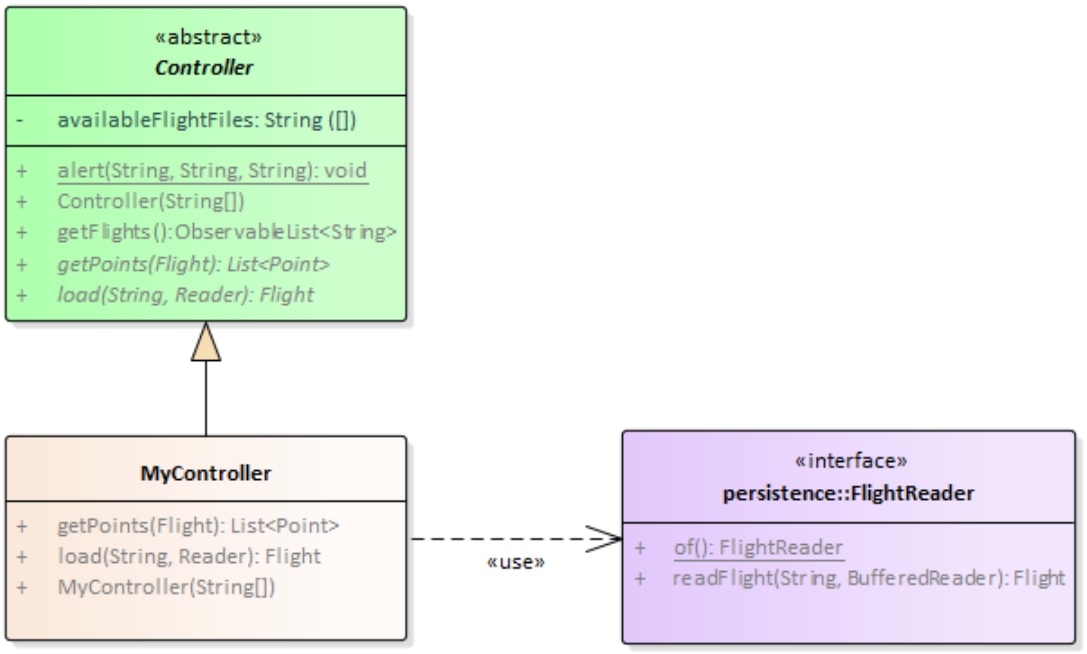
Parte 2 (punti: 14)

IMPORTANTE: per consentire il test della GUI anche nel caso di *reader* non funzionanti, è fornita la classe **FlightTrackerAppMock** che replica **FlightTrackerApp** utilizzando dati fissi pre-cablati al posto di quelli letti da file (utilizzando a tal fine un apposito **ControllerMock**).

L'applicazione deve permettere di scegliere il volo desiderato fra quelli disponibili e vederne il grafico sull'emisfero.

Con riferimento alle figure seguenti: la combo in alto a sinistra contiene i nomi dei file disponibili (Fig. 1). Avvicinandosi alla combo, compare un tooltip che invita a scegliere un volo (Fig. 2). Scegliendolo (Fig. 3), viene mostrato il relativo grafico. Se successivamente l'utente sceglie un altro volo, di default viene mostrato solo il nuovo volo (Fig. 4). Tuttavia, è possibile selezionare la checkbox a lato per abilitare la selezione multipla, nel qual caso scegliendo altri voli le relative tracce vengono aggiunte al grafico esistente, in colori casuali via via differenti (Fig. 5).

Controller (package flightTracker.ui.controller) *(punti 5)*

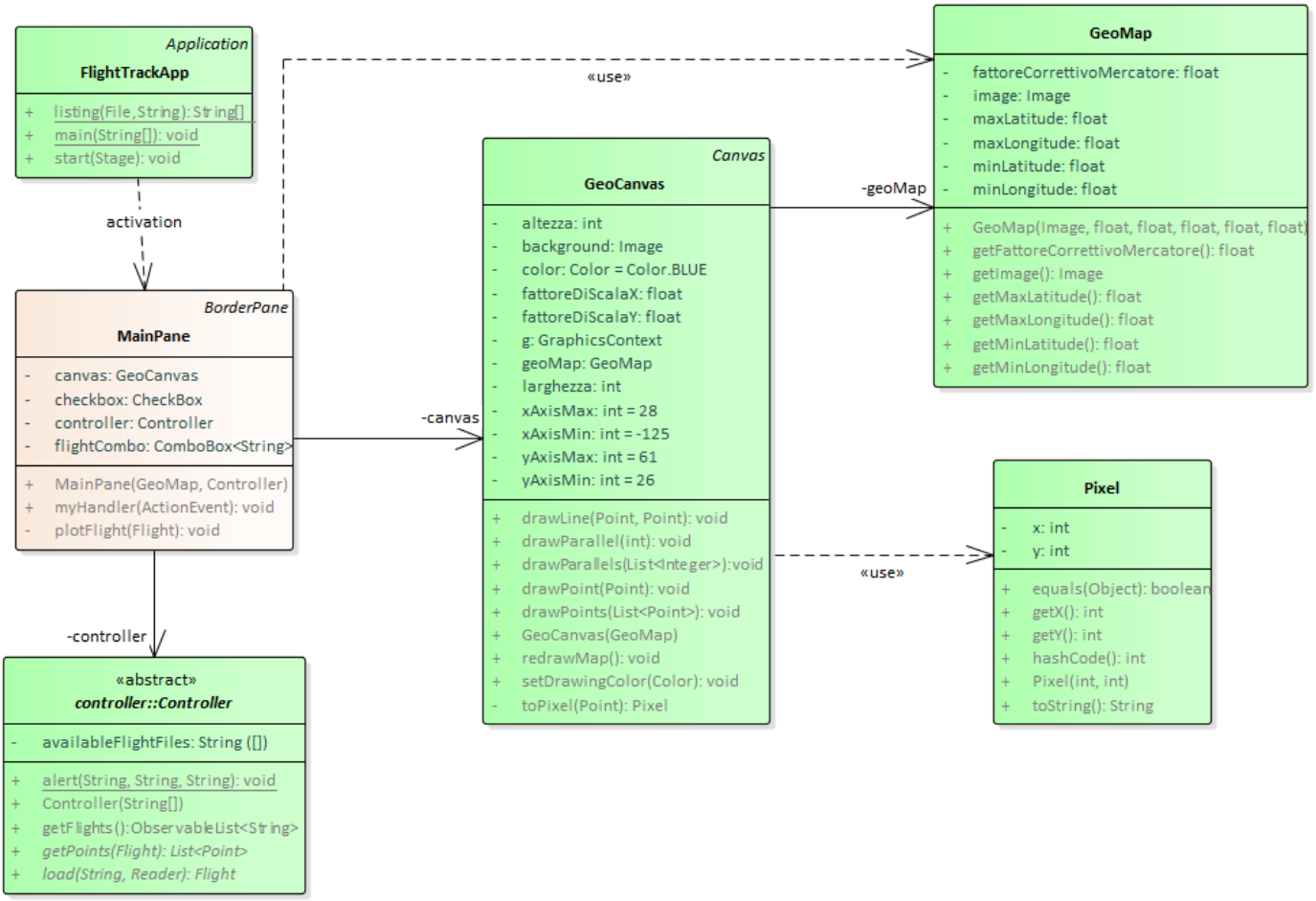


SEMANTICA

a) La classe astratta ***Controller*** (fornita) fornisce un'implementazione parziale del controller dell'applicazione:
   - Il costruttore riceve una ***GeoMap*** e la lista dei nomi dei file .csv disponibili: effettua opportuni controlli sulla validità degli argomenti ricevuti, lanciando ***IllegalArgumentException*** in caso contrario;
   - Il metodo `getFlights` restituisce la lista osservabile dei nomi di file corrispondenti ai voli disponibili
   - Il metodo statico `alert` permette di far comparire un dialogo di avviso per segnalare errori (eccezioni).

b) La classe ***MyController*** (da realizzare) estende ***Controller*** implementando
   a. il metodo `getPoints` che restituisce la lista di ***Point*** corrispondenti al volo passato come argomento: in pratica, dato un volo, recupera le singole posizioni (latitudine e longitudine) delle varie rilevazioni, le converte ciascuna nel ***Point*** corrispondente e restituisce la lista così ottenuta (pronta per il metodo `plotFlight` del ***MainPane***)
   b. Il metodo `load` che, dato un ***Reader***, legge – tramite ***FlightReader*** – il file con la descrizione del volo: se qualcosa va storto (errore di I/O o errori di formato), l'eccezione va semplicemente rilanciata all'esterno, in modo da essere gestita nella GUI.

Interfaccia utente (package flightTracker.ui) *(punti 9)*



L'applicazione deve permettere di scegliere il volo desiderato fra quelli disponibili e vederne il grafico sull'emisfero.

La struttura generale è quella di un *pannello a bordi*, di cui sono utilizzati solo la parte top per le barra controlli e il centro per l'emisfero con la traccia di volo. Se il caricamento preliminare (realizzato nella ***FlightTrackApp*** fornita nello start kit) ha esito positivo, deve comparire subito la finestra principale dell'applicazione, con la combo pre-popolata con tutti i nomi dei file .csv presenti nella cartella corrente (Fig. 1). Avvicinandosi alla combo, un tooltip suggerisce all'utente cosa fare (Fig. 2). Quando l'utente sceglie un volo, l'applicazione visualizza immediatamente la traccia corrispondente (Fig. 3): cambiando volo, la visualizzazione si aggiorna (Fig. 4) mostrando quest'ultimo.

Se tuttavia viene attivata la checkbox "Voli multipli", cambiando voli i nuovi si aggiungono, *con colori diversi,* alla mappa preesistente, che quindi mostra tutte le tracce insieme (Fig. 5).

SEMANTICA

a) La classe ***FlightTrackApp*** (fornita) contiene il main dell'applicazione, il cui metodo `start` costruisce la ***GeoMap***, produce la lista dei nomi dei file .csv disponibili, istanzia il controller e il ***MainPane***, e infine attiva la scena

b) La classe ***Pixel*** (fornita) rappresenta una posizione grafica con coordinate intere

c) La classe ***GeoMap*** (fornita) rappresenta una mappa geografica, caratterizzata da un'immagine e dalla corrispondente estensione orizzontale (da longitudine minima a longitudine massima) e verticale (da latitudine minima a latitudine massima); la descrizione è completata da un fattore di scala (*fattore correttivo Mercatore*) che tiene conto, in modo approssimato, della non linearità delle mappe nella proiezione di Mercatore comunemente utilizzata. Tale valore (1 per mappe che non necessitano di correzione) è normalmente molto prossimo a 1 per mappe che inquadrano solo una piccola porzione di territorio e/o si estendono poco in senso

verticale, mentre dev'essere proporzionalmente ridotto per mappe che abbracciano un'ampia fetta dell'emisfero. Per la mappa fornita nello start kit, un valore ragionevole è 0.87-0.88, impostato nel main.

d) La classe ***GeoCanvas*** (fornita) incapsula tutta la logica di disegno di una lista di punti su un piano cartesiano sovrapposto a una ***GeoMap.*** Sono forniti metodi per:

- disegnare un singolo ***Point***, una lista di ***Point***, o una retta fra due ***Point*** (`drawPoint`/`-s`, `drawLine`)
- disegnare un parallelo o una lista di paralleli alla/e latitudine/i specificata/e (`drawParallel`/`-s`)
- impostare il colore di disegno (default: blu) (`setDrawingColor`)
- ridisegnare la mappa allo stato iniziale (ossia senza tracce di voli sopra) (`redrawMap`)

e) La classe ***MainPane*** (da realizzare) implementa il pannello principale che contiene la barra comandi (in alto) e il GeoCanvas (al centro), gestendo gli opportuni eventi per ottenere il comportamento desiderato. In particolare

- la gestione dell'evento dev'essere incapsulata in un apposito metodo `myHandler`
- nella gestione dell'evento, in caso la lettura del file non vada a buon fine deve comparire un apposito dialogo col messaggio appropriato, distinto per tipo di eccezione, avvalendosi del metodo statico ***Controller.alert***

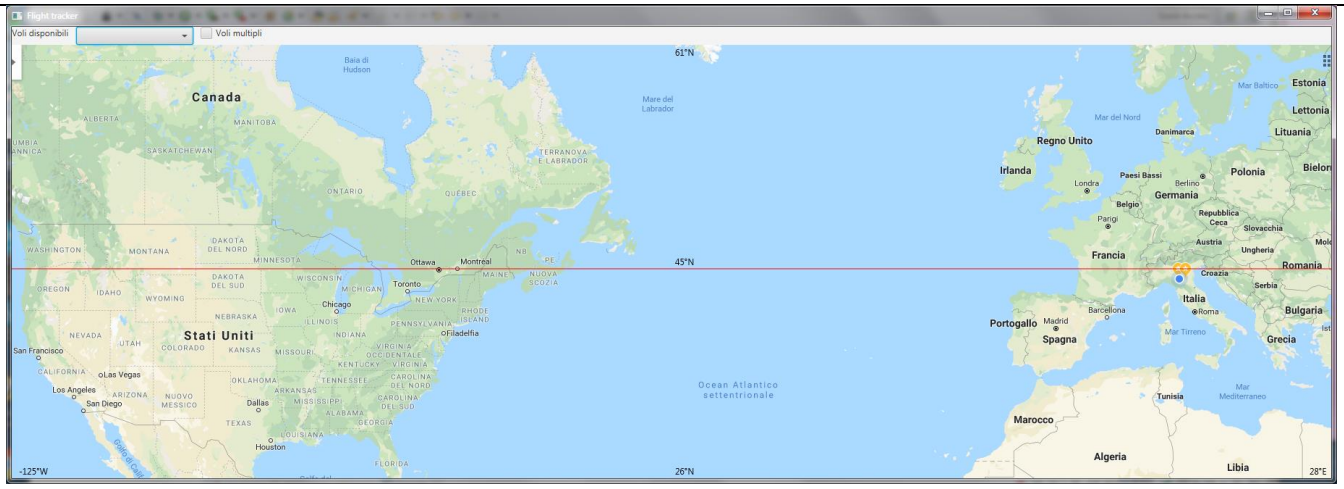


Fig. 1

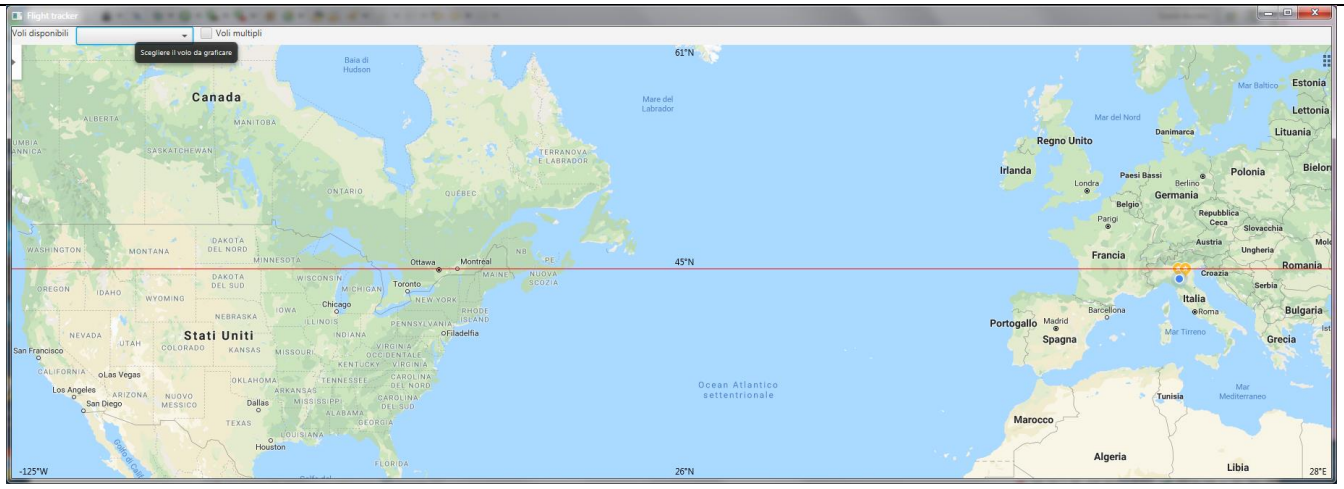


Fig. 2

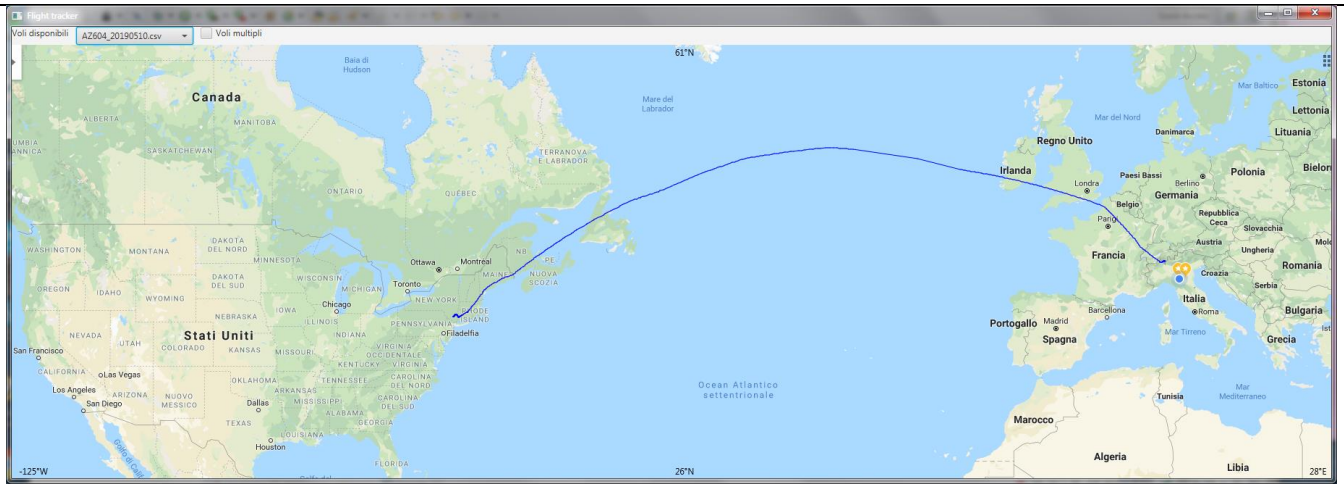


Fig. 3

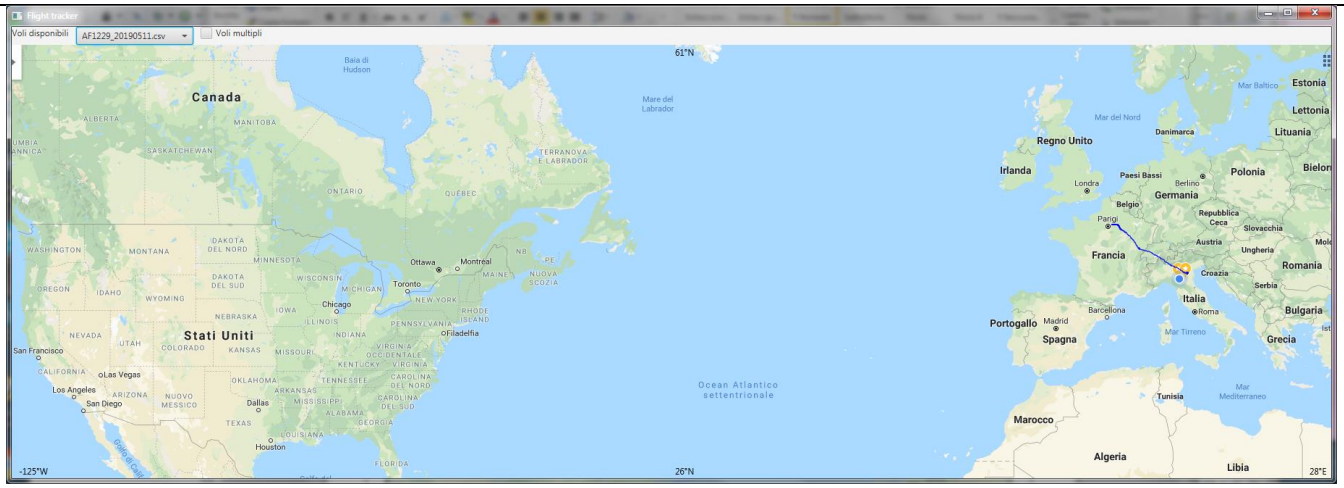


Fig. 4

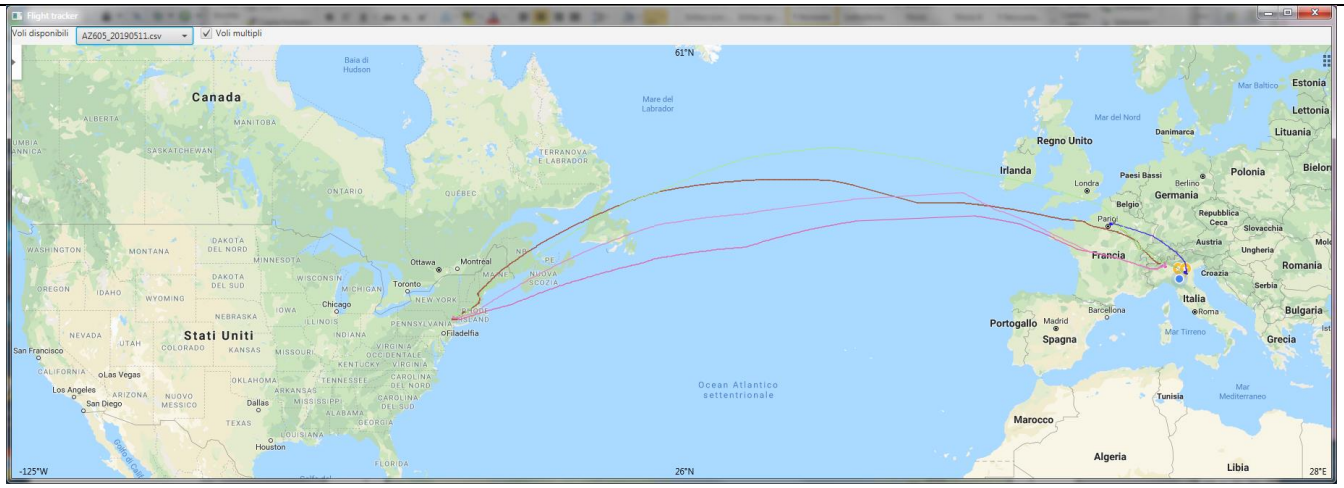


Fig. 5