Corso di Programmazione a Oggetti,
Università degli Studi di Padova,

Progetto: **DroneSim**

di Bicciato Federico,
matricola 1046373

Anno accademico 2017 – 2018

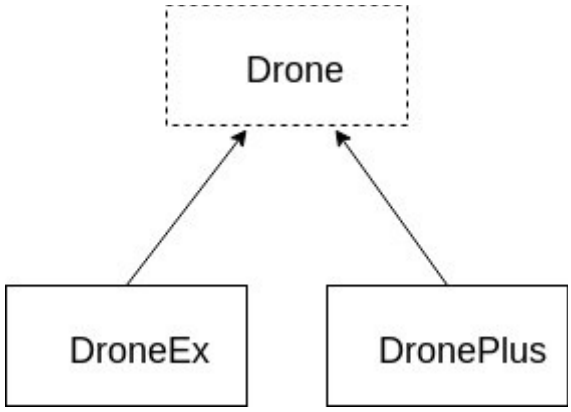
Introduzione e scopo del progetto

Il progetto richiede la progettazione e codifica di una calcolatrice che utilizzi una gerarchia di tipi di dato a piacere.
Il tipo di dato scelto è il drone. Il progetto è quindi stato realizzato nella forma di un simulatore di guida di un drone.
Il simulatore offre varie e semplici funzionalità, la gran parte delle quali sono semplici forme di calcolo: accensione, spegnimento, utilizzo della batteria, movimento in due dimensioni lungo gli assi x e y, e cambio della modalità di guida.
Le modalità di guida sono state scelte arbitrariamente, e consentono di effettuare movimenti differenti consumando differenti quantità di batteria. La modalità Plus consente spostamenti verso i quattro punti cardinali (N,S,E,W), mentre la Ex consente di spostarsi di 45° rispetto a essi (NW, NE, SW, SE). Si può cambiare da una modalità all'altra e, a seconda della destinazione da raggiungere, una modalità può essere più vantaggiosa di un'altra, e giustifica quindi il cambio.

Lo scopo del drone è quello di raggiungere una destinazione nel modo più efficiente possibile, cioè consumando la minima quantità di batteria. Ciò può (e verosimilmente accade) il cambio da una all'altra modalità di volo. Una volta raggiunta la meta, la batteria si resetta e una nuova destinazione è impostata.

Descrizione delle gerarchie di tipi

La gerarchia del modello utilizza tre tipi di dato:



La classe base astratta Drone è estesa dalle concrete DroneEx e DronePlus. Vediamole.

Drone: rappresenta un generico drone, e implementa le sue funzionalità di base.
I suoi campi dati indicano lo stato di accensione o spegnimento (*on*), la quantità di batteria rimasta (*battery*), il consumo standard di batteria per le operazioni della classe base che la richiedono (*batteryUsage*) e la posizione del drone (*position*), espressa dalla struttura dati *coord*.
Per quanto riguarda i metodi, è fornito un costruttore per essere invocato dalle sottoclassi, e un distruttore virtuale puro che rende Drone una classe astratta; *isOn* rileva lo stato di accensione; *turnOn* e *turnOff* tentano di accendere o spegnere il drone; *getBattery* informa circa la disponibilità di batteria, mentre *getPosition* ritorna la posizione corrente; *getName* fornisce l'interfaccia per le sottoclassi per ottenere il nome della modalità del drone; infine *move* consente il movimento indiscriminato (sempre possibile) del drone.

DroneEx / DronePlus: le sottoclassi rappresentano due modalità specifiche in cui il drone può trovarsi. Esse ne influenzano consumo di batteria e movimento. Ridefinendo il campo *batteryUsage*, le operazioni definite nelle classi derivate costano più batteria; *getName* è implementato e dice la modalità attiva (Ex o Plus) di volo del drone; *mode* è ridefinito, e controlla che il tipo di movimento preso in input sia appropriato, lanciando un eccezione altrimenti.

Motivazioni per la scelta implementativa del metodo *move*
Il metodo *move*, definito nella classe *drone* e ridefinito in *droneEx* e *dronePlus*, è il cuore delle classi, costituendo la funzionalità più significativa di un drone. É stato scelto di implementarlo parzialmente nella classe base, completandolo poi nella classe privata.
I metodi privati *useBattery* e *shiftPosition*, chiamati da *move* nella classe base, *sono* legati al concetto di drone, e non a una sua derivazione specifica: un drone potrà sempre muoversi e dovrà sempre consumare batteria. Come questo accade nei dettagli è poi specificato e controllato nelle sottoclassi.
Se si fosse scelto di rendere *move* un metodo virtuale puro, esso non avrebbe potuto sfruttare l'ereditarietà di implementazione dei metodi privati della classe base, che sarebbero dovuti essere resi pubblici o protetti per consentirne l'accesso nella sottoclasse. Relegare tutto il codice di *move* alle sottoclassi avrebbe causato ripetizione e difficoltà di manutenzione del codice.

Descrizione uso codice polimorfo

I frammenti di codice polimorfo si trovano nella gerarchia del modello. In particolare, si tratta del distruttore virtuale puro *~drone()*, il metodo *getName* e il metodo *move* (non vi sono gerarchie nella GUI, in cui le classi sono connesse attraverso puntatori).
Il distruttore virtuale è prassi comune nelle gerarchie di tipi, poiché garantisce che i sottotipi siano completamente distrutti nei casi di istanziazione di un tipo derivato agganciato a un puntatore o riferimento del tipo della classe base. Il fatto che sia puro è anch'esso prassi, per rendere astratta la classe base.
Il metodo *getName* è la semplice stampa di una stringa che riporta il nome della modalità del drone attiva. Poiché la classe *drone* non è istanziabile e nemmeno rappresenta una modalità, il metodo è virtuale puro nella classe base.
Il metodo *move* è virtuale in quanto è necessario conoscere il sottotipo in uso per dire quali mosse esso possa fare. Utilizza comunque al suo interno delle funzioni della classe base (*useBattery* e *shiftPosition*), definite nella classe base e richiamate con binding statico poiché non dipendenti dalla modalità ma dall'essere drone.

Estensibilità dei tipi del modello

La gerarchia può essere estesa orizzontalmente aggiungendo modalità che comportano tipi diversi di movimento e consumi diversi di batteria. Ad esempio, si pensi a un drone che possa muoversi in tre direzioni disposte triangolarmente, come N, SW e SE.
Pensare di aggiungere una terza dimensione z alle coordinate è interessante e realistico. Si tratta di editare la struttura *coord* e i metodi di spostamento *shiftPosition* e *move*. A riguardo di ciò, sono due alternative. Scegliendo di implementare nella classe base il movimento verso l'alto e il basso, si creerebbe un prototipo di drone in grado di mantenere la quota in aria: appare giusto allora rendere la classe drone concreta. Se invece la si volesse conservare astratta, implementando tutto il movimento nei sottotipi, la gerarchia potrebbe essere molto varia.
Per semplicità, potrebbe rimanere essere estesa del tutto in orizzontale, definendo modalità indipendenti tra loro e con diversi gradi di bilanciamento tra volo verticale, orizzontale e angolare.

Breve manuale utente

Lo scopo dell'utente è di raggiungere la destinazione, ovvero far combaciare le coordinate del drone, mostrate negli LCD in alto, con quelle del target, negli LCD in basso. Ciò va fatto tenendo conto dell'uso della batteria, che si resetta a ogni meta raggiunta.
Nella parte superiore, come già accennato, si trova l'interfaccia, composta di label e schermi LCD.
Nella parte inferiore, invece, stanno i controlli, composti interamente di bottoni.
I quattro bottoni a destra consentono di accendere e spegnere il drone, e switchare tra modalità.
I bottoni direzionali, invece, si attivano a seconda della modalità in uso, e permettono il movimento del drone.

Considerazioni sullo svolgimento del progetto

Laddove la codifica del modello in C++ ha richiesto dei brevi ripassi sul libro, la parte Java ha richiesto di mantenere consistenti le funzionalità già definite usando un linguaggio e un paradigma leggermente diverso.
La parte predominante del lavoro è stato l'apprendimento e l'utilizzo delle librerie di Qt: in particolar modo, la comprensione della sua gerarchia di classi, il corretto uso dei layout annidati, e il passaggio e la rielaborazione dei segnali tra classi mediante puntatori.

Uso delle ore

Progettazione: 3h
Codifica:
- Model: 12h
- GUI: 30h
- Java: 7h
- relazione: 2h

Ambiente di sviluppo

OS: Linux, distro Ubuntu 16.04.5 LTS (xenial), 64bit
g++: versione 5.4.0
Qt: QtCreator 4.7.0, basato su Qt 5.11.1
Java: OpenJDK v. 1.8.0_181

Istruzioni per eseguire

Importare DroneSim0.pro in QtCreator; build; run.