Polygonator 2.0

Relazione sul progetto "Kalk"

Sonia Menon, matricola 1144731

Generali sul progetto:

Realizzato da Sonia Menon, matricola 1144731 e Riccardo Dalla Via, matricola 1142860

Titolo del progetto: Polygonator versione 2.0

Tempo impiegato:

- Analisi del problema, progettazione e bozza grafica: 6h + 1h
- Codifica Model: 18h + 3h
- Codifica View: 8h + 2h
- Codifica Java: 8h + 1h
- Debugging: 6h + 30m

Al tutto si aggiungono circa 15h + 2h totali di ricerca e apprendimento durante le fasi di codifica.

I tempi massimi di circa 50h sono stati superati, per la necessità di apprendimento della libreria Qt e di ricerca di metodi ottimali per la risoluzione di problemi riscontrati durante la codifica del progetto.

Le ore totali comprendono quelle impiegate per la prima versione del progetto, con un'aggiunta di ore per la versione 2.0: la nuova versione è stata pensata per migliorare alcune lacune della vecchia, ampliando la gerarchia (con Pentagon ed Hexagon), aggiungendo una funzionalità (possibilità di

scelta unità di misura e calcolo dell'apotema), perfezionando la grafica(menù di scelta figure con frecce per lo scorrimento, grafico migliorato) e migliorando alcuni algoritmi nella view e nella model.

SUDDIVISIONE DEL LAVORO:

La prima parte del progetto, cioè la progettazione e creazione delle prime classi base (Polygon, Point, Line), è stata fatta principalmente in collaborazione, con successive eventuali modifiche minori effettuate dai singoli, come anche la progettazione della versione 2.0. Anche il manuale utente è stato creato univocamente in coppia.

La mia parte del lavoro comprendeva principalmente:

- Ideazione grafica dell'applicazione
- Documentazione del codice C++ e Java
- Debugging e revisione del codice C++ e della parte grafica

C++/ Model:

- Creazione e implementazione della classe Function
- Implementazione di buona parte dei metodi delle classi della gerarchia (comprese le nuove classi Pentagon ed Hexagon) e gestione dell'ereditarietà

C++/View-Controller, creazione e gestione delle classi:

- OutputWindow
- FunctionChoiceModule
- ResultModule
- RoundedToggleSwitch
- RoundedToggleSwitchLabel
- Parte di MainWindow e MainWindowController
- PointChoiceModule
- Scelta e gestione unità di misura

Java:

- Buona parte del porting del progetto

AMBIENTE DI SVILUPPO:

SISTEMA OPERATIVO	WINDOWS 10 HOME
IDE UTILIZZATI	CLION 2018.1.5 (C++) , INTELLIJ IDEA 2018.1.5 (JAVA)
VERSIONE DI C++	C++11
VERSIONE DI QT	QT 5.5
VERSIONE DI JAVA	JAVA 1.8

Compilazione ed esecuzione:

Il nostro progetto prevede l'utilizzo di C++ 11 (ad esempio con le *lambda expressions*, la keyword '*nullptr*' e *'auto'*), pertanto forniamo il file ***Polygonator.pro***, diverso da quello ottenuto dall'esecuzione del comando ***qmake -project***.

I comandi necessari sono quindi ***qmake*** (che genera il Makefile), e ***make***, mentre per l'esecuzione si utilizza ***./Polygonator***.

Polygonator

Polygonator è un'applicazione desktop per eseguire calcoli su delle figure, a partire dall'inserimento di punti da parte dell'utente:

- Un punto: è possibile calcolare il fascio di rette passante per quel punto.
- Due punti: è possibile calcolare la lunghezza del segmento, il punto medio e l'equazione della retta passante per due punti.
- Tre punti: è possibile calcolare area, perimetro, base, altezza, apotema, bisettrici e mediane del triangolo ottenuto
- Quattro punti: è possibile calcolare area, perimetro, base, altezza (non disponibile per i quadrilateri generici), apotema (disponibili solo per i quadrati) e diagonali del quadrilatero ottenuto (inoltre, viene riconosciuta la figura specifica se non si tratta di un quadrilatero generico)

Versione 2.0:

Sono stati aggiunti due nuovi tipi di figure, visibili sul rinnovato menù di scelta:

- Cinque punti: è possibile calcolare area, perimetro, base, altezza e apotema (solo nei pentagoni regolari) e diagonali del pentagono ottenuto
- Sei punti: è possibile calcolare area, perimetro, base, altezza e apotema (solo negli esagoni regolari) e diagonali dell'esagono ottenuto

La nuova applicazione prevede ora la possibilità di selezionare l'unità di misura desiderata (mm, cm, metri, inches, foot, yards) e vedere i risultati con l'unità scelta.

È stato perfezionato il grafico che ora mostra la figura risultante in maniera più chiara e visibile, grazie ad una nuova gestione del ridimensionamento; inoltre, nella schermata dei risultati, vengono disegnate diagonali, bisettrici e mediane, quando selezionate.
Sono state aggiunte delle toolTip che mostrano maggiori informazioni in determinati elementi (ad esempio nei bottoni disabilitati).
Ora gli errori vengono mostrati in una barra a scomparsa nella parte alta della finestra (Figura 4).
Infine, è stato rivisto il metodo di riconoscimento di base e altezza dei poligoni.

Pattern Model-View-Controller

È stato scelto il pattern model-view controller per avere una migliore separazione e gestione di model e view.

Il pattern ha permesso di creare una model indipendente dalla grafica, che gestisce la gerarchia e le classi base, e fornisce tutti gli strumenti per eseguire i calcoli su di esse e gestire gli eventuali errori che si possono creare.

La view nel nostro caso implementa solamente la grafica e le interazioni con l'utente, senza però utilizzare in alcun modo la model.

Infine il controller si occupa di fornire alla view le informazioni necessarie da visualizzare utilizzando gli strumenti forniti dalla model ed eventualmente rielaborandoli.

Contenitori

Sono stati scelti contenitori diversi tra la model e la view per permettere al massimo il riutilizzo della model, eventualmente con librerie e framework diversi da Qt.
Gli elementi utilizzati nel progetto avevano generalmente necessità di essere ordinati, quindi la nostra scelta del contenitore è ricaduta su ***list<T>***, soprattutto per la possibilità di utilizzare metodi quali ***sort()***, tramite l'ausilio della ridefinizione degli operator (== , < , <= , > , >=) dei vari elementi T. La scelta ha sicuramente avuto dei lati negativi nel momento di dover utilizzare contemporaneamente più elementi T non consecutivi, cosa che sarebbe stata risolta scegliendo ad esempio un ***vector<T>*** , essendo esso indicizzato; tuttavia, un contenitore ordinato ad accesso sequenziale si è rivelato nella maggior parte dei casi più adatto.

Nella view, di conseguenza, è stata usata ***QList<T>*** : avendo la quasi esclusiva necessità di un accesso sequenziale degli elementi, una ***QList<T>*** rendeva più veloce ed efficiente lo scorrimento degli elementi.
Inoltre, nella view, è stata utilizzata anche ***QMap<k, V>*** , per poter avere facilmente l'associazione di alcuni valori keys (nel nostro caso, delle stringhe univoche necessarie ad identificare delle funzioni) ai loro values (nel nostro caso, i risultati corrispondenti).

Il contenitore usato in Java è di nuovo una ***List<T>***, che permetteva un funzionamento simile a quello usato in C++, e inoltre risolveva l'esigenza dell'accesso casuale.

GERARCHIA DI TIPI E USO DEL POLIMORFISMO

Di seguito è illustrata la gerarchia utilizzata nel progetto:

POLYGON

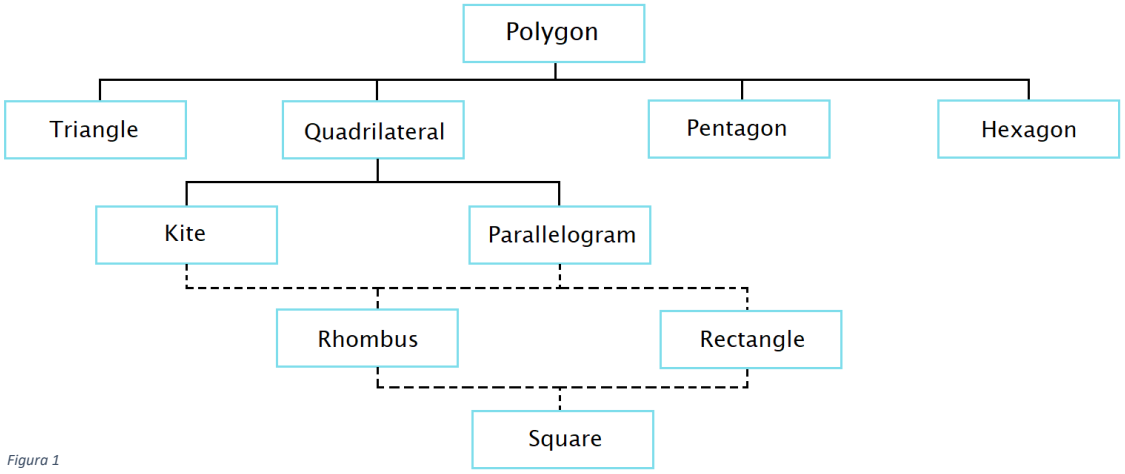

Figura 1

È la classe base della gerarchia; la classe è concreta, in quanto si prevede la presenza di poligoni irregolari non compresi nella gerarchia, che tuttavia può essere agevolmente ampliata per comprendere nuovi tipi di poligono.
Presenta i seguenti campi dati:

- ***bool concave*** è segnato come privato, e serve a controllare se il poligono è concavo o convesso; pubblicamente accedibile in lettura con il metodo *isConcave()*.
- ***bool regular [v. 2.0]*** è segnato come privato, e serve a controllare se il poligono è regolare (cioè se è equilatero ed equiangolo); pubblicamente accedibile con il metodo isRegular().
- ***list<Point> vertices*** è privato, in quanto non si prevede che possa essere modificato, nemmeno nelle sottoclassi, e memorizza la lista di punti che formano il poligono; pubblicamente visibile in lettura con il metodo *getVertices().*

Polygon mette a disposizione una serie di metodi non virtuali, in quanto non si prevede che la loro implementazione generica debba essere rivista nelle sottoclassi, dei metodi privati utilizzati per effettuare controlli durante la creazione di un poligono, e dei metodi virtuali polimorfi:

- ***virtual double getApothemConstant() [v. 2.0]:*** il metodo ritorna, nei poligoni regolari, il numero fisso (cioè il rapporto tra apotema e lato caratteristico di ognuno);
  (il calcolo dell'apotema, invece, è supportato da un metodo nella classe base, non virtuale);
- ***virtual double getBase() [v. 2.0]***: il metodo lancia un'eccezione nel caso in cui il poligono sia concavo (non si prevede a fini logici la base in tali poligoni), negli altri casi pone come lato base quello più parallelo all'asse delle ascisse.
- ***virtual double getHeight()***: lancia un'eccezione nei poligoni concavi o non regolari (tranne nel caso di triangoli e parallelogrammi, che ridefiniscono il metodo per poter effettuare il calcolo), in tutti gli altri casi calcola la distanza tra la base e il vertice opposto;
- ***virtual list<Line> getDiagonals()***: il metodo ritorna la lista delle diagonali di un poligono; viene rivisto solo nella classe Triangle per l'impossibilità di calcolarne le diagonali;
- ***virtual double getArea()***: ritorna l'area di un poligono generico; viene reimplementato in maniera più specifica ed efficiente nelle sottoclassi;

- ***virtual double getPerimeter()***: calcola il perimetro con una formula generica; anche per questo metodo, ci saranno implementazioni specifiche nelle sottoclassi;
- ***virtual string getName()***: fornisce una stringa identificativa per ogni classe.

TRIANGLE

La sottoclasse Triangle fa eccezione in quanto, oltre a non permettere il calcolo delle diagonali(come precedentemente spiegato), mette a disposizione dei nuovi metodi non previsti in Polygon:

- ***list<Line> getMedians()***: ritorna una lista di tutte le mediane del triangolo;
- ***Line getMedian(const Point &)***: calcola una mediana a partire da un punto del triangolo;
- ***list<Line> getBisectors()***: ritorna una lista di tutte le bisettrici di un triangolo;
- ***Line getBisector(const Point &)***: calcola una bisettrice a partire da un punto del triangolo.

Metodo ***static bool canBeCreatedWith(const Polygon&)***: il metodo, implementato in tutte le sottoclassi di Polygon, ritorna un booleano e controlla, appunto, se il poligono passato come parametro può formare il tipo di figura corrispondente la classe in cui viene definito; in altri termini, controlla se le specifiche geometriche del poligono generico soddisfano quelle della figura.

EREDITARIETÀ MULTIPLA

Le sottoclassi Rhombus e Square sono derivati rispettivamente da Kite e Parallelogram, e da Rhombus e Rectangle: per risolvere i problemi derivanti dall'ereditarietà multipla (la creazione multipla di sottoggetti delle classi basi comuni ai padri) è stata quindi usata una derivazione virtuale in tutte le classi padri.
La derivazione multipla è stata necessaria sia a fini logici di caratteristiche geometriche delle figure, sia anche alla necessità di ereditare metodi da entrambi i padri.

POLYGONFACTORY

La classe astratta PolygonFactory mette a disposizione un metodo pubblico
static Polygon *makePolygon(const list<Point> &)
e un metodo privato templatizzato
template<class T>
static T *checkPolygon(const Polygon &)

PolygonFactory non fa parte della gerarchia, ma ne fornisce un supporto.
Seguendo lo stile del factory, costruisce un poligono a partire da una lista di punti forniti al metodo *makePolygon*: il poligono ritornato sarà quello che meglio si adatta, cioè il più specifico possibile della gerarchia; per fare ciò, la classe implementa un metodo privato *checkPolygon*.

JAVA

In Java, non essendo prevista la possibilità di ereditare da più classi contemporaneamente, abbiamo ricostruito la gerarchia derivando unicamente dalla classe più affine (Rhombus deriva da Kite e Square deriva da Rectangle).
Inoltre, in sostituzione ai template, vengono usati i generics in PolygonFactory.

MANUALE UTENTE

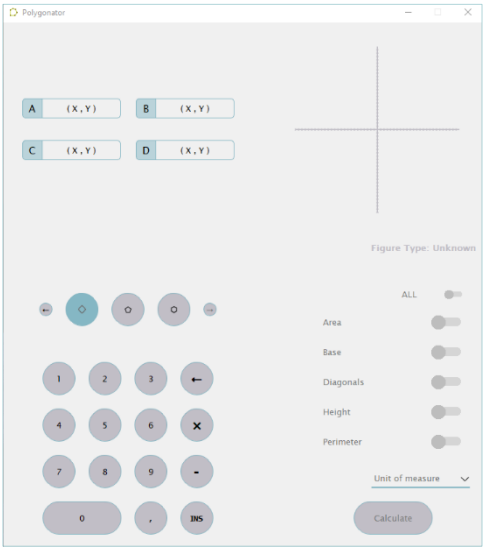

Figura 2

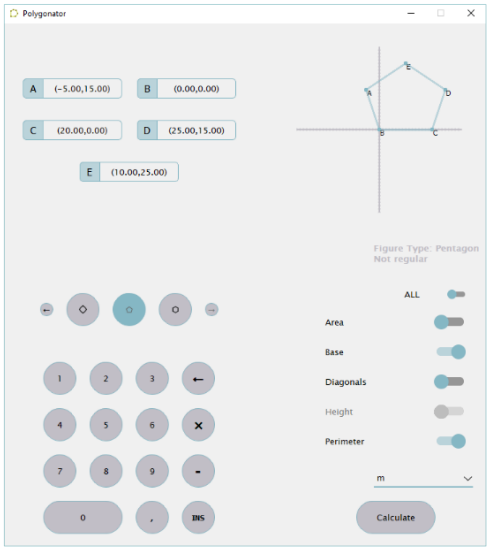

Figura 3

La calcolatrice, all'avvio, mostrerà la sua finestra principale di interazione con l'utente.

L'utente dovrà scegliere il numero di punti desiderato cliccando sull'iconcina rappresentante la figura adatta, o usando le shortcut da tastiera (Ctrl+P per il singolo punto, Ctrl+L per la linea, Ctrl+T per il triangolo, Ctrl+Q per il quadrilatero, Ctrl+P per il pentagono, Ctrl+H per l'esagono); di default, il numero di punti selezionato è quattro (*Figura 2*).

A seconda della figura scelta, appariranno corrispondenti box per l'inserimento dei punti necessari (in alto a sinistra), e tutte le operazioni possibili (in basso a destra).

Cliccando sui box di inserimento dei punti, si dovranno fornire le coordinate della x e y di ogni punto inserendole tramite tastiera, o usando il tastierino numerico fornito dall'applicazione: il tasto ← è associato all'usuale comportamento del DEL, mentre × elimina tutto il testo nella cella selezionata;
durante la digitazione, premendo INVIO si può confermare l'inserimento della prima coordinata, e successivamente dell'intero punto.
La scelta dei punti può far insorgere alcuni errori (compariranno in rosso, nella parte alta della finestra, *Figura 4*), i principali sono:

- "Point must be vertex": tre o più punti inseriti sono allineati, e non possono quindi formare un poligono;
- "Duplicated points found": due o più punti di un poligono sono uguali;
- "Cannot generate a line with two equal points": i due punti di una linea sono uguali.

Figura 4

Una volta definiti tutti i punti, la calcolatrice si occuperà di riconoscere la figura inserita: verrà creato automaticamente un grafico corrispondente, e mostrato il tipo di figura risultante (le figure possibili sono: Point, Line, Triangle, Quadrilateral, Kite, Rhombus, Parallelogram, Rectangle, Square, Pentagon, Hexagon) in base alle sue caratteristiche geometriche; verrà inoltre detto se la figura è regolare o no (*Figura 3*).
Inoltre, sarà ora possibile selezionare le operazioni desiderate (la scelta è disabilitata fino al completo inserimento dei punti), tramite switch On/Off di ognuna o di "ALL" per selezionarle/deselezionarle tutte, premendo poi semplicemente il tasto "Calculate"(in basso a destra) per ottenere i valori risultanti.

Si aprirà ora la finestra conclusiva, che mostra nuovamente il grafico, il tipo di figura, e la lista delle operazioni richieste con i rispettivi risultati (*Figura 5*).
Il tasto "Back" presente in questa pagina, permette all'utente di tornare alla pagina precedente per continuare ad usare l'applicazione ed effettuare nuovi calcoli.

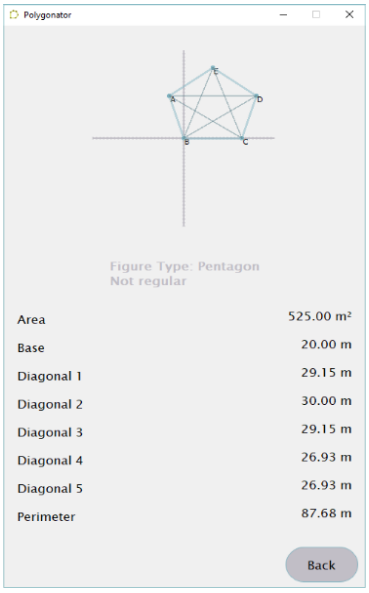

Figura 5