pyalgeo

NON ENTRI NESSUNO CHE NON CONOSCA LA GEOMETRIA

Motto all'entrata dell'Accademia di Platone

PyAlGeo (PYthon ALGorithmic GEOmetry) è un ambiente di *geometria dinamica* del piano realizzato in Python. Permette di gestire solo poche tipologie di entità geometriche: punti, segmenti, semirette, rette, circonferenze. Tutte le entità sono dinamiche, cioè modificabili interattivamente. Anche le espressioni (geometriche, numeriche e testuali) sono dinamiche nel senso che vengono ricalcolate (e visualizzate) contestualmente con le modifiche interattive (trascinamento dei punti mediante il mouse) alle entità geometriche coinvolte. All'interno dell'ambiente di PyAlGeo è possibile scrivere ed eseguire programmi in Python.

Il disegno viene realizzato mediante la grafica della tartaruga oppure mediante la costruzione e manipolazione di entità grafiche (punti, linee, ...).

PyAlGeo permette di sperimentare con la geometria in diversi contesti:

- geometria della tartaruga
- uso di oggetti dinamici (*Point*, *Segment*, *Ray*, *Line*, *Circle*)
- uso delle coordinate cartesiane

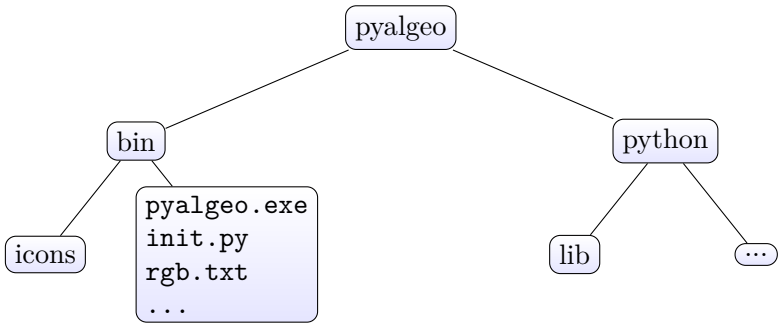
Attenzione: questa è una versione provvisoria ed incompleta del manuale; contiene sicuramente errori e ci sono delle funzionalità descritte che non sono perfettamente allineate con quanto effettivamente realizzato. [lc-1mar23]

0.1 Installazione

Per installare PyAlGeo si può procedere come segue:

1. scaricare il file `pyalgeo.zip` da `github.com/algmath/pyalgeo`
2. decomprimere il file `pyalgeo.zip` in una cartella (in una generica posizione)

Si ottiene la seguente struttura di cartelle:



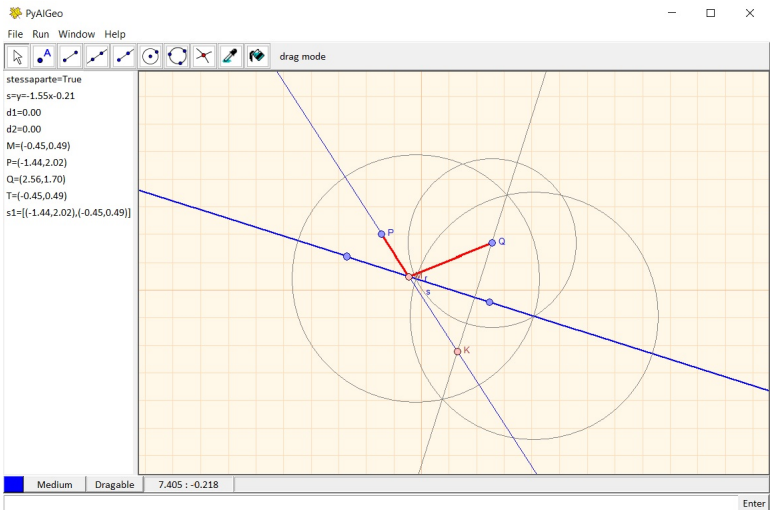
La cartella `bin` contiene il programma PyAlGeo; in particolare: il file `pyalgeo.exe` è il programma che attiva la finestra principale dell'applicazione; il file `init.py` viene automaticamente eseguito inizialmente e contiene le istruzioni di inizializzazione dell'ambiente; il file `rgb.txt` contiene la codifica dei nomi dei colori. Nella sotto cartella `python` vengono memorizzati i programmi Python scritti dall'utente; la cartella `lib` contiene i moduli Python che vengono importati nei programmi scritti dall'utente.

Il file `init.py` contiene alcuni comandi di configurazione iniziale dell'ambiente di PyAlGeo:

- `setviewport(`*xmin,ymin,xmax,ymax*`)`: setta la finestra dell'applicazione
- `setwindow(`*npixelunit*`)`: numero di pixel per unità logica
- `setprgpath(`*path*`)`: setta il percorso da dove caricare i programmi
- `setlibpath(`*path*`)`: setta il percorso delle librerie, da dove importare i moduli e le classi
- `setcolor(`*colore*`)`: setta il colore di disegno
- `setwidth(`*spessore*`)`: setta lo spessore delle linee di disegno
- `setstate(`*stato*`)`: setta lo stato delle primitive

0.2 Modalità d'uso di PyAlGeo

PyAlGeo si attiva mediante l'esecuzione del programma `pyalgeo.exe`. Al momento dell'esecuzione viene eseguito il file `init.py` presente nella cartella del programma; tale file contiene alcune istruzioni (in linguaggio Python) di inizializzazione dell'ambiente. Si presenta la seguente finestra.



Nell'ambiente di questa finestra sono predisposte le seguenti funzionalità:

- attivare le voci di menù corrispondenti alle seguenti funzionalità:
  - *File* > *Edit new* : scrittura di un nuovo programma
  - *File* > *Edit ...* : scrittura di un file che viene selezionato
  - *File* > *Exit* : termina il programma
  - *Run* > *Import ...* : selezione ed importazione di un modulo
  - *Run* > *Exec program ...* : selezione ed esecuzione di un programma
  - *Run* > *Exec last program* : esecuzione dell'ultimo programma
  - *Window* > *Grid* : attiva/disattiva il reticolo del piano
  - *Window* > *Clear* : elimina il disegno presente nella finestra
  - *Help* > *About* : visualizza la versione del programma
  - *Help* > *Manual* : visualizza il manuale del programma
- selezionare le funzioni di disegno mediante le icone; nell'ordine:
  - trascinamento dei punti base delle figure
  - inserimento di un *punto*
  - inserimento di un *segmento*
  - inserimento di una *retta*
  - inserimento di una *semiretta*
  - inserimento di una *circonferenza centro-punto*

  - inserimento di una *circonferenza per* 3 *punti*
  - intersezione di 2 linee
  - acquisizione degli attributi di una primitiva
  - impostazione degli attributi di una primitiva
- creare, modificare, muovere le entità grafiche in modo interattivo, selezionando la specifica icona
- selezionare gli attributi correnti di disegno (colore, stato, spessore) la specifica icona
- eseguire singole istruzioni sul campo di comando in fondo alla finestra

0.3 La grafica della tartaruga

La grafica della t. è integrata all'interno dell'ambiente *PyAlGeo* dove si può utilizzare la geometria della tartaruga come descritto a seguire.

0.3.1 Comandi elementari di disegno

A seguire sono elencati i comandi elementari di disegno tipici della grafica della tartaruga.

- `penup()`: alza la penna dal foglio
- `pendown()`: abbassa la penna sul foglio
- `ispendown()`: *TRUE* se la penna è abbassata sul foglio
- `forward(n)`: avanza di n passi
- `jump(n)`: salta in avanti di n passi
- `left(a)`: ruota a sinistra di a gradi
- `right(a)`: ruota a destra di a gradi
- `hide()`: nasconde la tartaruga
- `show()`: rende visibile la tartaruga
- `home()`: ripristina la condizione iniziale della tartaruga
- `clear()`: cancella quanto disegnato dalla t. (senza spostarla)

I passi della tartaruga sono misurati in unità virtuali, coerentemente con l'unità di misura del grid che si vede nella finestra grafica. Gli angoli vengono misurati in gradi. Inizialmente la penna è abbassata sul foglio; pertanto si può evitare il ricorso a *penup* e *pendown*, usando solamente *forward* e *jump*.

0.3.2 Gestione dello stato della tartaruga

La t. è un automa con stato, caratterizzato dai seguenti attributi:

- posizione
- angolo di avanzamento
- stato su/giù della penna
- colore della penna
- spessore della penna

Questi attributi vengono gestiti mediante i seguenti comandi:

- `setpos(x,y)`: sposta la t. sul punto di coordinate (x, y)
- `getpos()`: ritorna la coppia (x, y) delle coordinate della posizione della t.
- `setangle(a)`: imposta l'angolo a di avanzamento della t.
- `getangle()`: ritorna l'angolo di avanzamento della t.
- `setcolor(c)`: imposta il colore c di disegno della t.
- `getcolor()`: ritorna il colore corrente di disegno
- `setwidth(s)`: imposta lo spessore delle linee (valori: $1, 2, 3$)
- `getwidth()`: ritorna lo spessore corrente delle linee

I valori dei colori sono delle stringhe corrispondente al nome del colore: `'black'`, `'red'`, `'green'`, `'blue'`, ..., oppure una stringa che rappresenta valore esadecimale nella tradizionale codifica rgb: `'#000000'`, `'#FF0000'`, `'#00FF00'`, `'#0000FF'`, ...

È possibile salvare e ripristinare lo stato della t. mediante i seguenti comandi:

- `push()`: salva in una pila lo stato corrente
- `pop()`: ripristina l'ultimo stato salvato
- `init()`: ripristina lo stato iniziale

0.3.3 La tartaruga nel piano punteggiato

Un modo per dare evidenza grafica alle entità geometriche dei punti consiste nell'unire i punti mediante un segmento; in questo modo i punti diventano lo scheletro portante della figura. Il sistema grafico costituito dalla tartaruga e dalla associata penna può essere manipolato nel piano punteggiato dando vita alla geometria del piano punteggiato; i comandi, che possono essere pensati come rivolti alla penna, sono i seguenti:

- `pos()`: punto corrispondente alla posizione attuale della penna
- `move(P)`: muove la penna sul punto P senza tracciare linea; non modifica la direzione di avanzamento
- `draw(P)`: trascina la penna sul punto P tracciando una linea; non modifica la direzione di avanzamento
- `close()`: chiude la spezzata trascinando la penna sul punto dell'ultima *move*
- `look(P)`: orienta la tartaruga verso il punto P
- `dist(P)`: distanza della penna dal punto P
- `point()`: disegna un punto alla posizione attuale della penna

Una volta eseguito uno dei comandi descritti sopra, se si sposta il punto P la tartaruga non viene modificata; solo la funzione *dist* ritorna un valore dinamico che dipende dall'eventuale trascinamento del punto P.

Nota. E' possibile ambientare e gestire la tartaruga in un sistema di riferimanto cartesiano (geometria cartesiana) usando le funzioni di *Point* (costruttore e funzioni che ritornano le coordinate di un punto).

Nota. Il disegno della penna e della tartaruga non è né spostabile né agganciabile; per interfacciare la tartaruga con gli elementi grafici (*Line*, ...) bisogna usare le primitive grafiche della tartaruga (quelle elencate sopra) con argomento di tipo *Point* (quelle elencate sopra); in ogni caso il disegno tracciato rimane fisso e non trascinabile.

Nota. Un'assegnazione della forma

$$A = pos()$$

ha l'effetto di creare un punto A avente le coordinate uguali a quelle attuali della tartaruga; un eventuale spostamento della tartaruga non modifica il punto A; l'istruzione equivale all'azione di piantare una bandierina con nome A sul punto dove si trova la tartaruga.

0.3.4 La grafica della tartaruga in modalità ad oggetti

La grafica della tartaruga può essere impostata anche secondo la modalità orientata agli oggetti; in particolare:

- per creare una tartaruga t con specifici argomenti di inizializzazione viene richiamato il costruttore della classe ed utilizzata un'assegnazione della forma
$$t \leftarrow Turtle(\dots)$$
Il costruttore *Turtle* può avere degli argomenti quali il colore e lo spessore della penna
- per fare eseguire un'*azione* (comando, settaggio, funzione) viene utilizzata la notazione puntata
$$t.azione(\dots)$$

Esempio 0.3.1 - La seguente porzione di codice disegna di un quadrato rosso di lato di 5 unità:

```
t = Turtle('red')    # oppure t = Turtle()
t.pendown()
t.show()
for _ in range(4):
    t.forward(5)
    t.left(90)
```

Adottando l'impostazione orientata agli oggetti risulta semplice disegnare delle figure composte da parti, ciascuna delle quali viene disegnata da una diversa tartaruga. Se, in qualche modo [1], si muovono contemporaneamente più tartarughe, si ottiene l'effetto visivo del movimento contemporaneo delle varie tartarughe.

Esempio 0.3.2 - In questo esempio viene creato un array di tartarughe che vengono inizialmente orientate verso diverse direzioni, a raggiera; successivamente vengono fatte avanzare contemporaneamente ottenendo l'effetto di una spirale a più rami che si sviluppa dal centro verso l'esterno; l'apparente simultaneità del movimento viene ottenuta avanzando sequenzialmente di poco ciascuna tartaruga

```
n = 17 #numero di tartarughe
t = n*[None]
colors = ['red','green','blue','orange','brown']
for i in range(n):
    t[i] = Turtle()
    t[i].color(colors[i % len(colors)])
    t[i].left(360/n*i)
    t[i].show()
```

[1] Con un ciclo in cui si fanno eseguire brevi spostamenti a tutte le tartarughe oppure associando un *thread* di esecuzione a ciascuna tartaruga.

```
for _ in range(40):
    for i in range(n):
        t[i].forward(.2)
        t[i].left(5)
```

E' possibile far interagire tartarughe diverse.

Esempio 0.3.3 - Nella porzione di programma che segue vengono create due tartarughe: la prima si muove lungo una circonferenza; la seconda si muove orientando ad ogni passo la propria direzione verso la prima, fino a raggiungerla.

```
t1 = Turtle(color='red')
t2 = Turtle(color='blue')
t1.move(Point(-2,1))
t2.move(Point(7,3))
t1.show()
t2.show()
while t1.dist(t2.pos())>.02:
    t1.forward(.1)     # piu' lenta
    t2.forward(.11)    # piu' veloce
    t1.left(3)         # ruota di poco
    t2.look(t1.pos())  # orienta verso t1
```

0.3.5 Estensione della classe Turtle

La classe *Turtle* può essere estesa, aggiungendo ulteriori attributi e nuovi metodi e ridefinendone alcuni di quelli già realizzati. Questa tecnica è tipica della *programmazione ad oggetti* e viene detta *estensione* o *ereditarietà*.

```
class Tarta(Turtle):
    # costruttore
    def __init__(self, col='black'):
        Turtle.__init__(self)
        super().color(col)
        self._passi = 0   # numero di passi percorsi

    # metodo ridefinito
    def forward(self, passi:float):
        super().forward(passi)
        self._passi += passi

    # numero passi - valore dinamico
    def npassi(self):
        return _Value(self.dpos(),lambda:self._passi,'npassi')

    # percorso fatto - valore statico
    def percorso(self):
        return self._passi

    # metodo aggiunto
    def poligono(self, n:int, lato:float):
        for _ in range(n):
            self.forward(lato)
            self.left(360/n)

    # metodo disinnescato
    def color(self, col:str):
        pass

#---- main ----
t = Tarta('red')
t.poligono(7,3)
write('percorso:',t.percorso()) # valore statico
write('npassi:',t.npassi())     # valore dinamico
t.color('blue')   # non ha effetto
k = 1
for _ in range(100):
    t.forward(k)
    t.left(90)
    k += .1
```

0.4 Tipologie di oggetti e loro modalità di generazione

È possibile gestire diverse tipologie di primitive grafiche:

- *Point*: punto
- *Segment*: segmento
- *Ray*: semiretta
- *Line*: retta
- *Circle*: circonferenza

0.4.1 Costruttori degli oggetti grafici

Ogni oggetto grafico viene creato mediante un costruttore che viene attivato interattivamente agendo sulle icone della GUI, oppure mediante l'esecuzione di una specifica istruzione Python avente il nome della classe dell'oggetto (*Point*, *Line*, *Segment*, *Ray*, *Circle*) dell'oggetto che si intende creare e degli argomenti che ne definiscono la geometria e gli attributi grafici. Nel caso in cui il primo argomento sia `INPUT` significa che l'oggetto viene specificato (parzialmente o completamente) interattivamente, agendo sulla GUI; gli attributi grafici (*state*, *width*, *color*) che non vengono specificati esplicitamente, vengono assunti da quelli al momento impostati sulla GUI.

Esempio 0.4.1 - [*costruttore puro*] L'istruzione

```
Line(Point(0,0),Point(1,2),color='red',state=VISIBLE,width=THIN)
```

genera la retta passante per i punti di coordinate $(0,0)$ e $(1,2)$. □

Esempio 0.4.2 - [*costruttore misto*] L'istruzione

```
Segment(INPUT,P,color='blue',state=DRAGABLE,width=THICK)
```

acquisisce il secondo punto di un segmento avente un vertice in P; il segmento assume il colore *rosso*, mentre gli attributi *width* e *state* sono quelli impostati; questi possono essere modificati prima di acquisire in input il secondo punto del segmento. □

Esempio 0.4.3 - [*input puro*] L'istruzione

```
Circle(INPUT)
```

acquisisce in input una circonferenza, con gli attributi correnti. □

0.4.2 Operazioni sugli oggetti grafici

Sulle primitive si possono eseguire diverse tipologie di operazioni:

- accesso alle componenti o proprietà delle primitive:
  - coordinate di un punto
  - estremi di un segmento
  - ...
- operazioni che generano altre primitive vincolate agli argomenti; ad esempio:
  - somma di due punti
  - intersezione fra due rette
  - ...
- entità calcolate mediante espressioni o algoritmi:
  - punto medio (fra due punti)
  - asse di un segmento
  - retta perpendicolare passante per un dato punto
  - bisettrice di un angolo
  - ...

0.4.3 Il valore nullo

Il valore nullo θ rappresenta un valore che è il risultato di un'operazione in un caso estremo, impossibile; esempi:

- intersezione di due rette parallele
- retta individiata da due punti coincidenti
- risultato di un'operazione in cui uno degli argomenti è θ

0.5 Struttura degli oggetti grafici

Ogni oggetto è composto da una sovrapposizione di 3 livelli di informazioni:

1. componente *geometrica*: definisce la *forma* dell'oggetto e ne costituisce l'*essere*
2. componente *grafica*: definisce come l'oggetto *appare* su video
3. componente *dinamica*: definisce come l'oggetto *reagisce* alle sollecitazioni dell'utente

Questi livelli sono specificati mediante degli attributi che definiscono la struttura interna, l'apparenza su video ed il comportemento a seguito di operazioni interattive dell'utente. Questi diversi livelli sono caratterizzati da specifici attributi.

0.5.1 Attributi geometrici

Gli attributi geometrici specificano la forma, la posizione e la struttura degli oggetti grafici e sono

- coordinate geometriche dell'oggetto: dipendono dalla particolare tipologia dell'oggetto: (x, y) per un *Point*, $[(x_1, y_1), (x_2, y_2)]$ per un *Segment* e similmente per le altre categorie di entità

0.5.2 Attributi grafici

Gli attributi grafici definiscono le caratteristiche di visualizzazione. Vengono impostati al momento della creazione dell'oggetto; se non vengono specificati vengono presi quelli di default impostati; possono essere cambiati successivamente alla creazione.

- `color`: colore dell'oggetto; viene specificato mediante una stringa del colore: `'red'`,`'orange'`, `'#FF00A7'`
- `name`: nome dell'oggetto; è un attributo opzionale che viene visualizzato vicino all'oggetto a cui si riferisce, con lo stesso colore dell'oggetto; si sposta contestualmente al movimento dell'oggetto corrispondente
- `width`: spessore della linea: è un numero intero positivo: 1: sottile, 2: media, 3: spessa; al posto dei valori numerici si possono usare le costanti simboliche `THIN`, `MEDIUM`, `THICK`

0.5.3 Attributi di interazione

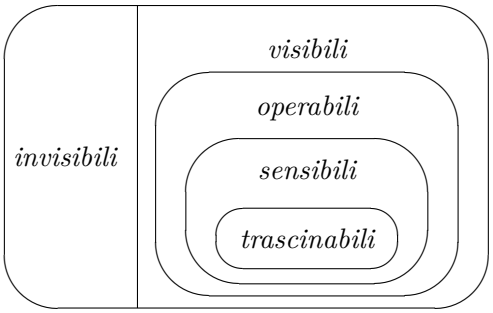
Gli oggetti possono essere manipolati interattivamente; le possibilità dipendono dal loro stato di attivazione specificato dal seguente attributo:

- `state`: stato di attivazione

In ogni istante un oggetto grafico si trova in uno dei seguenti stati di attivazione:

- *invisibile* (`INVISIBLE`=0): oggetto invisibile; quindi non può essere manipolato interattivamente (trascinato, interrogato, modificato nei suoi attributi, ...). Gli oggetti invisibili non sono operabili visivamente (trascinabili, ...) ma possono essere usati mediante programma, per eseguire calcoli. Possono essere elevati di stato.
- *visibile* (`VISIBLE`=1): oggetto visibile su video; non può essere oggetto di operazione, agganciato, trascinato o cambiato di formato grafico; esempio: le rette del grid del piano
- *operabile* (`OPERABLE`=2): oggetto visibile su video; reagisce, cambiando colore, quando viene sovrapposto dal mouse; può essere argomento di operazione (cancellazione, intersezione, cambio di attributo grafico, ...); non può essere agganciato o trascinato
- *sensibile* (`SENSIBLE`=3): reagisce, cambiando colore, quando viene sovrapposto dal mouse; con un clic può essere selezionato oppure "agganciato" se è attiva la modalità di inserimento; non può essere trascinato
- *trascinabile* (`DRAGABLE`=4): oggetti mobili; possono essere oggetto di operazione, agganciati e trascinati agganciando un loro punto

Si può evidenziare il seguente schema insiemistico:



Da questo schema si deducono le seguenti implicazioni:

$$trascinabile \Longrightarrow sensibile \Longrightarrow operabile \Longrightarrow visibile$$

Date queste implicazioni, lo stato di un oggetto può essere specificato mediante un singolo attributo `state` univalore, e sullo stato si possono fare comparazioni di ordine.

Per quanto riguarda la mobilità dei singoli punti si possono avere le seguenti situazioni:

- **punto libero**: può essere mosso liberamente nel piano; viene creato con attibuto di stato `DRAGABLE`
- **punto fisso**: una volta creato non può essere spostato; viene creato con attibuto di stato ≤`DRAGABLE`
- **punto vincolato**: può essere spostato su una linea vincolo
- **punto dipendente**: è in funzione di altri punti; viene generato mediante un'espressione; può essere spostato solo indirettamente muovendo i punti da cui dipende; ad esempio:

```
A = Point(2,3)
B = Point(4,1)
M = (A+B)/2      # punto medio dipendente da A e da B
```

Un oggetto composto, individuato da più punti, può essere modificato e spostato in varie modalità:

- *fisso*: nessuna sua componente può essere spostata
- *modificabile*: se ne possono spostare i singoli punti, cambiandone la forma
- *traslabile*: può essere mosso in modo rigido, senza ruotare
- *ruotabile* se può essere fatto ruotare attorno ad un punto di perno

Componendo i movimenti di base si possono generare altri movimenti:

- ribaltamento (simmetria centrale o assiale)
- ...

Combinando gli attributi di stato delle varie componenti di un'entità grafica si riescono ad ottenere diverse situazioni:

- circolo con centro fisso e raggio variabile: basta creare il centro sensibile (ma non trascinabile) e poi agganciarsi al centro

0.5.4 Modalità di selezione e settaggio degli attributi

In ogni istante sono specificati dei valori correnti degli attributi, visibili sulla GUI; questi attributi. Gli attributi degli oggetti possono essere selezionati e settati con diverse modalità:

1. Il valore degli attributi può essere specificato nel costruttore in fase di creazione dell'oggetto; per default gli oggetti che vengono creati mediante il costruttore hanno l'attributo di stato `INVISIBLE` ed hanno solo le caratteristiche geometriche che servono per i calcoli; per le primitive grafiche che vengono costruite in modo interattivo (argomento `INPUT` del costruttore) i valori degli attributi non specificati nel costruttore vengono assunti da quelli correnti impostati nella finestra grafica dell'applicazione;
2. È possibile acquisire i valori attuali correntemente impostati sulla GUI mediante le funzioni *getcolor*(), *getwidth*(), *getstate*() che ritornano il valore dell'attributo e settare sulla GUI dei valori mediante i corrispondenti comandi *setcolor*(c), *setwidth*(w), *setstate*(s). Questi funzioni e comandi possono essere invocati su singole istanze di oggetti grafici mediante la tradizionale notazione puntata; ad esempio *g.getcolor*() ritorna il colore dell'oggetto grafico g.
3. E' possibile modificare uno o più attributi di un oggetto mediante il metodo d'istanza *config*; ad esempio *g.config*(*color*='red',*state*=`DRAGABLE`) setta il colore rosso ed il valore di stato `DRAGABLE` sull'oggetto g. Il metodo *config* ha il parametro *expand* che può avere uno dei seguenti due valori:
   - `False`: (valore di default) viene configurata solo la componente lineare e non eventuali punti base (estremi del segmento, ...)
   - `True`: vengono configurati anche i punti base; in automatico, se si setta l'attributo *state*=`INVISIBLE`, viene attivata la modalità *expand*=`True`
4. E' possibile acquisire e settare i valori degli attributi degli oggetti grafici mediante le icone *getformat* e *setformat*, agendo in modalità interattiva sulla GUI.

Esempio 0.5.1 - La seguente porzione di codice illustra alcune possibilità sulla gestione degli attributi grafici:

```
A = Point((2,3),name='A',state=DRAGABLE)
B = Point((3,1),name='B',color='#EEBB00',state=DRAGABLE)
A.config(name='A1',color='red',state=VISIBLE)
C = Point(INPUT,name='C',msg='punto C?',state=DRAGABLE)
D = Point(INPUT,name='D',msg='punto D?',color='red',state=DRAGABLE)
C.config(color=B.color(),state=A.state(),name=C.name()+'1')
```

0.6 Modi operativi

Durante l'interazione dell'utente mediante mouse in ogni istante è attivo uno dei seguenti modi operativi, ciascuno caratterizzato da una specifica forma del cursore:

- modo *generale*: appare un cursore a forma di freccia; si possono avere diversi effetti a seconda di come si agisce:
  - *spostare elemento*: avvicinandosi ad un elemento grafico trascinabile esso si evidenzia ed è possibile trascinarlo; se l'elemento è vincolato esso si muove sul vincolo; se l'elemento appartiene ad un gruppo, viene spostato l'intero gruppo di elementi
  - *spostare foglio*: puntanto il mouse su un punto vuoto con il tasto sinistro e trascinando si ottiene l'effetto di traslare il foglio
  - *zoomare*: è possibile zoomare ruotando in avanti o indietro la rotellina del mouse; lo zoom avviene attorno al punto corrispondente al cursore del mouse
  - *enquiry*: con il tasto destro si apre un menu contestuale per interrogare o modificare un elemento
- modo *disegno*: appare un cursore a croce; si può inserire una nuova primitiva grafica

In ogni momento, indipendentemente dal modo operativo corrente, è possibile scrivere un comando sulla linea di comando.

0.7 Modalità di input degli oggetti

L'input avviene selezionando singoli punti con clic/drag del mouse. In ogni momento è attivo uno dei seguenti modi operativi per l'inserimento delle primitive:

- modo *libero* (*FREE*): i punti vengono presi sul punto dove si fa clic/drag col mouse; l'input di ciascun punto è libero e non si risente della presenza degli altri elementi presenti
- modo *aggancio* (*JOIN*): avvicinandosi il cursore del mouse ad un elemento grafico sensibile, esso si evidenzia e:
  - se si è vicini ad un punto: il punto viene condiviso, diventando un riferimento condiviso ad un punto già presente;
  - se si è vicini ad una linea: il punto selezionato diventa un punto vincolato sulla linea

E' possibile forzare l'input mediante i costruttori delle primitive (vedi `INPUT` ...).

Esempio 0.7.1 - Con l'assegnazione

```
P = Point(INPUT,color='red',state=DRAGABLE)
```

vengono settati sulla GUI i due valori 'red' e DRAGABLE degli attributi di colore e stato, mentre con

```
P = Point(INPUT).config(color='red',state=DRAGABLE)
```

viene eseguito l'input con i valori correnti degli attributi e poi avviene il settaggio dei nuovi valori sul singolo oggetto acquisito, senza modificare sulla GUI i valori correnti degli attributi.

0.8 Modalità di output degli oggetti

Gli oggetti grafici creati, se hanno l'attributo di stato `VISIBLE` (o superiore), vengono visualizzati direttamente sulla finestra grafica. Sulla finestra testo, posta sulla parte sinistra della finestra grafica, è possibile visualizzare, in formato testo, delle stringhe, dei valori e degli oggetti grafici; questi ultimi vengono convertiti in formato testo. Mediante il comando

```
write(s)
```

viene visualizzata la stringa *s*. Ad esempio: `write('Punto P='+str(P))`. Similmente a *write* si può usare il comando

```
show(s,obj)
```

che visualizza la stringa *s* seguita dalla conversione in stringa dell'oggetto *obj*, con la differenza che, rispetto a *write*, eventuali modifiche all'oggetto vengono direttamente riportate sulla corrispondente stringa sulla finestra testo.

0.9 Assegnazioni, alias e costruttori per copia

Se x è un (riferimento a) un oggetto ($Point$, ...), con l'assegnazione

```
y = x
```

y diventa un riferimento (alias) all'oggetto referenziato da x. Una modifica a y influenza l'oggetto (referenziato da) x.

La funzione $clone(x)$ crea una copia di un generico oggetto x; l'oggetto creato viene solitamente referenziato mediante un identificatore utilizzando la tradizionale istruzione di assegnazione

```
y = clone(x)
```

y diventa così un riferimento ad un oggetto uguale a x ma distinto da esso; un'eventuale modifica a y non interferisce con l'oggetto x.

Un'alternativa all'uso della funzione *clone* consiste nell'uso del *costruttore per copia*: se x è un oggetto di classe C ($Point$,...), $C(x)$ è il *costruttore per copia* che crea un oggetto uguale a x ma distinto da x e con l'assegnazione

```
y = C(x)
```

l'oggetto creato viene referenziato da y.

Esempio 0.9.1 - ...

```
P = Point((2,3),color='black',state=VISIBLE)
s = Segment(P,Point(INPUT,color='red',state=DRAGABLE),name=...)
t = s
t.config(color='blue',name='t',state=DRAGABLE)
t.config(name='newt')
w = Segment(t,color='black',state=DRAGABLE)     #costruttore per copia
w.config(name='W',color='brown',state=DRAGABLE) #espandere il config ai figli
```

Esempio 0.9.2 - ...

```
#---- test con le assegnazioni e clone dei punti ---- [dal file test_point.py]
A = Point(INPUT,name='A',msg='input punto...',color='red',state=DRAGABLE)
B = Point(INPUT,name='B',msg='input punto...',color='green',state=DRAGABLE)
write('dist1(A,B)=',dist(A,B))
T = A   #T e' un alias di A
T.config(name='T',color='orange')
write('dist2(T,B)=',dist(T,B))

#B = A        #B e' un alias di A
#B = Point(A) #B e' un clone autonomo da A
B = Point(T)  #B e' clone autonomo (il vecchio B rimane zombi attivo)

B.config(state=DRAGABLE,name='B',color='blue')
write('dist3(A,B)=',dist(A,B))
```

0.10 Point

I punti sono gli elementi fondamentali dell'ambiente *PyAlGeo*: costituiscono gli elementi grafici, possono essere condivisi fra le primitive e possono essere trascinati e spostati in modalità interattiva.

I punti sono utilizzabili mediante le operazioni elencate a seguire; nella descrizione vengono omessi i parametri opzionali relativi alla specifica degli attributi grafici; se non vengono specificati vengono assunti quelli correnti.

- *costanti*: sono definiti i seguenti identificatori di punti costanti:

  `O`: punto *origine* di coordinate $(0, 0)$
  `U`: punto *unità* di coordinate $(1, 0)$
  `I`: punto *indice* di coordinate $(0, 1)$

- *costruttori* : punti del piano costruibili mediante i seguenti costruttori:

  `Point(P)`: copia del punto P (solo attributi geometrici (e non grafici)
  `clone(P)`: copia integrale del punto P (attributi geometrici e grafici)
  `Point((x,y),name=...,color=...,state=...)`: costruttore di un punto di coordinate (x, y), di specificato *nome*, *colore* e *stato*
  `Point(INPUT,msg=...,on=...)`: acquisizione interattiva con mouse; *msg* (opzionale) specifica il messaggio guida; *on* (opzionale) specifica l'elemento grafico su cui è vincolato il punto
  `Point(RANDOM,on=...)`: generazione casuale di un punto; *on* denota l'eventuale elemento grafico su cui è vincolato il punto; se non viene indicato l'argomento *on* (oppure *on* = *None*) viene selezionato un punto casuale nella finestra correntemente visibile; se viene indicato *on* = (x_1, y_1, x_2, y_2) il punto casuale viene scelto all'interno del rettangolo avente (x_1, y_1) e (x_2, y_2) come vertice in basso a sinistra e vertice in alto a destra
  `Point(PARAM,on=..,param=...)`: punto parametrico sull'elemento grafico *on* con parametro *param* compreso nell'intervallo $[0, 2]$

- *confronti*: un punto può essere confrontato con un altro mediante con i seguenti operatori, coerenti con la semantica di Python:

  `A == B`: A e B hanno uguali coordinate geometriche
  `A != B`: A e B hanno coordinate geometriche diverse
  `A is B`: A e B sono riferimenti allo stesso punto
  `not A is B`: A e B sono riferimenti a punti distinti

- *operazioni*: un punto può essere generato mediante un'espressione costituita dalle seguenti operazioni elementari:

  `hom(P,Q,`λ`)`: omotetico di Q rispetto a P di rapporto λ
  `rot(P,Q,`α`)`: ruotato di Q rispetto a P di angolo α
  `med(P,Q)`: punto medio fra P e Q
  `sym(P,Q)`: simmetrico del punto P rispetto a Q
  `sym(P,Q,R)`: simmetrico del punto P rispetto alla retta QR
  `ali(P,Q,R)`: i punti P, Q, R sono allineati
  `par(P,Q,R,S)`: la retta PQ è parallela alla retta RS
  `P + Q`: somma vettoriale fra punti
  `P - Q`: differenza vettoriale fra punti
  `k * P`: moltiplicazione per uno scalare (anche a destra)
  `-P`: opposto del punto (rispetto all'origine)

- *funzioni*: danno come risultato un valore dinamico di classe *Value*:

  `coordx(P)`, `coordy(P)` : coordinate cartesiane del punto P
  `coords(P)` : coppia delle coordinate cartesiane del punto P
  `dist(P,Q)`: distanza fra i due punti P e Q
  `ang(A)`: misura dell'angolo angolo della retta OA
  `ang(A,B)`: misura dell'angolo angolo della retta AB
  `ang(A,V,B)`: misura dell'angolo angolo $A\widehat{V}B$

0.11 Segment, Ray e Line

Segmenti, semirette e rette sono entità strutturalmente ed operativamente equivalenti; si differenziano solo nel modo in cui vengono visualizzate e da come gestiscono i punti vincolati su di esse.

In DyGeo le rette sono rappresentate dagli oggetti della classe *Line*.

- *costruttori* : le rette del piano costruibili mediante i seguenti costruttori:

  `Line(P,Q)`: retta passante per due punti

  `Line(P,ang)`: retta per P e formante un angolo ang (in senso antiorario) con l'asse x

  `Line(P,(dx,dy))`: retta per P e per il punto $Q = P+(dx, dy)$; la coppia (dx, dy) definisce l'inclinazione della retta

  Una retta può essere acquisita interattivamente con il mouse; si possono così avere delle situazioni come le seguenti:

  `Line(INPUT)`: acquisizione interattiva con mouse, selezionando due punti sulla retta

  `Line(INPUT,node=P`: acquisizione interattiva con mouse di una retta del fascio proprio di centro P selezionando il secondo punto della retta

  `Line(P,Point(INPUT,on=c))`: acquisizione interattiva con mouse della retta per P e secondo punto selezionato sulla circonferenza c

- *operazioni*:

  `inters(r,s)` : coppia delle coordinate cartesiane del punto di intersezione fra due linee (segmenti, semirette, rette, corconferenze

  `slope(r)` : coefficiente angolare della retta r dato dalla coppia (dx, dy) delle coordinate del punto P della circonferenza unitaria tale che la retta OP sia parallela ad r

- *esempi*:

```
s = Line(T,S).config(color='gray')
C = inters(r,s).config(color='red')
```

0.12 Circle

…

0.13 Nomi, testi, label e scritte

In PyAlGeo vengono gestite diverse classi di stringhe: *nomi* degli oggetti, *testi* autonomi, *label* degli oggetti.

Nomi

Ogni oggetto grafico che viene creato può avere un nome (attributo `name`); il nome deve essere una stringa; se non viene specificato viene assunta la stringa vuota come nome. Dal nome (se specificato) viene generata automaticamente una label ancorata all'oggetto (nel caso l'oggetto non sia un punto, viene generato automaticamente un punto ancora a cui agganciare la label); la label generata viene mossa contestualmente all'oggetto e mantiene la vicinanza all'oggetto in caso di operazioni di zoom. Esempio:

```
A = Point(INPUT,name='A'color='green',state=DRAGABLE)
B = Point(INPUT,color='blue',state=DRAGABLE)
C = (A+B).config(state=DRAGABLE,color='red',name='A+B')
```

Testi

Un testo è un oggetto grafico autonomo di tipo stringa posizionato in un punto del piano. Un volta creato, il testo si stacca dal punto ed ha una vita separata (in drag). Un testo può essere dinamico, avendo dei parametri `{0}`, `{1}`, ... che vengono istanziati dai valori specificati nel successivo parametro `variables`. Esempio:

```
Text((5,6),"Questo e' un testo",color='blue',state=VISIBLE)
```

Label

Una label è un testo che viene agganciato ad un oggetto e rimane solidate con esso. Una label può essere dinamica (similmente ai testi dinamici). In zoom una label rimane alla stessa distanza (in pixel) dall'oggetto al quale è agganciata. Una label è dragabile indipendentemente dall'oggetto a cui è ancorata. Creare label su punto equivale (quasi) a creare punto con nome (a parte i colori). Un esempio:

```
Label(Point(2,2),"Testo della label",color='brown',state=DRAGABLE)
```

Scritte

Una scritta (*Writing*) è un oggetto analogo ad un *Text* con la differenza che rimane fisso sulla finestra dell'applicazione e non risente delle operazioni di trascinamento e zoom; le coordinate di posizionamento sono espresse in pixel e vengono riferite all'angolo alto a sinistra della finestra di disegno (angolo di coordinate $(0, 0)$). Un esempio:

```
Writing((80,30),"Testo della scritta",color='brown',state=DRAGABLE)
```

0.14 Valori ed espressioni dinamiche

In un ambiente di geometria dinamica è necessario disporre di *espressioni dinamiche* legate a delle quantità che dipendono da elementi geometrici (distanza, lunghezza, ...) che variano dinamicamente. Ciò deve valere sia per i numeri che per gli oggetti grafici. Di conseguenza gli attributi geometrici sono dinamici.

In PyAlGeoi punti e le altre antità grafiche sono modificabili dinamicamente; di conseguenza vengono modificati anche i valori numerici e logici che dipendono dalle entità grafiche. Un *Value* è un valore che dipende da altre entità grafiche; una modifica alle entità grafiche aggiorna automaticamente i valori che dipendono da esse.

Esempio 0.14.1 - Sono oggetti di classe *Value* i seguenti: le coordinate dei punti, la distanza fra due punti.

Esempio 0.14.2 - Un cerchio avente come raggio la distanza fra due punti, si automodifica se vengono spostati i punti che definiscono la distanza.

Se v è un valore dinamico, con $val(v)$ si ottiene il valore attuale (statico) di v.

Un *DynExp* è un'espressione (denotabile anche mediante un identificaore) il cui valore è legato ad un'espressione i cui argomenti sono funzione degli elementi geometrici. Se vengono spostati, l'espressione modifica contestualmente il suo valore e tutto viene riportato sul video. DynExp può essere anche una primitiva grafica (ad esempio il punto medio fra due dati punti).

Il tipo di un'espressione dinamica può essere:

- un oggetto grafico; ad esempio: `1.4*P`
- un valore numerico; ad esempio: `dist(P,Q)`

Un'espressione coinvolgente almeno un argomento modificabile (ad esempio un punto spostabile) è un'espressione dinamica. Un'espressione dinamica prevede un metodo *eval* che ritorna il valore dell'espressione in funzione del valore attuale degli argomenti coinvolti nell'espressione. Ogni modifica ad un argomento modificabile causa un'immediata rivalutazione di tutte le espressioni dinamiche che dipendono dal dato argomento, e, nel caso queste espressioni dinamiche siano visibili, il valore attuale viene visualizzato.

0.15 Valori e punti condizionali

I *valori condizionali* sono oggetti della forma

$$CondValue(c, x, y)$$

dove c è una espressione logica dinamica ed x ed y sono delle generiche espressioni (anche dinamiche).

Usando i valori condizionali si possono costruire degli *oggetti condizionali.* (cfr. *CondPoint* ...)

Indice