LUIGINO CALVI

ALGMATH

UN AMBIENTE PER LA SPERIMENTAZIONE DI ALGORITMI

2025

Indice

1

ALGMATH

AlgMath (ALGorithmic MATHematics) è un ambiente per lo sviluppo e l'analisi di algoritmi di carattere matematico, di tipo numerico e grafico. AlgMath permette di scrivere programmi ed eseguirli, ottenendo il risultato su apposite finestre (testuali e grafiche).

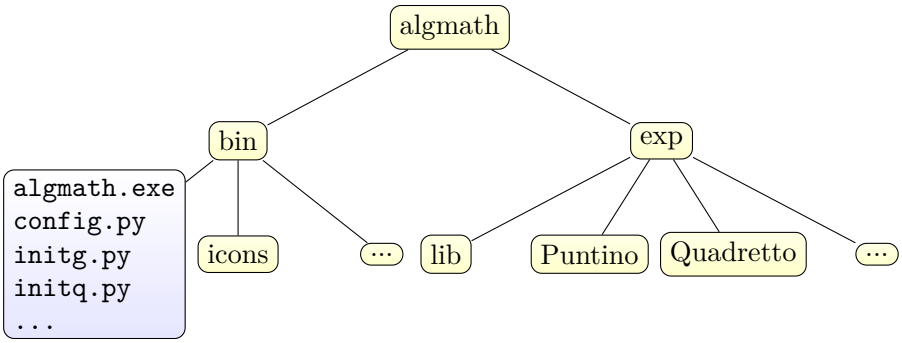
AlgMath si conforma alla metafora del laboratorio virtuale che può assumere diverse configurazioni ciascuna delle quali può essere pensata come una *stanza* nella quale svolgere degli *esperimenti*; in ciascuna stanza del laboratorio è utilizzabile uno specifico *automa* comandabile mediante un corrispondente *linguaggio*; in ciascuna stanza c'è un *tavolo* di lavoro suddiviso in due parti: in una (a sinistra) lavora il solutore che conduce l'esperimento, scrivendo il programma mediante il quale comanda un automa che agisce nella parte destra.

Nota. Questa è una versione provvisoria ed incompleta del manuale; contiene sicuramente errori e ci sono delle funzionalità descritte possono non essere allineate con quanto effettivamente realizzato nell'applicazione AlgMath.

1.1 Installazione di ALGMATH

Per installare AlgMath si può procedere come segue:

1. scaricare da `github.com/algmath/lab` il file `algmath.zip`
2. decomprimere il file `algmath.zip` in una cartella; si ottiene la seguente struttura di cartelle:



3. copiare un collegamento sul desktop al file `algmath.exe` presente nella sotto cartella `bin`

La cartella `bin` contiene il file `algmath.exe` che è il programma che attiva la finestra principale dell'applicazione ed altri file. Ci sono poi diverse altre sottocartelle contenenti diversi file necessari all'esecuzione dell'applicazione AlgMath, in particolare alcune cartelle per l'esecuzione di programmi in linguaggio Python.

La cartella `exp` contiene gli *esperimenti* che vengono eseguiti nelle varie *stanze* del *laboratorio* AlgMath; al di là della metafora, un esperimento corrisponde più prosaicamente ad un file sorgente scritto in un linguaggio di programmazione. La figura sopra propone una ripartizione ad albero della cartella `exp`: il primo sottolivello corrisponde alle stanze del laboratorio, ciascuna caratterizzata dal nome dell'automa utilizzato in quella stanza; al successivo livello sono riportati gli argomenti degli esperimenti, ciascuno caratterizzato dal nome di un capitolo. La cartella `lib` contiene moduli (in linguaggio Python e `LFR`) che vengono importati nei programmi scritti dall'utente e moduli scritti dall'utente stesso.

1.2 Configurazione iniziale di ALGMATH

Il file `config.py` contiene alcuni comandi (in linguaggio Python) di configurazione iniziale della finestra principale (*GUI*, *G*raphical *U*ser *I*nterface) dell'ambiente di AlgMath (dimensioni, posizione, ...) descritti nella lista che segue:

- `setconfig(`*room*`)`: setta la configurazione iniziale, all'avvio dell'applicazione AlgMath
- `setguipos(`*xpos, ypos*`)`: setta le coordinate (in pixel) dell'angolo in alto a sinistra della GUI
- `seteditor(`*nrows, ncols*`)`: setta il numero di righe e di colonne della finestra dell'editor (posta a sinistra)
- `setviewport(`*width, height*`)`: setta le dimensioni (in pixel) della finestra grafica (posta a destra)
- `setwindow(`*npixelunit*`)`: setta il numero di pixel per unità logica del piano cartesiano della finestra grafica [per le configurazioni *Puntino*, *PyTurtle* e *PyPen*]
- `setlibpath(`*path*`)`: setta il percorso delle librerie, da dove importare i moduli e le classi
- `setboard(`*nrows, ncols, size*`)`: numero di righe e di colonne della scacchiera ed ampiezza (in pixel) di ciascuna casella [per le configurazioni *Quadretto* e *PyQuad*]
- `setangle(a)`: imposta l'angolo a di avanzamento della t. [per le configurazioni *Puntino* e *PyTurtle*]
- `setcolor(c)`: imposta il colore c di disegno delle linee [per le configurazioni *Puntino*, *PyTurtle* e *PyPen*]
- `setwidth(s)`: imposta lo spessore delle linee (valori: $1, 2, 3$) [per le configurazioni *Puntino*, *PyTurtle* e *PyPen*]

Con `seteditor` e `setviewport` rimangono indirettamente definite anche le dimensioni della finestra grafica dell'intera applicazione.

1.3 Le configurazioni

AlgMath può essere impostato secondo diverse configurazioni, ciascuna orientata allo sviluppo ed esecuzione di algoritmi di una specifica categoria. Per ciascuna configurazione è predisposto:

- un linguaggio di programmazione
- un editor orientato al linguaggio
- un insieme di funzioni di libreria
- un ambiente grafico interattivo di esecuzione

Ciascuna configurazione è selezionabile alle voci del menu *Room* della finestra principale di AlgMath, oppure mediante il comando `setconfig(`*a*`)` scritto nel file di configurazione `config.py` che precisa l'esecutore *a* che viene utilizzato. Il parametro *a* può essere uno dei seguenti (fra parentesi è riportato il linguaggio per comandare l'automa):

- `Puntino`: robot elementare di movimento sul piano (`LFR`)
- `Quadretto`: robot elementare su scacchiera (`LFR`)
- `PyQuad`: robot elementare su scacchiera (*Python*)
- `PyTurtle`: grafica della tartaruga (*Python*)
- `PyPen`: ambiente di geometria dinamica (*Python*)
- `PyTerm`: esecutore di programmi Python testuali (*Python*)
- `TMachine`: simulatore della macchina di Turing (*TM*)

La configurazione può essere attivata automaticamente al momento del caricamento del programma nella finestra dell'editor oppure al momento dell'esecuzione del programma dalla finestra dell'editor, scrivendo `<`*configurazione*`>` nel programma stesso (ad esempio scrivendo la stringa `<Puntino>`).

1.4 L'interfaccia di ALGMATH

La figura che segue descrive l'interfaccia grafica del programma AlgMath:

Il menu principale di questa finestra presenta le seguenti voci:

- *Room*: permette di scegliere la stanza in cui lavorare
- *Table*: permette di scegliere le componenti (finestra di editing ed esecuzione) da attivare
- *Help*: la sottovoce *About* visualizza la versione del programma mentre la sottovoce *Manual* apre il manuale d'uso del programma

La finestra dell'applicazione è suddivisa in due sottofinestre, corrispondentemente alla tradizionale suddivisione dei ruoli fra solutore ed esecutore: nella sottofinestra alla sinistra il solutore scrive il programma che regola il funzionamento dell'automa mentre nella sottofinestra alla destra si vede l'effetto dell'esecuzione del programma da parte dell'automa.

La sottofinestra di editing

La sottofinestra di editing, posizionata sulla sinistra, permette di scrivere programmi mediante un editor di testo orientato al linguaggio; vengono gestiti i seguenti linguaggi: `LFR`, *Python* o *TM* (linguaggio per le macchine di Turing). In questa sottofinestra sono presenti le seguenti icone:

 (*New*) elimina il contenuto corrente della finestra

 (*Open*) apre un file (programma) selezionato dall'utente

 (*Edit*) propone le tradizionali funzioni di editing: *Copia*, *Taglia* ed *Incolla*

 (*Save*) salva su file il programma corrente

 (*Save as...*) salva il programma corrente richiedendo il nome del file su cui salvare

 (*Run*) esegue il programma corrente

La sottofinestra di esecuzione

Nella sottofinestra di esecuzione, posizionata sulla destra, si vede il risultato dell'esecuzione. La sottofinestra di esecuzione è costituita da:

- una toolbar di icone; alcune specifiche voci di queste icone permettono di selezionare e mandare in esecuzione dei programmi (senza passare dalla finestra dell'editor)
- una finestra grafica o testuale per la lettura interattiva dei dati e la visualizzazione dei risultati dell'esecuzione

La sottofinestra di esecuzione è diversificata a seconda della configurazione impostata; è descritta, per i diversi casi, nei capitoli che seguono.

La linea di comando e di comunicazione

In varie configurazioni (*Puntino*, *Quadretto*, *PyQuad*, *PyTurtle*, *PyPen*), alla base della finestra di esecuzione, è presente una linea testuale di interazione che serve per inviare comandi all'automa, visualizzare dei messaggi ed errori prodotti dalla compilazione/esecuzione dei programmi e per la comunicazione dei dati di input/output.

Esecuzione dei programmi

L'esecuzione dei programmi che movimentano i vari automi di AlgMath può avvenire in tre diverse modalità:

- eseguendo il programma della finestra dell'editor mediante l'icona 
- in alcune configurazioni è possibile selezionare ed eseguire un programma dalla finestra di esecuzione mediante la voce *Run* dell'icona corrispondente all'automa della configurazione corrente
- in alcune configurazioni è possibile eseguire songoli comandi dalla linea di comando

1.5 La programmazione in linguaggio Python

In alcune configurazioni di AlgMath è abilitata la programmazione con linguaggio Python. A seconda della configurazione attivata è possibile utilizzare un corrispondente insieme di funzioni di libreria (descritte a seguire nei capitoli corrispondenti a ciascuna configurazione). In tutte le configurazioni l'interazione testuale con l'utente viene realizzata mediante le seguenti due funzioni che estendono le omonime funzioni del linguaggio Python aggiungendo alcuni argomenti opzionali.

- `input`(*msg*, *init*, *check*): legge da tastiera una stringa; ritorna la stringa letta:
  - *msg* : messaggio guida per l'input
  - *init* : valore iniziale proposto per l'input
  - *check*(*str*) : funzione utente di controllo: se la stringa *str* è accettata ritorna `True`, altrimenti ritorna una stringa che costituisce il messaggio di avvertimento per l'utente; in questo caso viene richiesto un nuovo valore
- `print`(**args*, *end*, *tag*, *time*): visualizza la lista di argomenti *args*:
  - *args* : lista di argomenti che vengono convertiti in stringa e visualizzati
  - *end* : (opzionale; default: *end=”*) stringa di fine linea; se *end*='`\n`' si va a nuova linea
  - *tag* : stringa del colore di visualizzazione; sono possibili le seguenti scelte: '`message`', '`warning`', '`error`'
  - *tim* : tempo di attesa (in sec) al termine della visualizzazione; valori ammessi:
    - *tim* = −1 : si attende un clic per continuare
    - *tim* = 0 : si continua subito
    - *tim* > 0 : numero di secodi di attesa

2

PUNTINO

PUNTINO è un ambiente caratterizzato da un automa, detto *Puntino* (nel seguito semplicemente P), che si muove sui vertici del reticolo di coordinate intere di un piano quadrettato lasciando una linea di traccia al suo passaggio, disegnando in tal modo delle figure secondo il tradizionale approccio della *geometria della tartaruga*. P è comandabile mediante un semplice linguaggio che permette di descrivere dei programmi in cui i pochi comandi di movimento elementari sono organizzati da semplici strutture di controllo, come descritto nel cap. *Muovere*.

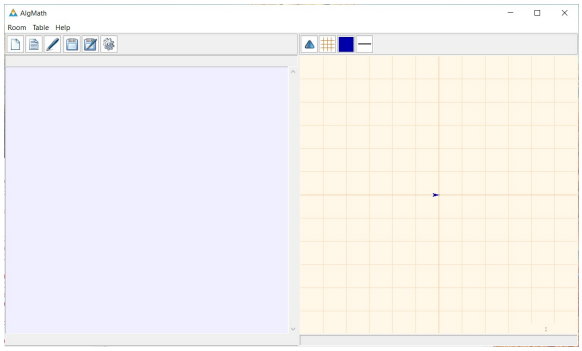
2.1 Attivazione dell'ambiente PUNTINO

L'ambiente PUNTINO viene attivato all'interno di ALGMATH selezionando, alla voce di menu *Room*, l'opzione *Puntino*. È possibile fare questa scelta scrivendo nel file di configurazione `config.py` l'istruzione `setconfig('Puntino')`. Al momento dell'esecuzione viene eseguito il file `initg.py`; tale file contiene alcune istruzioni (in linguaggio Python) per l'inizializzazione dell'ambiente (caricamento di librerie, posizionamento iniziale di P, ...).

Una volta attivato l'ambiente PUNTINO si può scrivere il programma in linguaggio `LFR` nella finestra dell'editor e poi mandarlo in esecuzione con il bottone *Run* presente nella finestra dell'editor. Si possono anche eseguire singole espressioni `LFR` dalla linea di comando posta alla base della finestra del piano di movimento di P. In alternativa si può scrivere dapprima il programma in linguaggio `LFR` in un file sorgente (con estensione `.lfr`) mediante un qualsiasi editor di testo e successivamente eseguirlo mediante l'icona *Puntino* della toolbar, selezionando la voce *Run...* mediante la quale viene richiesto di selezionare il programma da eseguire.

2.2 Interfaccia di PUNTINO

Dopo aver attivato l'ambiente PUNTINO si presenta la seguente finestra:



La finestra è suddivisa in due sottofinestre: nella finestra di editor di sinistra viene scritto, in linguaggio `LFR`, il programma da eseguire mediante l'icona *Run*; nella sottofinestra di destra si vede l'effetto dell'esecuzione del programma, ossia il movimento di P. Lo stato (posizione e verso di avanzamento) di P viene visualizzato mediante un triangolino. Eventuali errori (di sintassi) vengono segnalati nella linea di comunicazione posta a destra, in fondo alla sottofinestra della scacchiera.

Nella sottofinestra di destra è presente una toolbar di due icone dove sono predisposte le seguenti funzionalità:

 (*Puntino*) vengono presentate le seguenti sottovoci:

- *Home*: riporta P all'origine degli assi e rivolto verso Nord
- *Visible*: rende visibile il triangolino di stato di P
- *Stop*: interrompe l'esecuzione del programma e ferma P
- *Run...*: seleziona ed esegue un programma
- *Rerun*: esegue l'ultimo programma eseguito

 (*Plane*) vengono presentate le seguenti sottovoci:

- *Grid*: attiva/disattiva il reticolo quadrettato del piano
- *Clear*: elimina quanto disegnato sul piano

 (*Color*) apre una finestra di selezione del colore di disegno

 (*Width*) apre una finestra di selezione dello spessore della penna di disegno

In fondo a questa sottofinestra è presente una *linea di comando* dove è possibile scrivere singoli comandi in linguaggio `LFR`; il tasto ⏎ manda in esecuzione il comando scritto sulla linea. La linea di comando viene attivata mediante un clic del mouse sulla linea stessa e si esce con il tasto esc.

2.3 Programmazione in linguaggio LFR

Puntino può essere comandato mediante un programma di movimento scritto nella finestra dell'editor oppure nella linea di comando posta sotto la finestra grafica.

I comandi elementari per muovere Puntino sono i seguenti:

`F` : avanzamento di un passo unitario (`F`*orward*)

`B` : indietreggiamento di un passo unitario (`B`*ackward*)

`L` : rotazione a sinistra di un angolo retto (`L`*eft*)

`R` : rotazione a destra di un angolo retto (`R`*ight*)

`J` : salto in avanti di un passo unitario (`J`*ump*)

I comandi elementari descritti nella precedente lista possono essere composti mediante i seguenti schemi strutturali per generare delle espressioni:

$n\ \alpha$: ripete n volte il programma α

`[` $\alpha_1\,\alpha_2\,\ldots\,\alpha_n$ `]` : raggruppa la sequenza di elementi interni alle parentesi quadre

`-`α : inversione del programma α

Viene così definito un vero e proprio linguaggio (denominato *linguaggio* `LFR`) per la programmazione di P. Altre caratteristiche del linguaggio `LFR` sono descritte nel cap. *Muovere*.

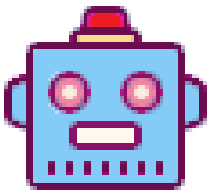
Esempio 2.3.1 - L'esecuzione dell'espressione `4[2FR]` ha l'effetto far tracciare a Puntino un percorso quadrato avente lato di due unità. □

Per eliminare il disegno tracciato e riportare Puntino alla posizione iniziale si può usare il comando

`I` : elimina il disegno tracciato e riporta P alla posizione iniziale (`I`*nit*)

3

QUADRETTO



QUADRETTO è un elementare ambiente di *robotica educativa* caratterizzato da un automa virtuale, chiamato *Quadretto* (nel seguito semplicemente Q), che si muove su una scacchiera quadrata visualizzata sul video. In ogni istante Q occupa esattamente una casella della scacchiera, ha un verso di avanzamento rispetto al quale può avanzare alla casella successiva e ruotare ad angolo retto, a sinistra o a destra, rimanendo sulla stessa casella. Q può essere movimentato mediante dei comandi diretti attivabili mediante i tasti freccia della tastiera oppure mediante analoghi comandi del linguaggio `LFR` (come descritto nel cap. *Interagire*).

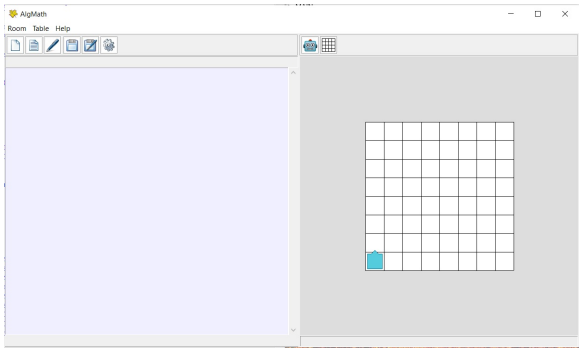
3.1 Attivazione dell'ambiente QUADRETTO

L'ambiente QUADRETTO viene attivato all'interno di ALGMATH selezionando, alla voce di menu *Room*, l'opzione *Quadretto*. È possibile fare questa scelta inizialmente scrivendo nel file di configurazione `config.py` l'istruzione `setconfig('Quadretto')`. Al momento dell'esecuzione viene eseguito il file `initq.py`; tale file contiene alcune istruzioni (in linguaggio Python) per l'inizializzazione dell'ambiente (caricamento di librerie, posizionamento iniziale di Q, caricamento di configurazioni della scacchiera, ...).

Una volta attivato l'ambiente QUADRETTO si può scrivere il programma `LFR` nella finestra dell'editor e poi mandarlo in esecuzione con il bottone *Run* presente nella stessa finestra. Si possono anche eseguire singole istruzioni `LFR` dalla linea di comando posta alla base della finestra di esecuzione, sotto alla scacchiera di movimento. In alternativa si può scrivere dapprima il programma `LFR` (con estensione `.lfr`) in un file sorgente mediante un qualsiasi editor di testo e successivamente eseguirlo mediante l'icona *Quadretto* della toolbar, selezionando la voce *Run...* mediante la quale viene richiesto di selezionare il programma da eseguire.

3.2 Interfaccia di QUADRETTO

All'attivazione dell'ambiente QUADRETTO si presenta la seguente finestra:



La finestra è suddivisa in due sottofinestre: sulla finestra di editor di sinistra viene scritto, in linguaggio `LFR`, il programma da eseguire mediante l'icona *Run*; eventuali errori (di sintassi) vengono segnalati nella linea di comunicazione posta a destra, in fondo alla sottofinestra della scacchiera. Nella sottofinestra di destra si vede il movimento di Q.

Nella sottofinestra di destra è presente una toolbar di due icone dove sono predisposte le seguenti funzionalità:

 (*Quadretto*) vengono presentate le seguenti sottovoci:

- *Home*: riporta Q alla casella $(1, 1)$, rivolto a Nord
- *Trace*: attiva la colorazione in nero delle caselle di passaggio
- *Speed*: permette di scegliere la velocità di avanzamento di Q
- *Start*: fa partire Q (si ferma quando incontra un ostacolo)
- *Step*: esegue un passo di avanzamento di Q
- *Stop*: interrompe l'esecuzione del programma e ferma Q
- *Run...*: seleziona ed esegue un programma
- *Rerun*: esegue l'ultimo programma eseguito

 (*Board*) vengono presentate le seguenti sottovoci:

- *Init*: riporta la scacchiera allo stato iniziale
- *Empty*: elimina gli oggetti sulla scacchiera (blocchi e muri)
- *Clear*: elimina scritte e colori dalle caselle
- *Load*: carica un file di configurazione della scacchiera

La parte centrale della finestra di esecuzione visualizza la scacchiera in cui si muove Q. Agendo su questa sottofinestra è possibile movimentare Q mediante i 4 tasti freccia della tastiera:

[↑] : avanzamento di una casella

[↓] : indietreggiamento di una casella

[←] : rotazione a sinistra

[→] : rotazione a destra

Per attivare questa modalità di movimento manuale è sufficiente un clic del mouse sulla finestra di esecuzione fuori dalla scacchiera.

Esempio 3.2.1 - La pressione in sequenza dei bottoni [↑] [↑] [←] [↑] ha l'effetto di far eseguire a Q un percorso ad "elle", come un cavallo degli scacchi. □

In fondo a questa sottofinestra è presente una *linea di comando* dove è possibile scrivere singoli comandi in linguaggio `LFR`; il tasto [⏎] manda in esecuzione il comando scritto sulla linea. Si entra in questa modalità con un clic sulla linea e si esce con il tasto [esc]. Una volta entrato in modo comando, viene disabilitato il movimento di Q mediante i tasti freccia.

3.3 Programmazione in linguaggio LFR

Q è comandabile mediante i seguenti comandi elementari del linguaggio LFR già visti per Puntino.

- F : avanzamento alla casella davanti (F*orward*)
- B : indietreggiamento di una casella (B*ackward*)
- L : rotazione a sinistra (L*eft*)
- R : rotazione a destra (R*ight*)

Oltre alle precedenti azioni dei movimenti alementari si possono utilizzare le seguenti funzioni relative ai sensori di stato ed allo stato di memoria di Quadretto (spiegate nel cap. *Interagire*):

- X : ritorna un valore da 1 a 8 corrispondente alla coordinata orizzontale della casella in cui è posizionato Quadretto
- Y : ritorna un valore da 1 a 8 corrispondente alla coordinata verticale della casella in cui è posizionato Quadretto
- A : ritorna l'angolo di avanzamento attuale, ossia uno dei quattro valori 0 (NORTH), 1 (WEST), 2 (SOUTH), 3 (EAST) (A*ngle*)
- D : ritorna la distanza (in caselle) dall'oggetto davanti (D*istance*)
- C : ritorna 0 se la casella davanti a Quadretto è di colore nero; ritorna 1 in tutti gli altri casi (C*olor*)
- S : ritorna 1 se la casella davanti è libera; ritorna 0 se la casella davanti è occupata da un oggetto (S*pace*)
- T : ritorna 1 se Quadretto è a contatto con un oggetto davanti; ritorna 0 se la casella davanti è libera (T*ouch*)
- P : ritorna 1 se l'ultimo tentativo di movimento non è riuscito a causa di una situazione di impedimento dovuto ad un blocco da parte di un oggetto; 0 altrimenti (P*ressure*)
- Q : ritorna 1 se la casella corrente è marcata, 0 altrimenti (Q*uery*)
- M : ritorna la sequenza delle azioni presenti nella pila (M*emory*)
- N : *ripristina* il contenuto della pila vuota (N*ew*)
- I : *inizializza* la scacchiera eliminando tutti gli oggetti presenti (muri e blocchi) e riporta Quadretto alla situazione iniziale (I*nit*)
- W : *marca* la casella corrente (W*rite*)
- U : *cancella* la marcatura della casella corrente (U*nmark*)

I comandi/funzioni elementari descritti sopra possono essere composti mediante i seguenti schemi strutturali:

$n\ \alpha$: ripete n volte il programma α

`[` $\alpha_1\,\alpha_2\,\ldots\,\alpha_n$ `]` : raggruppa la sequenza α

`-`α : inversione del programma α

Si possono inoltre utilizzare dei controlli sulle azioni tipici dei linguaggi di programmazione di tipo procedurale: `if`, `while`, `repeat`, `loop`.

Come la maggior parte dei linguaggi, il linguaggio di programmazione `LFR` offre la possibilità di definire *sottoprogrammi*, ossia assegnare un nome identificativo a delle parti di programma che vengono poi richiamate ed utilizzate in altre parti del programma. La sintassi è la seguente:

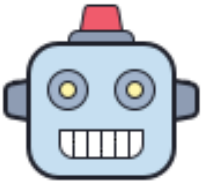
id `is` *exp*

dove *id* è l'identificatore del sottoprogramma, costituito da una sequenza di caratteri alfabetici minuscoli ed *exp* è l'espressione che viene assegnata all'identificatore *id*, costituita da un generico programma `LFR`; nel punto della chiamata del sottoprogramma viene scritto l'identificatore *id* e viene eseguita l'espressione *exp* ottenendo il corrispondente effetto.

Tutte le caratteristiche del linguaggio `LFR` sono descritte nel cap. *Interagire*.

4

PYQUAD



PyQuad (PYthon QUAdretto) è la versione di Quadretto comandabile mediante il linguaggio Python. Con questo linguaggio vengono superati i limiti del linguaggio `LFR` e si ha la possibilità di codificare algoritmi più articolati ed avanzati. In particolare si rende disponibile il ricorso a variabili ed espressioni numeriche e logiche e l'uso di sottoprogrammi con argomenti.

4.1 Attivazione dell'ambiente PYQUAD

L'ambiente PyQuad viene attivato all'interno di AlgMath selezionando, alla voce di menu *Room*, l'opzione *PyQuad*. È possibile fare questa scelta inizialmente scrivendo nel file `config.py` l'istruzione `setconfig('PyQuad')`. Al momento dell'esecuzione viene eseguito il file `initq.py`; tale file contiene alcune istruzioni (in linguaggio Python) per l'inizializzazione dell'ambiente (caricamento di librerie, posizionamento iniziale di Q, caricamento di configurazioni della scacchiera, ...).

Una volta attivato l'ambiente Quadretto si può scrivere il programma `Python` nella finestra dell'editor e poi mandarlo in esecuzione con il bottone *Run* presente nella stessa finestra. Si possono anche eseguire singole istruzioni Python dalla linea di comando posta alla base della finestra di esecuzione, sotto alla scacchiera di movimento. In alternativa si può scrivere dapprima il programma Python (con estensione `.py`) in un file sorgente mediante un qualsiasi editor di testo e successivamente eseguirlo mediante l'icona *PyQuad* della toolbar, selezionando la voce *Run...* mediante la quale viene richiesto di selezionare il programma da eseguire.

4.2 Interfaccia di PYQUAD

All'attivazione dell'ambiente PyQuad si presenta una finestra identica e con le stesse funzionalità di quella di Quadretto già descritta al paragrafo 3.2, con l'unica differenza che nella sottofinestra dell'editor a sinistra nella linea di comando alla base della finestra di esecuzione a destra vengono accettati comandi scritti in linguaggio Python.

4.3 Programmazione in Python

In alternativa al linguaggio `LFR`, i movimenti di Q possono essere programmati mediante il linguaggio Python. Le istruzioni Python per muovere Q sono:

- `forward()`: avanza alla casella successiva
- `back()`: indietreggia di una casella
- `left()`: ruota a sinistra
- `right()`: ruota a sinistra

Esempio 4.3.1 - Il programma che segue fa compiere a Q un cammino lungo un quadrato di lato 3.

```
for m in range(4):
    for n in range(2):
        forward()
    right()
```

□

Lo stato (posizione e verso di avanzamento) di Q sono gestiti mediante le seguenti funzioni e comandi:

- `posc()`: valore della colonna dove è posizionato Q
- `posr()`: valore della riga dove è posizionato Q
- `angle()`: valore del verso di avanzamento di Q
- `locate(`c`,`r`,`v`=NORTH)`: posizionamento alla colonna c, riga r e verso v

I valori dei versi di avanzamento sono rappresentati dalle costanti simboliche `NORTH`, `WEST`, `SOUTH`, `EAST`.

Esempio 4.3.2 - Il programma che segue ruota in senso orario Q verso est.

```
while angle() != EAST:
    right()
```

□

Sono inoltre utilizzabili i seguenti comandi:

- `setspeed(`v`)`: setta la velocità v di avanzamento di Q; sono ammessi i seguenti valori: 0 : lenta, 1 : media, 2 : veloce

Altri comandi in Python saranno presentati nei prossimi paragrafi.

4.4 La scacchiera

La scacchiera è composta da 3 livelli di elementi:

- oggetti
- colori
- scritte

Queste tipologie di elementi possono coesistere su una stessa casella.

Oggetti sulla scacchiera

Sulle caselle all'interno della scacchiera è possibile disporre degli oggetti, di due diverse tipologie: *muri* e *blocchi*. Questi oggetti vengono creati selezionando la voce *Wall* o la voce *Block* dal menu popup che si apre con clic con il tasto destro del mouse sulla casella desiderata. Ogni blocco è caratterizzato da un numero progressivo che appare sul blocco stesso.

I possibili oggetti che possono essere disposti sulle caselle della scacchiera sono descritti nella lista che segue; il nome indicato corrisponde all'identificatore utilizzabile nei programmi Python (fra parentesi è riportato il corrispondente codice numerico usato nei file `.brd` di configurazione della scacchiera):

- `ROBOT` (7): oggetto robot
- `BLOCK` (8): oggetto blocco
- `WALL` (9): oggetto muro
- `SPACE` (10): spazio, nessun oggetto

Il posizionamento degli oggetti avviene mediante clic con il tasto destro del mouse che propone un menu dal quale scegliere l'oggetto da posizionare sulla casella selezionata.

La gestione del movimento di Q in presenza di oggetti viene gestita mediante i seguenti sensori:

- `front()`: sensore di tatto che ritorna uno dei valori `WALL`, `BLOCK`, `SPACE` a seconda che nella casella davanti ci sia un muro, un blocco o spazio
- `touch()`: sensore di contatto che ritorna *True* se davanti c'è un muro o un blocco; equivale a `front() in {BLOCK,WALL}`
- `blocked()`: sensore di stato che ritorna *True* se l'ultima azione di movimento (`forward()` o `backward()`) non è riuscita a spostare Q a causa di un muro o di blocchi che non possono essere spostati
- `dist()`: sensore di distanza che ritorna il numero di caselle davanti libere da oggetti

Esempio 4.4.1 - Il programma che segue avanza Q fino ad arrivare ad una situazione di blocco.

```
while not blocked():
    forward()
```

□

I colori delle caselle

Le caselle della scacchiera possono essere colorate. A seguire sono elencate le costanti simboliche dei possibili colori ed il loro corrispondente effetto di quando Q vi transita (fra parentesi è riportato il valore numerico corrispondente a ciascuna costante):

- `WHITE` (1): casella neutra (senza indicazione di comando)
- `YELLOW` (2): rotazione a sinistra
- `RED` (3): rotazione a destra
- `BLUE` (4): inverte il senso di marcia
- `BLACK` (6): colore del percorso

Q percepisce il colore della casella davanti mediante il sensore

- `color()`: colore della casella davanti

ed è in grado di lasciare una traccia dove passa se viene attivata l'opzione

- `trace(TRUE)`: Q colora con `BLACK` la casella su cui transita

La colorazione delle caselle può essere fatta interattivamente con clic-dx sulla casella desiderata: si apre un menu popup mediante il quale è possibile selezionare il colore desiderato. Una volta selezionato un colore è sufficiente un semplice clic su una casella per colorarla con il colore selezionato precedentemente; un clic-sx su una casella colorata ha l'effetto di togliere il colore, portandolo a `WHITE`. In alternativa alla modalità interattiva sopra descritta, è possibile colorare le caselle con le seguenti istruzioni:

- `fill(`*col*`,`*c*`,`*r*`)`: colora con il colore *col* la casella (c, r)
- `fill(`*col*`)`: colora con il colore *col* la casella in cui si trova Q

Con *Coding-on-board* si indica una modalità di programmazione dei movimenti specificando il comportamento di Q quando arriva in una data casella: all'arrivo nella casella Q esegue l'azione (di rotazione) in base al colore della casella: se è *gialla* ruota a *sinistra*, se *rossa* ruota a *destra* (nel caso in cui sia *bianca* non esegue alcuna rotazione). Svolta l'eventuale rotazione, Q fa un passo in avanti nella direzione corrente. Il movimento viene attivato mediante l'icona *PyQuad* della finestra di esecuzione selezionado la voce *Start* che fa avanzare Q (ed eseguire l'eventuale azione di rotazione). Con questa modalità, anziché programmare Q, si programmano le caselle in modo che il robot che vi transiterà segua le direttive di movimento specificate in ciascuna casella.

Scritte sulle caselle

Sulle caselle è possibile scrivere dei caratteri o stringhe mediante le seguenti funzioni:

- `stamp(`s`,`c`,`r`)`: scrive la stringa s sulla casella (c, r)
- `stamp(`s`)`: scrive la stringa s sulla casella in cui si trova Q
- `label(`c`,`r`)`: ritorna la stringa presente sulla casella (c, r)
- `label()`: ritorna la stringa presente sulla casella in cui si trova Q

4.5 Configurazione della scacchiera

La configurazione della scacchiera può essere fatta interattivamente: con clic-dx del mouse su una casella, si apre un menu' contestuale mediante il quale si seleziona l'oggetto ed il colore da assegnare alla data casella.

Sono utilizzabili anche le seguenti istruzioni:

- `clear()`: elimina i colori e le scritte sulla scacchiera
- `empty()`: elimina gli oggetti (muri e blocchi) sulla scacchiera
- `init()`: elimina gli oggetti, i colori e le scritte sulla scacchiera e porta Q alla posizione $(1, 1)$ con orientamento verso nord

I file con estensione `.brd` sono file testo (editabili mediante un qualsiasi editor di testo) che contengono l'immagine di una configurazione della scacchiera. Un esempio è riportato a seguire.

```
# File: esempio.brd
5,4,3,9
2,6,2,7
3,4,1,8
```

Oltre ai commenti `#...` ed alle linee vuote, le altre linee del file descrivono delle quadruple aventi il formato "$colonna, riga, colore, oggetto$" Gli indici di colonna e riga partono da 1; $(1, 1)$ sono le coordinate della casella in basso a sinistra. I valori dei colori e degli oggetti sono i seguenti:

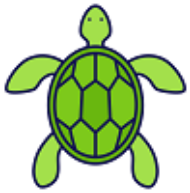
- *colori* : bianco : 1, giallo : 2, rosso : 3, blu : 4, nero : 6
- *oggetti* : quadretto : 7, blocco : 8, muro : 9

Il contenuto delle varie caselle della tastiera può essere gestito interamente mediante clic e tasto destro del mouse, oppure fare ricorso alle seguenti istruzioni Python:

- `loadboard(`*fname*`)`: carica il file *fname* di configurazione della scacchiera
- `resetboard()`: riporta la scacchiera allo stato iniziale (vuota e con Q in posizione $(1, 1)$)

5

PYTURTLE



PyTurtle è un ambiente in cui viene comandato ed agisce l'automa della tradizionale *geometria della tartaruga* secondo le modalità ereditate dal linguaggio Logo, con la differenza che in PyTurtle la tartaruga viene comandata mediante il linguaggio Python. Chiameremo questo automa con il nome Tartaruga o semplicemente T. Di fatto, l'automa Tartaruga costituisce un'estensione di Puntino in quanto T può girare di un qualsiasi angolo ed avanzare di una generica quantità, anche non intera.

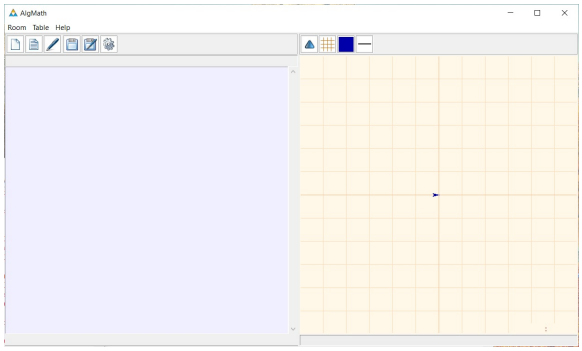
5.1 Attivazione dell'ambiente PYTURTLE

L'ambiente PyTurtle viene attivato all'interno di AlgMath selezionando, alla voce di menu *Room*, l'opzione *PyTurtle*. È possibile fare questa scelta scrivendo nel file `config.py` l'istruzione `setconfig('PyTurtle')`. Al momento dell'esecuzione viene eseguito il file `initg.py`; tale file contiene alcune istruzioni (in linguaggio Python) per l'inizializzazione dell'ambiente (caricamento di librerie, posizionamento iniziale di T, ...).

Una volta attivato l'ambiente PyTurtle si può scrivere il programma in linguaggio *Python* nella finestra dell'editor e poi mandarlo in esecuzione con il bottone *Run* presente nella finestra dell'editor. Si possono anche eseguire singole istruzioni *Python* dalla linea di comando posta alla base della finestra del piano di movimento di T. In alternativa si può scrivere dapprima il programma in linguaggio Python in un file sorgente mediante un qualsiasi editor di testo e successivamente eseguirlo mediante l'icona *PyTurtle* della toolbar, selezionando la voce *Run...* mediante la quale viene richiesto di selezionare il programma da eseguire.

5.2 L'interfaccia di PYTURTLE

All'attivazione dell'ambiente PyTurtle si presenta la seguente finestra:



La finestra è suddivisa in due sottofinestre: sulla finestra di editor di sinistra viene scritto, in linguaggio Python, il programma da eseguire mediante l'icona *Run*; eventuali errori (di sintassi) vengono segnalati nella linea di comunicazione posta a destra, in fondo alla sottofinestra del piano di movimento di T.

Nella sottofinestra di destra si vede il movimento di T. In questa sottofinestra è presente una toolbar di quattro icone dove sono predisposte le seguenti funzionalità:

 (*PyTurtle*) vengono presentate le seguenti sottovoci:

- *Home*: riporta T all'origine degli assi e rivolto verso Nord
- *Visible*: rende visibile il triangolino di stato di T
- *Stop*: interrompe l'esecuzione del programma e ferma T
- *Run...*: seleziona ed esegue un programma
- *Rerun*: esegue l'ultimo programma eseguito

 (*Plane*) vengono presentate le seguenti sottovoci:

- *Grid*: attiva/disattiva il reticolo quadrettato del piano
- *Clear*: elimina quanto disegnato sul piano

 (*Color*) apre una finestra di selezione del colore di disegno

 (*Width*) apre una finestra di selezione dello spessore della penna di disegno

La parte centrale della finestra di esecuzione visualizza il piano di disegno.

In fondo a questa finestra è presente una *linea di comando* dove è possibile scrivere singoli comandi in linguaggio Python; il tasto ↵ manda in esecuzione il comando scritto sulla linea. Si entra in questa modalità con un clic sulla linea e si esce con il tasto esc.

5.3 Programmazione in Python

La grafica della t. è integrata all'interno dell'ambiente PyTurtle dove si possono eseguire le istruzioni Python descritte a seguire.

Comandi elementari di disegno

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

- `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

I passi della tartaruga sono misurati in unità virtuali (numeri decimali), coerentemente con l'unità di misura del grid che si vede nella finestra grafica. Gli angoli vengono misurati in gradi (sessagesimali).

Comandi di configurazione

Oltre ai precedenti comandi di disegno sono utilizzabili i seguenti comandi di configurazione:

- `penup()`: alza la penna dal foglio
- `pendown()`: abbassa la penna sul foglio
- `hide()`: nasconde la tartaruga
- `show()`: rende visibile la tartaruga
- `home()`: ripristina la condizione iniziale della tartaruga
- `clear()`: cancella quanto disegnato dalla t. (senza spostarla)

Inizialmente la penna è abbassata sul foglio; pertanto si può evitare il ricorso a `penup` e `pendown`, usando solamente `forward` e `jump`.

Visualizzazione di testi

È possibile visualizzare testi mediante le seguenti funzioni in linguaggio Python:

- `print(`s`)` : visualizza la stringa s sulla linea di comando
- `write(`s`)` : visualizza la stringa s sulla finestra testo delle label

Gli stati dell'automa della grafica della tartaruga

L'automa della grafica della tartaruga può essere pensato come la composizione di un automa *Motore* che gestisce lo spostamento sul piano con un automa *Penna* che si incarica di disegnare. Questo automa composto è caratterizzato dai seguenti attributi:

- posizione della t. sul piano
- 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 s delle linee (valori: $1, 2, 3$)
- `getwidth()`: ritorna lo spessore corrente delle linee
- `setvisible(False/True)`: imposta lo stato di visibilità della t.
- `isvisible()`: ritorna `True` se e solo se la t. è visibile
- `setpendown(False/True)`: alza/abbassa la penna sul foglio.
- `ispendown()`: ritorna `True` se e solo se la penna è abbassata

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

5.4 La tartaruga nel piano punteggiato

Il sistema grafico costituito dalla tartaruga e dalla associata penna può essere gestito nel piano punteggiato mediante i seguenti 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

Un'assegnazione della forma

```
P = pos()
```

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

5.5 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 = Turtle(...)`

  Il costruttore `Turtle` può avere degli argomenti quali il colore (`color`) e lo spessore (`width`) della penna
- per fare eseguire ad una tartaruga `t` un'*azione* (comando, settaggio, funzione) viene utilizzata la notazione puntata

  `t.`*azione*`(...)`

Esempio 5.5.1 - La seguente porzione di codice disegna di un quadrato rosso di lato di 5 unità:

```
t = Turtle()
t.setcolor('red')
t.setwidth(2)
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 5.5.2 - 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.

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

```
n = 17 #numero di tartarughe
t = n*[None]
colors = ['red','green','blue','orange','brown']
for i in range(n):
    t[i] = Turtle(color=colors[i%len(colors)],width=1)
    t[i].left(360/n*i)
    t[i].show()

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

□

E' possibile far interagire tartarughe diverse.

Esempio 5.5.3 - 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
```

□

6

PYPEN

PyPen è un ambiente di *geometria dinamica* attivabile all'interno di AlgMath. Le entità grafiche (punti, segmenti, semirette, rette, circonferenze) possono essere create sia interattivamente, mediante specifiche icone presenti sull'interfaccia grafica, sia mediante istruzioni in linguaggio Python. In entrambi i casi, le entità sono dinamiche, cioè modificabili interattivamente, trascinandole mediante il mouse, oppure mediante istruzioni in linguaggio Python; in particolare si possono modificarne le coordinate ottenendo una loro movimentazione sul video. Le espressioni (geometriche, numeriche e testuali) sono dinamiche nel senso che vengono ricalcolate contestualmente con le modifiche. All'interno dell'ambiente di PyPen è possibile scrivere ed eseguire programmi in Python o singole istruzioni sul campo di comando in fondo alla finestra.

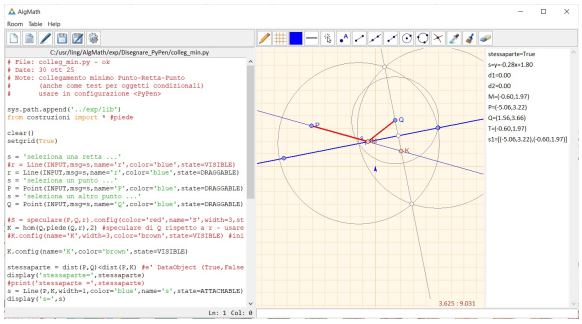
6.1 Attivazione dell'ambiente PYPEN

L'ambiente PyPen viene attivato all'interno di AlgMath selezionando, alla voce di menu *Room*, l'opzione *PyPen*. È possibile fare questa scelta inizialmente scrivendo nel file `config.py` l'istruzione `setconfig('PyPen')`. Inizialmente viene eseguito il file `initg.py` il quale contiene alcune istruzioni Python di inizializzazione dell'ambiente.

Una volta attivato l'ambiente si può scrivere il programma Python nella finestra dell'editor e poi mandare in esecuzione con il bottone *Run* presente nella finestra dell'editor. Si possono anche eseguire singole istruzoni Python dalla linea di comando posta alla base della finestra di esecuzione, sotto al piano di disegno. In alternativa si può scrivere dapprima il programma Python in un file sorgente mediante un qualsiasi editor di testo e successivamente eseguirlo mediante l'icona *PyPen* della toolbar, selezionando la voce *Run...* mediante la quale viene richiesto di selezionare il programma da eseguire.

6.2 L'interfaccia di PyPen

All'attivazione dell'ambiente PyPen si presenta la seguente finestra:



La finestra è suddivisa in due sottofinestre: sulla finestra di editor di sinistra viene scritto, in linguaggio `Python`, il programma da eseguire mediante l'icona *Run*; eventuali errori (di sintassi) vengono segnalati nella sottofinestra posta a destra.

Nella sottofinestra di destra si vede l'esito dell'esecuzione del programma. In questa sottofinestra è presente, in alto, una toolbar di icone dove, alle diverse voci, sono predisposte le seguenti funzionalità:

- (*Pencil*) vengono presentate le seguenti sottovoci:
  - *Import...*: seleziona ed importa un modulo
  - *Run...*: seleziona ed esegue un programma
  - *Rerun*: esegue l'ultimo programma eseguito
  - *Stop*: interrompe l'esecuzione del programma

 (*Plane*) vengono presentate le seguenti sottovoci:

- *Grid*: attiva/disattiva il grid del reticolo del piano
- *Clear*: elimina quanto presente sul terminale

 (*Color*) apre una finestra di selezione del colore di disegno

 (*Width*) apre una finestra di selezione dello spessore della penna di disegno

 (*State*) apre un menu per selezionare lo stato corrente degli elementi di disegno

 (*Point*) input di un punto mediante mouse

 (*Segment*) input di un segmento selezionando mediante mouse i 2 vertici

 (*Line*) input di una retta selezionando mediante mouse 2 punti di passaggio

 (*Ray*) input di una semiretta mediante mouse

 (*Circle*) input di una circonferenza selezionando mediante mouse il centro ed un punto di passaggio

 (*Circle3p*) input di una circonferenza selezionando mediante mouse 3 punti di passaggio

 (*Inters*) operazione di intersezione di 2 linee selezionate mediante mouse

 (*GetFormat*) acquisizione del formato (colore,stato,spessore) dell'elemento selezionato mediante mouse

 (*SetFormat*) imposizione del formato corrente (colore,stato,spessore) all'elemento selezionato mediante mouse

 (*Erase*) eliminazione elementi [da completare]

La parte centrale della finestra di esecuzione visualizza il piano di disegno.

In fondo a questa sottofinestra è presente una *linea di comando* dove è possibile scrivere singoli comandi ...; il tasto ⏎ manda in esecuzione il comando scritto sulla linea. Si entra in questa modalità con un clic sulla linea e si esce con il tasto esc.

6.3 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.

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

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`

Attributi di interazione

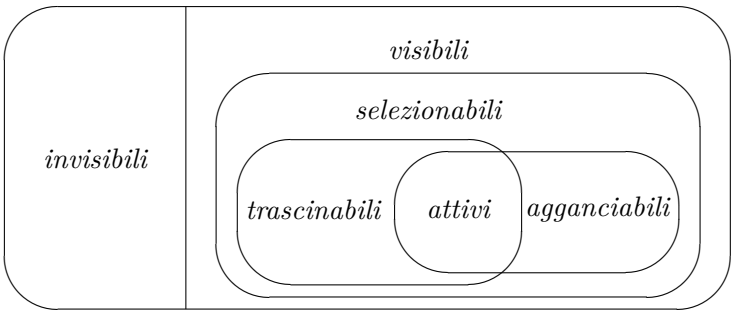
Gli oggetti possono essere manipolati interattivamente; le possibilità dipendono dal loro stato di attivazione specificato dal seguente attributo:

- `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. Sono utilizzati anche per realizzare degli input di punti vincolati ad un elemento non visibile.
- *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
- *selezionabile* (`SELECTABLE`=2): oggetto visibile su video; reagisce, cambiando colore, quando viene sovrapposto dal mouse; può essere selezionato con un clic per subire un'operazione (cancellazione, intersezione, cambio di attributo grafico, ...); non può essere agganciato o trascinato
- *agganciabile* (`ATTACHABLE`=3): reagisce, cambiando colore, quando viene sovrapposto dal mouse; in fase di inserimento, con un clic vicino (quando è evidenziato) può essere "agganciato" (modificando il punto di clic); non può essere trascinato
- *trascinabile* (`DRAGGABLE`=4): oggetti trascinabili con clic-sx del mouse; non vengono agganciati in fase di inserimento
- *attivi* (`ACTIVE`=5): oggetti che assommano le caratteristiche degli oggetti agganciabili e trascinabili

Si può evidenziare il seguente diagramma insiemistico:



Lo stato di un oggetto viene specificato mediante l'attributo univalore `state`.

6.4 Creazione degli oggetti grafici

In PyPen è possibile gestire le seguenti tipologie di primitive grafiche:

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

Ogni oggetto grafico viene creato mediante un costruttore che viene attivato agendo sulle icone della GUI, oppure mediante l'esecuzione di una specifica istruzione Python avente il nome della classe (*Point*, *Line*, *Segment*, *Ray*, *Circle*) dell'oggetto che si intende creare e degli argomenti che ne definiscono la geometria (coordinate) e gli attributi grafici (nome, colore, spessore, stato).

Esempio 6.4.1 - L'istruzione

```
Point((2,3),name='P',color='red',state=DRAGGABLE)
```

genera il punto di coordinate $(2,3)$, avente nome P, colore *rosso* e stato *trascinabile*. □

Se non vengono specificati alcuni attributi, vengono assunti da quelli al momento impostati sulla GUI.

Esempio 6.4.2 - L'istruzione

```
Segment(Point(0,0),Point(1,2),width=THIN)
```

genera il segmento di estremi i punti di coordinate $(0,0)$ e $(1,2)$, avente spessore *sottile* ed avente colore e stato uguale a quello correntemente impostato sulla GUI. □

Dopo aver creato un oggetto grafico se ne possono modificare gli attributi mediante il metodo `config`, come si vede nel seguente esempio.

Esempio 6.4.3 - L'istruzione

```
a = Line(Point(0,0),Point(1,-2),state=ACTIVE,width=THIN)
```

genera la retta `a` passante per i punti di coordinate $(0,0)$ e $(1,-2)$, avente spessore *sottile* ed avente colore e stato uguale a quello correntemente impostato sulla GUI; successivamente mediante l'istruzione

```
a.config(name='r',color='green',width=THICK)
```

la retta viene modificata assumendo il nome r, il colore *verde* e spessore *spesso*. Si noti che `a` è l'identificatore dell'oggetto grafico nel contesto del programma mentre r è il nome dell'etichetta che compare a fianco della retta sull'interfaccia grafica. □

6.5 Settaggio e selezione degli attributi

Gli attributi degli oggetti possono essere settati e selezionati con diverse modalità:

1. Il valore degli attributi può essere specificato nel costruttore in fase di creazione dell'oggetto; per le primitive grafiche che vengono costruite in modo interattivo i valori degli attributi non specificati nel costruttore vengono assunti da quelli correntemente impostati sulla GUI
2. È possibile acquisire i valori attuali correntemente impostati sulla GUI mediante i metodi `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). Queste 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=DRAGGABLE)
   ```

   setta il colore *rosso* ed il valore di stato `DRAGGABLE` 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.

6.6 Modalità di input degli oggetti

L'input avviene selezionando singoli punti con clic-sx del mouse; la modalità di input è evidenziata dalla presenza di un cursore del mouse a forma di croce. I punti vengono generalmente presi sul punto dove si fa clic/drag col mouse. Se si è vicini al un elemento grafico con stato di interazione `ATTACHABLE` esso si evidenzia con un colore più chiaro; in questo caso se si è vicini ad un punto esso viene condiviso ed il punto in input diventa un riferimento al punto già presente; se si è vicini ad una linea (segmento, retta, semiretta, circonferenza) il punto selezionato diventa un punto vincolato sulla linea.

A seguire sono riportati alcuni esempi di possibili modalità di input interattivo di primitive grafiche.

Esempio 6.6.1 - Con la seguente porzione di codice:

```
P = Point(INPUT)
s = Segment(INPUT,color='red',state=VISIBLE)
Q = Point(INPUT,color='black',state=DRAGGABLE,on=s)
```

viene acquisito un punto `P` con gli attributi correntemente impostati sulla GUI; successivament un segmento `s` di colore *rosso*, stato di interazione `VISIBLE` e gli altri attributi impostati sulla GUI; per ultimo viene acquisito un punto `Q` di colore *nero* vincolato sul segmento `s`. □

Esempio 6.6.2 - Con l'assegnazione

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

vengono settati sulla GUI i valori dei due attributi `color` e `state`, mentre con

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

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. □

Esempio 6.6.3 - Con la seguente porzione di codice:

```
a = Circle(INPUT,msg='seleziona centro e punto')
A = Point(2,3)
B = Point(-2,-1)
b = Circle(INPUT,(A,B))
```

viene acquisita una circonferenza `a`, con il messaggio guida indicato, selezionando con clic-sx il centro ed un punto di passaggio, ed una circonferenza `b` passante per i due punti `A` e `B`. □

6.7 Modalità di output degli oggetti

Gli oggetti grafici possono essere visualizzati in diverse modalità:

- *modalità grafica*: gli oggetti grafici creati, se hanno l'attributo di stato `VISIBLE` (o superiore), vengono visualizzati direttamente sulla finestra grafica
- *modalità testo statica*: gli oggetti grafici possono essere convertiti in stringa con `str`(*obj*); la stringa descrive le coordinate al momento della conversione dell'oggetto; successive modifiche dell'oggetto non cambiano la stringa; la stringa può essere visualizzata sulla linea di comando con `print` oppure sulla finestra testo con `write`; ad esempio:

  ```
  write('Punto P='+str(P))
  ```

  oppure con:

  ```
  print('Punto P='+str(P))
  ```

- *modalità testo dinamica*: con

  `display`(s, obj)

  l'oggetto *obj* viene visualizzato sulla finestra testo mediante una stringa dinamica, preceduta dal prefisso statico s; eventuali modifiche dinamiche all'oggetto vengono direttamente riportate sulla corrispondente stringa dinamica (la finestra testo può essere aperta con clic-sx sulla finestra grafica ed attivando la finestra attivando la voce *TextWindow* del menu contestuale che si apre, oppure mediante l'istruzione Python `settextwin(True)`)

6.8 Assegnazioni, alias e cloni

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.

Per realizzare la copia di oggetti distinti è utilizzabile la la funzione *clone*(p) crea una copia del punto p; 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.

Mediante la funzione `clone` è possibile realizzare la copia di altre classi di oggetti, come si vede nel seguente esempio.

Esempio 6.8.1 - Con la seguente porzione di codice:

```
A = Point(3,4,name='A',color='red',state=DRAGGABLE)
B = Point(4,2,name='B',color='blue',state=DRAGGABLE)
s = Segment(A,B,name='s',color='green')
t = Segment(clone(head(s)),clone(tail(s)))
```

viene creato un segmento s ed una copia t di s, uguale ma indipendente da s. □

6.9 Point

I punti sono gli elementi fondamentali dell'ambiente *PyPen*: 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((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 $[0, 2]$
  `clone(P)`: copia integrale del punto P (attributi geometrici e grafici)

- *modificatori* : i seguenti modificatori delle coordinate accettano valori numerici; nel caso di valori dinamici, vengono convertiti in valori statici: punti del piano costruibili mediante i seguenti costruttori:

  `setx(`P, x`)`: modifica la coordinata x del punto P con il valore x
  `sety(`P, y`)`: modifica la coordinata y del punto P con il valore y
  `setxy(`P, x, y`)`: modifica le coordinate x ed y del punto P con i valori x ed y

- *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 dei punti P e Q
  P `-` Q : differenza vettoriale dei punti P e Q
  k `*` P : moltiplicazione di P per lo scalare k (anche a destra)
  `-`P: opposto del punto P (rispetto all'origine)

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

  `getx`(P) : coordinata x del punto P
  `gety`(P) : coordinata y del punto P
  `getxy`(P) : coppia (x,y) delle coordinate 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 $A\widehat{V}B$

6.10 Line, Segment e Ray

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 PyPen le rette sono rappresentate dagli oggetti della classe *Line* e vengono gestite mediante le seguenti funzioni.

- *costruttori* : le rette del piano costruibili mediante i seguenti costruttori (possono essere specificati gli argomenti `color` (colore), `state` (stato) e `width` (spessore)):

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

  `Line(`P`,`m`)`: retta per P e con coefficiente angolare m

  `Line(`P`,(`dx, dy`))`: retta per il punto 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, mediante i seguenti costruttori (può essere specificato anche l'argomento opzionale `msg` che specifica il messaggio guida dell'input):

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

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

  `Line(INPUT,(`dx, dy`))`: acquisizione interattiva con mouse di una retta avente inclinazione individuata dal vettore (dx, dy) selezionando il secondo punto della retta

  `Line(INPUT,`m`)`: acquisizione interattiva con mouse di una retta avente coefficiente angolare m selezionando il secondo punto della retta

- *operazioni*:

  `inters(`r, s`)` : punto di intersezione fra due linee (segmenti, semirette, rette, corconferenze)

  `slope(`r`)` : coefficiente angolare della retta r

Analogamente a *Line* vengono gestiti i segmenti (classe *Segment*) e le semirette (classe *Ray*).

6.11 Circle

Le circonferenze sono rappresentate dagli oggetti della classe *Circle* e vengono gestiti mediante le seguenti funzioni:

- *costruttori* : le circonferenze del piano costruibili mediante i seguenti costruttori (possono essere specificati gli argomenti `color` (colore), `state` (stato) e `width` (spessore)):

  `Circle(`C, r`)`:circonferenza di centro C e raggio r

  `Circle(`C, P`)`:circonferenza di centro C e passante per P

  `Circle(`P, Q, R`)`:circonferenza per i 3 punti P, Q, R

  Una circonferenza può essere acquisita interattivamente con il mouse, mediante i seguenti costruttori (in aggiunta agli argomenti `color`, `state` e `width` può essere specificato anche l'argomento opzionale `msg` che specifica il messaggio guida dell'input):

  `Circle(INPUT)`: acquisizione di una circonferenza selezionando il centro ed un punto di passaggio

  `Circle(INPUT,`C`)`: acquisizione di una circonferenza di centro C e passante per il punto selezionato

  `Circle(INPUT,`r`)`: acquisizione di una circonferenza di raggio r centrata sul punto selezionato

  `Circle(INPUT,())`: acquisizione di una circonferenza passante per 3 punti selezionati

  `Circle(INPUT,(`P`,))`: acquisizione di una circonferenza passante per il punto P e per i 2 punti selezionati

  `Circle(INPUT,(`P, Q`))`: acquisizione di una circonferenza passante per i punti P e Q e per il punto selezionato

- *operazioni*:

  `inters(`a, b`)` : punto di intersezione fra due linee (segmenti, semirette, rette, corconferenze)

  `center(`a`)` : centro della circonferenza a

  `radius(`a`)` : raggio della circonferenza a

6.12 Nomi, testi, etichette e scritte

In PyPen vengono gestite diverse classi di stringhe: *nomi* degli oggetti, *testi*, *etichette* e *scritte*.

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. L'attributo `name` può essere una stringa della forma $(nome, dx, dy)$ dove *nome* è la stringa del nome e (dx, dy) è una coppia di numeri che specificano l'offset in pixel del posizionamento del nome rispetto al punto di riferimento. 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((1,1),name='A',color='green',state=DRAGGABLE)
B = Point((3,2),name=('B',10,0),color='blue',state=DRAGGABLE)
C = (A+B).config(state=DRAGGABLE,color='red',name='A+B')
```

Testi

Un *testo* è un oggetto grafico (classe `Text`) 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),"Testo rosso costante",color='red',state=DRAGGABLE)
P = Point(1,4,color='brown',state=DRAGGABLE)
Text(P,'P({0},{1})',[getx(P),gety(P)],color='blue',state=VISIBLE)
```

Per default un testo viene centrato sul punto di riferimento; è possibile specificare, mediante l'argomento `anchor`, uno dei seguenti valori numerici per indicare il posizionamento rispetto al punto di riferimento: `A_CENTER` (default), `A_NORTH`, `A_NORTHEAST`, `A_EAST`, `A_SOUTHEAST`, `A_SOUTH`, `A_SOUTHWEST`, `A_WEST`, `A_NORTHWEST`.

Esempio:

```
Text((2,1),'Testo a destra di (2,1)',color='black',anchor=A_EAST)
```

Etichette

Una *etichetta* è un oggetto grafico (classe `Label`) costituito da 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 etichetta rimane alla stessa distanza (in pixel) dall'oggetto al quale è agganciata. Oltre all'offset di spiazzamento (in pixel) rispetto al punto di riferimento, per un'etichetta è possibile specificare, mediante l'attributo `anchor`, il verso di posizionamento del testo rispetto al punto addizionato dell'offset, con i valori numerici precisati precedentemente per i testi.

Esempio:

```
Label((2,3),"Etichetta blu",color='blue',state=DRAGGABLE)
P = Point(1,2,name=('P',10,0),color='black',state=DRAGGABLE)
x, y = getx(P),gety(P)
Label(P,"({0},{1})",[x,y],color='red',offset=(25,0),anchor=A_EAST)
```

Scritte

Una *scritta* è un oggetto grafico (classe `Writing`) che rimane fisso sulla finestra dell'applicazione e non risente delle operazioni di trascinamento e zoom; le coordinate di posizionamento sono espresse in pixel; possono essere riferite all'angolo alto a sinistra della finestra di disegno (angolo di coordinate $(0, 0)$) mediante l'argomento `corner=1` (default) oppure all'angolo in basso a destra mediante l'argomento `corner=2`.

Esempio:

```
Writing((80,-30),"In alto a sinistra",color='brown',state=DRAGGABLE)
P = Point((2,3),name='P',color='green',state=DRAGGABLE)
x, y = getx(P),gety(P)
Writing((80,-50),"P=({0},{1})",variables=[x,y],state=DRAGGABLE)
Writing((-180,50),"In basso a destra",color='red',corner=2,state=VISIBLE)
```

6.13 Valori ed espressioni dinamiche

In PyPen i punti e le altre entità grafiche sono oggetti dinamici in quanto modificabili interattivamente mediante il mouse oppure mediante specifici comandi.

Un *valore dinamico* (oggetto della classe *Value*) è un valore numerico che dipende da un oggetto dinamico.

Esempio 6.13.1 - Sono oggetti di classe *Value* i seguenti: le coordinate dei punti, la distanza fra due punti, la lunghezza del raggio di una circonferenza.

La modifica di un oggetto grafico attiva automaticamente la modifica dei valori dinamici che dipendono da esso.

Esempio 6.13.2 - Un cerchio avente come raggio la distanza fra due punti, si automodifica se vengono spostati i punti che ne definiscono la distanza.

Sui valori dinamici sono applicabili le seguenti operazioni che generano come risultato dei valori numerici (interi o decimali):

- operazioni aritmetiche elementari: x `+` y, x `-` y, x `*` y, x `/` y
- resto della divisione intera: x `%` y
- opposto (prefisso): `-`x
- elevamento a potenza: `pow(`x`,`n`)`
- valore assoluto: `abs(`x`)`
- funzioni trigonometriche: `sin(`a`)`, `cos(`a`)`, `tan(`a`)`

Sui valori dinamici si possono inoltre applicare le seguenti operazioni relazionali, in notazione infissa, che generano valori logici dinamici:

- confronti di uguaglianza e diversità: x `==` y
- confronti di ordine: x `<` y, x `<=` y, x `>` y, x `>=` y,

Sui valori logici dinamici si possono applicare i tradizionali operatori logici espressi in notazione funzionale:

- *and* logico: `andv(`x`,`y`)`
- *or* logico: `orv(`x`,`y`)`
- *not* logico: `notv(`x`)`

Un'operazione aritmetica coinvolgente un valore dinamico produce un valore dinamico, anche se qualche valore non è dinamico.

Un'*espressione dinamica* è un'espressione contenente qualche valore dinamico. Un'espressione dinamica può essere assegnata ad un identificatore mediante la tradizionale operazione di *assegnazione*. La modifica di un oggetto grafico comporta automaticamente la rivalutazione delle espressioni dinamiche che dipendono da esso e la modifica degli identificatori assegnati mediante l'espressione. Nel caso un'espressione dinamica sia visualizzata in formato testuale, il valore visualizzato viene automaticamente aggiornato.

Un'espressione dinamica può essere di diverso tipo:

- valore numerico; esempi: `dist(P,Q)`, `radius(c)`
- valore logico: `getx(P)<gety(P)`
- una stringa; ad esempio: label comprendente i valori delle coordinate di un punto
- un oggetto grafico; ad esempio: `dist(A,B)*P`

Mediante i valori dinamici è possibile, ad esempio, costruire delle entità grafiche che dipendono da altre.

Se e è un'espressione dinamica, con

$$\texttt{val}(e)$$

si ottiene il valore attuale (statico) ottenuto dalla valutazione dell'espressione e in funzione del valore attuale degli argomenti coinvolti nell'espressione.

6.14 Valori e punti condizionali

I *valori condizionali* sono espressi nella forma funzionale

$$\texttt{IfValue}(c, x, y)$$

dove c è una espressione logica dinamica ed x ed y sono delle generiche espressioni (anche dinamiche); il risultato è x se c è vera, altrimenti y.

Usando i valori condizionali si possono costruire dei *punti condizionali* con la notazione funzionale

$$\texttt{IfPoint}(c, A, B)$$

dove c è una espressione logica dinamica ed A e B sono delle generiche espressioni che producono un punto.

Esempio: nella seguente porzione di codice vengono generati 4 punti A, B, C, P di colore diverso ed un segmento che congiunge il punto P al punto più vicino collegandolo con un segmento del colore del punto a cui viene collegato:

```
A = Point(2,3,name='A',color='blue',state=DRAGGABLE)
B = Point(1,2,name='B',color='red',state=DRAGGABLE)
C = Point(4,1,name='C',color='green',state=DRAGGABLE)
P = Point(3,4,name='P',color='black',state=DRAGGABLE)
condA = andv(dist(P,A)<=dist(P,B),dist(P,A)<dist(P,C))
condB = andv(dist(P,B)<=dist(P,A),dist(P,B)<dist(P,C))
Q = IfPoint(condA,A,IfPoint(condB,B,C))
col = Q.getcolor()
P.config(color=col)  #cambia colore dinamicamente come Q
x = Segment(P,Q,color=col,state=DRAGGABLE,width=MEDIUM)
```

7

PYTERM

PyTerm (PYthon TERMinal) è un'applicazione che consente di

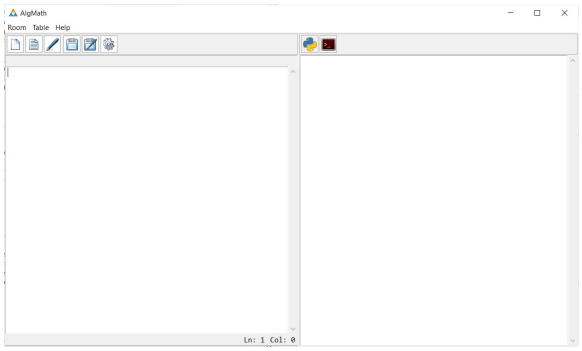
- caricare/editare/salvare file testo mediante un editor orientato alla sintassi del linguaggio Python
- eseguire programmi Python (senza installare il compilatore Python)
- raccogliere l'output di un programma Python
- editare il testo dell'output dell'output prodotto
- salvare su file l'output prodotto
- eseguire singole istruzioni Python
- gestire l'input da tastiera dei dati in formato testuale

7.1 Attivazione dell'ambiente PYTERM

PyTerm viene attivato all'interno di AlgMath selezionando alla voce di menu *Room* l'opzione *PyTerm*. È possibile attivare questa scelta inizialmente scrivendo nel file `config.py` l'istruzione `setconfig('PyTerm')`. Al momento dell'esecuzione viene eseguito il file `initp.py` presente nella stessa cartella; tale file contiene alcune istruzioni (in linguaggio Python) per l'inizializzazione dell'ambiente (caricamento di librerie, import di moduli, ...).

7.2 Interfaccia di PYTERM

Dopo aver attivato l'ambiente PyTerm si presenta la seguente finestra:



La finestra è suddivisa in due sottofinestre: sulla finestra di editor di sinistra viene scritto, in linguaggio `Python`, il programma da eseguire mediante l'icona *Run*; eventuali errori (di sintassi) vengono segnalati nella sottofinestra posta a destra.

Nella sottofinestra di destra si vede l'esito dell'esecuzione del programma. In questa sottofinestra è presente, in alto, una toolbar di due icone dove, alle diverse voci, sono predisposte le seguenti funzionalità:

 (*Python*) vengono presentate le seguenti sottovoci:

- *Import...*: seleziona ed importa un modulo
- *Run...*: seleziona ed esegue un programma
- *Rerun*: esegue l'ultimo programma eseguito
- *Stop*: interrompe l'esecuzione del programma

 (*Xterm*) vengono presentate le seguenti sottovoci:

- *Clear*: elimina quanto presente sul terminale
- *Command*: attiva la modalità *comando*
- *Edit*: attiva la modalità di editing
- *Freeze*: disattiva la modalità di editing
- *Save*: salva quanto presente sul terminale
- *Save as...*: salva il contenuto del terminale richiedendo il nome del file su cui salvare

In fondo a questa sottofinestra è presente una *linea di comando* dove è possibile scrivere singoli comandi ...; il tasto ⏎ manda in esecuzione il comando scritto sulla linea. Si entra in questa modalità con un clic sulla linea e si esce con il tasto esc.

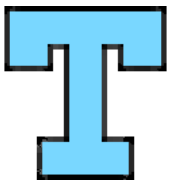
Una modalità alternativa consiste nello scrivere il programma in un file testo (con estensione `.py`) e poi eseguirlo mediante la voce di menu *Python* > *Run...* con la quale si seleziona il programma da eseguire.

7.3 Input e visualizzazione di stringhe in PyTerm

Il terminale di PyTerm estende le usuali istruzioni *input* e *print* di Python con delle omologhe funzioni come descritto al cap. 1, par.5 di AlgMath.

8

TMACHINE



TM*achine* è un ambiente per l'esecuzione di programmi mediante una *macchina di Turing* (in seguito *MdT*). Permette di:

- caricare/editare/salvare file contenenti programmi per la MdT
- eseguire programmi per la MdT
- controllare passo-passo l'esecuzione del programma

8.1 Attivazione dell'ambiente TMACHINE

TM*achine* viene attivato all'interno di ALGMATH selezionando alla voce di menu *Room* l'opzione *TMachine*. È possibile attivare questa scelta inizialmente scrivendo nel file `config.py` l'istruzione `setconfig('TMachine')`.

Il file `tmachine.cfg` è un file di configurazione che definisce alcuni parametri della finestra dell'applicazione:

- `inputfont` *font size attr*: definisce il font per i caratteri del campo di input
- `tapefont` *font size attr*: definisce il font per i caratteri del nastro
- `labelfont` *font size attr*: definisce il font per i caratteri delle etichette dei campi testo
- `statefont` *font size attr*: definisce il font per i caratteri dei campi di stato

Nel file `tmachine.cfg` si possono inserire dei commenti di linea che iniziano dal carattere `#` fino alla fine della linea.

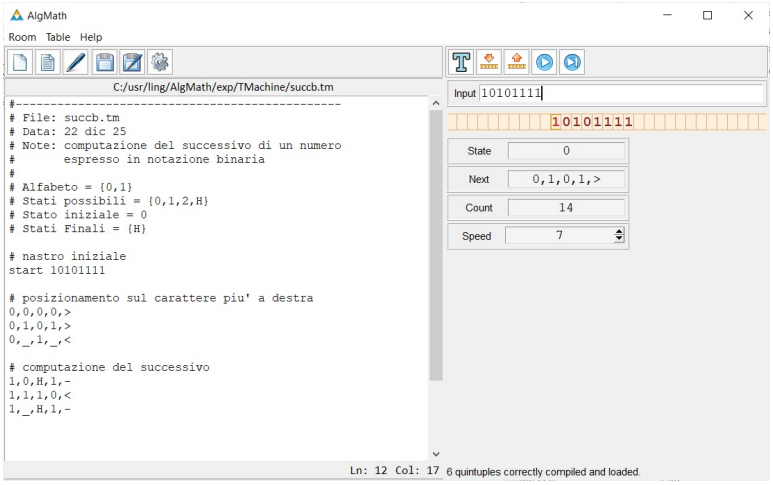
Esempio 8.1.1 - A seguire è riportato un esempio del file di configurazione:

```
# File: tmachine.cfg
# Date: 22 dic 25
# Note: configuration file of TMachine

tapefont  courier 16 bold
inputfont courier 14
labelfont Helvetica 11
statefont courier 13
```

8.2 Interfaccia di TMACHINE

Dopo aver attivato l'ambiente TM*achine* si presenta la seguente finestra:



La finestra è suddivisa in due sottofinestre: sulla finestra di editor di sinistra viene scritto, in linguaggio TM, il programma da caricare sulla MdT mediante l'icona *Run*; il programma è costituito da una sequenza di quintuple; tale programma serve per definire il comportamento della MdT; eventuali errori (di sintassi) vengono segnalati nella linea di comunicazione posta a destra, in fondo alla sottofinestra ddi esecuzione della MdT.

Nella sottofinestra di destra si vede l'esecuzione della MdT. In questa sottofinestra è presente una toolbar di quattro icone dove sono predisposte le seguenti funzionalità:

(*TMachine*) vengono presentate le seguenti sottovoci:

- *Reset*: elimina il contenuto del nastro
- *Run...*: seleziona ed esegue un programma
- *Rerun*: esegue l'ultimo programma eseguito

: scrive sul nastro il contenuto del campo testo dell'input

: copia il contenuto del nastro sul campo testo dell'input

Mediante l'uso combinato delle funzionalità *Write tape* e *Read tape* risulta indirettamente possibile l'editing sul nastro della MdT.

8.3 Modalità di esecuzione della TM

Le ultime due icone della toolbar regolano l'esecuzione della MdT:

 : copia l'input sul nastro ed esegue dall'inizio il programma caricato

 : esegue un passo del programma

 : sospende l'esecuzione

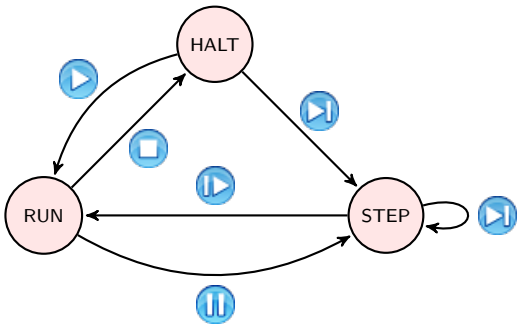
 : riprende l'esecuzione

 : arresta l'esecuzione

La TM può trovarsi nelle seguenti modalità di esecuzione:

- `HALT` : la TM è ferma
- `RUN` : la TM è in esecuzione ciclica
- `STEP` : la TM è in esecuzione "passo-passo"

In ciascuna di queste modalità possono essere attivate due possibili transizioni di stato, come descritto nel grafo di transizione di stato che segue:



8.4 Sintassi dei programmi

Il testo che segue riporta un esempio di programma per la TMachine dal quale si può desumere il formato delle quintuple e la definizione (facoltativa) del nastro iniziale.

```
#-----------------------------------------------
# File: succb.tm
# Note: computazione del successivo di un numero
#       espresso in notazione binaria
#
# Alfabeto = {0,1}
# Stati possibili = {0,1,2,H}
# Stato iniziale = 0
# Stati Finali = {H}

# nastro iniziale
start 10101111

# posizionamento sul carattere piu' a destra
0,0,0,0,>
0,1,0,1,>
0,_,1,_,<

# computazione del successivo
1,0,H,1,-
1,1,1,0,<
1,_,H,1,-

# posizionamento sul carattere piu' a sinistra
2,0,2,0,<
2,1,2,1,<
2,_,H,_,>
```