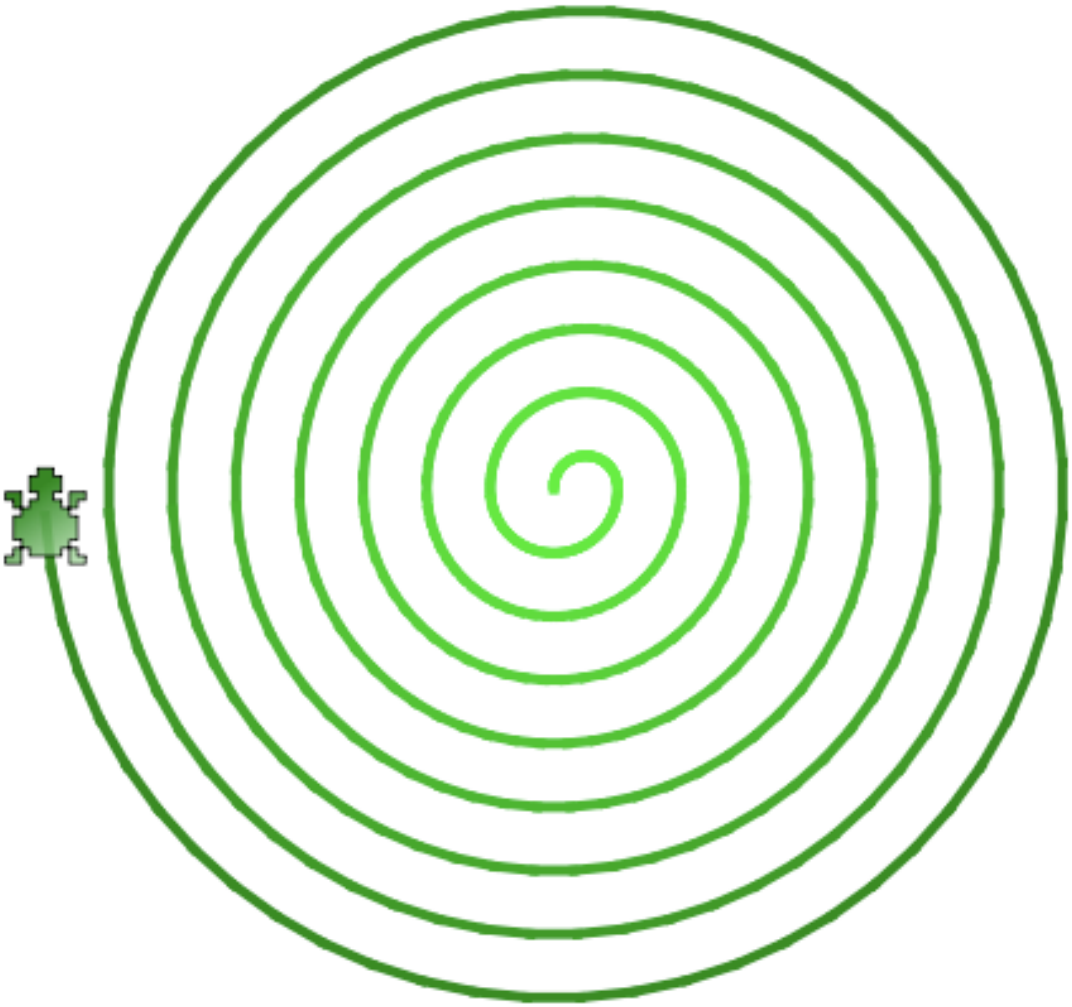
C O R S O D I L O G O

MANUALE DI RIFERIMENTO
ED ESERCIZIARIO, LEZIONE N.3



LEZIONE N.3

Copyright (c) Ugo Landini

versione 1.0.0 01/04/2009

Lavoro rilasciato sotto licenza

Commons Creative, Attributions, Non Commercial, Share Alike

http://creativecommons.org/licenses/by-nc-sa/3.0/

Lezione n.3

Comandi per migliorare l'aspetto estetico delle nostre creazioni

Aggiungiamo ora al nostro arsenale di comandi base delle nuove istruzioni: **pencolor, penwidth** e **canvascolor.** Queste istruzioni sono molto semplici ma ci permettono di ottenere dei nuovi effetti grafici e migliorare l'aspetto esteriore del risultato.

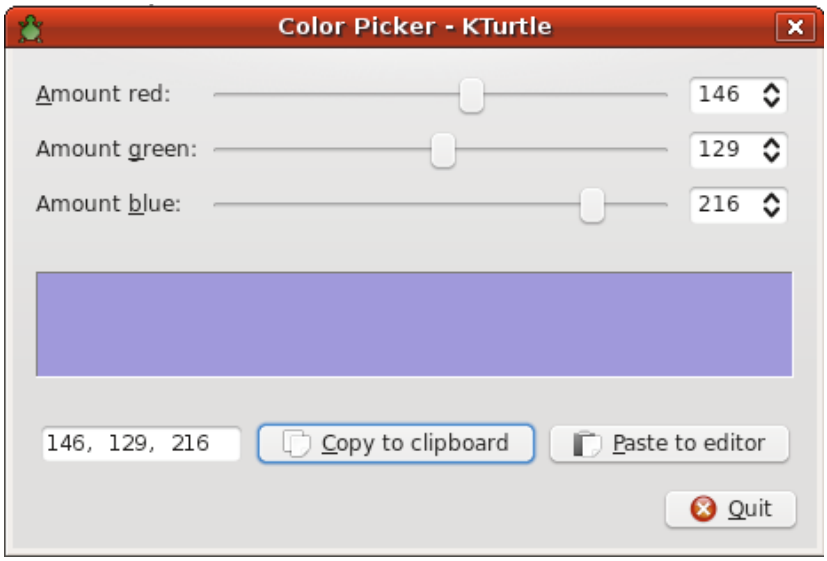
Pencolor e **penwidth** non disegnano nulla, nel senso che non hanno effetti immediati sulla tartaruga o sul canvas: permettono però di decidere i colori e la grandezza del tratto disegnato dalle istruzioni *successive*. La parola *successive* è molto importante, molti programmatori alle prime armi usano queste istruzioni alla fine del programma, o le usano da sole e si chedono come mai non succeda nulla: semplicemente bisognerà aspettare un comando di disegno (**forward**, per esempio) per osservare dei risultati!

penwidth decide la larghezza del tratto della penna. 1 è il valore di **default** (ossia il valore di penwidth quando non lo esplicitiamo, come abbiamo fatto finora), noi possiamo usare qualsiasi numero: attenzione però alle dimensioni dello schermo! Il tratto potrebbe essere così grande da ricoprire tutta la superficie visibile (*canvas*).

pencolor decide il colore del tratto disegnato dalla tartaruga. Qui la faccenda è più complicata, poichè per decidere il colore bisogna usare tre valori: **R**, **G** e **B**.

RGB sono i tre componenti di rosso (**R**ed), verde (**G**reen) e blu (**B**lue), che, mixati, danno il colore finale: un po' come succede per la luce, in cui ogni diversa radiazione (lunghezza d'onda) corrisponde al colore finale della luce stessa. Ogni componente **RGB** è un numero che va da 0 (scuro) a 255 (chiaro), e dunque per ogni componente ci sono 256 possibili valori.

Il valore di default è 0,0,0 (ossia il nero, assenza di colore). Il bianco corrisponde al massimo della luminosità di tutti e tre i componenti **RGB**, dunque a 255,255,255. Il rosso è 255,0,0. Il verde è 0,255,0. Il blu è 0,0,255. In mezzo, ci sono tutte le possibili combinazioni di questi tre valori (per chi è più matematico, queste combinazioni sono 256 x 256 x 256, ossia oltre 16 milioni di colori possibili!)



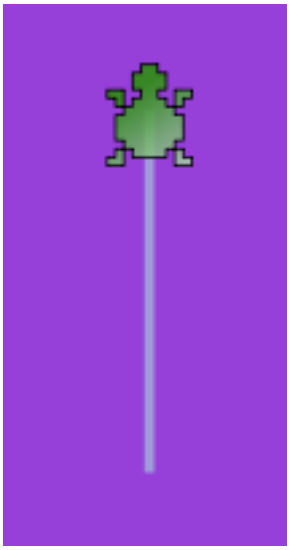
Nelle ultime versioni di kturtle è possibile anche "mischiare" insieme i colori ed ottenere il corrispondente **RGB** direttamente dall'interfaccia grafica: clickando su Tools -> Color chooser apparirà una schermata che permetterà di visualizzare immediatamente l'effetto del mix delle 3 componenti e di incollare i 3 valori **RGB** direttamente nell'editor del codice.

CORSO DI LOGO, Copyright (C) Ugo Landini

Canvascolor invece, a differenza di **pencolor** e **penwidth**, mostra immediatamente il colore **RGB** dello sfondo e non bisogna aspettare che la tartaruga disegni qualcosa per osservarne il risultato. Il valore di default di **canvascolor** è 255,255,255 (ossia il bianco)

```
1 reset
2 pw 3
3 pc 146, 129, 216
4 cc 146, 0, 216
5 fw 100
6
7
```

Facciamo un esempio che chiarirà meglio il tutto. Inserendo i seguenti comandi nell'editor otterremo una linea larga 3 pixel, colorata di un celeste chiaro, e su uno sfondo viola.



Osservate che **penwidth** è stato abbreviato in **pw, pencolor** in **pc** e **canvascolor** in **cc**.

Spunti e riflessioni:

- Rivedere gli esempi dei capitoli precedenti e colorarli
- Osservare che i tre valori **RGB** sono dei normali numeri, e quindi sono assegnabili a delle variabili e manipolabili come qualsiasi altra cosa in kturtle. (**pencolor** $r, $g-10, $b*2 per esempio)

Note di inglese:

pen = penna

color = colore (anche se in inglese-americano, poichè in inglese-inglese - british english - si scrive colour)

canvas = tela

width = larghezza

size = dimensioni

Per cui penwidth = larghezza della penna, pencolor = colore della penna, canvassize = dimensioni della tela, canvascolor = colore della tela e così via.

Il sistema delle coordinate

Il canvas su cui disegnamo è settato inizialmente (**default**) come un quadrato di 400 x 400 pixel.

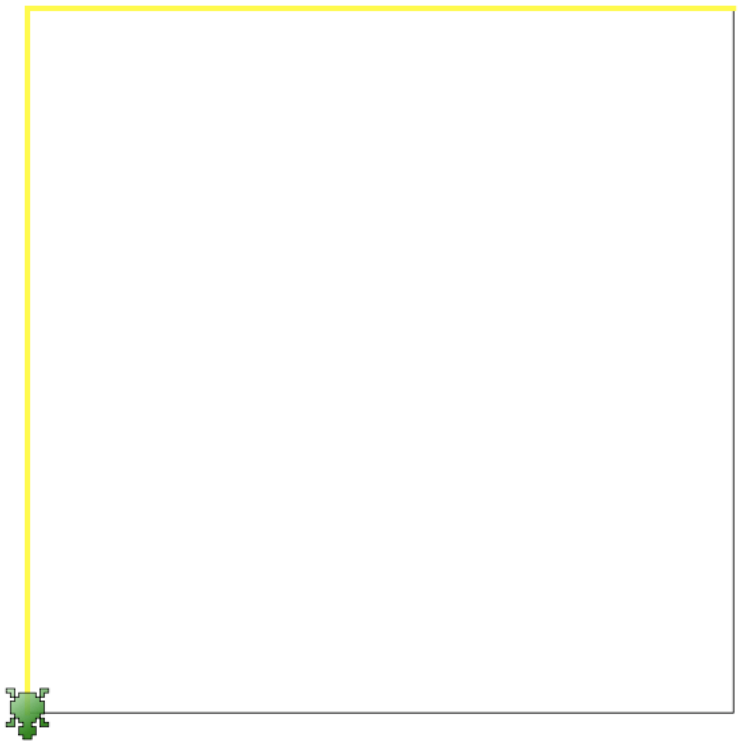
Ce lo possiamo immaginare come un megapiano di battaglia navale: per individuare un punto (pixel) sullo schermo bisognerà usare le sue due coordinate. Ad esempio 0,0 è l'origine, in alto a sinistra. 400,0 è l'angolo in alto a destra, 400,400 quello in basso a destra e 0,400 quello in basso a sinistra

```
reset

pw 3
pc 255,255,0
go 0,0
tr 90
fw 400

go 0,0
tr 90
fw 400

```



Proviamo ad usare la tartaruga per capire un po' meglio come funziona.

Il nuovo comando **go** fa si' che la tartaruga si sposti, *senza* disegnare, al punto indicato. **go** in inglese vuol dire "vai".

Questo programma, dopo aver allargato a 3 la dimensione del tratto e scelto il giallo come colore non fa altro che:

go 0,0

Sposta la tartaruga alle cordinate 0,0, ossia l'angolo in alto a sinistra.

tr 90

fw 400

Gira di 90 ° e disegna una riga di 400 pixel. Essendo il default del canvas 400x400, la riga arriva fino in fondo a destra.

Dopodichè ritorna al punto 0,0, rigira a destra (quindi ora è orientata verso il basso) e fa di nuovo una riga lunga 400 pixel. In questo modo abbiamo evidenziato in giallo i due assi.

E' chiaro che con **go** si può andare dove si vuole: (200,200) per esempio è il punto centrale dello schermo, (300,100) è un punto in alto a destra, (234, 121) uno più o meno al centro ma un po' in alto, e così via. I matematici chiamano la prima coordinata **x**, e la seconda **y**. Quindi **x** è la coordinata sull'"orizzontale", mentre **y** è la coordinata sul "verticale"

Conoscendo la formula dell'area del quadrato, possiamo anche calcolare quanti punti ci sono nel canvas di default (400 x 400 = 16.000)!

Approfittiamo per introdurre l'istruzione **center,** che non fa altro che posizionare la tartaruga al centro dello schermo. Sul canvas di default, è dunque equivalente a go 200,200. Ovviamente su un canvas di dimensione diversa, center va sempre al centro, go 200,200 invece no! Center in inglese vuol dire "centro", ovviamente, anche se in inglese britannico si scrive *centre*, come sono complicati questi anglosassoni!

Ma come si fa a creare un canvas di dimensione diversa da 400x400, per esempio di 800x600 pixel? Presto detto, basta usare l'istruzione canvassize 800,600 e avremo a disposizione uno spazio più grande (monitor permettendo!)

Possiamo anche farci "dire" dalla tartaruga qual'è la sua posizione sullo schermo. Per fare questo, introduciamo ben tre nuove istruzioni:

getx

chiede alla tartaruga la coordinata x (l'asse orizzontale) (*get x*, ossia *dammi* la *x*)

gety

chiede alla tartaruga la coordinata y (l'asse verticale) (*get y*, ossia *dammi* la *y*)

print "ciao ciao"

stampa sullo schermo la scritta "ciao ciao" . "Print" in inglese vuol dire proprio stampa.

print 5

stampa sullo schermo il numero 5

e dunque

print getx

stampa sullo schermo la posizione x della tartaruga: dopo un reset, sul canvas di default, la tartaruga è al centro, quindi stamperà 200. Torneremo su questa utile istruzione nei prossimi capitoli

Spunti e riflessioni:

- Rivedere gli esempi dei capitoli precedenti e provarli in canvas di dimensioni diverse. Cosa cambia?
- go 0,0 (o anche go 400,400) è un ottimo modo per spostare la tartaruga quando copre una parte importante del nostro disegno
- provare a stampare le coordinate della tartaruga dopo alcune istruzioni di movimento: sono le coordinate che vi aspettavate?

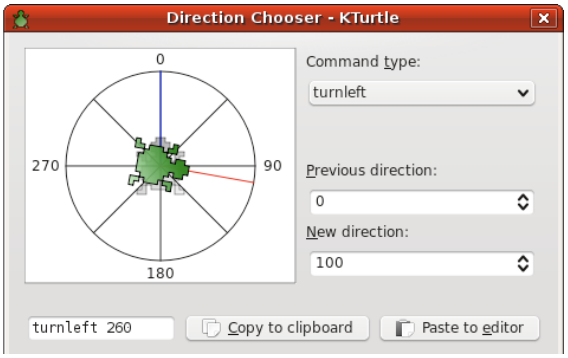
Disegnamo un labirinto

Per disegnare un labirinto, utilizzeremo essenzialmente l'istruzione **random**. Sarà un'occasione però anche per introdurre tre nuovi comandi: **penup, pendown** e **direction**.

penup dice alla tartaruga di "alzare" la penna (up = su). Immaginando che la tartaruga disegni con la sua coda che striscia sul foglio, è facile immaginare cosa faccia questo comando: "alzando" la penna la tartaruga non disegnerà più, e anche eseguendo un comando come forward si otterrà solo uno spostamento e non un tratto sul canvas.

pendown ritorna invece alla situazione "normale" o di **default**, quella in cui la tartaruga, muovendosi, lascia un tratto sul foglio. (down = giù)

direction serve per far puntare una direzione *assoluta* e non *relativa* alla tartaruga (direction = direzione). turnright e turnleft sono comandi relativi, nel senso che la tartaruga si sposta per esempio di 90° partendo dalla posizione in cui è in quel momento. direction 90 invece ruota la tartaruga di 90° partendo da una posizione fissa e assoluta in cui 0 è il nord. Nelle ultime versioni di Kturtle è possibile utilizzare il comodo "Direction Chooser" (Tools -> Direction Chooser) per scegliere graficamente l'angolo: utile per mostrare ai bambini visivamente il concetto.



random invece ci permette di far "pensare" alla tartaruga un numero caso, compreso fra due estremi. Ad esempio la seguente istruzione:

```
$a = random 10,30
```

assegna alla variabile $a un valore qualsiasi, compreso fra 10 e 30.

```
$a = random 0,100
```

assegna invece ad $a un valore compreso fra 0 e 100.

Questa istruzione ci permette di inserire un po' di *casualità* nei nostri programmi: non è un caso che in inglese *random* voglia dire proprio *casuale*!

Attenzione che il valore "pensato" da random non è un numero intero: per alcuni comandi grafici questo potrebbe non essere un problema, poichè semplicemente la parte frazionaria di un pixel verrà ignorata (**forward** 10 o 10.2345 sono equivalenti, poichè il pixel è indivisibile) ma per altri comandi di certo non è così:

```
$a = random 10,30
print $a
```

produrrà qualcosa come 23.687687!

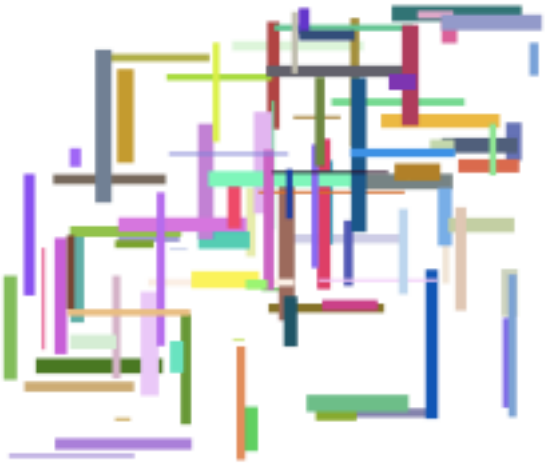
Se si vuole arrotondare il numero all'intero più vicino, si deve usare l'istruzione **round**:

```
$a = round random 10,30
print $a
```

```
1 reset
2
3 repeat 100 {
4
5  $x = random 1, 100
6  fw $x
7  tr 90
8
9 }
```

La più semplice versione di "labirinto" è mostrata in figura.

E' un semplice ciclo repeat di righe di lunghezza casuale: per i bambini è molto divertente osservare l'esecuzione del programma, "scommettendo" se riesce o non riesce a rimanere nei limiti del canvas.

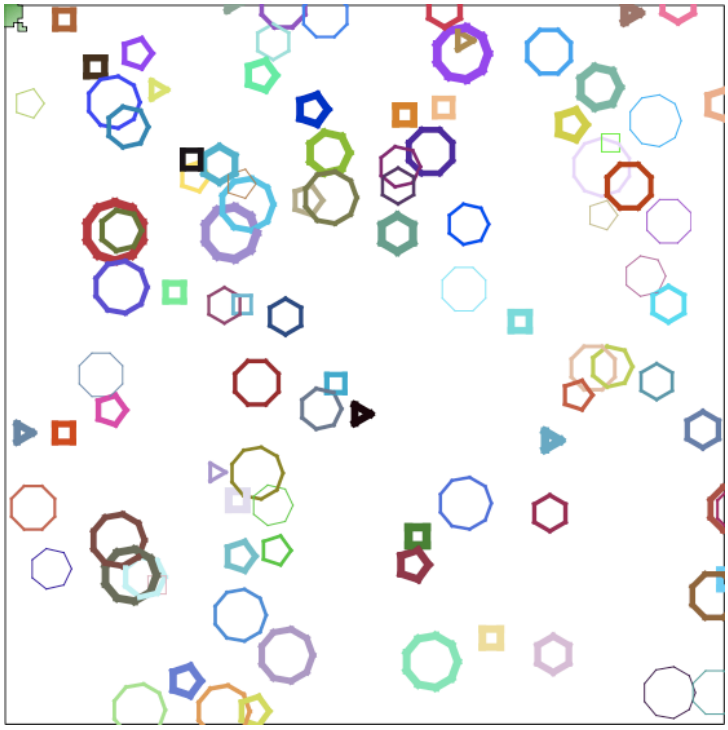


```
reset

repeat 100 {
 $x = random 1,40
 $y = random 1,40
 $w = random 1,5
 $r = random 0,255
 $g = random 0,255
 $b = random 0,255
 pw $w
 pc $r, $g, $b
 fw $x
 tr 90
 penup
 fw $y
 pendown
}
```

Una versione solo leggermente più sofisticata mostra invece come sfruttare random e penup per un effetto "artistico" simile ad un quadro astratto. La struttura del programma è identica al precedente: si estraggono casualmente, oltre alla distanza del segmento da disegnare ($x) , anche un segmento "invisibile" ottenuto con penup e pendown ($y), che le dimensioni del tratto ($w) ed i tre colori $r, $g, $b.

Esercizi



- Scrivere un programma che disegni sullo schermo 100 figure, ispirandosi alla figura qui accanto, con le seguenti caratteristiche
    - colore e spessore delle righe casuali
    - coordinate casuali (suggerimento: usare go)
    - numero dei lati casuale (suggerimento: il numero di lati può essere compreso fra 3 e 10, ed usare learn)
- Riscrivere il programma del labirinto utilizzando direction invece di turnright ed estraendo casualmente con random la direzione che la tartaruga deve prendere.

Disegnamo una spirale

Ora che abbiamo una certa padronanza dell'uso dei colori e delle variabili, possiamo provare ad ottenere una spirale con colore sfumato come quella mostrata in apertura di capitolo.

Per fare questo esercizio introdurremo anche un nuovo tipo di ciclo, simile al repeat ma leggermente più potente, il ciclo for.

Il ciclo for si differenzia dal repeat perchè è in grado di farci sapere ad ogni iterazione (ossia ogni volta che ripete, che "fa un giro") il valore di una variabile associata al ciclo stesso.

Prendiamo ad esempio il programma in figura, che scrive 10 volte sullo schermo "Ciao!".

```
reset
$y=100
repeat 10 {
 go 100, $y
 print "Ciao!"
 $y = $y + 20
}
```

Abbiamo dovuto dichiarare una variabile $y all'esterno del repeat, ed esplicitamente effettuare una somma per spostare la coordinata. Con un ciclo for è molto più semplice realizzare la stessa cosa.

```
reset

for $y=100 to 300 step 20 {
 go 100, $y
 print "Ciao!"

}
```

Ecco come il programma si trasforma sostituendo il repeat con il for. La dichiarazione della variabile $y ora è implicita nel ciclo. Si dichiara che $y andrà da 100 (valore iniziale) a 300 (valore finale), incrementandosi di 20 ogni volta (lo step)

Se non si indica esplicitamente lo step, il ciclo for incrementerà di 1 ad ogni iterazione. Dovrebbe quindi essere evidente che, quando non serve solo ripetere un numero n di volte, ma anche usare il numero di iterazione (per esempio per qualche calcolo o per visualizzazione), è sì possibile usare il repeat e fare tutto a manina ma il for ci fa trovare già un bel po' di lavoro fatto! Il for (che in inglese vuol dire "per") si può "leggere" in questo modo: "per $y che va da 100 a 300 ..."

E ora, sfruttiamo il ciclo for appena imparato per disegnare finalmente la nostra spirale!

Il piano per creare una spirale è questo: si disegna un pezzo di cerchio (ormai siamo capaci), poi si "stringe" il raggio e se ne disegna un altro pezzo, e così via. Il cerchio non si chiude mai, poichè il raggio continua a stringersi, ed avremo un bell'effetto spirale. Ovviamente è possibile anche disegnarla al contrario, partendo dal centro ed aumentando il raggio pian piano.

E' possibile disegnare spirali facendo mezzo cerchio alla volta, ma anche un quarto di cerchio od altri valori: provando diversi numeri è possibile ottenere molti effetti esteticamente gradevoli.

Analizziamo insieme il programmino della spirale.

La procedura arco non è altro che una variante del cerchio o del poligono già sviluppata nei capitoli precedenti, con l'unica differenza di utilizzare un numero predefinito di lati (20) e fare mezzo giro (180°).

Il cuore della spirale è nel ciclo for. Vengono in questo caso disegnati 16 mezzi cerchi: la variabile $x infatti assume il valore iniziale 1 e poi cresce di uno in uno (infatti non c'è step) fino a 16. (si legge "per x che va da 1 a 16")

Il ciclo viene dunque ripetuto per 16 volte (attenzione: se fosse partito da zero sarebbe stato dunque ripetuto 17 volte!), ed ogni volta viene disegnato un arco di lunghezza crescente. Per ottenere un effetto grafico esteticamente gradevole, ogni mezzo arco è disegnato con un verde via via più scuro. Questo è ottenuto diminuendo ogni volta il valore 255 nella componente GREEN di RGB di 8 volte ($x * 8), lasciando invece invariate le due componenti di rosso e blu.

```
1 reset
2 learn arco $l {
3 $r = 180/20
4  repeat 20 {
5   fw $l
6   tr $r
7  }
8 }
9
10 pw 3
11 for $x = 1 to 16 {
12  pc 0,255 - $x * 8, 0
13  arco $x
14 }
```

Giocando con $x, il valore 8 che lo moltiplica, 180 (lunghezza dell'arco) e 20 (numero dei "lati" dell'arco di cerchio) è possibile ottenere molte spirali graficamente diverse.

Esercizi

- Disegnare una spirale "restringendo" il raggio, ossia partendo dall'arco di cerchio più grande.
- Disegnare una spirale fatta di triangoli via via crescenti.