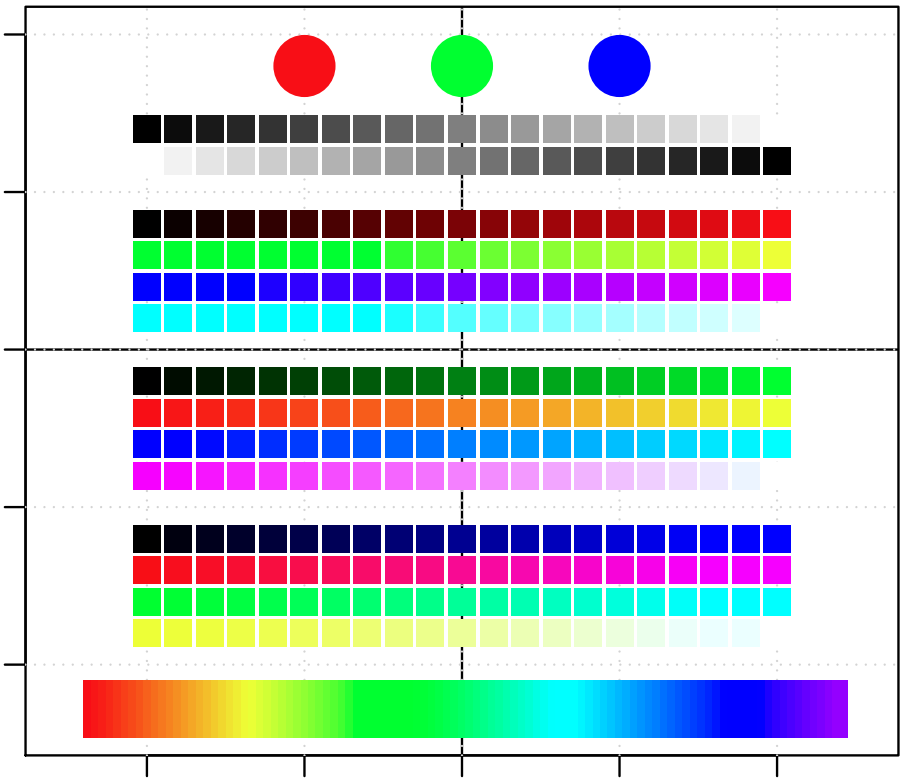


Figura 2.7: Scale di grigi e tavolozza di colori ottenuta con una piccola variante dello script descritto nel testo (a cui sono state aggiunte le istruzioni per produrre l'arcobaleno, vedi nota 10).

fra due colori mediante la funzione `colorRampPalette()`. Ad esempio

```
> pal <- colorRampPalette(c("yellow","red"))
```

ci crea la funzione[11] `pal()` la quale produce una sequenza di `n` colori dal giallo a rosso, ove `n` è il parametro con cui viene chiamata. Ad esempio

```
> pal(7)
[1] #FFFF00 #FFD400 #FFAA00 #FF7F00 #FF5500 #FF2A00 #FF0000
```

Questi codici risultanti, apparentemente strani, sono numeri in rappresentazione esadecimale e sono particolarmente comodi per rappresentare in modo compatto i codici RGB. Innanzitutto notiamo come gli 'FF' iniziali e gli zeri finali restano costanti, mentre cambiano i due valori centrali che vanno da 'FF', in corrispondenza del giallo (ovvero rosso più verde), a '00' in corrispondenza del rosso. Le cifre del sistema esadecimale sono 16 e sono rappresentate dalle normali cifre del sistema decimale a cui si aggiungono le prime lettere dell'alfabeto. Quindi: 0, 1, . . . , 9, a, b, c, d, e , f. (in genere vengono usate indifferentemente lettere minuscole e maiuscole). Quindi 'F' corrisponde a 15 e 'F0' 15 volte la base (ovvero $15 \times 16 = 240$). Ne segue che 'FF' vale $15 \times 16 + 15 = 255$. Infatti, come con due cifre decimali si possono rappresentare 100 numeri diversi ($10^2 = 100$), compreso lo zero, in modo analogo con due cifre esadecimali si possono rappresentare $16^2 = 256$ numeri, da 0 a 255. Per esercizio, calcoliamo l'equivalente dei valori intermedi dei livelli del verde, da D4 a 2A:

[11] In effetti, abbiamo un caso di una funzione [`colorRampPalette()`] che come risultato dà una funzione!

```
> str(pal)
function (n)  x <- ramp(seq.int(0, 1, length.out = n)) rgb(x[, 1], x[, 2], x[, 3],
maxColorValue = 255)  <bytecode: 0x9c62308> <environment: 0x9e4ac9c>
```

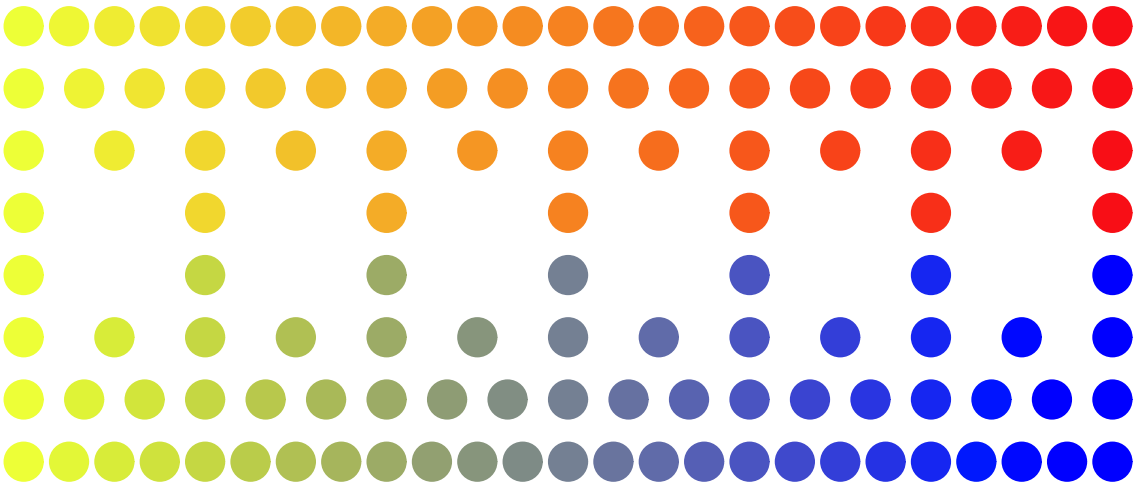


Figura 2.8: Esempi di tavolozze personalizzate ottenute interpolando fra due colori.

```
> 13*16 + 4
[1] 212
> 10*16 + 10
[1] 170
> 7*16+15
[1] 127
> 5*16 + 5
[1] 85
> 2*16 + 10
[1] 42
```

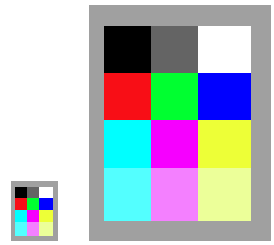
Per vedere questa tavolozza personalizzata all'opera, oltre a una che va dal giallo al blu, usiamo il seguente script, il cui risultato è mostrato in figura 2.8:[12]

```
pal <- colorRampPalette(c("yellow","red"))
n=7; plot(0:(n-1), rep(0,n), ylim=c(-1,1), axes=FALSE,
          xlab='', ylab='', pch=20, col=pal(n), cex=5)
n=13; points(0:(n-1)/2, rep(0.2,n), ylim=c(-1,1), pch=20, col=pal(n), cex=5)
n=19; points(0:(n-1)/3, rep(0.4,n), ylim=c(-1,1), pch=20, col=pal(n), cex=5)
n=25; points(0:(n-1)/4, rep(0.6,n), ylim=c(-1,1), pch=20, col=pal(n), cex=5)

pal <- colorRampPalette(c("yellow","blue"))
n=7; points(0:(n-1), rep(-0.2,n), pch=20, col=pal(n), cex=5)
n=13; points(0:(n-1)/2, rep(-0.4,n), ylim=c(-1,1), pch=20, col=pal(n), cex=5)
n=19; points(0:(n-1)/3, rep(-0.6,n), ylim=c(-1,1), pch=20, col=pal(n), cex=5)
n=25; points(0:(n-1)/4, rep(-0.8,n), ylim=c(-1,1), pch=20, col=pal(n), cex=5)
```

Chi è interessato a tavolozze di vario tipo può dare un'occhiata al pacchetto di R `RColorBrewer`, sul quale non ci soffermiamo oltre.

[12] Chi è appassionato di grafica e ha a disposizione un programma che 'sonda' i colori sullo schermo (ad esempio *KColorChooser* di *KDE* Linux) può controllare che il codice colore rilevato (in *KColorChooser* è indicato con codice 'HTML') corrisponde esattamente alla rappresentazione esadecimale che abbiamo visto nel testo. (Nota: una parte dei programmi di *KDE*, fra cui *KColorChooser*, è disponibile anche per Windows grazie al progetto *KDE on Windows*, descritto in `http://techbase.kde.org/Projects/KDE_on_Windows`.)



(000,000,000) (100,100,100) (255,255,255)
(255,000,000) (000,255,000) (000,000,255)
(000,255,255) (255,000,255) (255,255,000)
(128,255,255) (255,128,255) (255,255,128)
[cornice: (160,160,160]

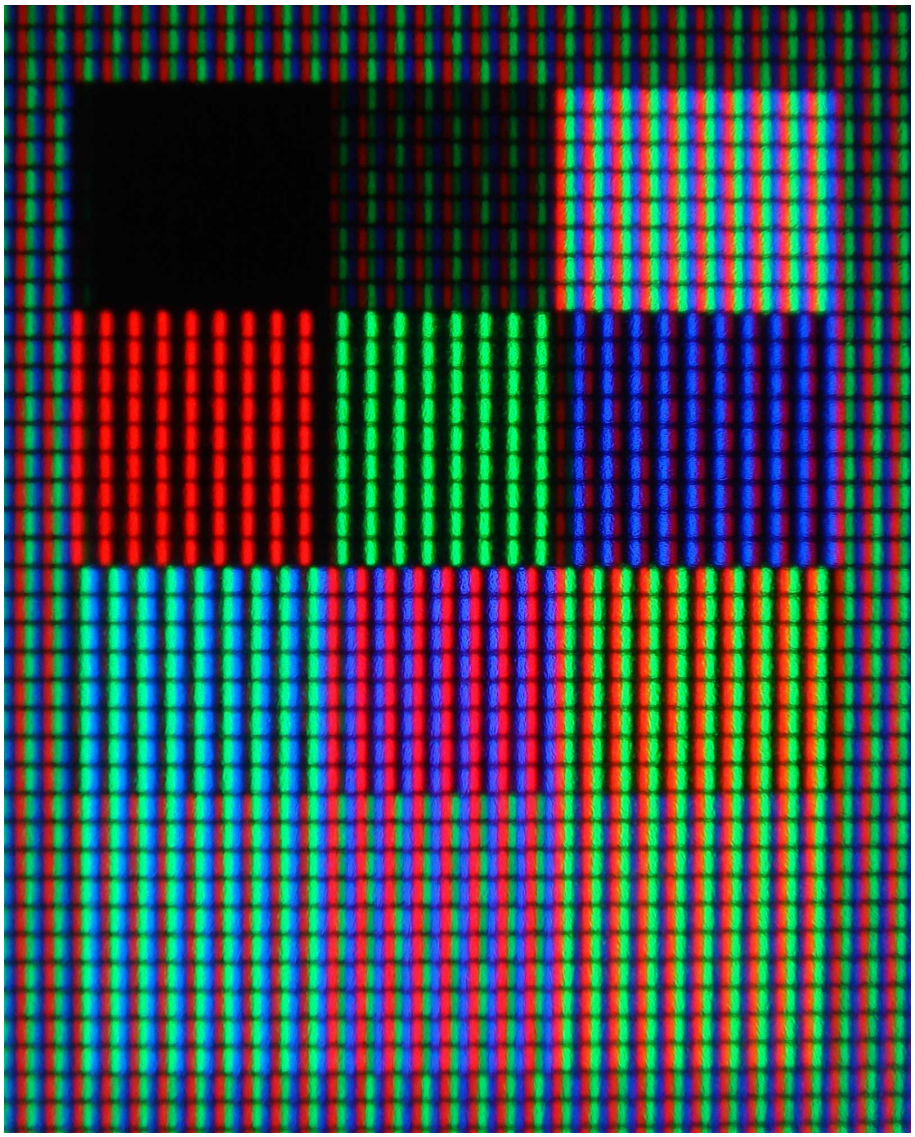


Figura 2.9: Ingrandimento di una piccola porzione dello schermo (32×40 pixel – si possono contare!) corrispondente all'immaginetta piccola in alto a sinistra quando è vista con le dimensioni normali ('100%'). L'immagine alla sua destra serve solo per mostrare meglio i colori. La tabella in alto riporta il codice RGB delle diverse parti dell'immagine.

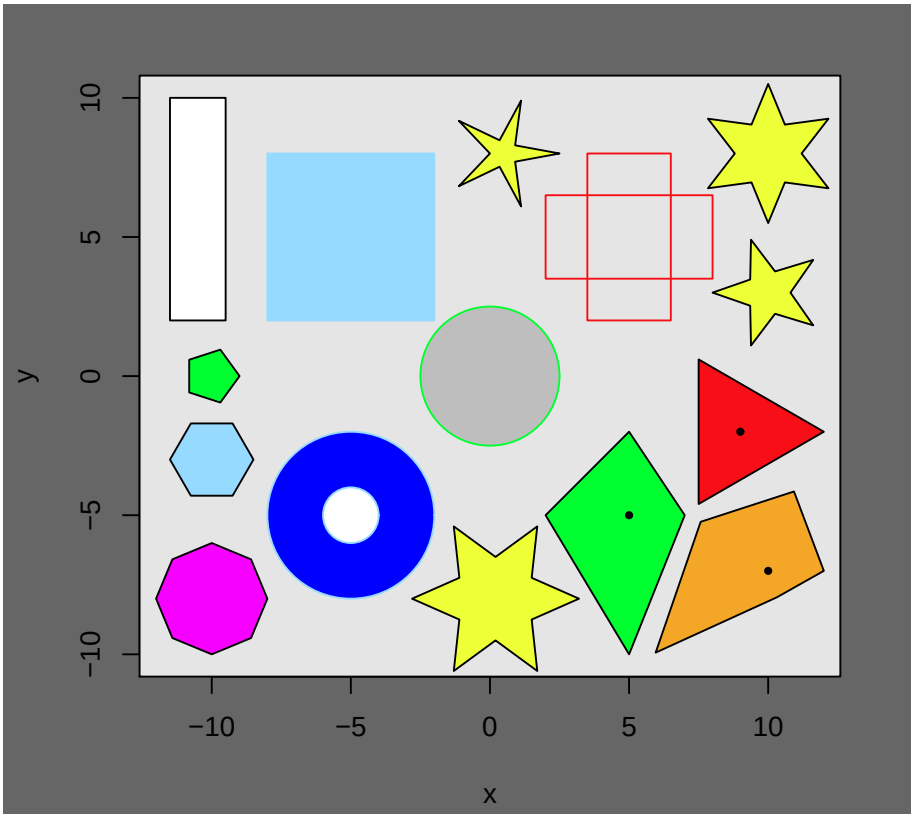


Figura 2.10: Cerchi, rettangoli, quadrati e ‘stelle’ (in senso lato) ottenuti con le funzioni `rect` e `symbols`.

Proseguiamo con altre prove: una bandiera italiana, ottenuta con tre rettangoli affiancati e un rettangolo riempito con tratteggio. Ecco il resto dello script, il cui risultato complessivo è mostrato in figura 2.11.

```
x = c(0,0,1/3,1/3,0)-2+1
y = c(1.5,2,2,1.5,1.5)-1+1
polygon(x,y, col='green')
polygon(x+1/3,y, col='white')
polygon(x+2/3,y, col='red')

x = c(1,2,2,1,1)-0.5
y = c(1,1,0.5,0.5,0.5)+0.5
polygon(x,y, density=20, col='gold')
```

2.6 Figure trasparenti

Finora è stato sottinteso che qualsiasi oggetto grafico si aggiungesse sulla finestra grafica, esso andasse a sovrapporsi su un eventuale oggetto precedente. (vedi ad esempio la serie di simboli in basso di figura 2.1). A volte può essere interessante poter sovrapporre una figura all’altra, potendo però vedere anche quella sotto. Affinché ciò sia possibile i vari oggetti grafici devono avere un certo grado di *trasparenza*. Ciò è possibile in R giocando sul quarto parametro di `rgb()` che a suo

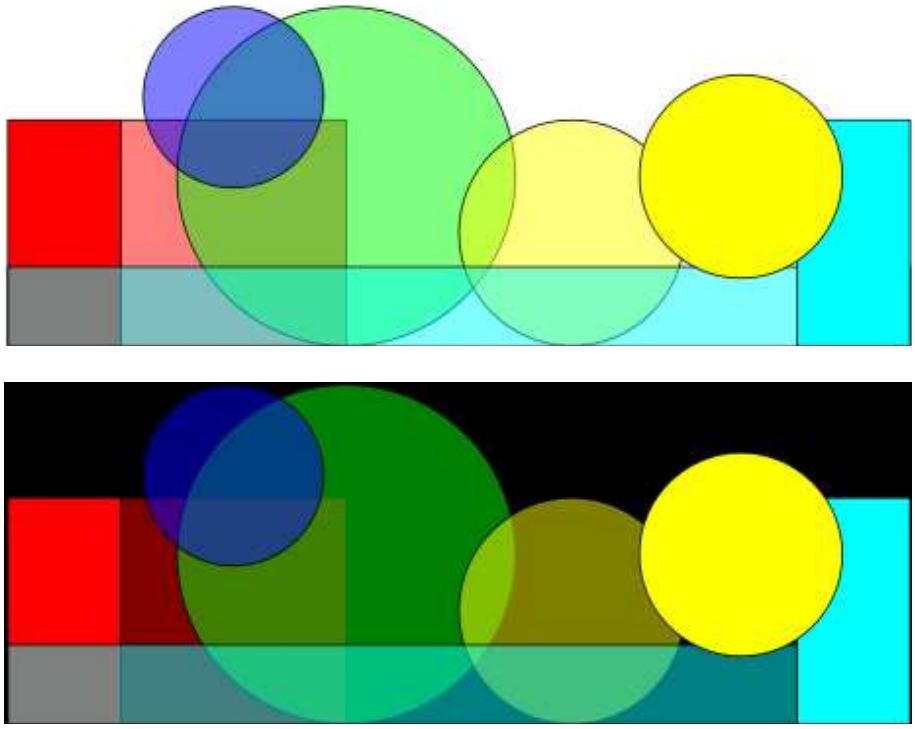


Figura 2.12: Esempio dell'uso del quarto parametro di `rgb()` che permette di disegnare oggetti trasparenti.

2.7 Annotazioni sui plot

Abbiamo già incontrato nel paragrafo 2.1 la funzione `text()` per porre dei semplici caratteri alfanumerici in un plot. Vediamo ora delle varianti, in particolare lettere greche, simboli matematici e altro, come mostrato in figura 2.13. Siccome le possibilità di grafica di R vanno ben al di là dell'intento di questo testo, diamo direttamente lo script con il quale è stata prodotta la figura, invitando chi è interessato ad approfondire.

```
foglio.bianco(tit='')
text(-5,5,' orizzontale',pos=4, col='gold')
text(-5,5,' ruotato di 45 gradi',srt=45, pos=4, col='gold')
text(-5,5,' ruotato di 90 gradi',srt=90, pos=4, col='gold')
text(-5,5,' ruotato di 135 gradi',srt=135, pos=4, col='gold')
text(-5,5,' ruotato di 180 gradi',srt=180, pos=4, col='gold')

text(-5,4,'orizzontale',pos=1, col='lightblue',cex=2)
text(-5,4,'capovolto',srt=180, pos=3, col='lightblue',cex=2)

text(-7.5,1.5,expression(alpha+beta+delta+gamma), cex=1.5,col='magenta')
text(-7.5,1.5,expression(alpha+beta+delta+gamma), cex=1.5,col='magenta')

text(-2.5,1.5,expression(sqrt(x^2+y^2)), cex=1.5,col='blue')

text(2.5,7.5,expression(frac(x,sqrt(x^2+y^2))), cex=1.5,col='red')
text(7.5,7.5,expression(integral(f(x)*dx, a, b)), cex=2,col='gray')

text(5, 2.5, expression(paste(frac(1, sigma*sqrt(2*pi)), " ",
```

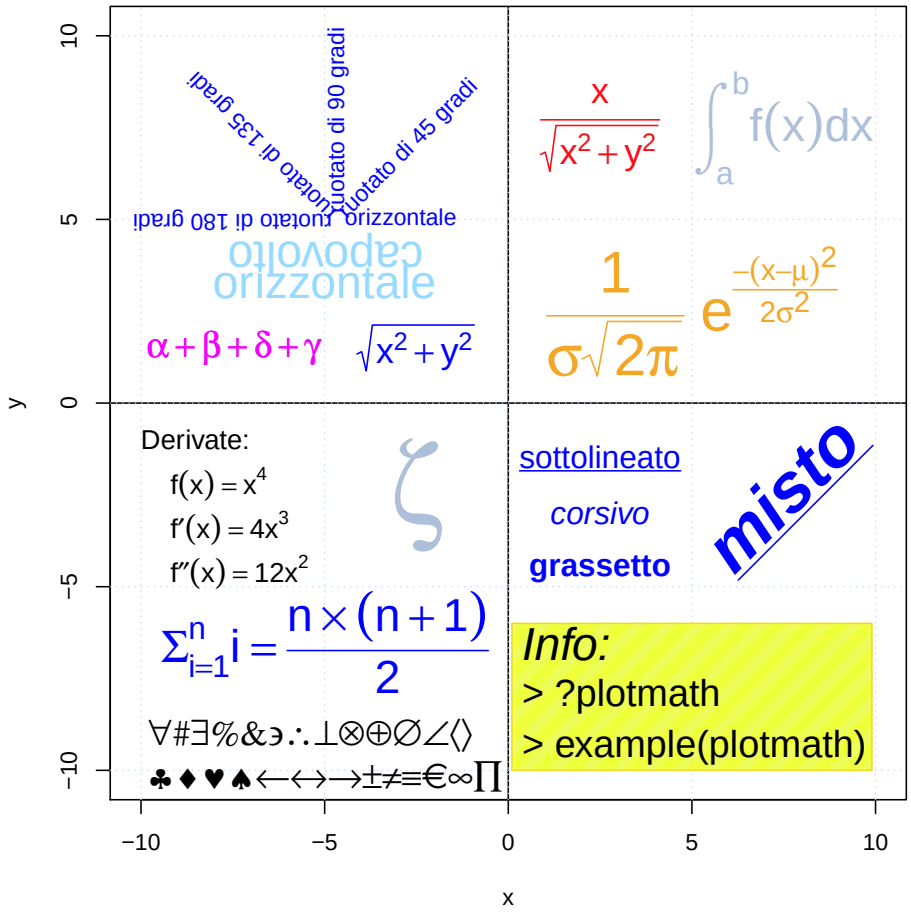


Figura 2.13: Alcune delle possibilità di annotazioni ‘speciali’ in R.

```
                          plain(e)^{frac(-(x-mu)^2, 2*sigma^2)})),
     cex = 2.5, col='orange')

text(-10, -1,  "Derivate:", adj = 0,cex=1.2)
text(-9.2, -2.5, expression(f*minute(x^2) == 2 * x), adj = 0, cex=1.2)
text(-9.2, -4, expression(f*second (x^4) == 12 * x^2), adj = 0, cex=1.2)

text(-2.5, -2.5, expression(zeta), cex=5, col='gray')

text(2.5, -1.5, expression(underline("sottolineato")), cex=1.4, col='blue')
text(2.5, -3, expression(italic("corsivo")), cex=1.4, col='blue')
text(2.5, -4.5, expression(bold("grassetto")), cex=1.4, col='blue')
text(7.5, -2.5, expression(bolditalic(underline("misto"))), cex=3, srt=45, col='blue')

text(-5, -6.5, expression(Sigma[i==1]^n*i == over(n%*%(n+1),2)), cex=2, col='blue')

text(-10, -9.2, expression(symbol("\042")*symbol("\043")*symbol("\044")
    *symbol("\045") *symbol("\046")*symbol("\047") *symbol("\134")
    *symbol("\136") *symbol("\304")*symbol("\305") *symbol("\306")
    *symbol("\320") *symbol("\341")*symbol("\361")), cex=1.5, pos=4)
```

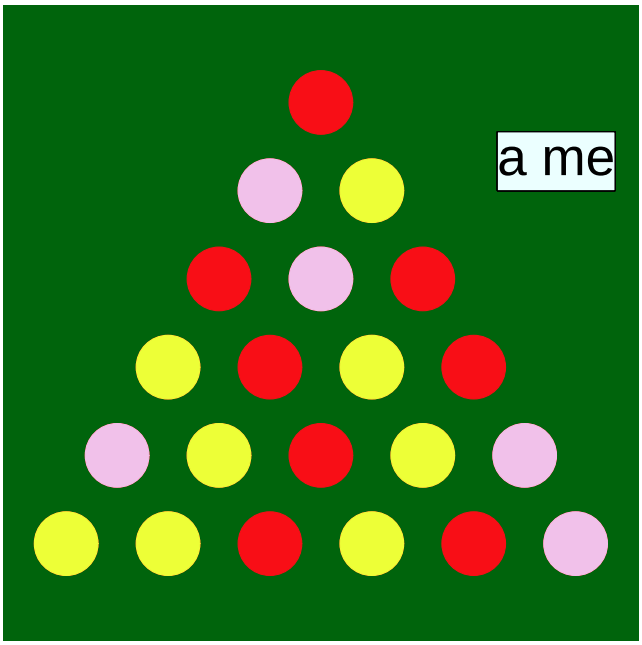


Figura 10.5: Interfaccia grafica al gioco delle 21 palline.

```
  dh <- L6/5           #     ''     orizzontale
  p <- matrix(1:42, 21, 2)
  np <- 0
  for (v in 0:5) { # loop verticale dal basso all'alto
    y <- -a6 + v * dv
    for (h in 0:(5-v)) {
      x <- -L6/2 + dh/2*v + h*dh
      np <- np + 1
      p[np,1] <- x
      p[np,2] <- y
    }
  }
  return(p)
}

inizia.grafica <- function(p, col, bg) {
  if(!is.null(dev.list())) lapply(as.numeric(dev.list()), dev.off)
  dev.new(width=6, height=6)
  old.mar = par("mar")
  par(mar=c(0., 0., 0., 0.))
  par(bg=bg)
  plot(NULL, xlim=c(-10., 10.)  , ylim=c(-5, 10), axes=FALSE,
       xlab='', ylab='', asp=1)
  cambia.colore(p[,1], p[,2],  col=col)
  par(mar=old.mar)
}

cambia.colore <- function(x, y, col) {
  points(x, y, pch=19, cex=8, col=col)
```

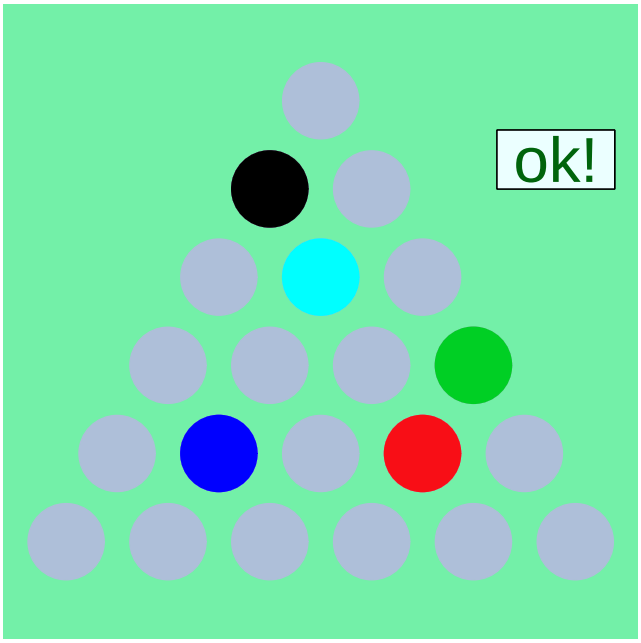


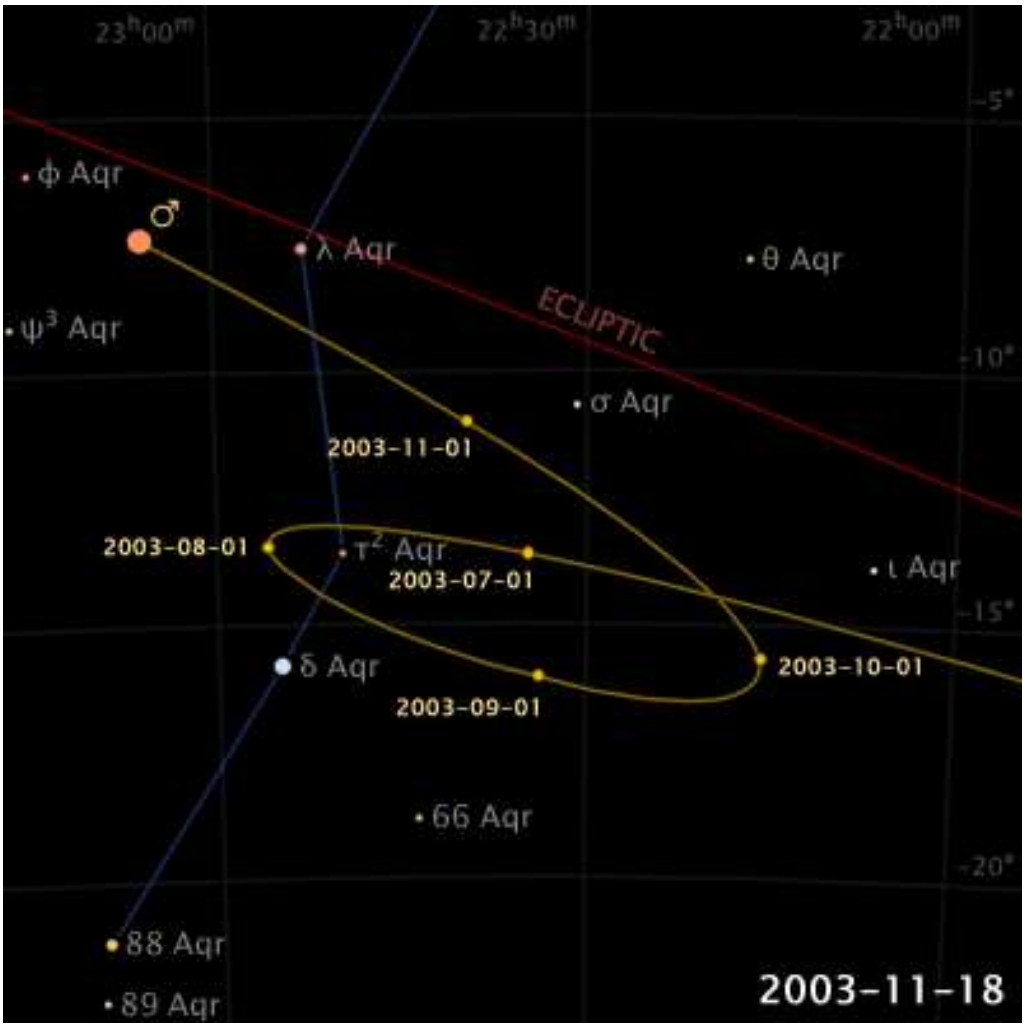
Figura 10.6: Grafica del gioco strategico delle 21 palline trasformato in un gioco di memoria (vedi testo). Ecco la situazione in cui il giocatore ha ricostruito con successo una sequenza di cinque palline

```
        colori <- sample( rainbow(balls) ) # arcobaleno
      )


if(balls > length(colori)) colori <- rep(colori, ceiling(balls/length(colori)))

ind <- sample(1:21, balls)       # palline scelte
messaggio('via!', col='blue')
for(i in 1:balls) {
  cambia.colore(p[ind[i],1], p[ind[i],2], col=colori[i])
  Sys.sleep(dt)
  cambia.colore(p[ind[i],1], p[ind[i],2], col='gray')
}
messaggio(sprintf("%d", balls))

nc <- 0
ok <- fine <- FALSE
repeat {
  l <- locator(1)
  if( (l$x > 6 & l$x < 10) & (l$y > 7 & l$y < 9)) {
    messaggio('fine', 'red')
    fine <- TRUE
    break
  } else {
    # cerchiamo la pallina beccata
    cand <- which( (sqrt((l$x - p[,1])^2 + (l$y - p[,2])^2) < 1 ) & vive, TRUE)
    if(length(cand) == 0) next # è andata a vuoto: clicca ancora
    vive[cand] = FALSE
```



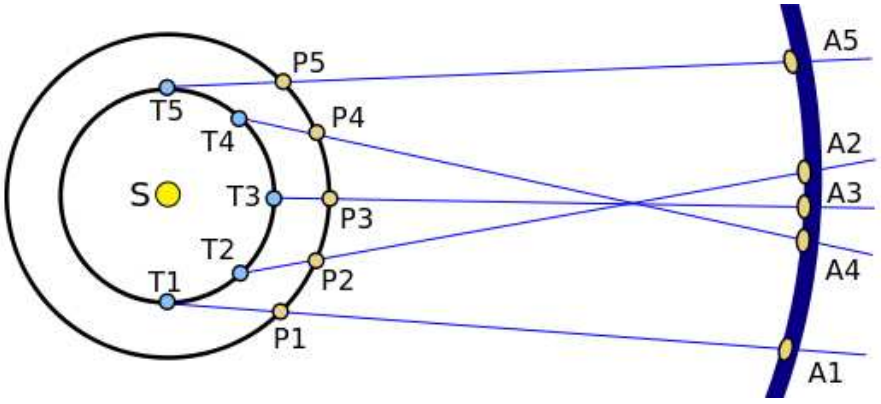


Figura 11.15: Moto apparente di Marte visto dalla Terra (figure da `http://en.wikipedia.org/wiki/Apparent_retrograde_motion`, ove la prima è disponibile come gif animata).