

## Quesiti da svolgere esclusivamente su carta

### Mini-robot

*Difficoltà: facile*

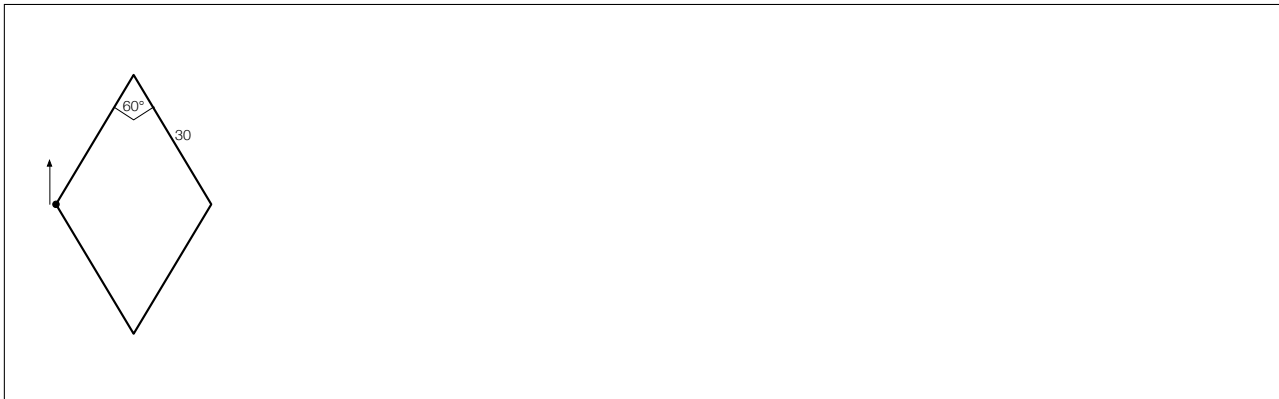
Abbiamo un mini-robot in grado di eseguire questi comandi:

- $V$ : avanza di un passo;
- $S(g)$ : ruota a sinistra di un angolo di  $g$  gradi;
- $D(g)$ : ruota a destra di un angolo di  $g$  gradi.

Più comandi in sequenza sono separati dal simbolo  $+$ , mentre la ripetizione per  $n$  volte di un comando o una sequenza  $C$  di comandi può essere concisamente indicata da  $n * (C)$ . Ad esempio:

- la sequenza  $V + S(30) + 5 * (V) + D(45)$  significa: (dalla posizione in cui ti trovi e nel verso in cui guardi) avanza di un passo, poi girati di 30 gradi a sinistra, poi avanza di 5 passi, infine girati di 45 gradi a destra;
- il comando  $3 * (10 * (V) + D(60))$  significa: ripeti per 3 volte queste due azioni: avanza di 10 passi e poi girati di 60 gradi a destra.

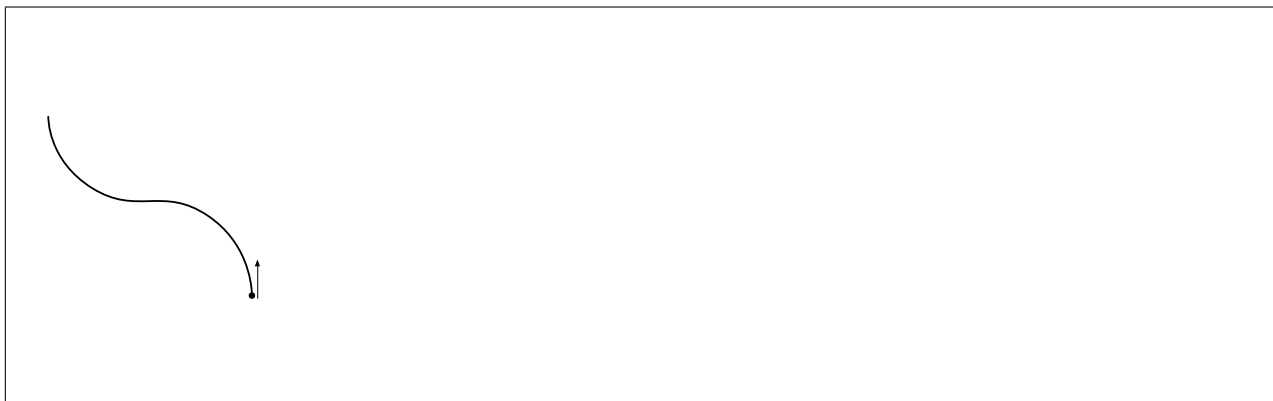
Se il mini-robot si trova rivolto verso nord nella posizione indicata dal pallino nero nella figura qui sotto, quali comandi gli faranno percorrere la traiettoria romboidale, dove il lato del rombo è percorso in 30 passi? Attenzione: alla fine il mini-robot dovrà ritrovarsi nella stessa precisa posizione di partenza (rivolto a nord).



Se il mini-robot riceve il comando  $180*(V+S(1))$ , percorrerà una traiettoria "semicircolare", come indicato nella figura qui sotto, e alla fine sarà rivolto verso sud.



Quali comandi faranno invece percorrere al mini-robot la traiettoria sotto indicata?

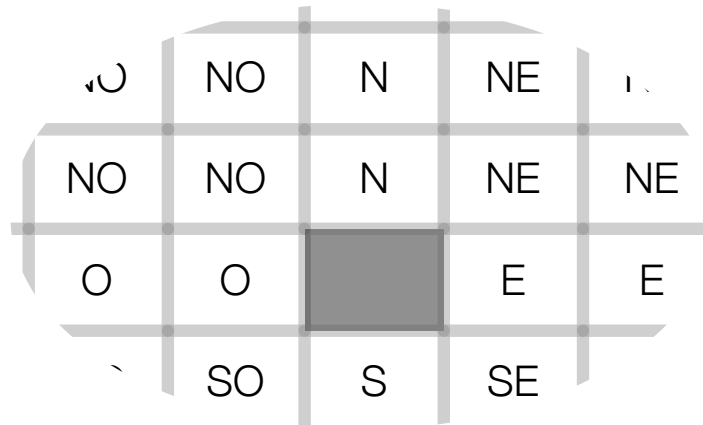


## Caccia al tesoro

*Difficoltà: media*

C'è un tesoro nascosto da qualche parte in un'area quadrata, suddivisa in quadrati piú piccoli, come una scacchiera con  $n$  caselle per lato. Ad ogni vostro tentativo, voi indicate una casella e ottenete una tra queste nove risposte: Ok, N, O, S, E, NO, SO, SE, NE.

La risposta Ok significa che il tesoro è nascosto proprio nella casella da voi indicata; la risposta N significa che il tesoro si trova in una delle caselle a Nord, sulla stessa colonna di quella da voi indicata; la risposta O significa che il tesoro si trova in una delle caselle a Ovest, sulla stessa riga di quella da voi indicata; la risposta NO significa che il tesoro si trova in una delle caselle piú a Nord e piú a Ovest di quella da voi indicata. Analogamente per le altre risposte, dove S sta per Sud ed E sta per Est. Per maggior chiarezza, nella figura qui sotto è disegnata una porzione dell'area quadrata e voi avete indicato la casella in grigio; in ciascuna casella è riportata la risposta che voi otterreste se il tesoro fosse nascosto proprio in quella casella. (Naturalmente se il tesoro fosse nella casella in grigio, la risposta ottenuta sarebbe Ok.)



Si vuol sapere qual è la dimensione massima dell'area quadrata (in numero di caselle per lato) affinché possiate avere la certezza di indovinare dove si trova il tesoro al piú al quarto tentativo. Giustificare la risposta!

## Attraversamento del fiume in notturna

*Difficoltà: media*

Quattro castori devono attraversare un ponte sul fiume, ed è notte fonda. Per fortuna hanno una torcia elettrica, ma il ponte può essere percorso da al più due castori per volta, e chi attraversa il ponte deve avere con sé la torcia, mentre gli altri possono aspettare al buio. I quattro castori non sono ugualmente veloci: per attraversare il ponte, Alex impiega 1 minuto, Bob 2 minuti, Chris 5 minuti e Dan, che è il più lento, 10 minuti. Qual è il tempo minimo necessario affinché tutti e quattro i castori si ritrovino sulla riva opposta del fiume? Giustificare la risposta!

## Un vecchio calcolatore

*Difficoltà: difficile*

Il castoro Adriano ha trovato in soffitta un vecchio calcolatore, che dispone di tre soli *registri* (aree di memoria in ciascuna delle quali può essere memorizzato un numero), chiamati  $R_1$ ,  $R_2$  e  $R_3$ .

Per programmare questa macchina, è necessario codificare in una opportuna sequenza le operazioni che essa dovrà svolgere. Le operazioni che la macchina può compiere sono di sei tipi:

- $\text{Zero}(i)$ : memorizza 0 nel registro  $R_i$ ;
- $\text{Inc}(i)$ : incrementa di 1 il contenuto del registro  $R_i$ ;
- $\text{Dec}(i)$ : decrementa di 1 il contenuto del registro  $R_i$ ;
- $\text{Store}(i, j)$ : copia il contenuto del registro  $R_j$  nel registro  $R_i$ ;
- $\text{Jump}(i, j, n)$ : se  $R_i$  e  $R_j$  contengono lo stesso valore, allora salta all'operazione numero  $n$
- $\text{JumpNeg}(i, j, n)$ : se  $R_i$  e  $R_j$  contengono valori diversi, allora salta all'operazione numero  $n$

dove  $i$  e  $j$  indicano il numero di un registro (e quindi possono essere 1, 2 o 3) e  $n$  indica il numero d'ordine di un'operazione nella sequenza.

Ad esempio, il programma seguente scambia il contenuto dei registri  $R_1$  e  $R_2$ :

```
1: Store(3,2)
2: Store(2,1)
3: Store(1,3)
```

Il castoro Adriano memorizza due numeri nei registri  $R_1$  e  $R_2$  e vuole scrivere un programma che calcoli la loro somma, memorizzandola in  $R_1$  al posto del primo addendo.

Sapreste aiutarlo?

## Quesiti da svolgere con l'aiuto del computer

### Un programma affollato di eventi

Il parco di divertimento *Kangourandia* offre ai suoi visitatori un ricco calendario di eventi. Ogni giorno sono previsti film 4D, esibizioni degli *stunt-men* e dei tuffatori, commedie musicali, animazione dei bambini, parate delle *mascotte*, e altro ancora.

Avete solo un giorno a disposizione e volete assistere al massimo numero di eventi possibili. Naturalmente non è possibile assistere ad eventi che si svolgono in contemporanea (anche parzialmente).

Il vostro amico Ale Gridi, genio dell'informatica, vi ha suggerito di usare questa strategia per scegliere gli eventi cui assistere:

Mettete in ordine gli eventi del programma, poi analizzateli uno alla volta rispettando quell'ordine: se l'evento in questione è il primo considerato oppure non si sovrappone agli altri eventi già scelti, allora assisterete all'evento, altrimenti lo scarterete e non lo prenderete più in considerazione.

In questo modo, garantisce Ale Gridi, riuscirete ad assistere ad un numero di eventi che nessuno potrà battere.

L'unico problema è che non vi ricordate in che modo Ale Gridi vi ha suggerito di ordinare gli eventi e le possibilità sono molte!

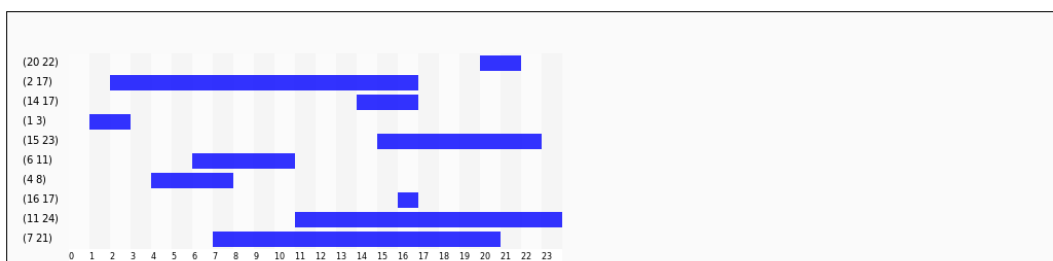
Per individuare un buon criterio di ordinamento, avete a disposizione un programma che vi aiuterà a fare degli esperimenti. Per avviarlo lanciate il programma *Eventi*, vi comparirà una pagina con tre riquadri.

- Nel riquadro in alto potete inserire e modificare le informazioni relative agli orari degli eventi; per semplicità assumiamo che gli eventi inizino e finiscano tutti allo scoccare dell'ora. Per ogni evento, va indicata la coppia di ora di inizio e ora di fine nel formato (inizio fine). Ad esempio la sequenza

**(20 22)(2 17)(14 17)(1 3)(15 23)(6 11)(4 8)(16 17)(11 24)(7 21)**

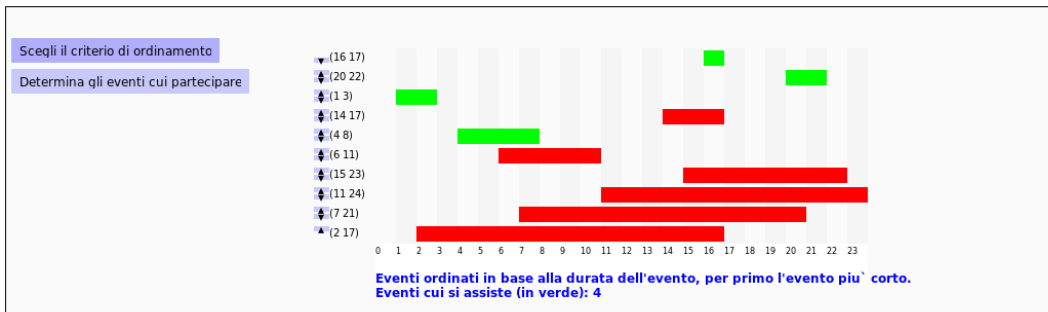
rappresenta dieci eventi, uno che inizia alle 20 e finisce alle 22, uno che inizia alle 2 e finisce alle 17, uno che inizia alle 14 e finisce alle 17, e così via. Possono esserci più eventi con lo stesso orario, in questo caso ci sarà una coppia (inizio fine) che si ripete nella sequenza.

- Nel riquadro centrale trovate una rappresentazione grafica degli eventi in base al loro orario.



- Nel riquadro in basso potete modificare l'ordine in cui vengono visualizzati gli eventi elencati nella sequenza del primo riquadro. Con le frecce a sinistra potete spostare verso l'alto o il basso ciascun evento. Potete inoltre scegliere un criterio di ordinamento e ottenere la visualizzazione degli eventi

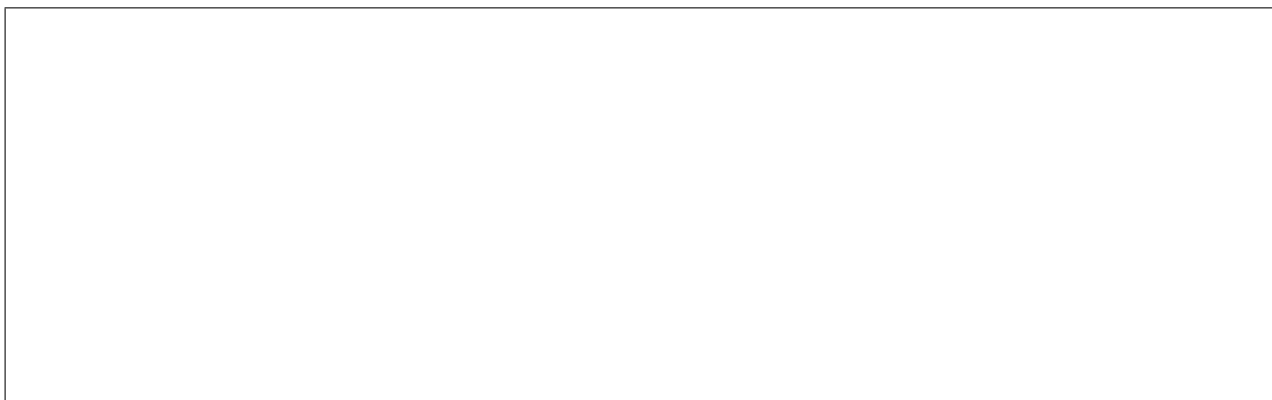
nell'ordine corrispondente. Infine, cliccando su “determina gli eventi cui partecipare”, potete applicare, agli eventi così ordinati, la strategia suggerita da Ale Gridi, ottenendo in verde l'elenco degli eventi cui assistere. Ad esempio, ordinando in base all'ora di inizio (per primo quello che inizia per primo) si otterrebbe questa selezione di eventi:



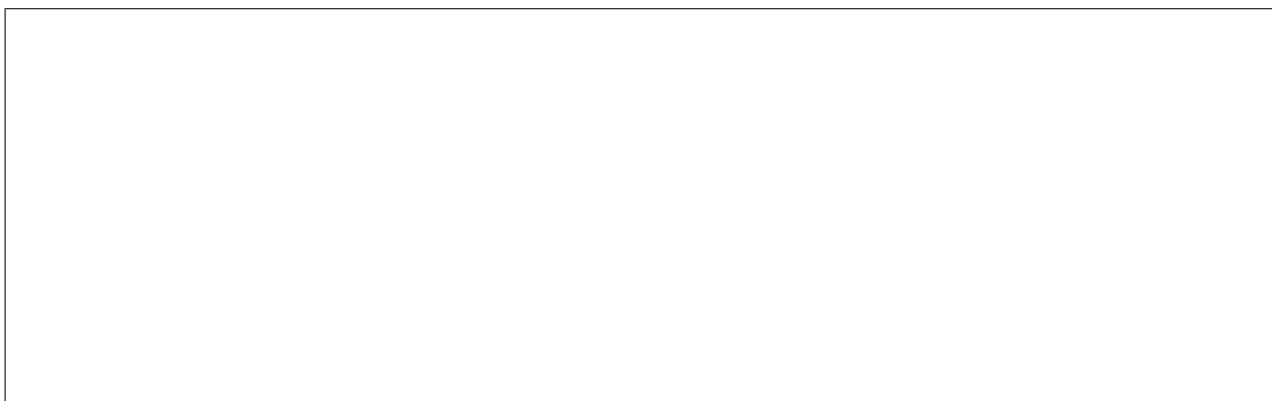
Per ciascuno dei seguenti criteri di ordinamento, scrivete se è un criterio *adatto* oppure no, nel senso che permette di selezionare il maggior numero di eventi possibili cui assistere. Inoltre, per ciascuno dei criteri che considerate non adatti, fornite un esempio (più semplice è l'esempio, maggiore sarà il punteggio) che mostra perché il criterio non è adatto. Per ciascuno dei criteri che considerate adatti, illustrate, nell'ultimo riquadro di questo quesito, il ragionamento che vi ha portato a questa conclusione.

1. Criterio: in base all'ora d'inizio, per primo quello che inizia prima.

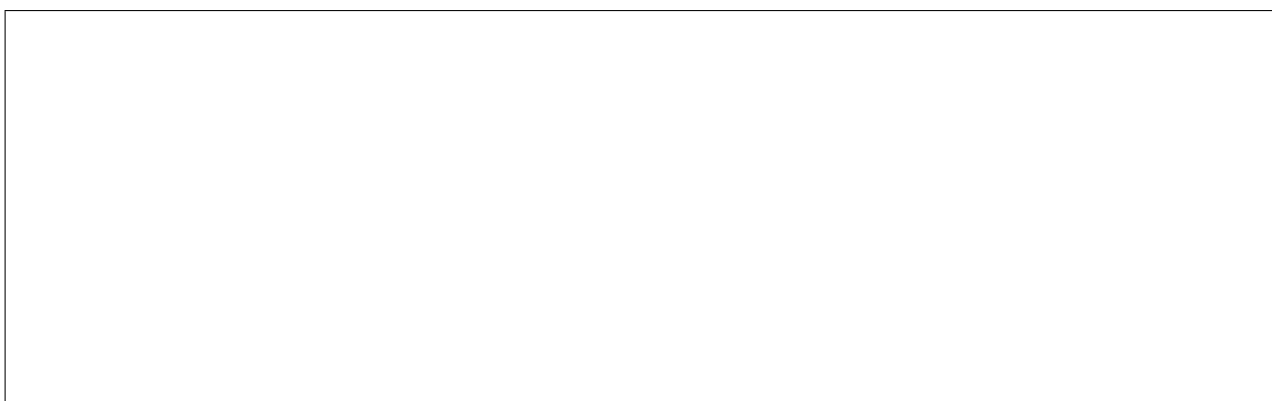
2. Criterio: in base all'ora di fine, per primo quello che finisce prima.



3. Criterio: in base all'ora d'inizio, per primo quello che inizia più tardi.

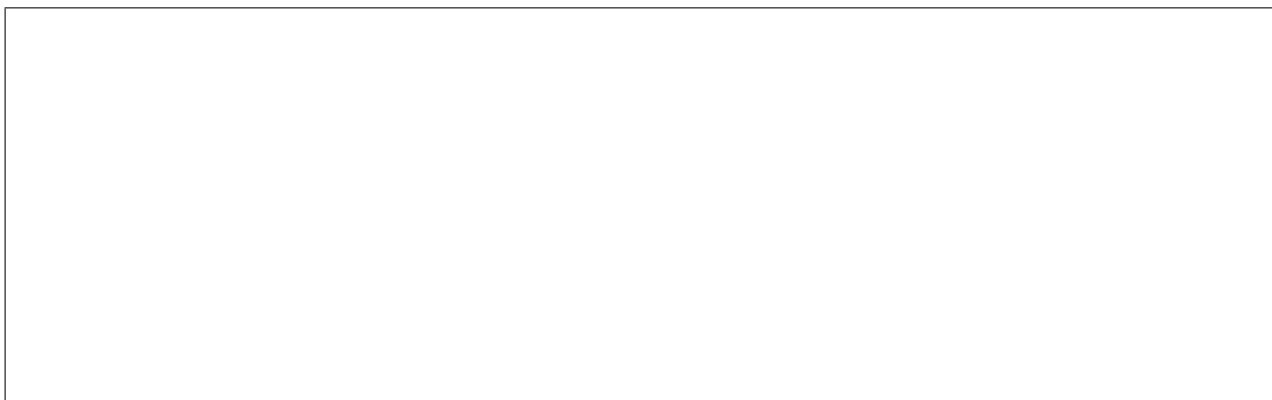


4. Criterio: in base all'ora di fine, per primo quello che finisce più tardi.

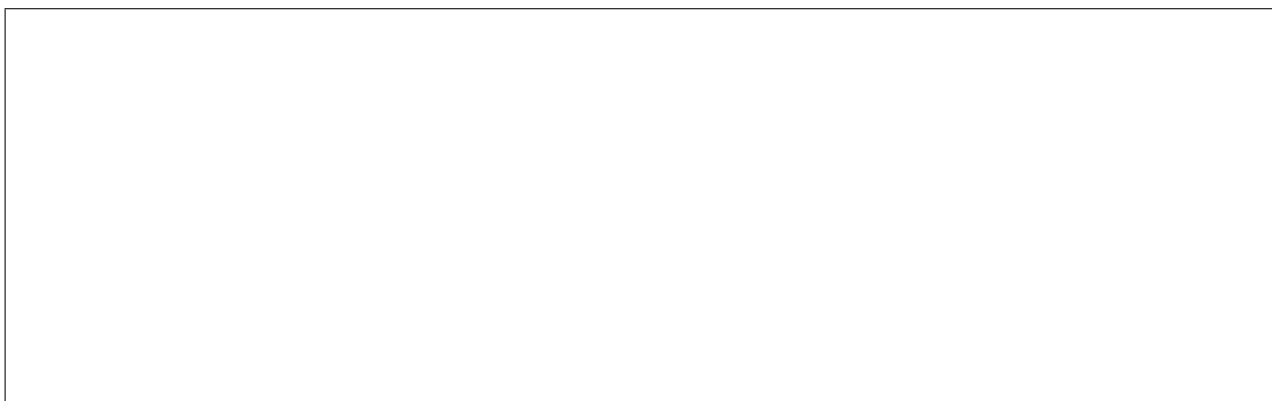




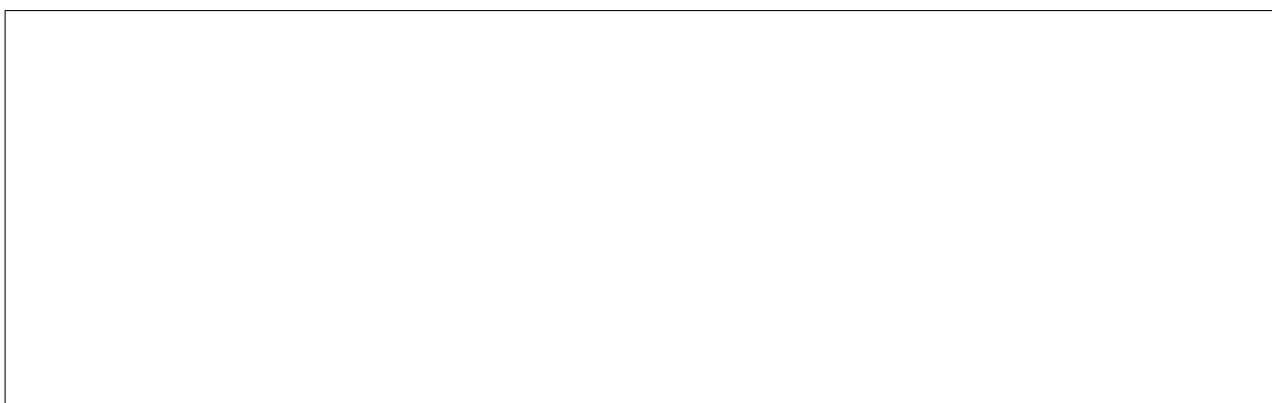
5. Criterio: in base alla durata dell'evento, per primo l'evento più corto.



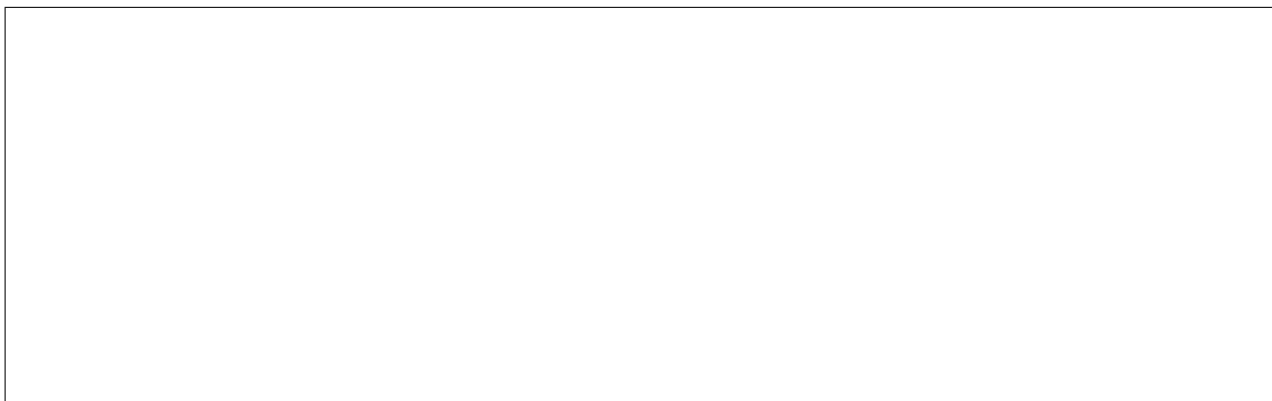
6. Criterio: in base alla durata dell'evento, per primo l'evento più lungo.



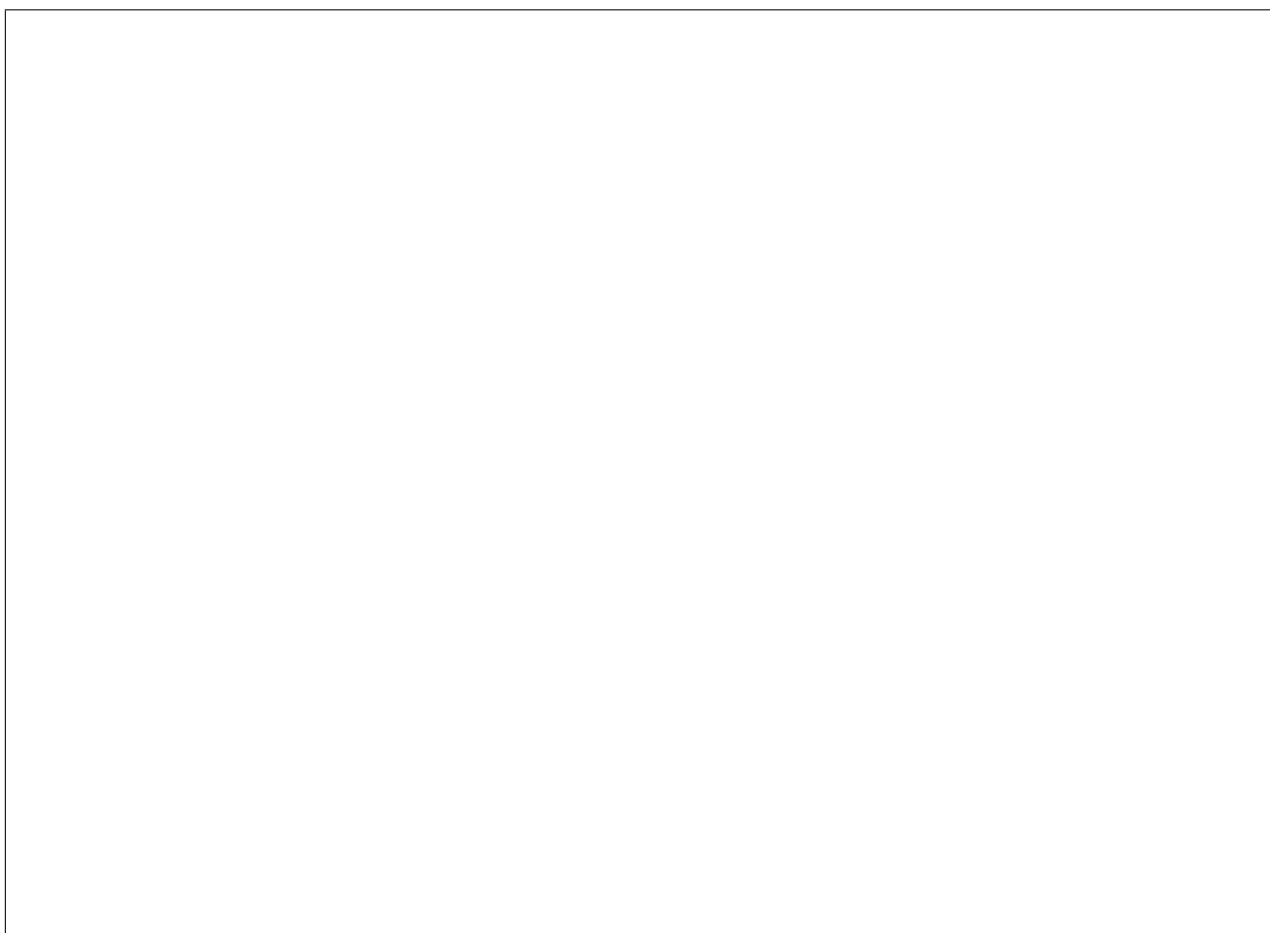
7. Criterio: in base al numero di sovrapposizioni, per primo l'evento che ha meno sovrapposizioni con gli altri eventi.



8. Criterio: in base al numero di sovrapposizioni, per primo l'evento che ha più sovrapposizioni con gli altri eventi.



Illustrate in modo convincente che i criteri (o il criterio) che avete considerato adatti sono imbattibili.



## Clickomania

Vi ricordate le regole di *Clickomania* che avete ricevuto ieri sera?

Vostro compito è realizzare un programma per giocare a Clickomania o, piú precisamente, un programma che aggiorni la parete di gioco in seguito a ogni mossa (click) del giocatore, utilizzando l'*ambiente di programmazione Blockly*.

### Blockly

Blockly consente di *costruire dei programmi* a partire da blocchi colorati predefiniti, che si possono incastrare come in un puzzle.

Non c'è un solo programma che permette di giocare a Clickomania: i blocchi a disposizione possono essere combinati a formare tanti diversi programmi, tutti funzionanti. Tuttavia ciascun blocco ha un *costo* e il vostro punteggio sarà tanto maggiore quanto minore sarà la somma dei costi dei blocchi usati.

Una volta costruito un programma, si può iniziare il gioco cliccando sul bottone "Gioca". A questo punto, ad ogni click del giocatore (voi stessi, mentre testate il vostro programma!!), la parete verrà aggiornata in base alle istruzioni del programma costruito.

Blockly consente inoltre di vedere gli effetti dell'esecuzione del programma costruito, attraverso un'animazione, direttamente sulla parete di gioco, e permette di salvare fino a 4 programmi costruiti.

### Interfaccia

Lanciate il programma *Clickomania*: vedrete una pagina composta di tre colonne.

- A sinistra trovate vari elementi.
  - La parete di mattoni.
  - Un cursore con ai lati una tartaruga e una lepre, per aumentare/diminuire la velocità dell'animazione.
  - Il bottone "Gioca" che consente di iniziare una partita e che si trasforma in "Reset" durante l'esecuzione del programma; cliccando su "Reset" è possibile iniziare una nuova partita.
  - Le icone relative a salvataggio/visualizzazione/caricamento dei programmi (o versioni di uno stesso programma): cliccando sul primo simbolo si può salvare il programma attualmente in costruzione, soffermandosi col mouse sul secondo simbolo si può visualizzare il programma salvato, cliccando sul terzo simbolo si può caricare il programma salvato in precedenza.
- Al centro trovate un menu di categorie tra cui scegliere i blocchi colorati; soffermandosi col mouse su un blocco si può leggere una descrizione di cosa fa il blocco.
- A destra trovate uno spazio bianco in cui potete trascinare i blocchi scelti dal menu, per costruire le istruzioni del vostro programma. Nell'angolo in basso a destra c'è il simbolo di un cestino nel quale possono essere trascinati i blocchi che non servono piú.

Durante l'aggiornamento della parete in seguito ad una mossa (click), i blocchi in esecuzione saranno via via evidenziati. Per fare degli esperimenti, tra una mossa e l'altra potete modificare il programma; però come soluzione del quesito accetteremo soltanto un programma che non debba essere modificato a seconda della mossa!

## Blocchi a disposizione

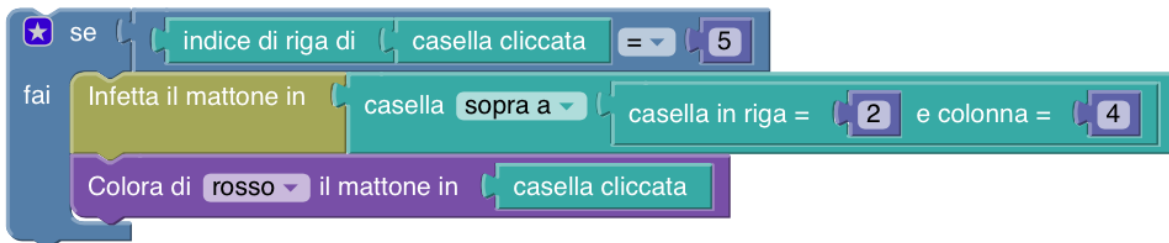
I blocchi disponibili sono raggruppati nelle seguenti categorie (alcuni blocchi sono presenti in più di una categoria): Aggiornamento parete, Caselle, Colori, Infezione, Logica, Cicli, Matematica, Variabili. Ciascun blocco ha un costo che si può visualizzare soffermandosi col mouse sopra il blocco stesso. L'elenco completo dei blocchi con i loro costi è disponibile nel foglio allegato.

## Esempi

Il programma più semplice che gioca a Clickomania è il seguente. Ad ogni click, vengono rimossi tutti i mattoni che fanno parte dell'area di contagio attorno alla casella cliccata, e gli spazi vuoti che si vengono a creare vengono eliminati dalla caduta dei mattoni che si trovano sopra.

Rimuovi i mattoni nell'area di contagio attorno a **casella cliccata** e fai cadere tutti i blocchi

Vediamo ora qualche esempio un po' più complicato (non è detto che siano di qualche utilità!!):



In questo caso, se la casella cliccata si trova nella riga di indice 5, allora viene infettato il mattone nella casella di coordinate (1, 4) e viene colorato di rosso il mattone nella casella cliccata.



In questo caso, viene considerata la casella più in alto tra quelle contenenti un mattone infetto, assieme alla casella cliccata; se le due caselle contengono un mattone dello stesso colore, allora verrà rimosso il mattone che si trova nella casella di coordinate (5, 4) e cadrà l'intera pila di mattoni posizionata a partire dalla casella di coordinate (4, 4).



In questo caso, se ci sono più di tre mattoni infetti, ne verranno rimossi tre a caso; se ce ne sono tre o meno, verranno rimossi tutti.

## Paleografia informatica

*Questa prova va svolta su carta. Se volete, potete usare il Web per fare delle ricerche.*

Essendo nato nel 4016 aveva ormai quasi quarant'anni, e da tutti era considerato un archeologo esperto, uno dei luminari nel suo campo, la *paleografia informatica*. Eppure non aveva mai partecipato prima di allora a una campagna di scavi che lo emozionasse tanto: sentiva che quello che stavano scoprendo avrebbe gettato una nuova luce sul passato dell'umanità e sui modi misteriosi con cui gli uomini, piú di due millenni prima, interagivano con i computer. Ben prima dell'avvento della programmazione olistica, prima del trasferimento osmotico della conoscenza, prima della costruzione della PCA (Pura Coscienza Algoritmica), gli umani usavano i computer in modo estremamente rudimentale, programmandone il funzionamento direttamente, e lo facevano attraverso misteriosi linguaggi (li chiamavano “**linguaggi di programmazione**”, appunto) il cui significato e la cui sintassi erano ancora in gran parte avvolti nel mistero.

Stava pensando a tutto questo quando, con la mano tremante, si avviava ad esaminare il contenuto della memoria USB trovata il giorno prima, alla fine di una lunga giornata sul campo, sotto il sole cocente dell'antica Silicon Valley, in California: stavano scavando a Mountain View, in quella che era stata la sede di un opificio informatico nato all'inizio del terzo millennio e chiamato GOGOL. Era in quell'area che, il giorno prima, avevano recuperato questo piccolo oggetto metallico. L'aveva riconosciuto subito, ne aveva già visti centinaia: era una memoria USB, forse posseduta da qualche dipendente di Gogol che la usava per trasportare i programmi da un computer all'altro, o magari per lavorarci da casa, durante la notte. Quella memoria conteneva con ogni probabilità del codice sorgente di qualche *programma informatico*.

Ora l'oggetto metallico era nel suo laboratorio; maneggiandolo con grande cautela lo aveva inserito nel lettore speciale (a prova di polvere e di contaminazione) che lui stesso aveva costruito negli anni e che gli permetteva di analizzare il contenuto di memorie di quel periodo. Dopo qualche istante, nello schermo olografico, iniziarono a comparire i piccoli involti che rappresentavano i “*file*” (cosí, nell'antichità, erano chiamati i contenitori dei codici sorgente dei programmi). Era bellissimo: c'erano decine di programmi tutti a sua disposizione, e aprendoli avrebbe potuto scoprire che cosa facevano e forse capire meglio i linguaggi in cui erano scritti e di cui ancora nessuno aveva compreso quasi nulla.

Prese in mano il pacchetto olografico di uno dei *file*, e ne svolse i lembi per aprirlo e vederne il contenuto. Quello che segue fu ciò che apparve, nella penombra della stanza.

Sapete identificare a) il linguaggio di programmazione e b) il contesto storico o informatico in cui sono stati creati i seguenti frammenti di codice?

1. Ecco un frammento:

```
/*  
 * You are not expected to understand this.  
 */  
if(rp->p_flag & SSWAP) {  
    rp->p_flag &= ~SSWAP;  
    aretu(u. u_ssav);  
}
```

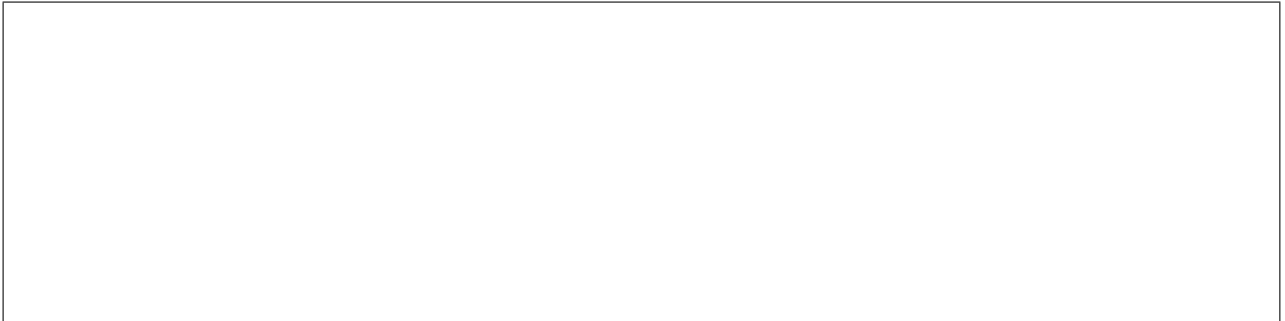
Di che si tratta?



2. Ecco un altro frammento:

```
INTEGER A,B,C
READ(5,501) A,B,C
501 FORMAT(3I5)
IF(A.EQ.0 .OR. B.EQ.0 .OR. C.EQ.0) STOP 1
S = (A + B + C) / 2.0
AREA = SQRT( S * (S - A) * (S - B) * (S - C))
WRITE(6,601) A,B,C,AREA
601 FORMAT(4H A= ,I5,5H B= ,I5,5H C= ,I5,8H AREA= ,F10.2,12HSQUARE UNITS)
STOP
END
```

Di che si tratta?



3. Ecco un altro frammento:

```
s' '$/=2048; while(<>){G=29;R=142; if ((@a=unqT="C*", _)[20]&48){D=89; _=unqb24, qT,@
b=map{ord qb8, unqb8, qT, _ ^$a[--D]}@INC; s / ... $/1$&/; Q=unqV, qb25, _; H=73; O=$b[4]<<9
|256|$b[3]; Q=Q>>8^(P=(E=255)&(Q>>12^Q>>4^Q/8^Q))<<17, O=O>>8^(E&(F=(S=O>>14&7^O)
^S*8^S<<6))<<9, _=(map{U=_%16orE^=R^=110&(S=(unqT, "\xb\ntd\xbz\x14d")[_/16%8]);E
^=(72,@z=(64,72,G^=12*(U-2?0:S&17)),H^=_%64?12:0,@z)[_ %8]}(16..271))[_]^(D>>=8
)+=P+(~F&E))for@a[128..$#a]} print+qT,@a}'; s/[D-HO-U_]//\ $$&/g; s/q/pack+/g; eval
```

Di che si tratta?



```
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;
```

Di che si tratta?

7. Ecco un altro frammento:

```
primes = sieve [2..]
sieve (p : xs) = p : sieve [x | x <- xs, x `mod` p > 0]
```

Di che si tratta?

8. Ecco un altro frammento:

```
((lambda (x)
  (list x (list (quote quote) x)))
 (quote
  (lambda (x)
    (list x (list (quote quote) x))))))
```

Di che si tratta?



9. Ecco un altro frammento:

```
mortal(X) :-  
    human(X).  
human(socrate).  
?- mortal(socrate).  
yes
```

Di che si tratta?

10. Ecco un altro frammento:

```
tell application "Finder"  
    make new folder at desktop  
end tell
```

Di che si tratta?

11. Ecco un altro frammento:

```
HAI
CAN HAS STDIO?
VISIBLE "HAI WORLD!"
KTHXBYE
```

Di che si tratta?

12. Ecco un altro frammento:

```
keeplooping=1;
while [[ $keeplooping -eq 1 ]] ; do
  read quitnow
  if [[ "$quitnow" = "yes" ]] ; then
    keeplooping=0
  fi
  if [[ "$quitnow" = "q" ]] ; then
    break;
  fi
done
```

Di che si tratta?