Informatica
e Tecnologie della Comunicazione Digitale

Docente:
Miguel Ceriani (ceriani@di.uniroma1.it)

Lezioni:
Mercoledì/Giovedì/Venerdì 9-11

Ricevimento (su appuntamento):
Mercoledì 14-16 a **viale Regina Elena 295**, **palazzina F**, **1° piano**

Lezione 18: Operatori, Espressioni e Diagrammi di Flusso

Operatori

- Funzionalità che potrei realizzare con funzioni predefinite ma sarebbe meno leggibile. Ad esempio:
- `a+b` invece che `somma(a,b)`
- `a-b` invece che `sottrazione(a,b)`
- `NOT a` invece che `not(a)`

Operatori Numerici

(alcuni)

Operatore	Num. Operandi	Descrizione	Esempio
+	2	somma	5 + indice
-	2	sottrazione	a - 4
*	2	moltiplicazione	base * altezza
/	2	divisione	lato / 2
-	1	opposto	- vel

Operatori Booleani

Operatore	Num. Operandi	Descrizione	Esempio
NOT	1	negazione	NOT qui_formaggio()
AND	2	congiuzione	strada_avanti() AND NOT qui_formaggio()
OR	2	disgiunzione	strada_destra() OR strada_sinistra()

Operatori Booleani: AND

- Corrisponde alla congiunzione “e”, nel senso di è vero qualcosa **e** qualcos’altro
- *espr1* **AND** *espr2* è un espressione booleana che vale vero solo se *espr1* e *espr2* valgono vero

a	b	a AND b
falso	falso	falso
falso	vero	falso
vero	falso	falso
vero	vero	vero

Operatori Booleani: OR

- Corrisponde alla congiunzione “e/o”, nel senso di è vero qualcosa **e/o** qualcos’altro
- *espr1* **OR** *espr2* è un espressione booleana che vale vero se almeno una tra *espr1* e *espr2* vale vero

a	b	a OR b
falso	falso	falso
falso	vero	vero
vero	falso	vero
vero	vero	vero

Espressioni

- Non le abbiamo ancora definite formalmente
- E' una delle due componenti fondamentali dei linguaggi di programmazione, insieme alle *istruzioni*
- Finora abbiamo definito sintassi e semantica delle istruzioni, ora completiamo la descrizione del linguaggio con sintassi e semantica delle *espressioni*

Espressioni: Sintassi

- *costante*

  dove *costante* è una costante come **45**, **7.5**, **vero**, **falso**
- *nome_variabile*

  dove *nome_variabile* è il nome di una variabile o parametro già definito in quel contesto
- *nome_funzione***(***espr1***,** *espr2***,** …**)**

  dove *nome_funzione* è il nome di una funzione precedentemente definita con DEF e *espr1*, *espr2*, … sono espressioni
- *espr1 op2 espr2*

  dove *op2* è un operatore su due operandi (es., **+**, **AND**) e *espr1*, *espr2*, … sono espressioni
- *op1 espr*

  dove *op1* è un operatore su un operando (es., **NOT**) e *espr* è un'espressione

Espressioni: Semantica

- *costante*

  il valore dell'espressione corrisponde alla costante
- *nome_variabile*

  il valore dell'espressione corrisponde al valore della variabile in quel punto dell'esecuzione
- *nome_funzione***(***espr1***,** *espr2***,** …**)**

  la funzione identificata da *nome_funzione* viene eseguita sui valori delle espressioni *espr1*, *espr2*, …; il valore complessivo dell'espressione è il valore restituito dalla funzione
- *espr1 op2 espr2*

  l'operatore *op2* viene eseguito sui valori delle espressioni *espr1* e *espr2*; il valore dell'espressione è il valore restituito dall'operatore
- *op1 espr*

  l'operatore *op1* viene eseguito sul valore dell'espressione *espr*; il valore dell'espressione è il valore restituito dall'operatore

Programma e Algoritmo

- un *programma* (di cui abbiamo visto tanti esempi in queste lezioni) è una procedura per realizzare un'applicazione scritta in un *linguaggio di programmazione* e quindi eseguibile da un computer;
- un *algoritmo* è una procedura descritta in modo da essere non ambigua, ma non in un linguaggio di programmazione; è l'idea dietro un programma, descritto in modo che sia comprensibile da altre persone;
- quando scrivo un programma corrispondente a un algoritmo, si dice che lo *implemento*; per uno stesso algoritmo ci possono essere tante diverse implementazioni, perché esistono tanti linguaggi diversi, ma anche usando uno stesso linguaggio perché possono cambiare i dettagli.

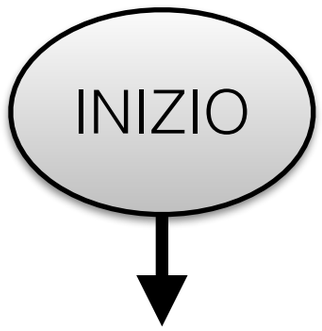
Diagrammi di Flusso

- ci sono tanti modi per descrivere un algoritmo, l'importante è che non rimanga nulla di ambiguo per chi lo legge;

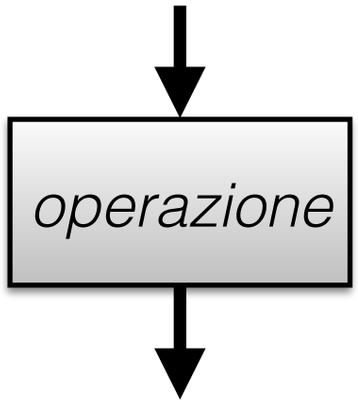
- mentre in un programma, tutto va dettagliato al livello al quale il linguaggio lo richiede, nell'algoritmo possiamo evitare i dettagli che risultino ovvi per chi lo legge;

- esistono vari modi per strutturare e descrivere un algoritmo, i *diagrammi di flusso* sono uno strumento molto conosciuto.

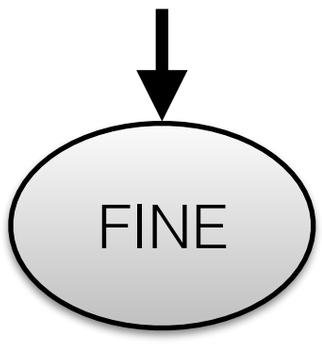
Diagrammi di Flusso: Basi



Rappresenta l'inizio dell'algoritmo, la freccia uscente porta alla prima istruzione da eseguire



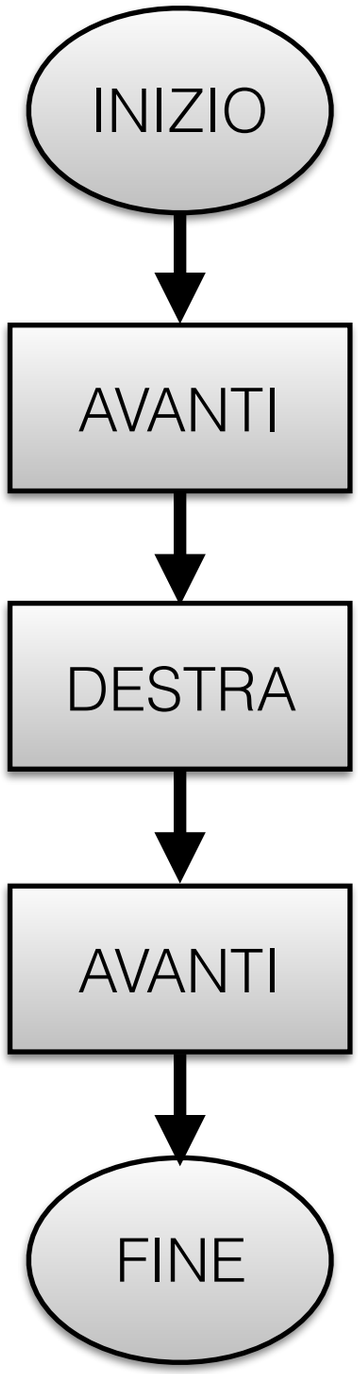
Rappresenta l'esecuzione di un'operazione, la freccia uscente porta alla successiva istruzione da eseguire



Rappresenta la fine dell'algoritmo

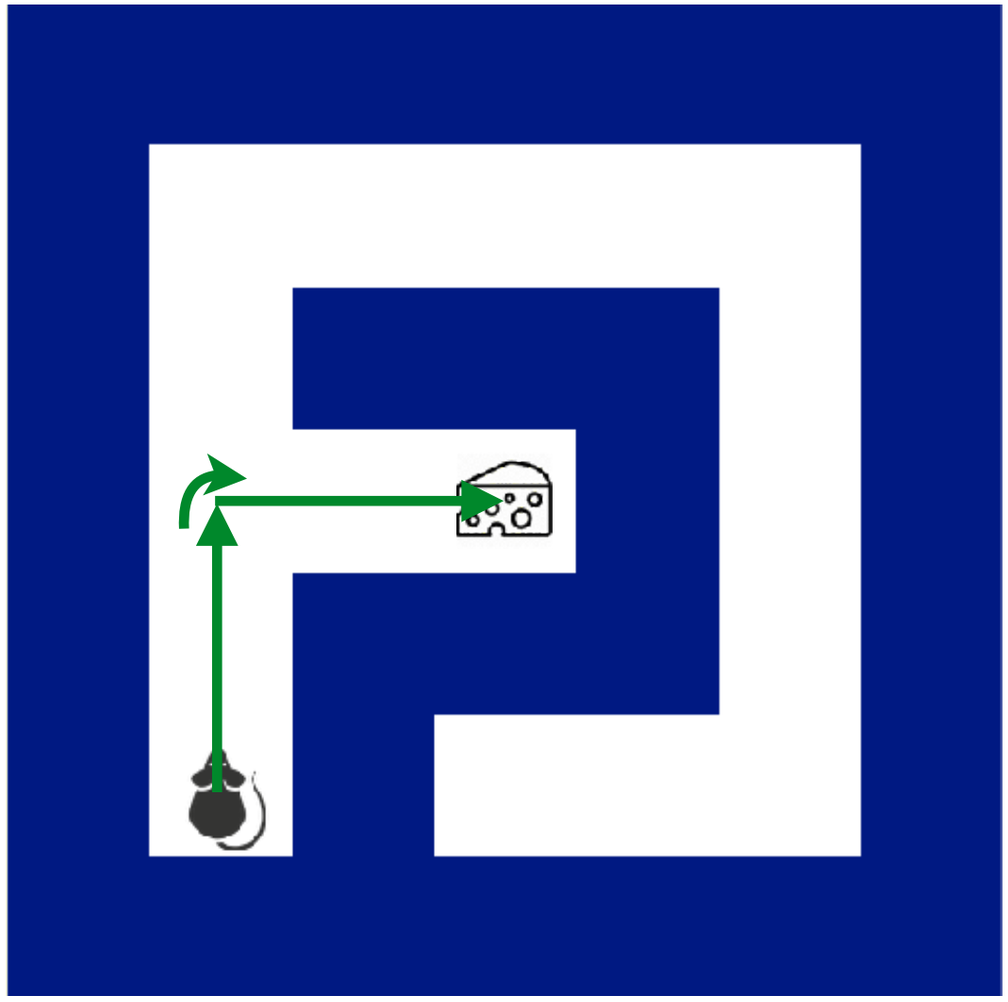
Caso Singolo

Diagramma di Flusso

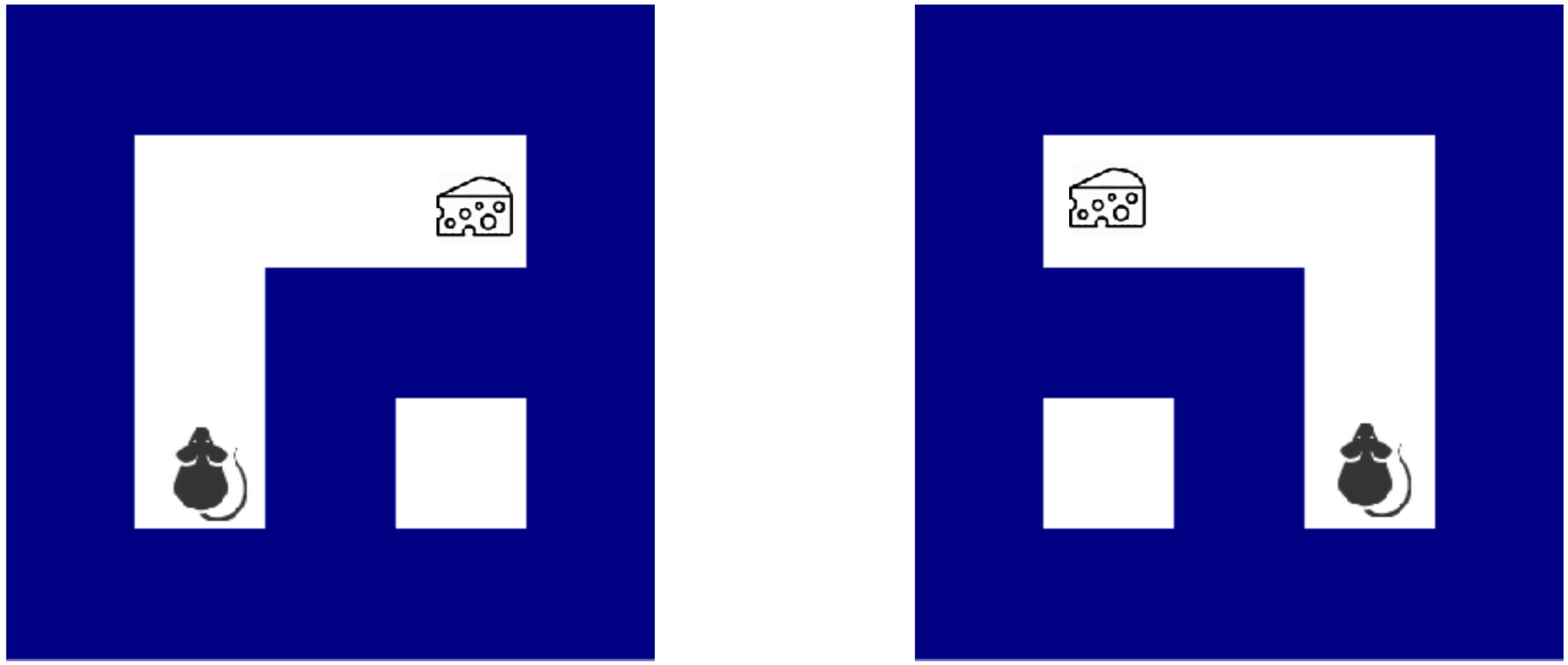


Programma

```
avanti()
destra()
avanti()
```

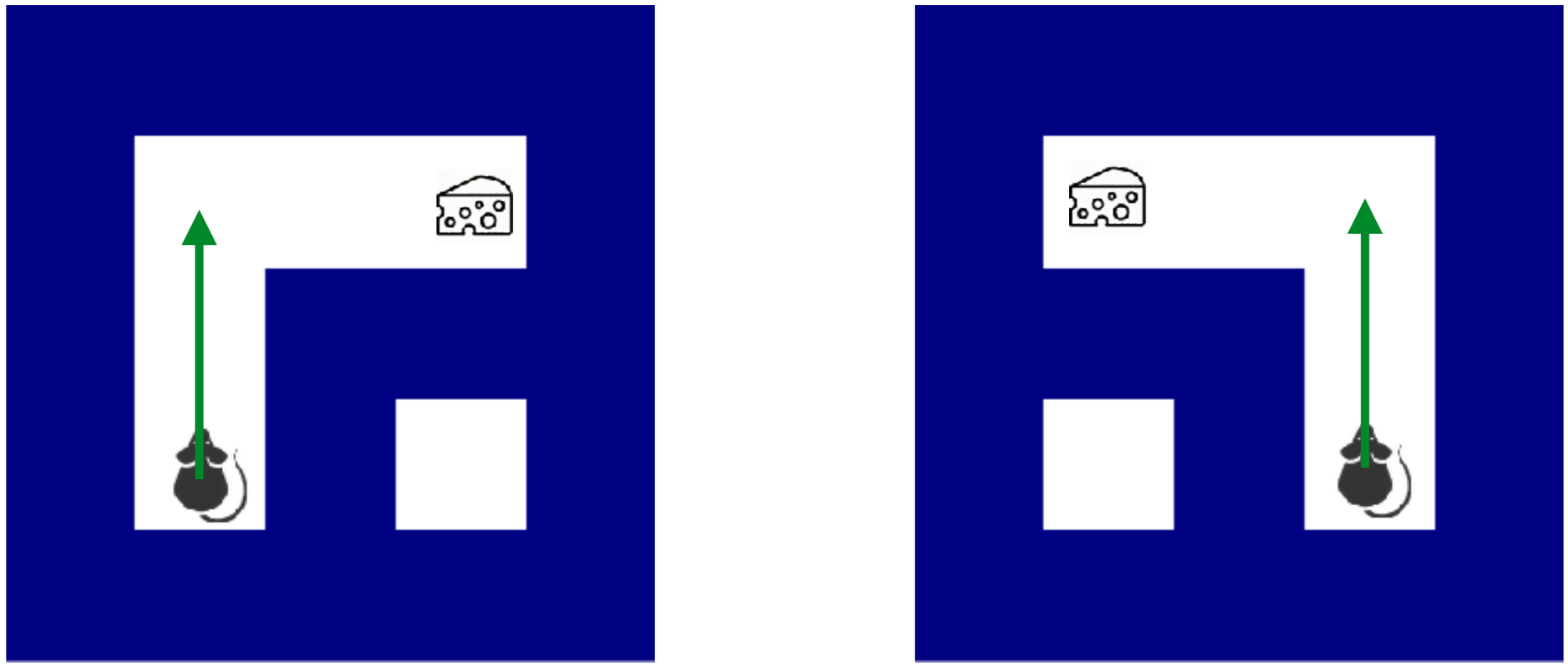


“Elle” di lunghezza 2



due casi

“Elle” di lunghezza 2

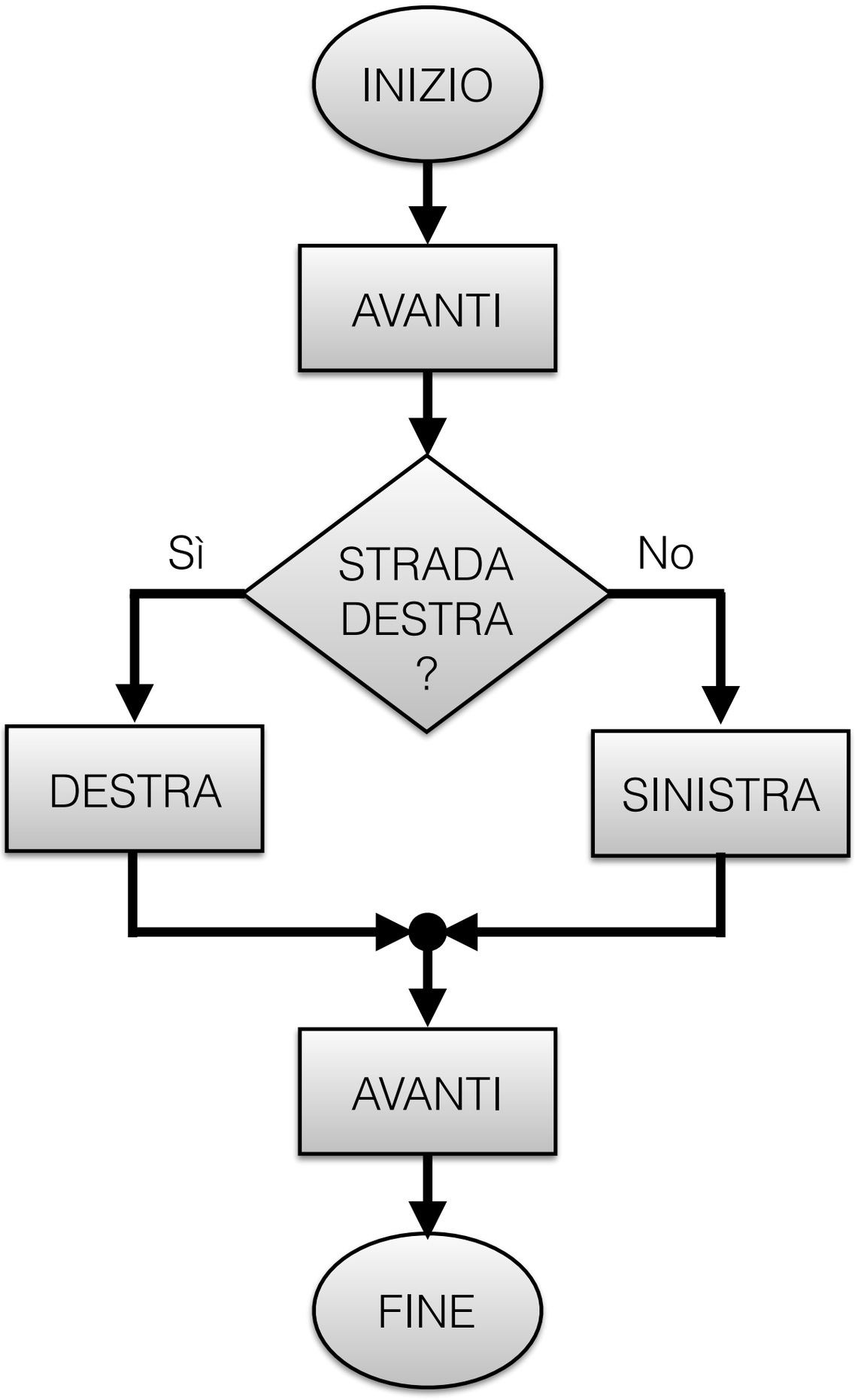


devo fare operazioni diverse a seconda di una condizione: come if-then-else

Diagrammi di Flusso: Condizioni

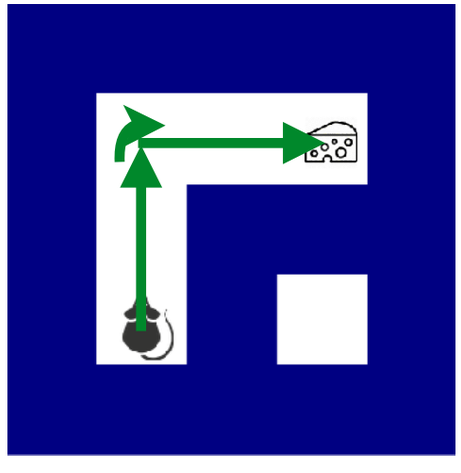
Rappresenta una condizione:

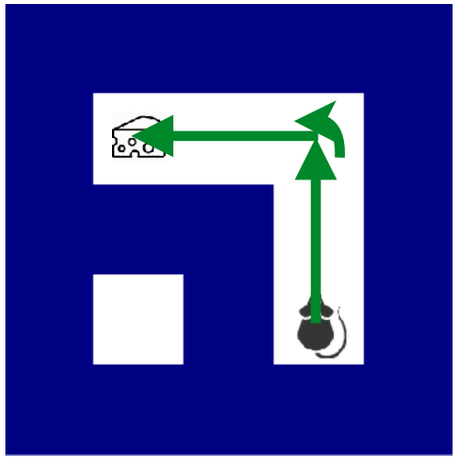
- la freccia “Sì” porta alla successiva istruzione nel caso in cui la condizione sia vera;
- la freccia “No” porta invece alla successiva istruzione nel caso in cui la condizione sia falsa.



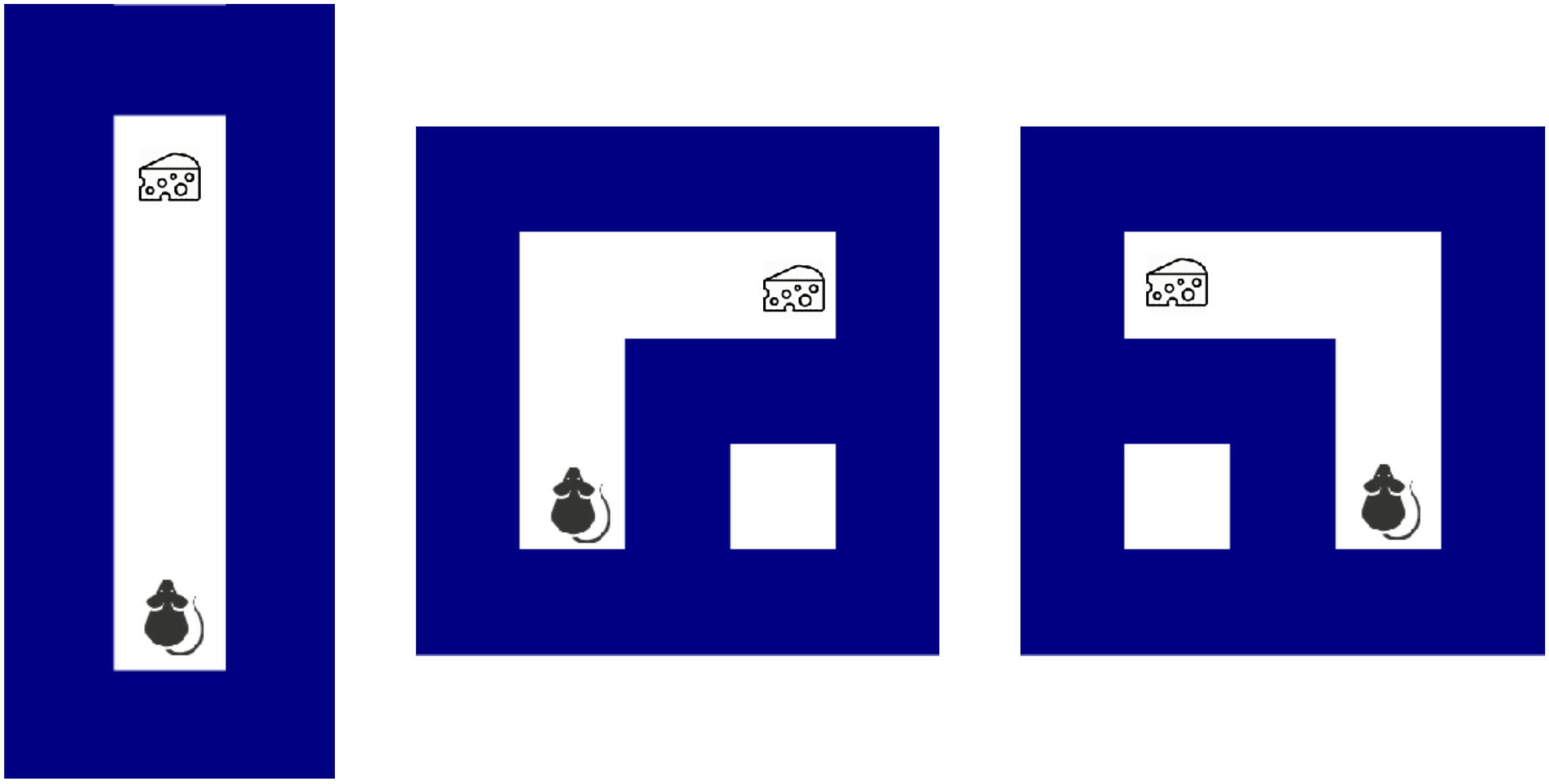
Programma

```
avanti()
IF (strada_destra())
    THEN {
        destra()
    }
    ELSE {
        sinistra()
    }
avanti()
```

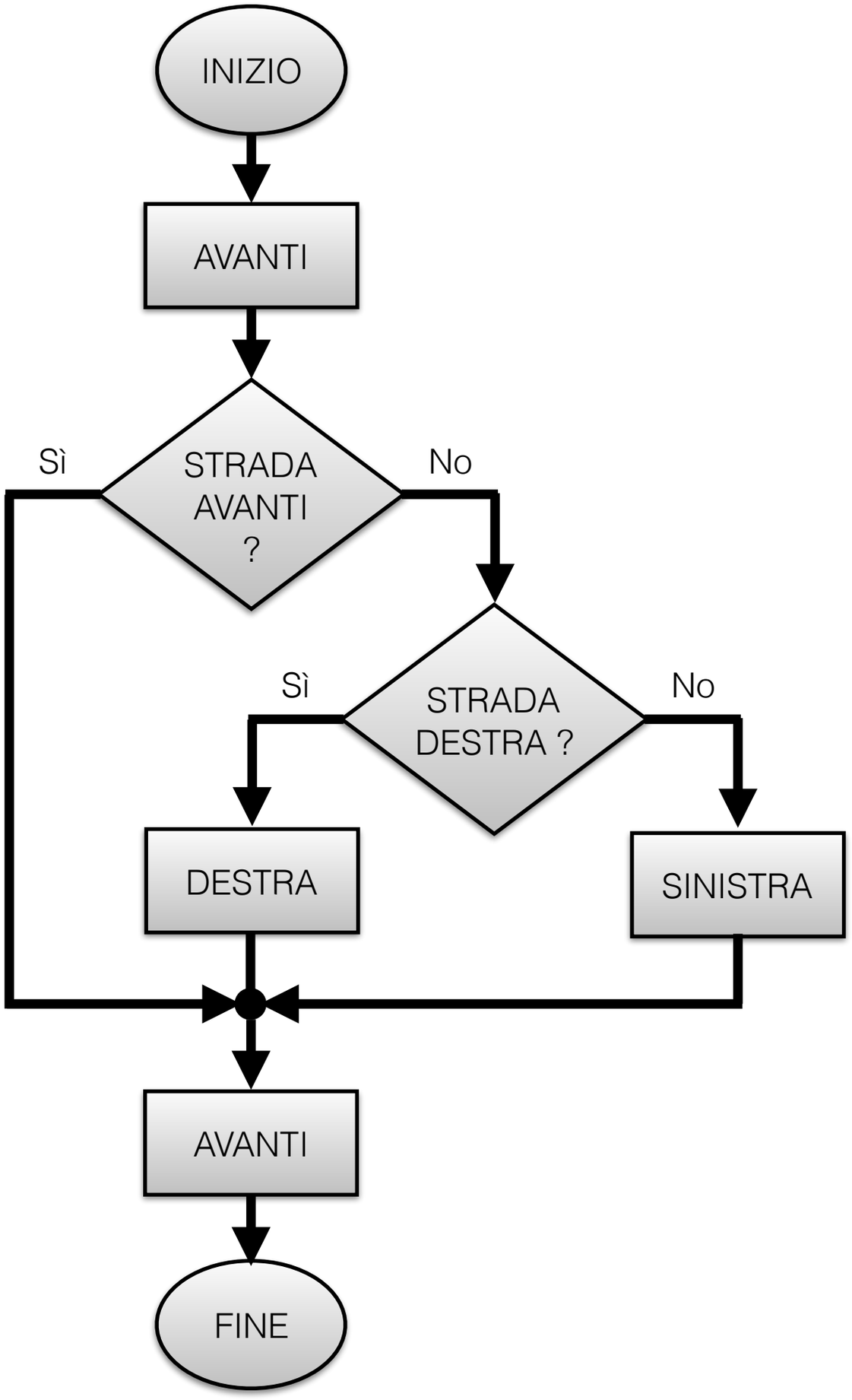




Percorso di lunghezza 2

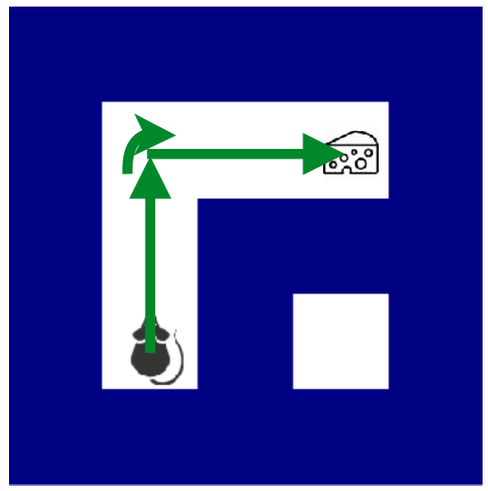


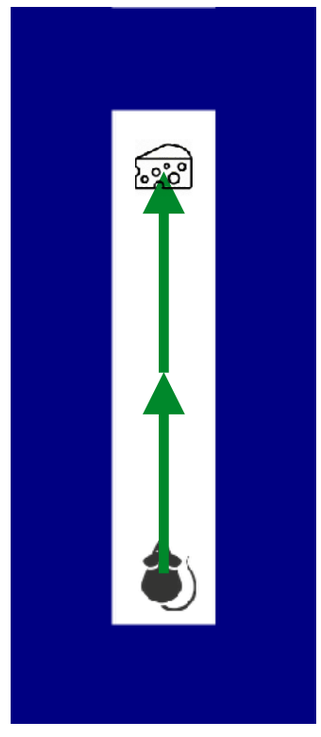
tre casi

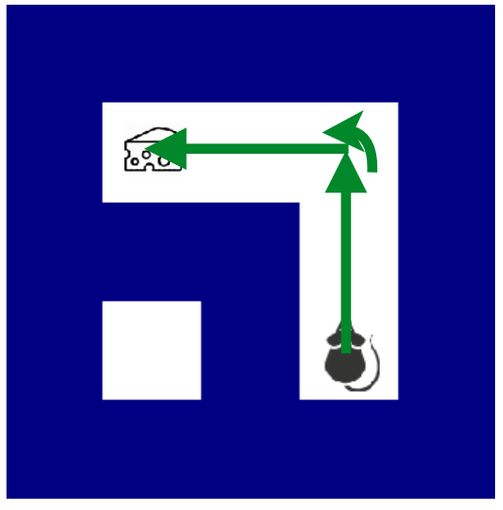


Programma

```
avanti()
IF (NOT strada_avanti())
    THEN {
        IF (strada_destra())
            THEN {
                destra()
            }
            ELSE {
                sinistra()
            }
    }
avanti()
```

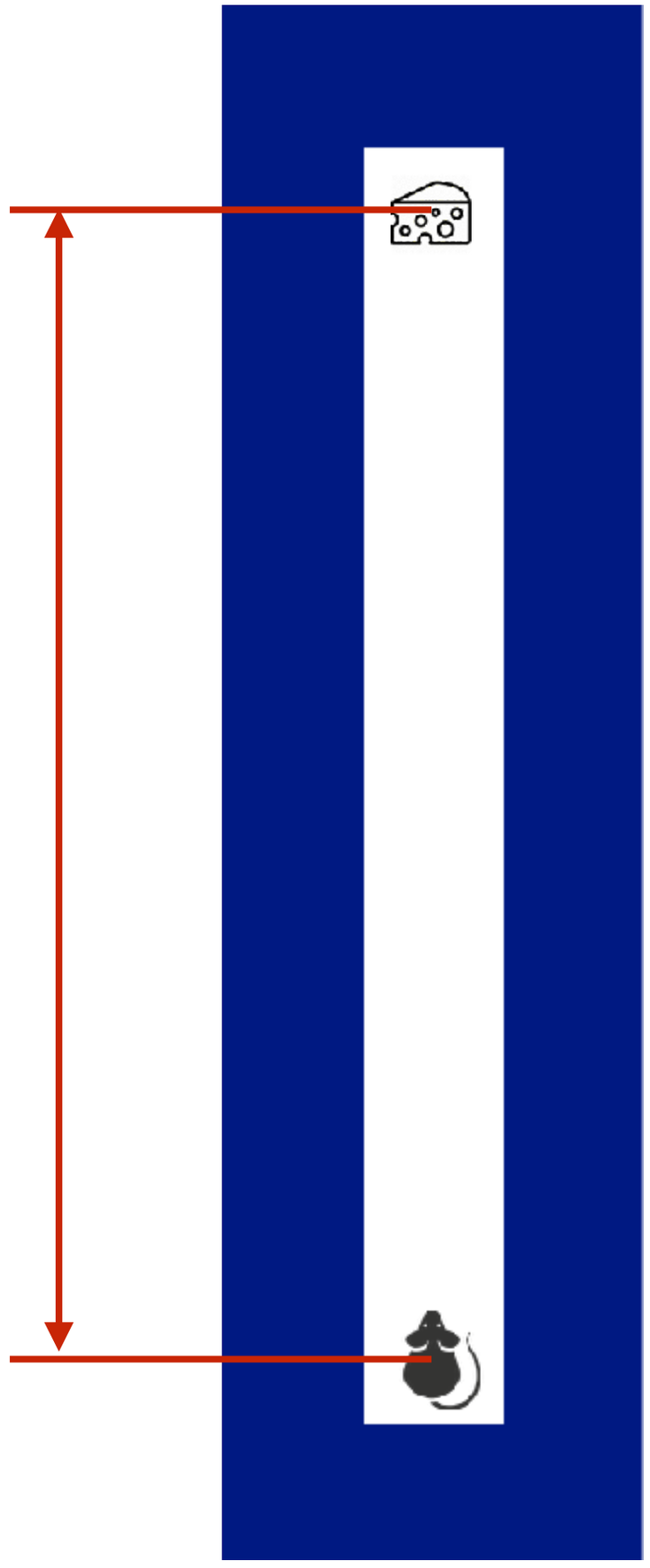


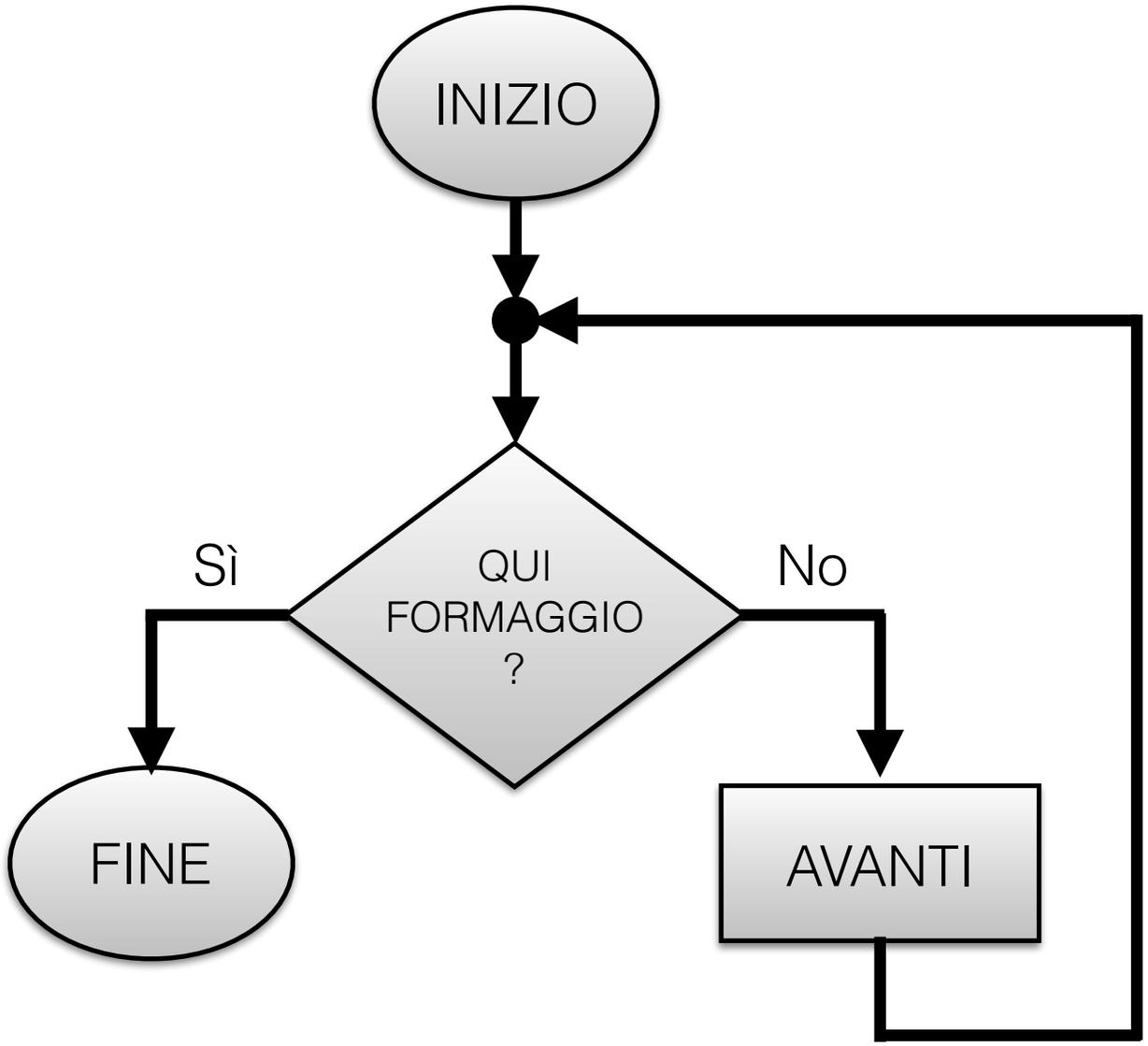


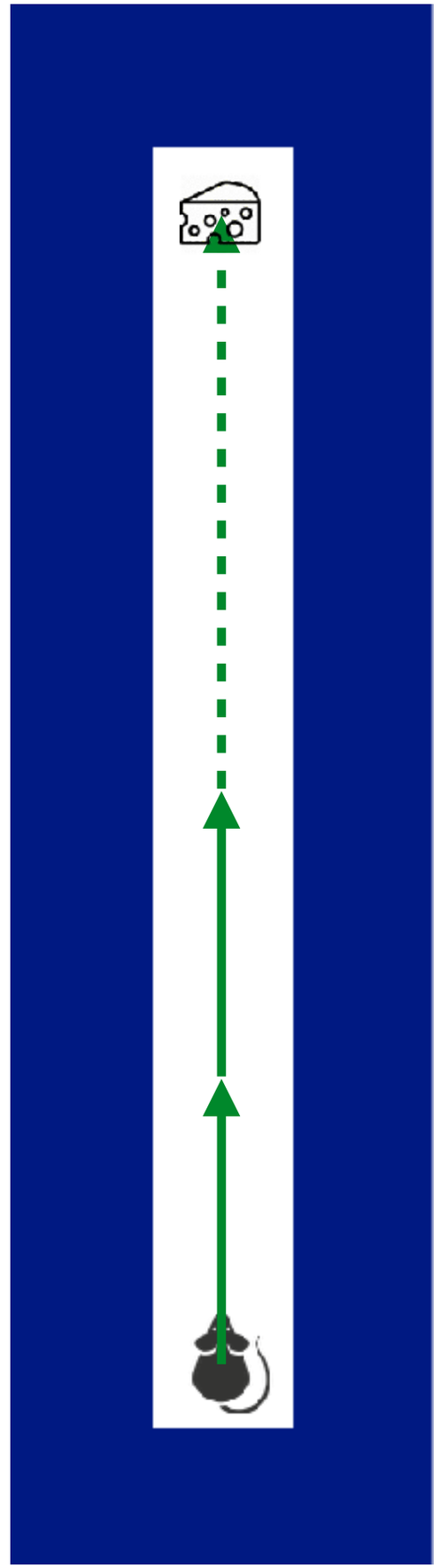


Labirinto dritto, ma di qualunque lunghezza

- dobbiamo trovare il modo di rappresentare un ciclo while con diagrammi di flusso;
- i componenti già definiti sono sufficienti, se sfruttiamo la possibilità di tornare a istruzioni precedenti grazie alle frecce.



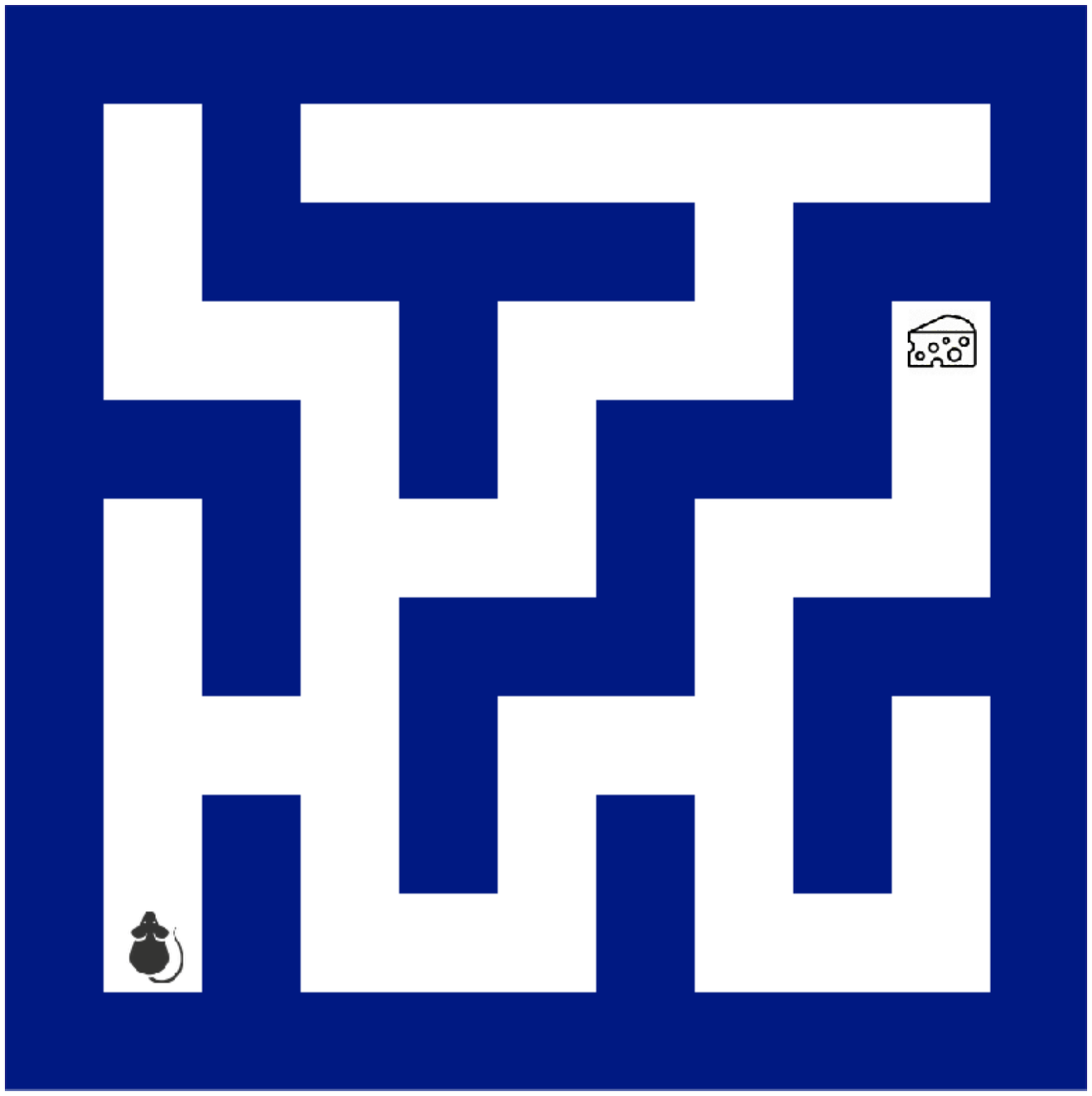




Programma

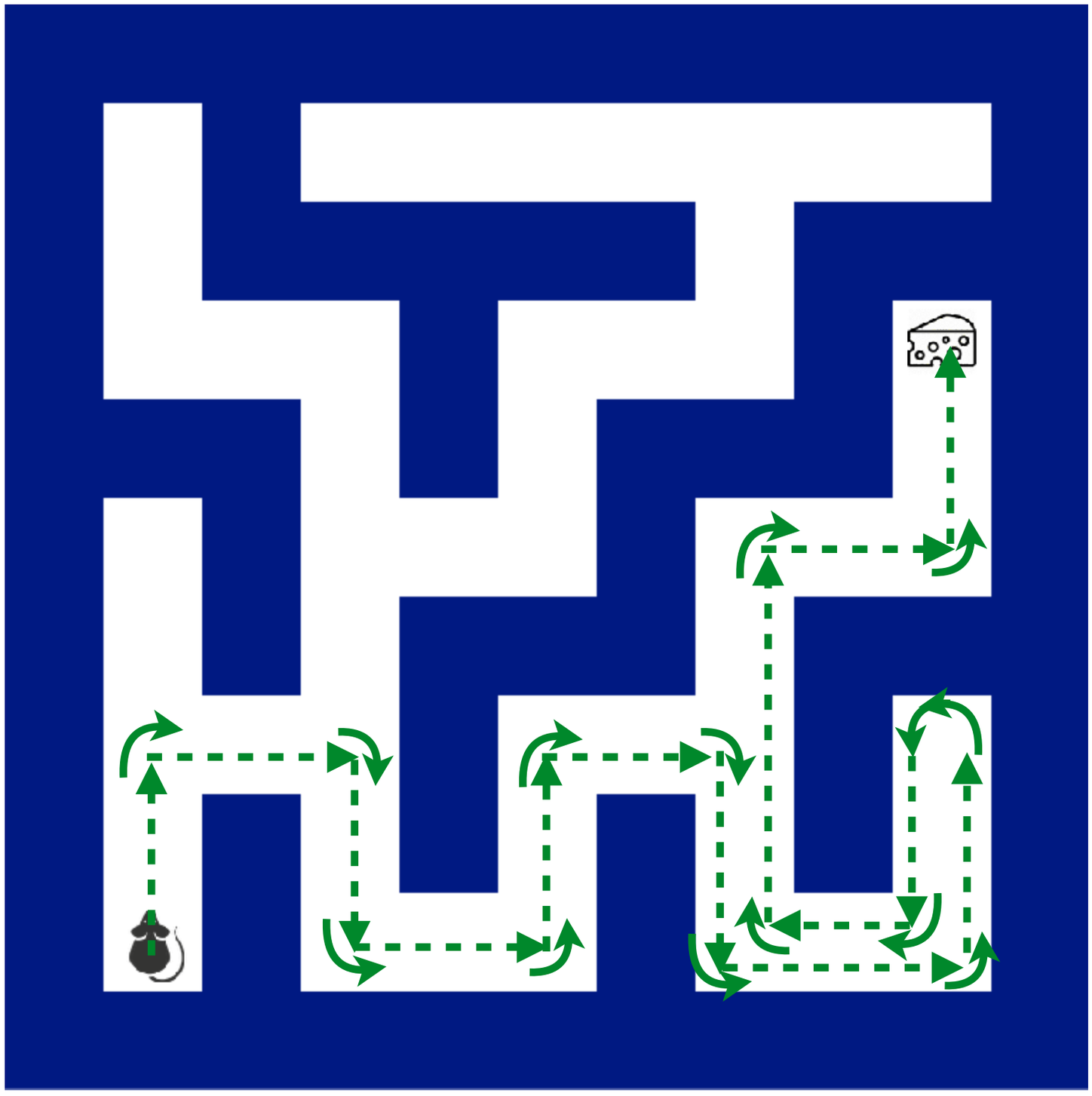
```
WHILE (NOT qui_formaggio()) {
    avanti()
}
```

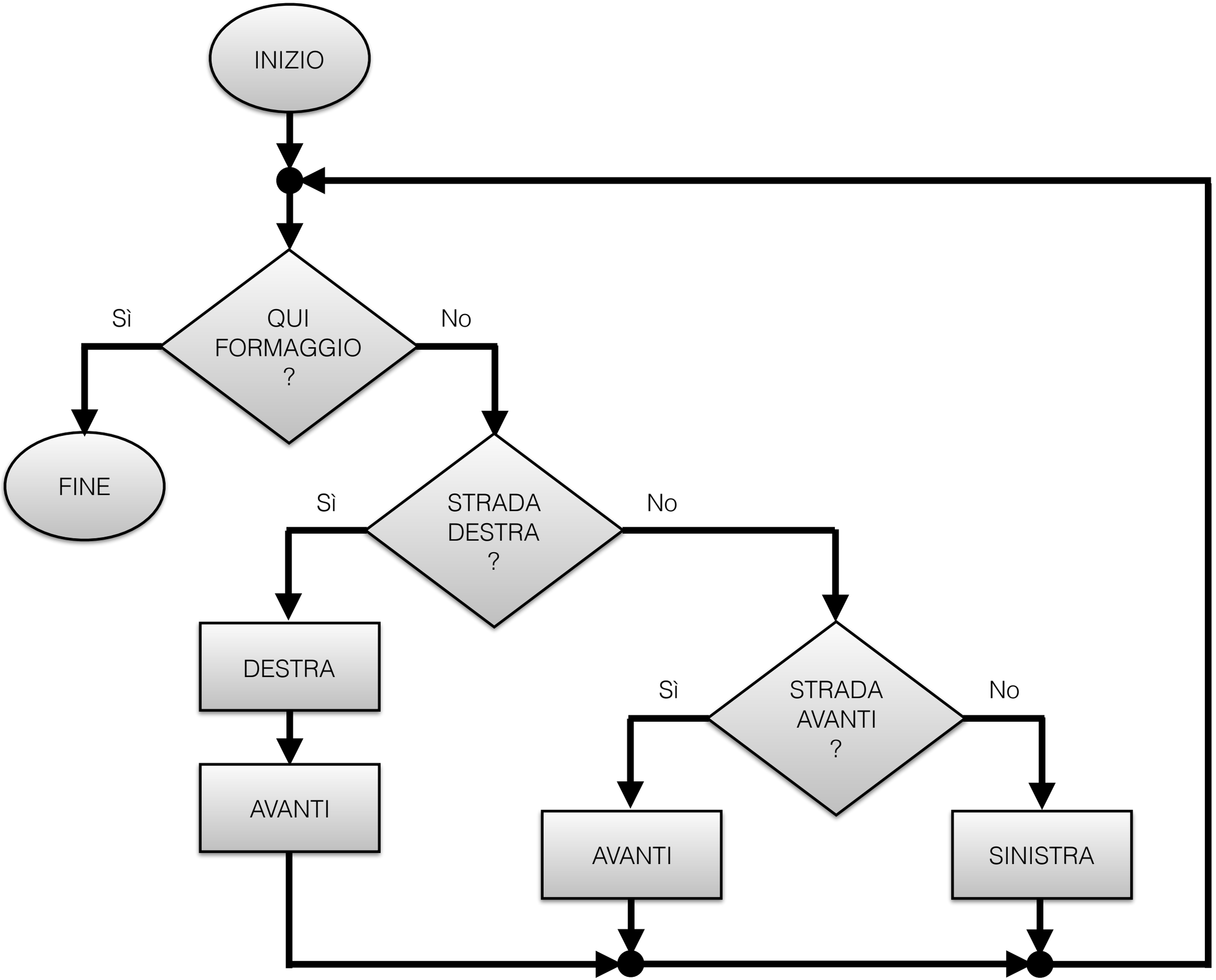
Ramificazioni senza “Circuiti”



Seguendo il Muro di Destra

```
WHILE (NOT qui_formaggio()) {
    IF (strada_destra())
        THEN {
            destra()
            avanti()
        }
        ELSE {
            IF (strada_avanti())
                THEN {
                    avanti()
                }
                ELSE {
                   sinistra()
                }
        }
}
```




INIZIO
QUI FORMAGGIO ?
Sì
No
FINE
STRADA DESTRA ?
Sì
No
DESTRA
AVANTI
STRADA AVANTI ?
Sì
No
AVANTI
SINISTRA