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 16: Linguaggi di Programmazione: If e While

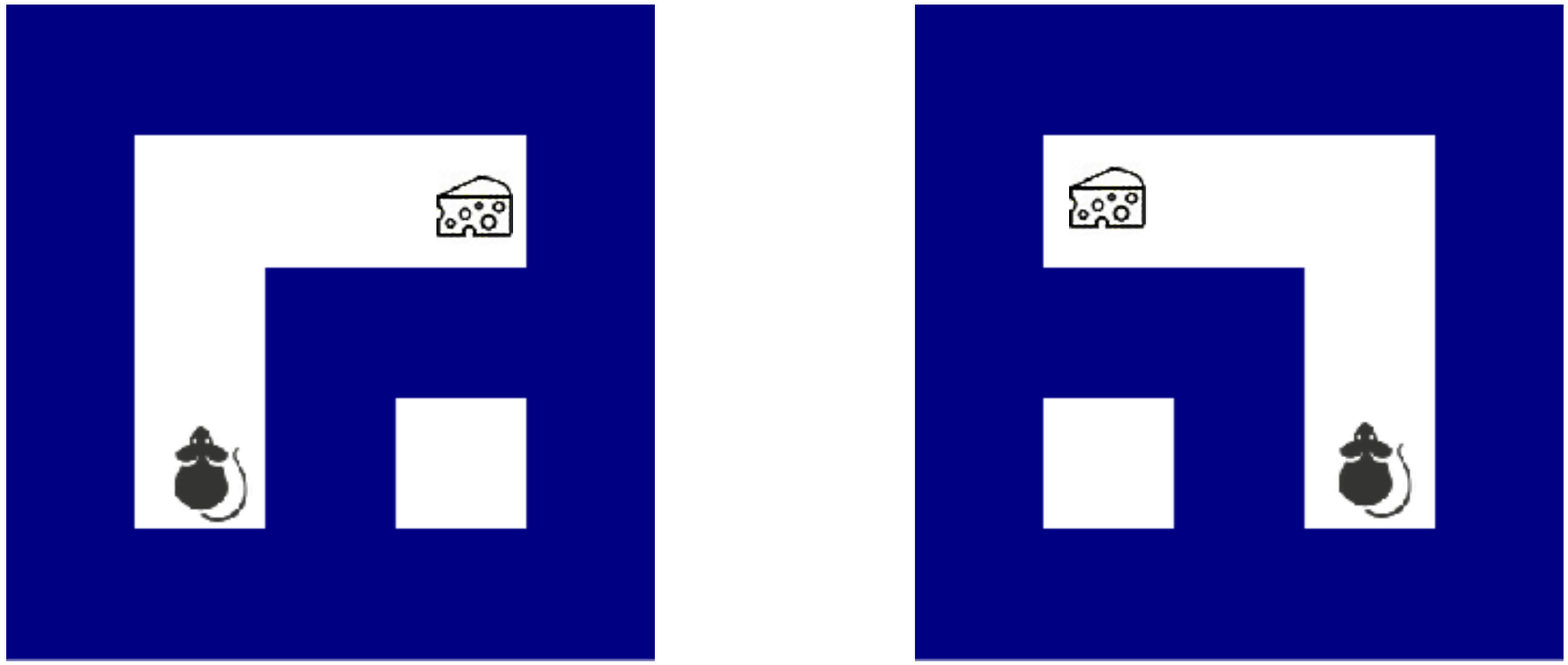
Risolvere Categorie di Labirinti

- nella precedente lezione abbiamo affrontato solo casi singoli, ovvero il programma serviva a risolvere uno specifico labirinto

- il problema generale sarebbe trovare un programma che risolve qualunque labirinto

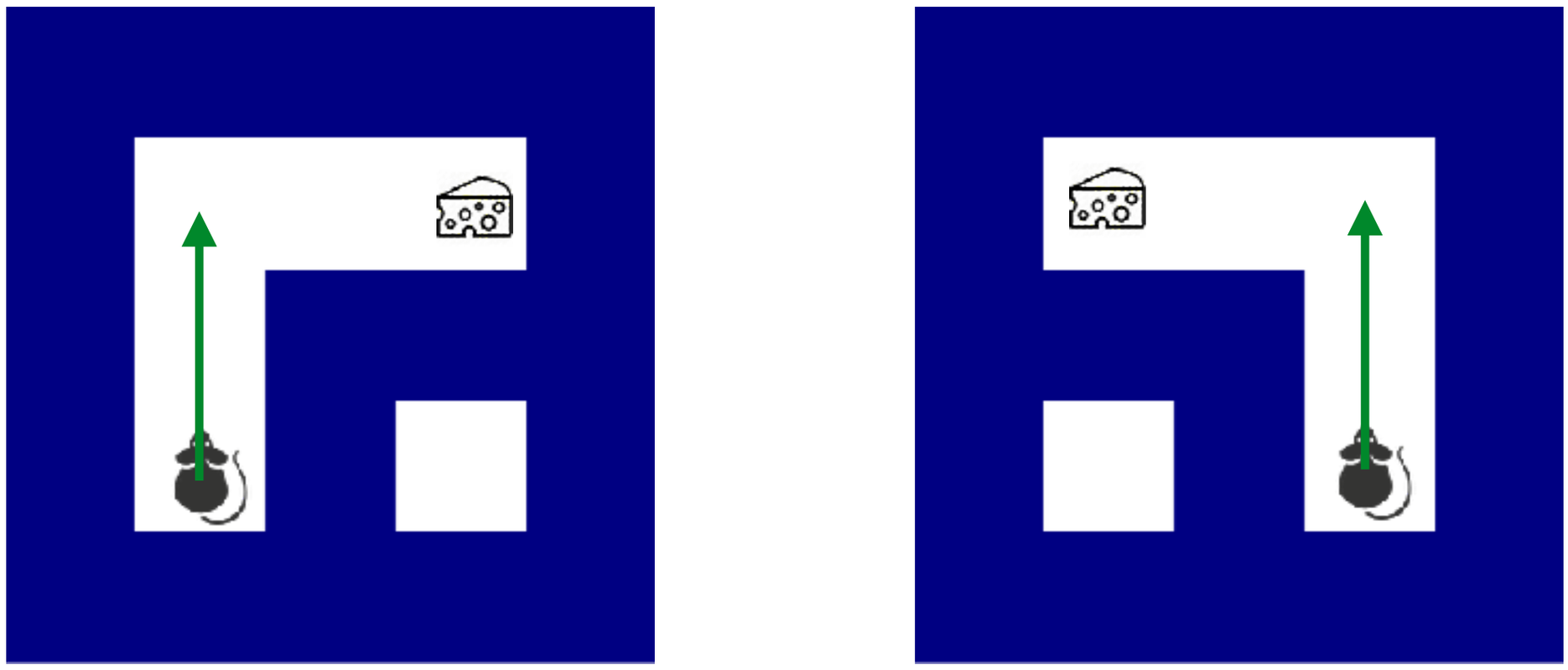
- per ora ci occuperemo di scrivere programmi che risolvano una certa categoria (insieme con certe caratteristiche) di labirinti

“Elle” di lunghezza 2



solo due casi!!!

“Elle” di lunghezza 2



Programma

avanti()

?

devo andare a sx o a dx
a seconda di dove
c’è strada libera

If-Then-Else

- nei programmi ho spesso bisogno di eseguire alcune istruzioni se una condizione è vera e altre se la condizione è falsa
- perciò definisco l'istruzione if-then-else, che in base ad un valore booleano (vero o falso) decide di eseguire una sequenza di istruzioni (then) oppure un'altra (else)
- l'else è lo posso usare oppure no

If-Then-Else: Sintassi

IF (*espr_booleana***)**
THEN {
then_istruzione1
then_istruzione2
…
}
ELSE {
else_istruzione1
else_istruzione2
…
}

dove *espr_booleana* è una espressione che assume valori booleani,
then_istruzione1, *then_istruzione2*, … e *else_istruzione1*, *else_istruzione2*, … sono istruzioni del linguaggio (incluso un'altra **IF (**…**) THEN {**…**} ELSE {**…**}**);
il ramo **ELSE** è facoltativo (posso usare anche la forma **IF (**…**) THEN {**…**}**).

If-Then-Else: Semantica

IF (*espr_booleana***)**
THEN {
then_istruzione1
then_istruzione2
…
}
ELSE {
else_istruzione1
else_istruzione2
…
}

valuta se il valore di *espr_booleana* è vero o falso;
se vero, esegui la sequenza *then_istruzione1*, *then_istruzione2*, …
se falso e se c'è l'**ELSE**, esegui la seq. *else_istruzione1*, *else_istruzione2*, …

Funzioni Predefinite 2/2: Informazioni

ricordiamo che abbiamo queste funzioni predefinite che restituiscono tutte valori booleani:

- **qui_formaggio()**
  restituisce vero se il topo si trova dove c'è il formaggio, falso altrimenti

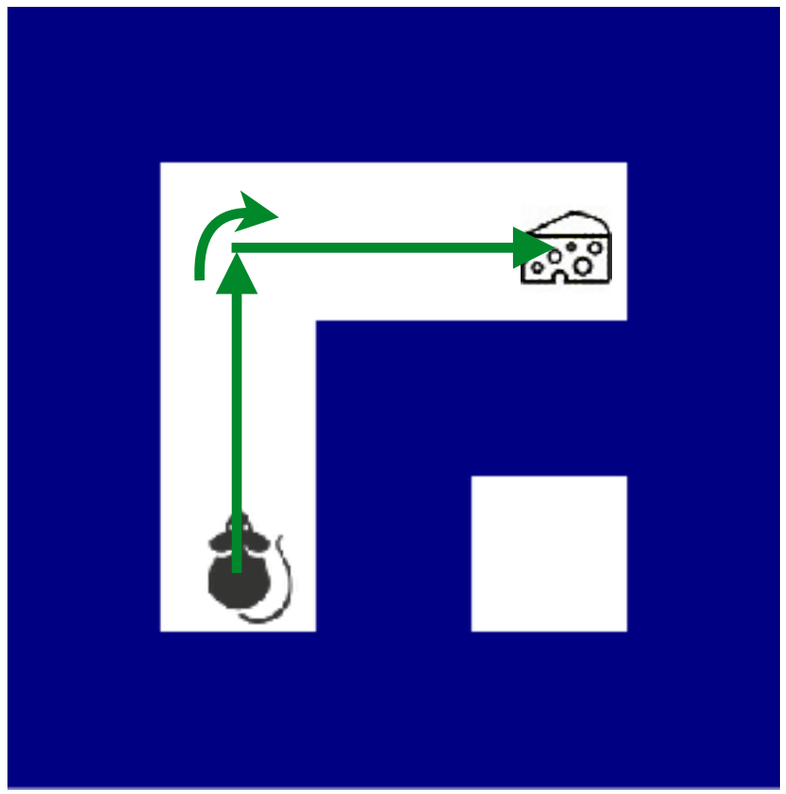
- **strada_avanti()**
  restituisce vero se si può andare avanti (non c'è un muro), falso altrimenti

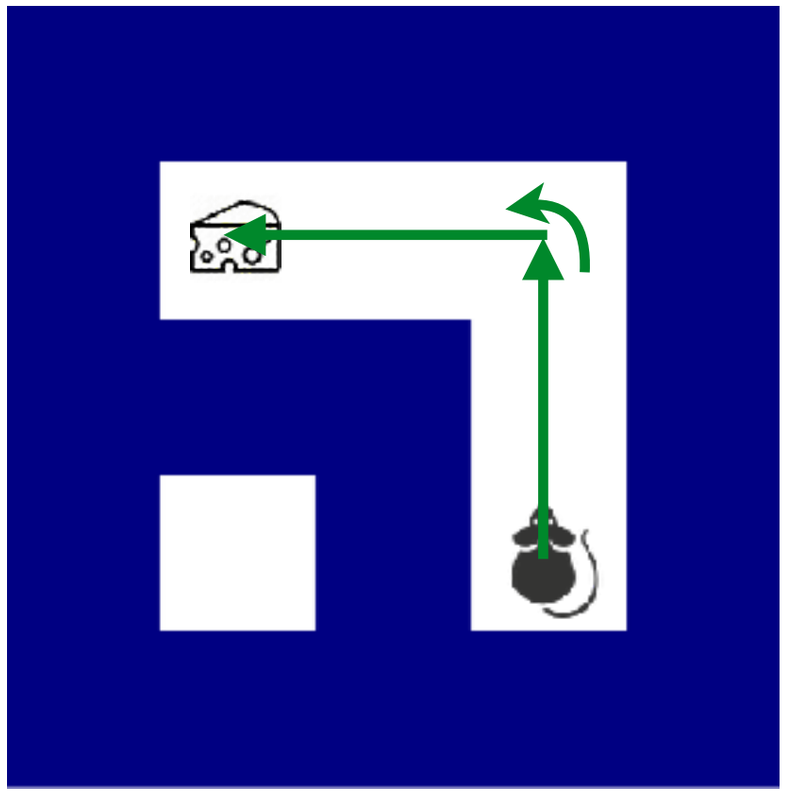
- **strada_destra()**
  restituisce vero se si può andare a destra, falso altrimenti

- **strada_sinistra()**
  restituisce vero se si può andare a sinistra, falso altrimenti

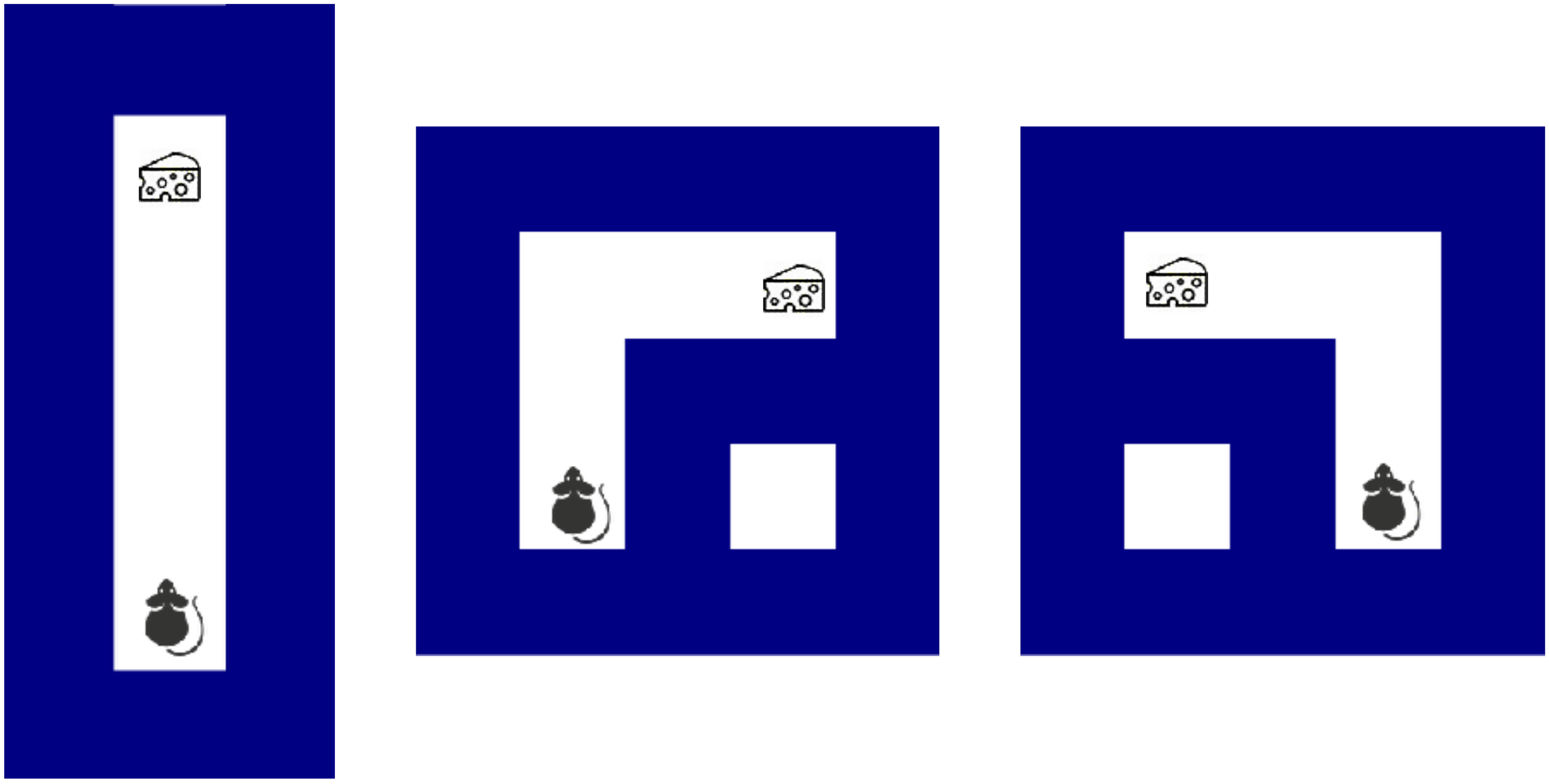
Programma

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





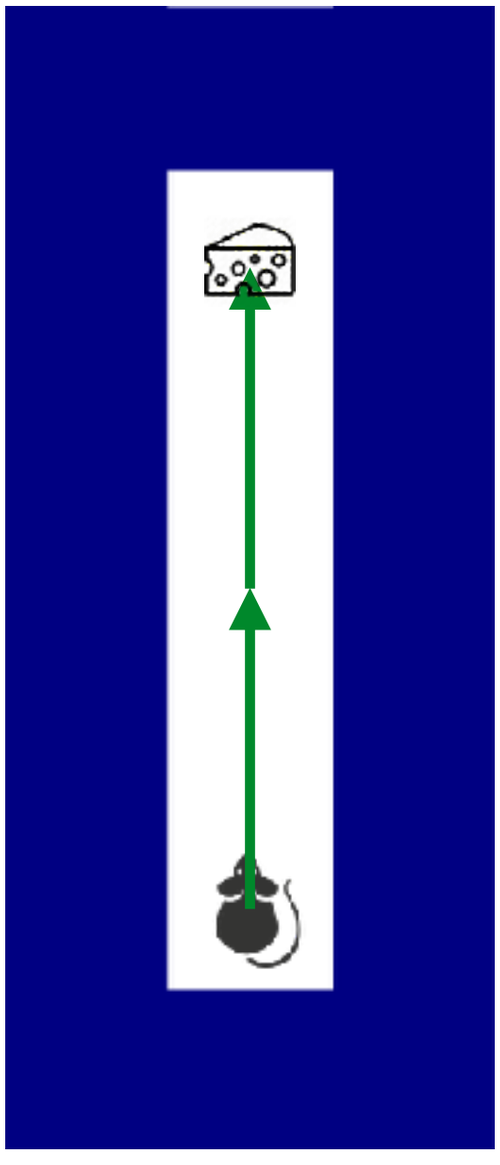
Un altro esempio:
Percorso di lunghezza 2

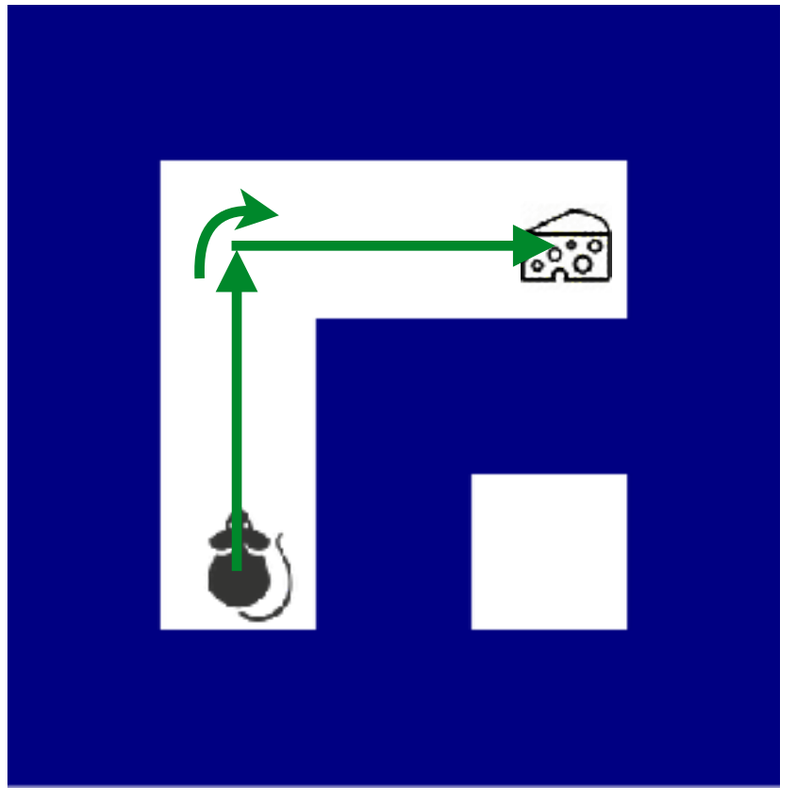


consideriamo questi 3 casi

Programma

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





Operatori Booleani: NOT

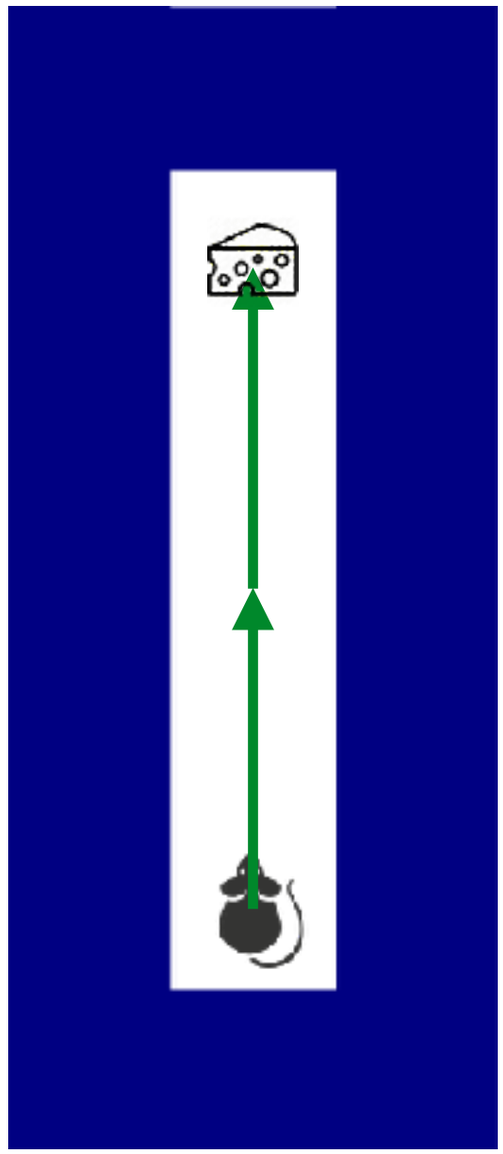
- capita che devo “manipolare” dei valori booleani
- per esempio mi interessa quando una cosa non è vera, uso l’operatore NOT

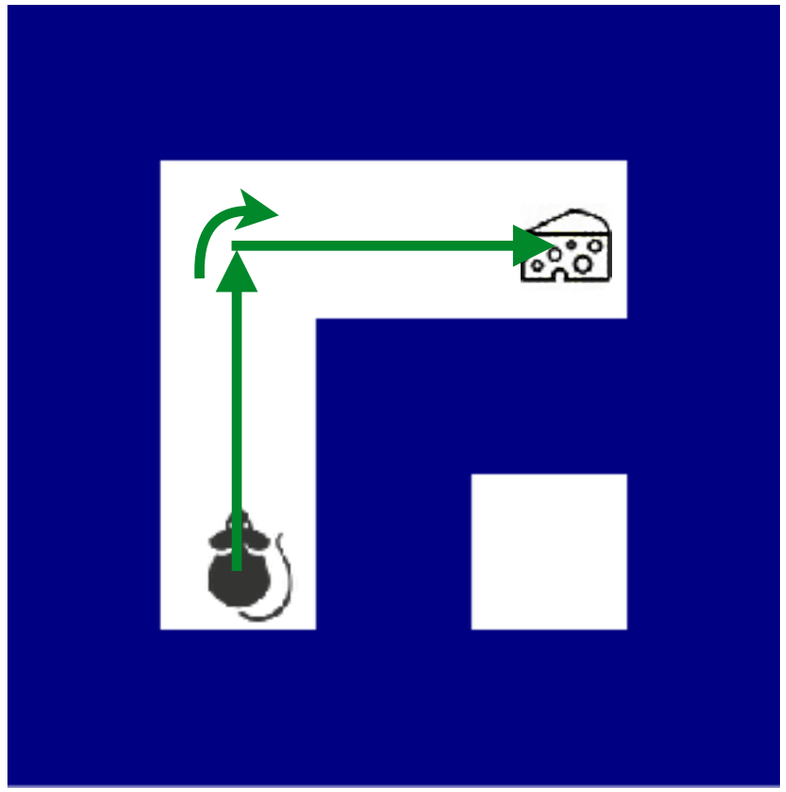
- esempio

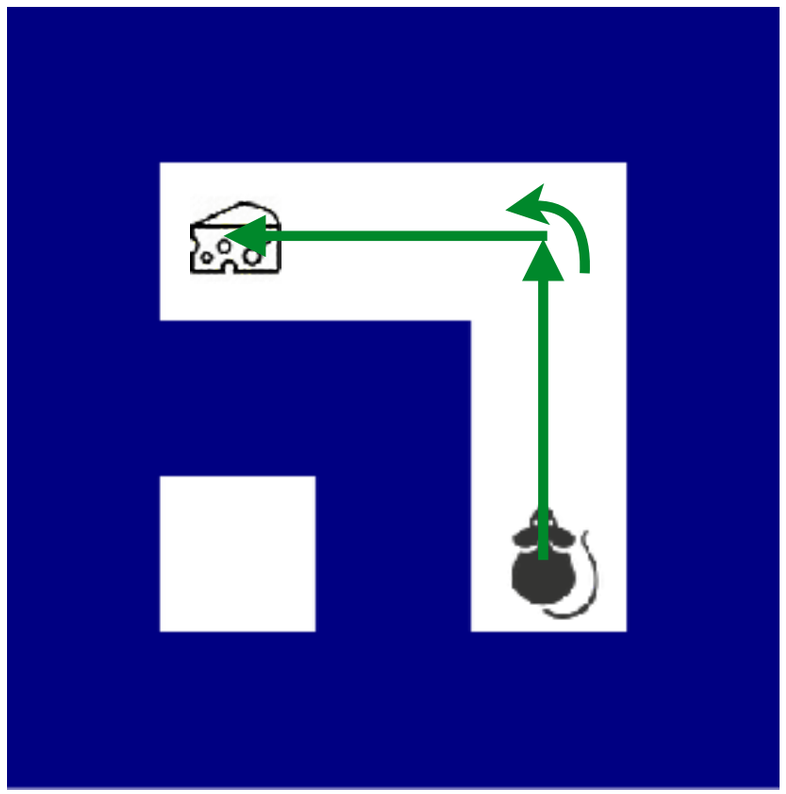
```
IF (NOT piove())
    THEN {
        prendi_bici()
        pedala()
    }
```

Programma (usando NOT)

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







Categorie di Labirinti

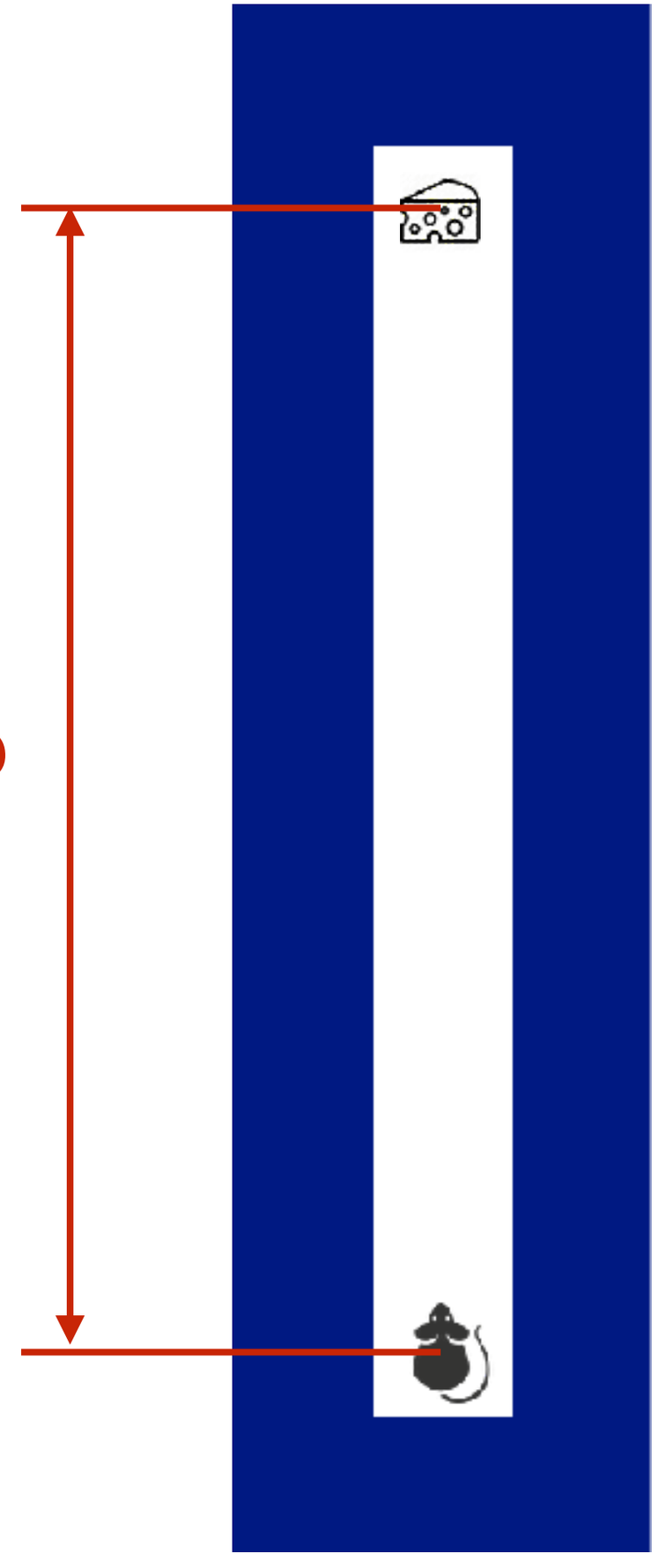
- nella precedente lezione abbiamo affrontato solo casi singoli, ovvero il programma serviva a risolvere uno specifico labirinto

- il problema generale sarebbe trovare un programma che risolve qualunque labirinto

- per ora ci occuperemo di scrivere programmi che risolvano una certa categoria (insieme con certe caratteristiche) di labirinti

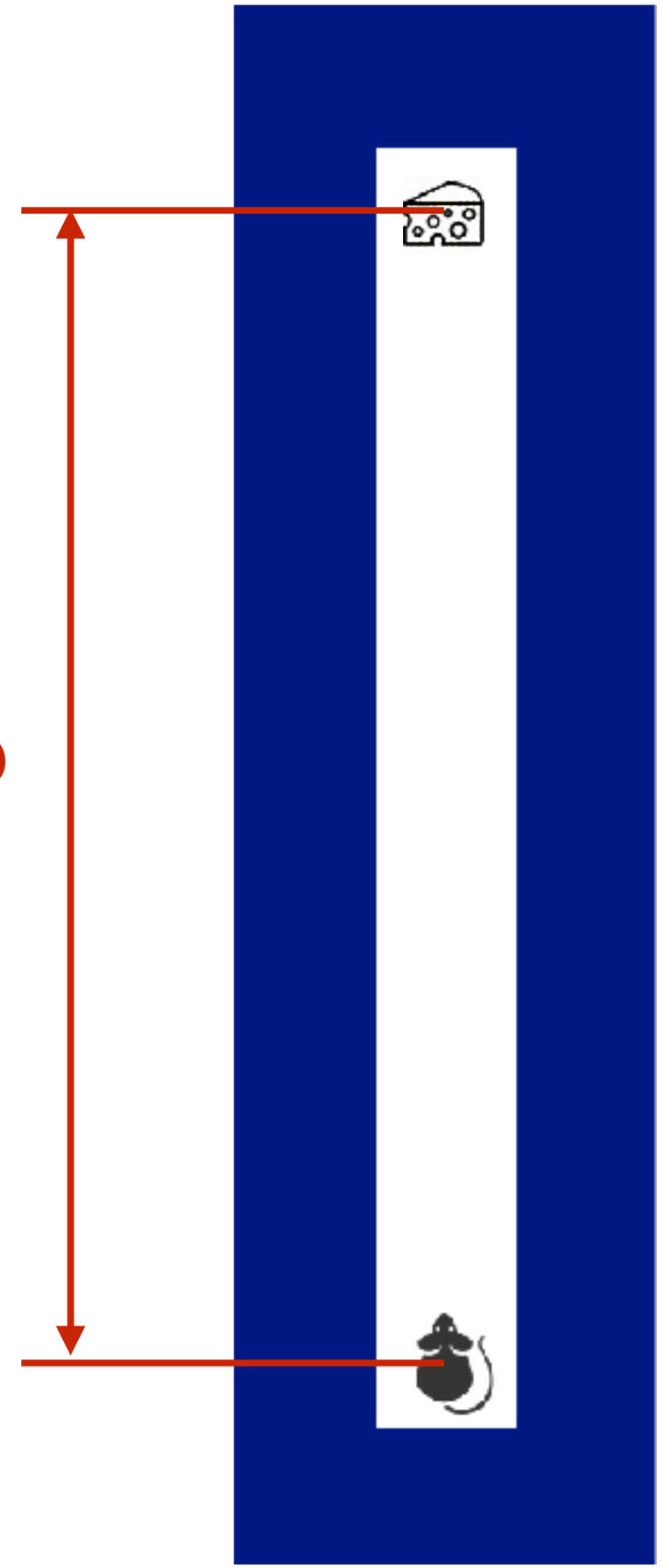
Labirinto dritto, ma di qualunque lunghezza

- per risolvere il problema più in generale non possiamo limitare la dimensione dell'input (labirinto)
- se la dimensione dell'input è variabile, il caso più semplice è un unico tratto dritto, di cui però non conosciamo la lunghezza



Come fare?

- dobbiamo ripetere tante volte avanti(), fino a che non arriviamo
- un ciclo!
- ma nei cicli che abbiamo visto finora (RIPETI e FOR) dobbiamo stabilire in anticipo quante ripetizioni
- qui all'inizio non sappiamo quante ripetizioni sono



While

- spesso (quando ho un'input di dimensione variabile) devo eseguire un ciclo fino quando una condizione è verificata (es., il fatto che non sono ancora arrivato al formaggio)
- il while utilizza un espressione booleana come l'if-then, ma in questo caso un blocco di istruzioni viene ripetuto ciclicamente fino a quando l'espressione vale vero

While: Sintassi

WHILE (*espr_booleana***) {**
istruzione1
istruzione2
…
}

dove *espr_booleana* è una espressione che assume valori booleani,
istruzione1, *istruzione2*, … sono istruzioni del linguaggio (incluso un'altra **WHILE (**…**) {**…**}**).

While: Semantica

WHILE (*espr_booleana***) {**

istruzione1

istruzione2

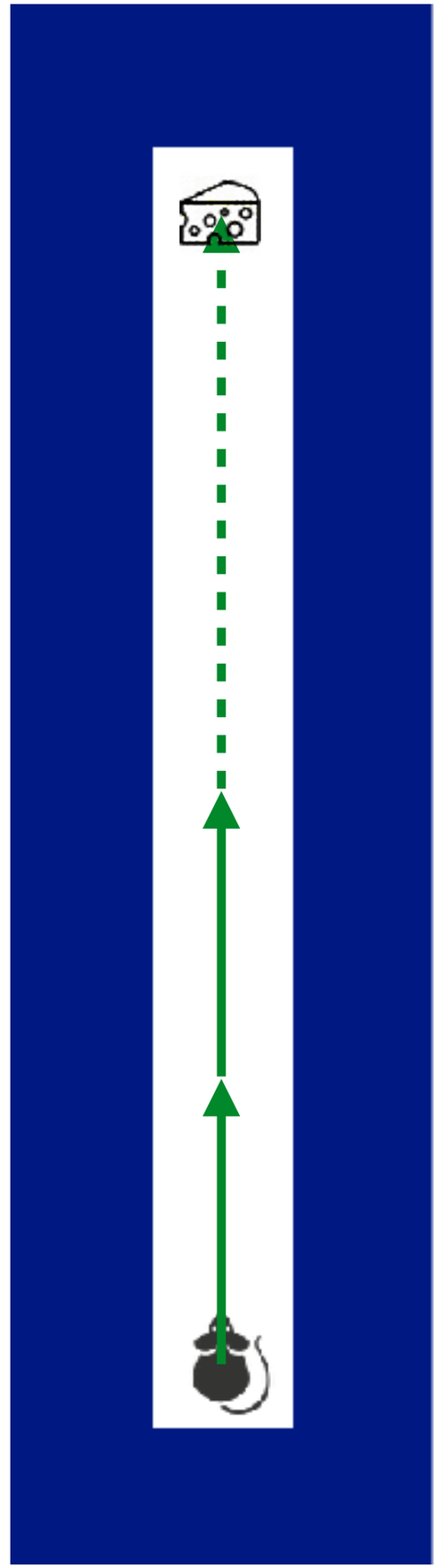
…

}

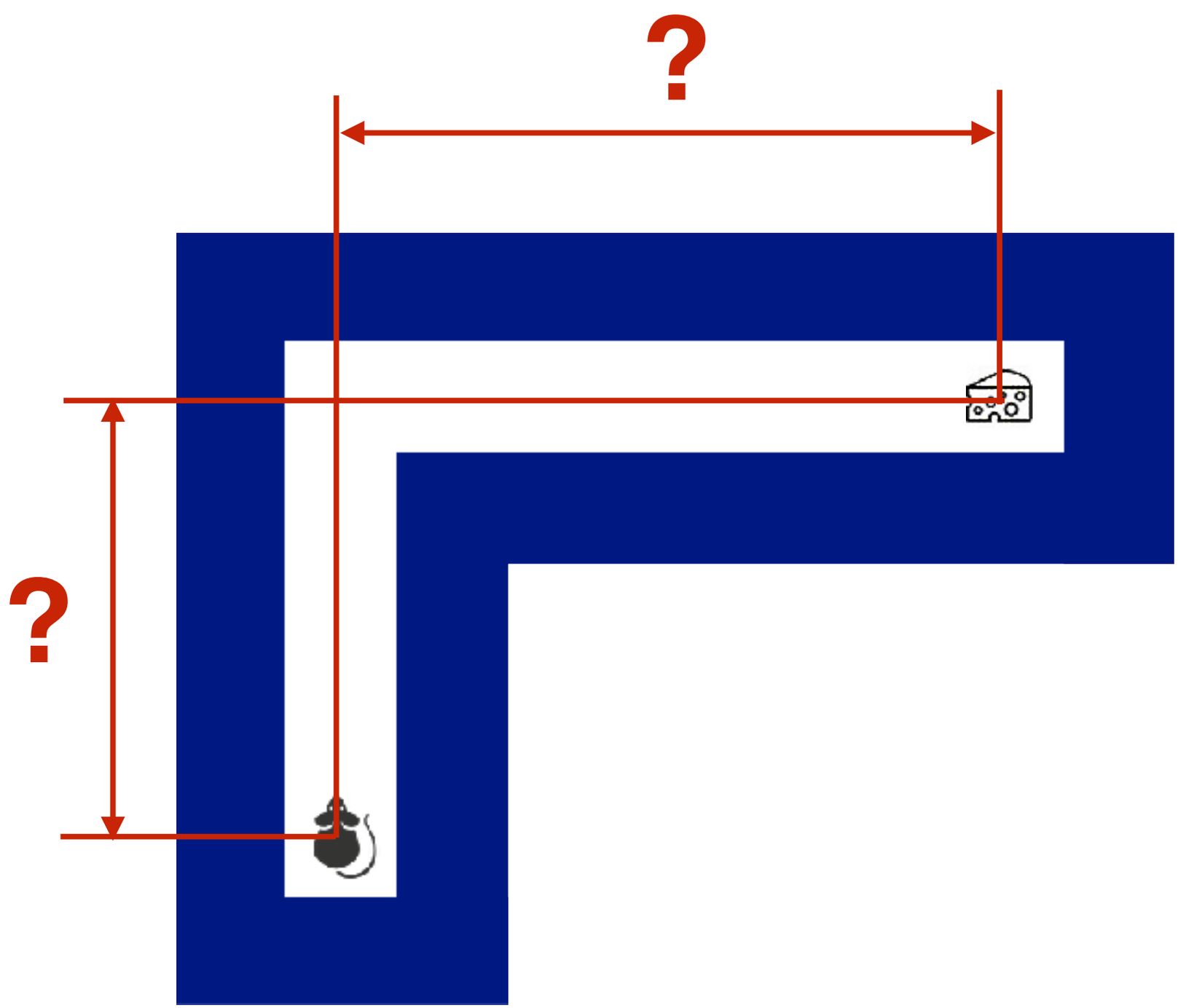
1. valuta se il valore di *espr_booleana* è vero o falso
2. se falso, esci (passa all'istruzione successiva al **WHILE**)
3. se vero, esegui la sequenza *istruzione1*, *istruzione2*, …
4. torna a 1

Programma

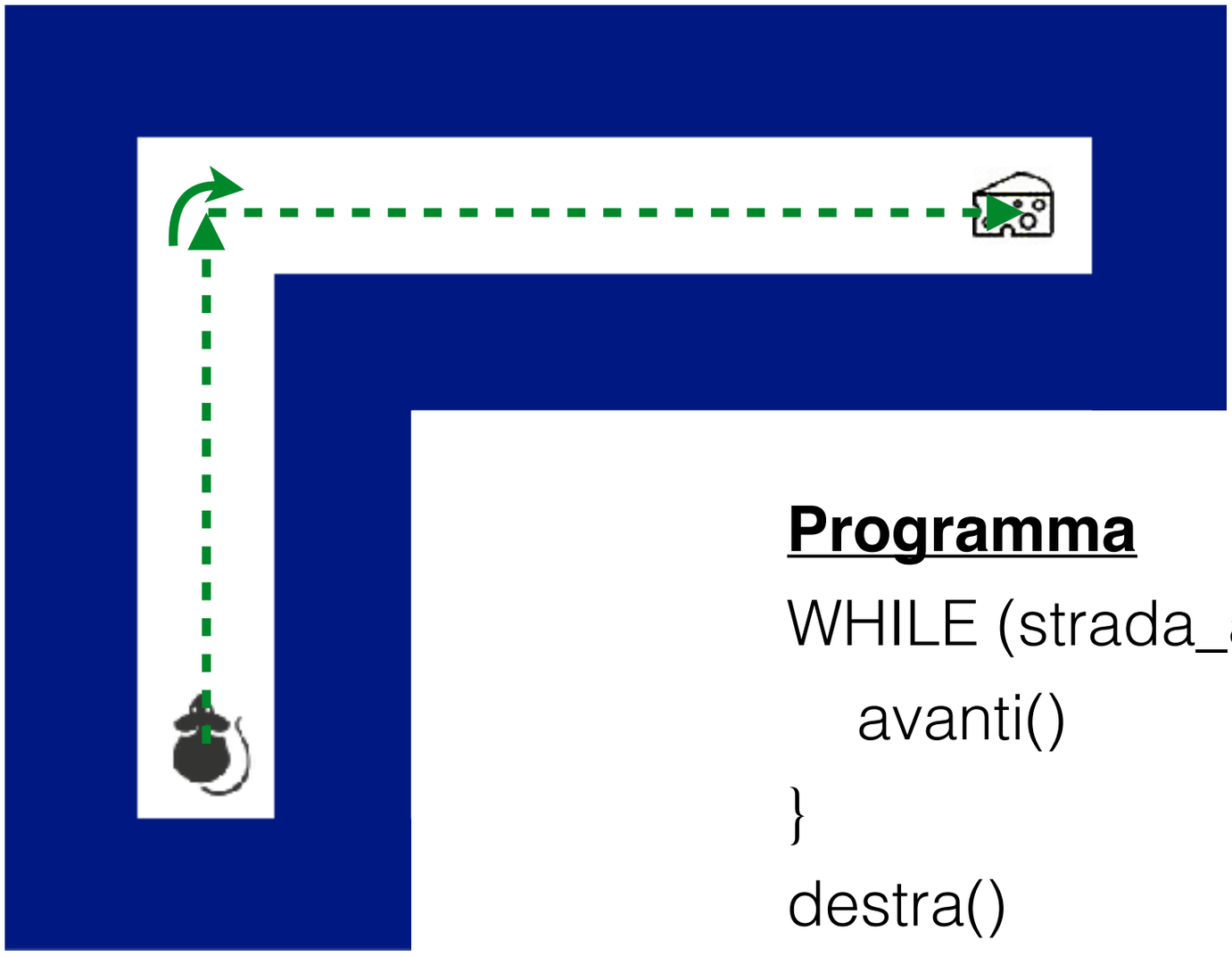
```
WHILE (NOT qui_formaggio()) {
    avanti()
}
```



Elle a destra
di qualunque lunghezza



Elle a destra
di qualunque lunghezza



Programma

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