Esercitazione 1: Agenti

Fondamenti di Intelligenza Artificiale

SAPIENZA
UNIVERSITÀ DI ROMA

A.A. 2022/2023

DIAG, Dipartimento di Ingegneria Informatica, Automatica, e Gestionale "Antonio Ruberti".

Installazione pygame

Installazione con pip:

```
pip3 install pygame
```

Installazione con anaconda:

```
conda install pygame
```

Si consigliano le versioni 3.7 di python e 2.3.0 di pygame

Repository codice esercitazioni

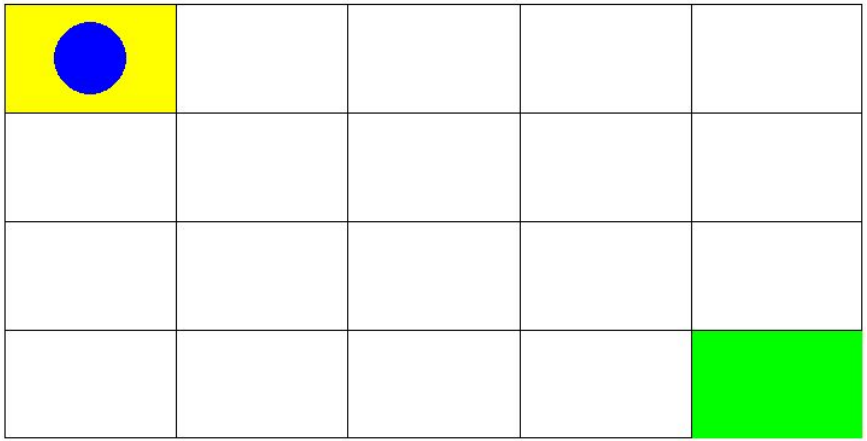
link: https://github.com/KRLGroup/FondamentiIA-2223

Download della repo con git (opzionale):

```
git clone https://github.com/KRLGroup/FondamentiIA-2223.git
```

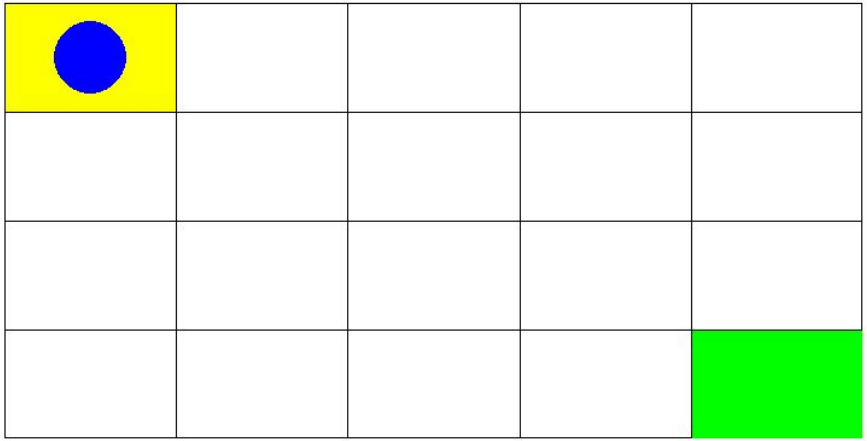
Il codice per questa esercitazione è in “`esercitazione01.py`”

Esercizio 1: environment



- goal: raggiungere la cella in basso a destra (4,3) partendo dalla cella in alto a sinistra (0,0)
- azioni: “N”, “S”, “E”, “W”, None
- osservabili: x, y, x_max, y_max

Esercizio 1: environment



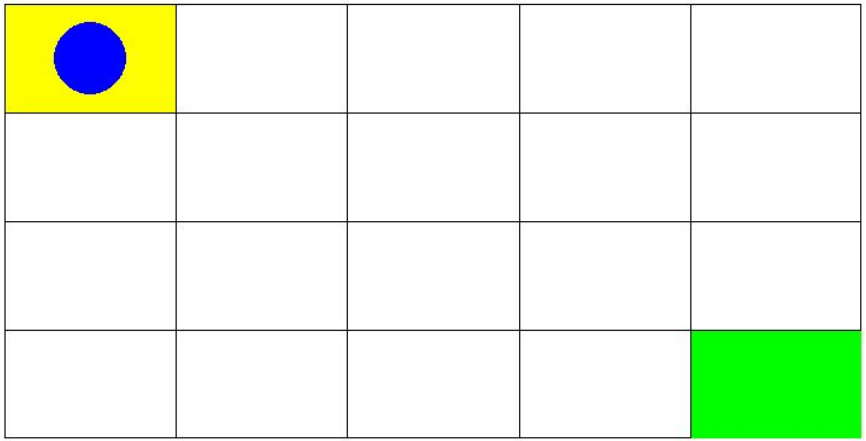
- misura di performance: numero di step necessari (da minimizzare)

Esercizio 1

Risolvere con un agente reattivo semplice (completare codice)

```
def simple_reflex_action(x, y, x_max, y_max):
    # da completare: ritornare l'azione da seguire
    pass
```

Esercizio 2: environment



- come nell'es 1, ma x_max e y_max **non sono osservabili**

Esercizio 2

A) risolvere con un agente reattivo model-based

```
# Note:
# - state è un dict che contiene le key "x", "y", "x_max", "y_max"
# - state["x_max"] e state["y_max"] sono inizialmente None (ignoti)
# - model è una funzione tale che model(state, action) "predice" la
#   prossima posizione (x, y) dell'agente, assumendo che parta da
#   state e segua l'azione action (considerando anche i limiti della
#   griglia)
def update_state(state, last_action, x, y, model):
    state["x"], state["y"] = x, y
    # da completare: aggiornare state["x_max"] e state["y_max"]
    pass

def model_based_reflex_action(state):
    x, y = state["x"], state["y"]
    x_max, y_max = state["x_max"], state["y_max"]
    # da completare: ritornare l'azione da seguire
    # suggerimento: una volta che x_max e y_max sono noti, si possono
    # seguire le stesse azioni dell'es1
    pass
```

Esercizio 2

B) la soluzione fornita è razionale rispetto alla misura di performance? Se sì, motivare. Se no, modificarla affinché lo sia

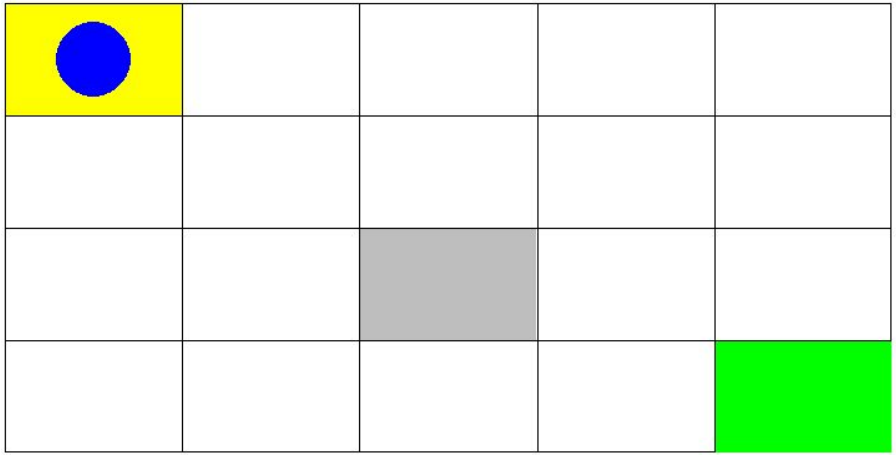
C) esiste una soluzione razionale con un agente reattivo semplice? Motivare la risposta

Esercizio 3

Quale dei 4 tipi di agente è più adatto al caso in cui si voglia passare per una cella intermedia prima di raggiungere il goal?

Motivare la risposta

Esercizio 4: environment



- come es 1, ma viene aggiunta una cella “piena” che l’agente **non può visitare**
- vengono anche aggiunte le azioni di movimento diagonale: “NE”, “NW”, “SE”, “SW”

Esercizio 4

A) risolvere con un agente razionale utility-based

```
es4_action_space = ["N", "S", "E", "W", "NE", "NW", "SE", "SW"]

# Note:
# - state è un dict che contiene:
#   - la posizione dell'agente in "x" e "y"
#   - la posizione del goal: "x_max" e "y_max"
def utility(state):
    # da completare: assegnare un valore di utility a state
    return 0.

# Note:
# - model è una funzione tale che model(state, action) ritorna un nuovo
#   dict next_state contenente lo stato a cui si arriva partendo da
#   state ed eseguendo action
def utility_based_action(state, model):
    # da completare: utilizzare model e la funzione utility definita
    # sopra per decidere l'azione migliore (i.e. che arriva in uno stato
    # a utility maggiore)
    pass
```

Esercizio 4

B) nel caso in cui non siano disponibili le azioni di movimento diagonale, sarebbe comunque possibile risolvere il task per **tutte le possibili configurazioni** dell'ostacolo?

Motivare la risposta e, in caso negativo, fornire una possibile modifica all'agente