Esercizio guidato

Settimana 13
10/01/2023

Si ringrazia il Dott. Giacomo Baruzzo per il materiale

Esercizio - Tema d'esame (2 ore e mezza)

Si implementi un programma che simula, in una **griglia quadrata 2D** di **LxL celle** (non è necessario visualizzarla), l'interazione tra **cellule batteriche** e **cellule del sistema immunitario**. La griglia è occupata in una certa **percentuale da cellule batteriche** e in un'altra **percentuale da cellule del sistema immunitario** (percentuali date in input dall'utente). Le celle rimanenti sono da considerare "libere".

Il tempo viene simulato attraverso un ciclo for e una variabile di iterazione t per un numero di iterazioni **t_max** fornito dall'utente. Ogni iterazione è divisa in **due fasi**. Quando la fase 1 è completata, inizia la fase 2.

FASE 1: ogni cellula (batterica e del sistema immunitario) si sposta di posizione in una casella nel suo intorno a una distanza massima di **raggio rmov** fornito dall'utente. Se nel raggio di azione non ci sono posizioni libere la cellula non si muove.

FASE 2: ogni cellula batterica che abbia raggiunto **volume >= 2** si divide (vd. DIVISIONE CELLULARE), altrimenti aumenta il suo **volume del 30%**. A questo punto, se una cellula batterica ha come vicine delle cellule del sistema immunitario, viene attaccata a turno da ciascuna delle vicine con una **probabilità di morire del 20%**. Se muore ovviamente non ci sono ulteriori attacchi. Se la cellula del sistema immunitario vince contro il batterio, il batterio scompare dalla griglia e al suo posto appare una nuova cellula del sistema immunitario. Dopo **9 attacchi** (con o senza successo) la cellula del sistema immunitario muore.

DIVISIONE CELLULARE: Le cellule batteriche, quando nascono, hanno volume iniziale = 1 e, quando raggiungono volume >= 2 si dividono in due cellule figlie ciascuna di volume 1. Una resta nella posizione della cellula genitore, l'altra compare nella griglia in una posizione random che non sia però occupata da cellule batteriche o del sistema immunitario. Se non ci sono più posizioni libere, la seconda cellula non compare nella griglia. La seconda cellula non subisce attacchi nella iterazione in cui è stata creata.

Il codice seguente implementa il programma **BACTvsIS.py** che fa uso di due classi: **Batterio** e **IScell**, entrambe figlie della superclasse **Cellula**.

Il programma salva in un file, ad ogni iterazione, il **numero di cellule batteriche** e il **numero di cellule del sistema immunitario** presenti nel sistema. Implementare il codice che definisce le tre classi in modo che il programma funzioni correttamente. Il codice implementato va salvato nel file **cellule.py**

Codice python (parte 1)

Primo passo: analisi del codice fornito per dedurre le specifiche per la progettazione delle classi

```
import random
from cellule import Cellula, Batterio, IScell

#inizializza la griglia 2D L per L con 0 in corrispondenza di cella vuota,
#e oggetti di tipo Batterio o IScell altrimenti con probabilità specificate
#dai parametri perc_batt e perc_is, rispettivamente
# @L lato griglia quadrata
# @perc_batt % batteri iniziale (un numero tra 0 e 1)
# @perc_is % cellule IS iniziale (un numero tra 0 e 1)
def gridinit(L = 20, perc_is = 0.1, perc_batt = 0.01):
  grid = []
  new = []
  for i in range(L):
    for j in range(L):
      if random.random() <= perc_is:
        new.append(IScell(i,j))
      elif random.random() <= perc_batt:
        new.append(Batterio(i,j))
      else:
        new.append(Cellula.EMPTY)
    grid.append(new)
    new = []
  return (grid)
```

Codice python (parte 1)

```
import random
from cellule import Cellula, Batterio, IScell

#inizializza la griglia 2D L per L con 0 in corrispondenza di cella vuota,
#e oggetti di tipo Batterio o IScell altrimenti con probabilità specificate
#dai parametri perc_batt e perc_is, rispettivamente
# @L lato griglia quadrata
# @perc_batt % batteri iniziale (un numero tra 0 e 1)
# @perc_is % cellule IS iniziale (un numero tra 0 e 1)
def gridinit(L = 20, perc_is = 0.1, perc_batt = 0.01):
  grid = []
  new = []
  for i in range(L):
    for j in range(L):
      if random.random() <= perc_is:
        new.append(IScell(i,j))
      elif random.random() <= perc_batt:
        new.append(Batterio(i,j))
      else:
        new.append(Cellula.EMPTY)
    grid.append(new)
    new = []
  return (grid)
```

Scorro la griglia per righe

Inserisco nella griglia un oggetto IScell

Inserisco nella griglia un oggetto Batterio

Inserisco nella griglia una cella vuota

Codice python (parte 1)

```
import random
from cellule import Cellula, Batterio, IScell

#inizializza la griglia 2D L per L con 0 in corrispondenza di cella vuota,
#e oggetti di tipo Batterio o IScell altrimenti con probabilità specificate
#dai parametri perc_batt e perc_is, rispettivamente
# @L lato griglia quadrata
# @perc_batt % batteri iniziale (un numero tra 0 e 1)
# @perc_is % cellule IS iniziale (un numero tra 0 e 1)
def gridinit(L = 20, perc_is = 0.1, perc_batt = 0.01):
  grid = []
  new = []
  for i in range(L):
    for j in range(L):
      if random.random() <= perc_is:
        new.append(IScell(i,j))
      elif random.random() <= perc_batt:
        new.append(Batterio(i,j))
      else:
        new.append(Cellula.EMPTY)
    grid.append(new)
    new = []
  return (grid)
```

Primo passo: analisi del codice fornito per dedurre le specifiche per la progettazione delle classi

NOTE per la progettazione delle classi:

- Costruttore IScell(i,j)
- Costruttore Batterio(i,j)
- Cella libera codificata con Cellula.EMPTY

Codice python (parte 2)

```
# itera tmax volte un processo di
# @ 1) movimento di tutte le cellule batteriche e IS
# @ 2) azione delle cellule batteriche (dividersi, morire, crescere)
# e rispettiva azione delle cellule IS se queste attaccano

def itera(t_max, grid, rmov, filename):
  L = len(grid)
  outfile = open(filename, "w")
  for t in range(t_max):
    print(t,"\t", Batterio.contaBatteri, "\t",IScell.contaIS)
    outfile.write(str(Batterio.contaBatteri)+"\t"+str(IScell.contaIS)+"\n")
    for i in range(L):
      for j in range(L):
        if (grid[i][j]!=Cellula.EMPTY):
          grid[i][j].move(grid, rmov)
    for i in range(L):
      for j in range(L):
        if (grid[i][j]!=Cellula.EMPTY):
          grid[i][j].act(grid)
  outfile.close()
```

Codice python (parte 2)

```
# itera tmax volte un processo di
# @ 1) movimento di tutte le cellule batteriche e IS
# @ 2) azione delle cellule batteriche (dividersi, morire, crescere)
# e rispettiva azione cellule IS se queste attaccano

def itera(t_max, grid, rmov, filename):
  L = len(grid)
  outfile = open(filename, "w")
  for t in range(t_max):
    print(t,"\t", Batterio.contaBatteri, "\t", IScell.contaIS)
    outfile.write(str(Batterio.contaBatteri)+"\t"+str(IScell.contaIS)+"\n")

    for i in range(L):
      for j in range(L):
        if (grid[i][j]!=Cellula.EMPTY):
          grid[i][j].move(grid, rmov)

    for i in range(L):
      for j in range(L):
        if (grid[i][j]!=Cellula.EMPTY):
          grid[i][j].act(grid)
  outfile.close()
```

Per ogni istante temporale

Scorro griglia per righe

Fase 1: movimento cellule

Scorro griglia per righe

Fase 2: azione (duplicazione o attacco)

Codice python (parte 2)

```
# itera tmax volte un processo di
# @ 1) movimento di tutte le cellule batteriche e IS
# @ 2) azione delle cellule batteriche (dividersi, morire, crescere)
# e rispettiva azione cellule IS se queste attaccano

def itera(t_max, grid, rmov, filename):
  L = len(grid)
  outfile = open(filename, "w")
  for t in range(t_max):
    print(t,"\t", Batterio.contaBatteri, "\t",IScell.contaIS)
    outfile.write(str(Batterio.contaBatteri)+"\t"+str(IScell.contaIS)+"\n")
    for i in range(L):
      for j in range(L):
        if (grid[i][j]!=Cellula.EMPTY):
          grid[i][j].move(grid, rmov)
    for i in range(L):
      for j in range(L):
        if (grid[i][j]!=Cellula.EMPTY):
          grid[i][j].act(grid)
  outfile.close()
```

NOTE per la progettazione delle classi:

- Variabile di classe Batterio.contaBatteri
- Variabile di classe IScell.contaIS
- Metodi move(grid, rmov) e act(grid)

Codice python (parte 3, **main**)

```
def main():


  L = int(input("Input size of square 2D grid. It must be an int: "))
  perc_is = float(input("Input % of immune system (IS) cells at time 0 (a number
between 0 and 1): "))
  perc_batt = float(input("Input % of bacteria cells at time 0 (a number between 0
and 1): "))
  griglia = gridinit(L, perc_is, perc_batt)


  filename = input("Input the name of the output file: ")
  t_max = int(input("Input number of iterations. It must be an int: "))
  raggioMov = int(input("Input max movement distance r. It must be an int: "))
  itera(t_max = t_max, grid = griglia, rmov = raggioMov, filename = filename)


main()
```

Progettazione delle 3 classi

Per ognuna delle 3 classi da progettare (Cellula, Batterio e IScell) seguiremo lo schema in 4 passi:

1. Definizione interfaccia pubblica
2. Definizione dello stato dell'oggetto
3. Definizione del costruttore
4. Definizione dei metodi

Cominciamo con la classe Cellula

Classe Cellula: informazioni dal testo

Cellula è la superclasse di Batterio e IScell. Definirà quindi le proprietà e i metodi di una cellula "generica".

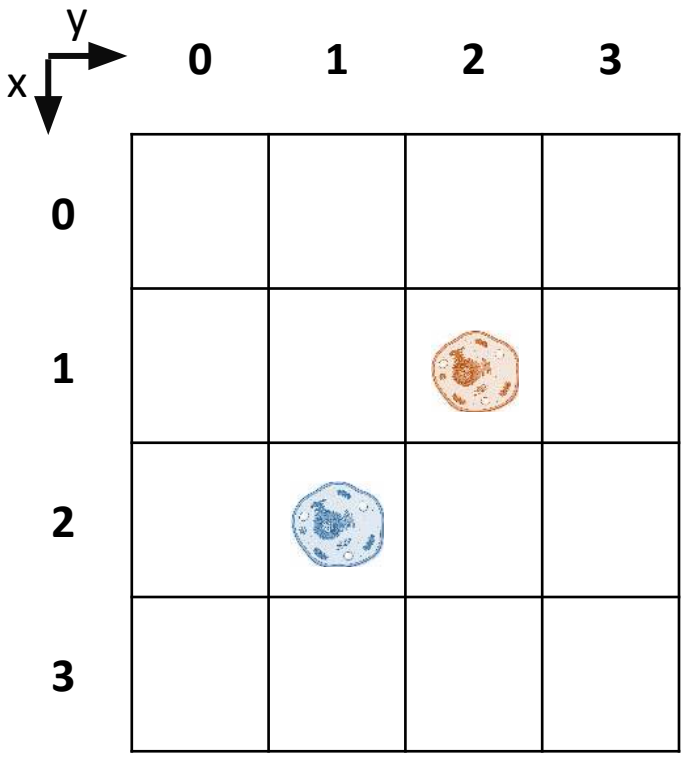
Dal testo si evince che la classe Cellula definirà:

- una **posizione x,y** della cellula nella griglia (vedi i costruttori IScell(i,j) e Batterio(i,j))
- una variabile di classe **Cellula.EMPTY** per indicare le celle vuote nella griglia
- un metodo **move(grid, rmov)** per "muovere" la cellula nella griglia **grid** in un intorno di raggio **rmov**
- un metodo (che chiameremo **die()**) per simulare la morte della cellula

Riguardo il metodo **act(grid)**, questo è un metodo che deve essere a disposizione di ogni oggetto Cellula, ma il cui comportamento è dipendente dal tipo di cellula.

Di conseguenza il metodo act(grid) si definisce **solo** nelle classi Batterio e IScell.

Classe Cellula: il metodo move()

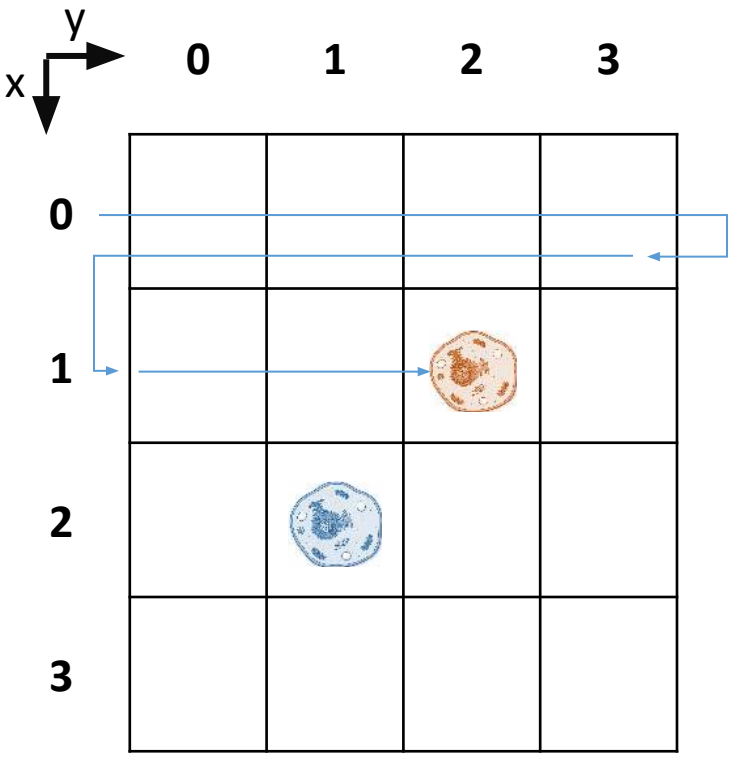


Nella fase 1 dell'iterazione, ogni cellula si sposta di posizione in un intorno di raggio **rmov**.

In dettaglio, si scorre la griglia per righe e si esegue lo spostamento di ogni cellula che si incontra.

Es: griglia 4x4, con due cellule.
La cellula 1 si sposta dalla posizione (1,2) alla posizione (0,1).
La cellula 2 si sposta da (2,1) a (3,1).

Classe Cellula: il metodo move()

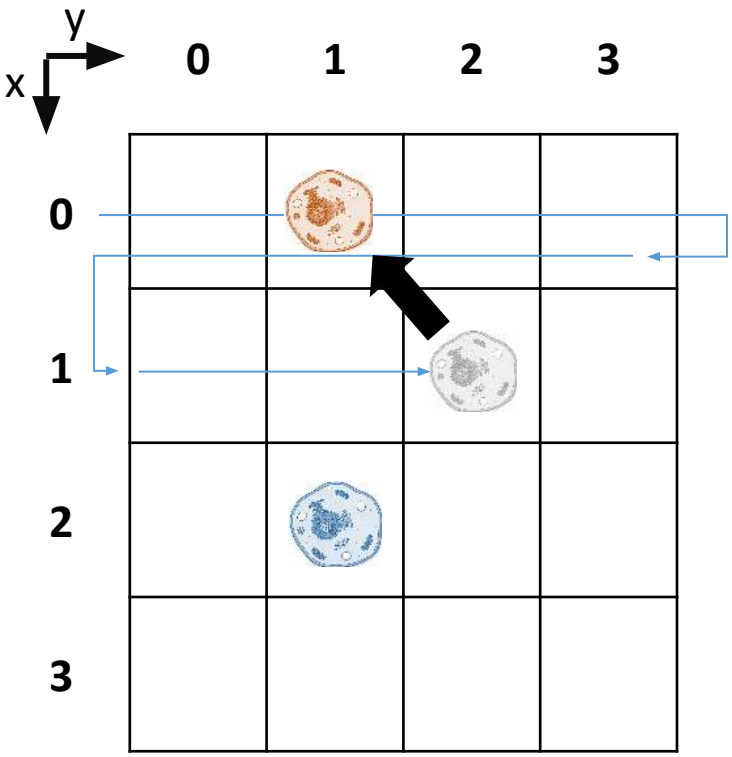


Nella fase 1 dell'iterazione, ogni cellula si sposta di posizione in un intorno di raggio **rmov**.

In dettaglio, si scorre la griglia per righe e si esegue lo spostamento di ogni cellula che si incontra.

Es: griglia 4x4, con due cellule.
La cellula 1 si sposta dalla posizione (1,2) alla posizione (0,1).
La cellula 2 si sposta da (2,1) a (3,1).

Classe Cellula: il metodo move()

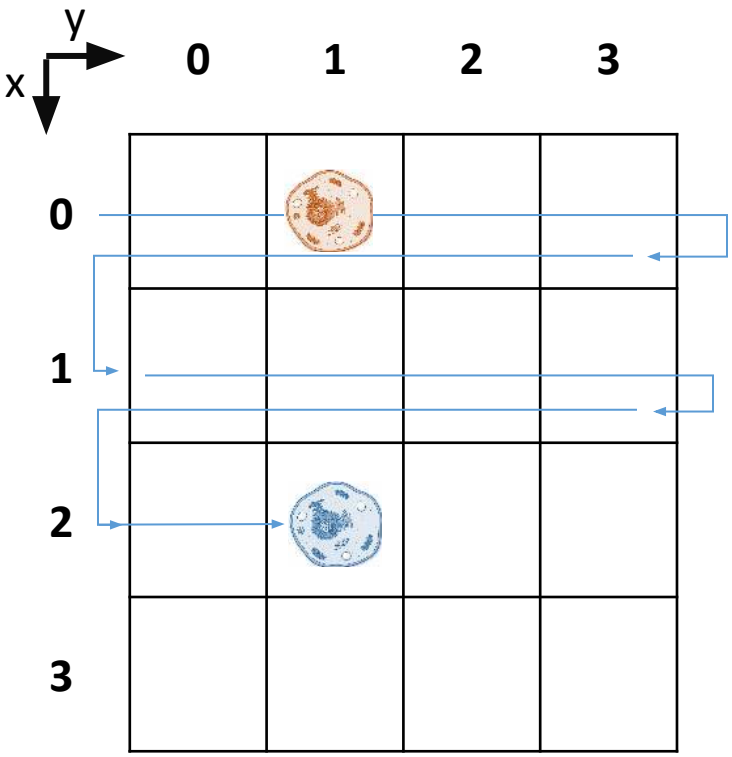


Nella fase 1 dell'iterazione, ogni cellula si sposta di posizione in un intorno di raggio **rmov**.

In dettaglio, si scorre la griglia per righe e si esegue lo spostamento di ogni cellula che si incontra.

Es: griglia 4x4, con due cellule.
La cellula 1 si sposta dalla posizione (1,2) alla posizione (0,1).
La cellula 2 si sposta da (2,1) a (3,1).

Classe Cellula: il metodo move()

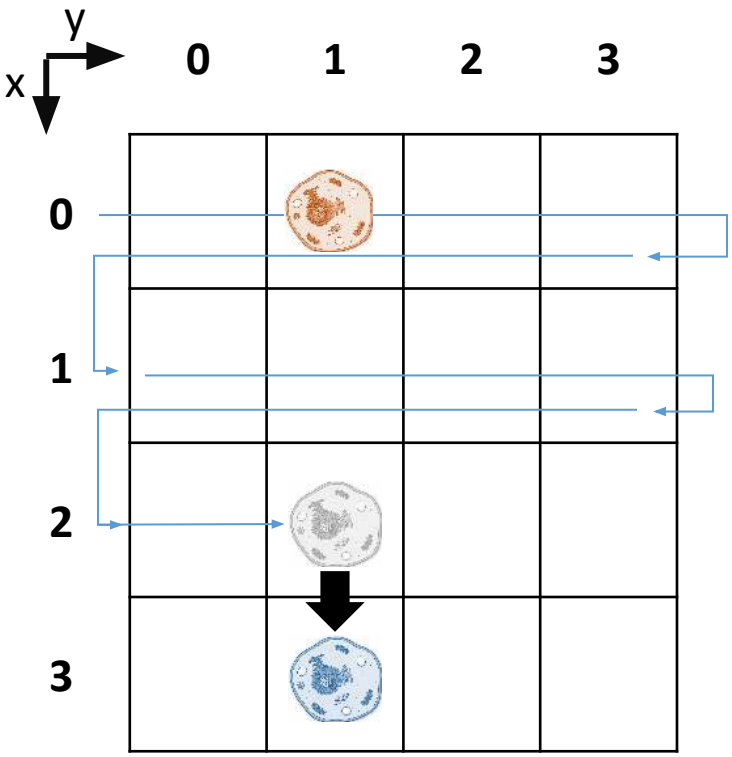


Nella fase 1 dell'iterazione, ogni cellula si sposta di posizione in un intorno di raggio **rmov**.

In dettaglio, si scorre la griglia per righe e si esegue lo spostamento di ogni cellula che si incontra.

Es: griglia 4x4, con due cellule.
La cellula 1 si sposta dalla posizione (1,2) alla posizione (0,1).
La cellula 2 si sposta da (2,1) a (3,1).

Classe Cellula: il metodo move()

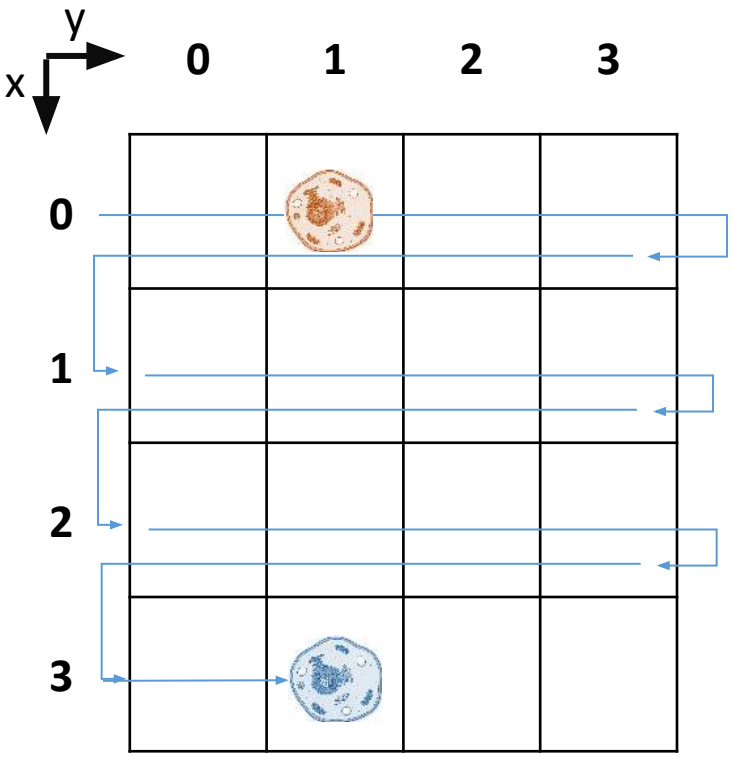


Nella fase 1 dell'iterazione, ogni cellula si sposta di posizione in un intorno di raggio **rmov**.

In dettaglio, si scorre la griglia per righe e si esegue lo spostamento di ogni cellula che si incontra.

Es: griglia 4x4, con due cellule.
La cellula 1 si sposta dalla posizione (1,2) alla posizione (0,1).
La cellula 2 si sposta da (2,1) a (3,1).

Classe Cellula: il metodo move()



Nella fase 1 dell'iterazione, ogni cellula si sposta di posizione in un intorno di raggio **rmov**.

In dettaglio, si scorre la griglia per righe e si esegue lo spostamento di ogni cellula che si incontra.

Es: griglia 4x4, con due cellule.
La cellula 1 si sposta dalla posizione (1,2) alla posizione (0,1).
La cellula 2 si sposta da (2,1) a (3,1).

Attenzione! In realtà, non dobbiamo spostare ogni cellula che incontriamo, perché potremmo spostare più volte una stessa cellula (come nel caso della cellula 2, che reincontreremo scorrendo la griglia, ma che non vogliamo rispostare). E' necessario **tenere quindi traccia di quali cellule sono già state spostate** nell'iterazione corrente, così da sapere quali "evitare" mentre si scorre la griglia.

Classe Cellula: informazioni dal testo

Cellula è la superclasse di Batterio e IScell. Definirà quindi le proprietà e i metodi di una cellula "generica".

Dal testo si evince che la classe Cellula definirà:

- una **posizione x,y** della cellula nella griglia (vedi i costruttori IScell(i,j) e Batterio(i,j))
- una variabile di classe **Cellula.EMPTY** per indicare le celle vuote nella griglia
- un metodo **move(grid, rmov)** per "muovere" la cellula nella griglia **grid** in un intorno di raggio **rmov**
- un metodo (che chiameremo **die()**) per simulare la morte della cellula
- **una variabile di esemplare (che chiameremo _acted) per memorizzare se la cellula è già stata spostata nell'iterazione corrente**

Riguardo il metodo **act(grid)**, questo è un metodo che deve essere a disposizione di ogni oggetto Cellula, ma il cui comportamento è dipendente dal tipo di cellula.

Di conseguenza il metodo act(grid) si definisce **solo** nelle classi Batterio e IScell.

Progettazione delle 3 classi

Per ognuna delle 3 classi da progettare (Cellula, Batterio e IScell) seguiremo lo schema in 4 passi:

1. **Definizione interfaccia pubblica**
2. Definizione dello stato dell'oggetto
3. Definizione del costruttore
4. Definizione dei metodi

Cominciamo con la classe Cellula

Classe Cellula – Definizione interfaccia pubblica (1/2)

```
class Cellula:

  ## Restituisce la posizione x y della cellula nella griglia
  # @return Coordinate x y della cellula
  def getpos(self):
    # codice metodo

  ## Fa morire la cellula
  # @param griglia Griglia di cellule
  def die(self, griglia):
    # codice metodo

  ## Restituisce se la cellula ha "agito" nell'iterazione corrente
  # @return "Azione" effettuata (0: no | 1: sì)
  def getacted(self):
    # codice metodo

  ## Definisce se la cellula ha già "agito" nell'iterazione corrente
  # @param v "Azione" effettuata (0: no | 1: sì)
  def setacted(self, v):
    # codice metodo
```

Classe Cellula – Definizione interfaccia pubblica (2/2)

```
class Cellula:

  . . .

  ## Restituisce una lista contenente le coordinate delle cellule di
  ## tipologia tipo entro un raggio r dalla posizione corrente
  # @param griglia Griglia di cellule
  # @param r Raggio attorno alla posizione corrente
  # @param tipo Tipologia di cellula
  # @return Lista contenente le coordinate
  def getvicini(self, griglia, r, tipo):
    # codice metodo

  ## Muove la cellula in una posizione libera nell'intorno di raggio r
  # @param griglia Griglia di cellule
  # @param r Raggio attorno alla cellula corrente
  def move(self, griglia, r):
    # codice metodo
```

Progettazione delle 3 classi

Per ognuna delle 3 classi da progettare (Cellula, Batterio e IScell) seguiremo lo schema in 4 passi:

1. Definizione interfaccia pubblica
2. **Definizione dello stato dell'oggetto**
3. Definizione del costruttore
4. Definizione dei metodi

Cominciamo con la classe Cellula

Classe Cellula – Definizione stato dell'oggetto

```
class Cellula:

  EMPTY = 0

  . . . # metodi
```

Classe Cellula deve definire:

- Variabile di classe Cellula.EMPTY che codifica le celle vuote
- Variabili di esemplare per le coordinate (x,y)
- Variabile di esemplare per memorizzare se la cellula è già stata spostata nell'iterazione corrente

Progettazione delle 3 classi

Per ognuna delle 3 classi da progettare (Cellula, Batterio e IScell) seguiremo lo schema in 4 passi:

1. Definizione interfaccia pubblica
2. Definizione dello stato dell'oggetto
3. **Definizione del costruttore**
4. Definizione dei metodi

Cominciamo con la classe Cellula

Classe Cellula – Definizione costruttore

```
class Cellula:

  EMPTY = 0

  def __init__(self, x, y, acted=0):
    self._x = x
    self._y = y
    self._acted = acted


  . . . # metodi
```

Classe Cellula deve definire:

- Variabile di classe Cellula.EMPTY che codifica le celle vuote
- Variabili di esemplare per le coordinate (x,y)
- Variabile di esemplare per memorizzare se la cellula è già stata spostata nell'iterazione corrente

Progettazione delle 3 classi

Per ognuna delle 3 classi da progettare (Cellula, Batterio e IScell) seguiremo lo schema in 4 passi:

1. Definizione interfaccia pubblica
2. Definizione dello stato dell'oggetto
3. Definizione del costruttore
4. **Definizione dei metodi**

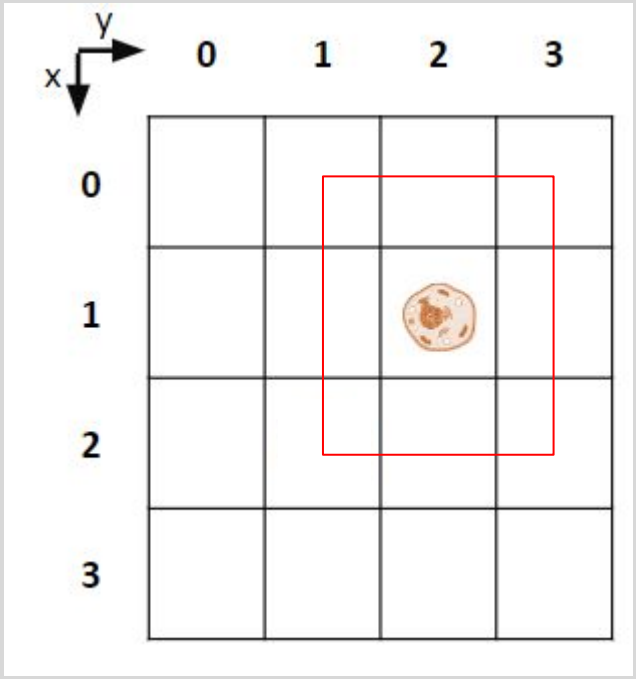
Cominciamo con la classe Cellula

Classe Cellula – Definizione metodi (1/3)

```
class Cellula:

  EMPTY = 0

  def __init__(self, x, y, acted=0):
    self._x = x
    self._y = y
    self._acted = acted

  def getpos(self):
    return(self._x,self._y)

  def die(self, griglia):
    griglia[self._x][self._y] = Cellula.EMPTY

  def getacted(self):
    return(self._acted)

  def setacted(self, v):
    self._acted = v
```

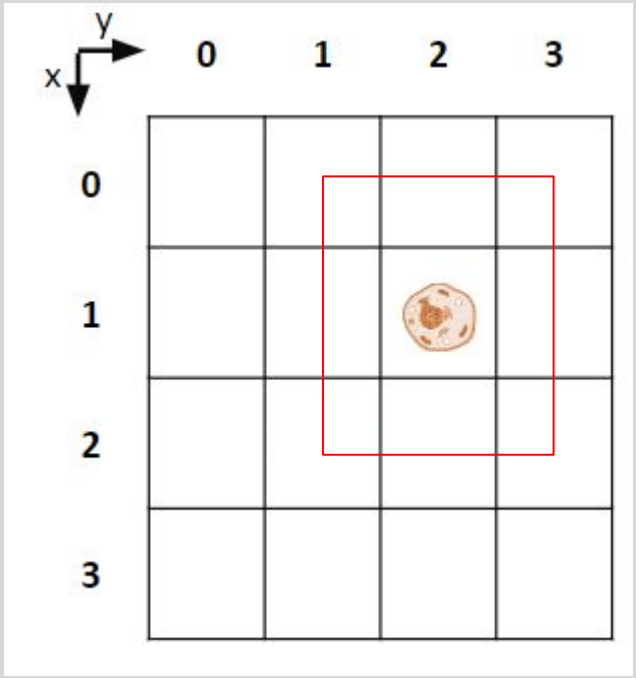
Classe Cellula – Definizione metodi (2/3)

```
class Cellula:

  . . .
Restituisce una lista contenente le coordinate delle cellule di
  ## tipologia tipo entro un raggio r dalla posizione corrente
  # @param griglia Griglia di cellule
  # @param r Raggio attorno alla posizione corrente
  # @param tipo Tipologia di cellula
  # @return Lista contenente le coordinate

  def getvicini(self, griglia, r, tipo):
    vicini = []
    L = len(griglia)
    min_x = max(0, self._x-r)
    max_x = min(L-1, self._x+r)
    min_y = max(0, self._y-r)
    max_y = min(L-1, self._y+r)
    for i in range(min_x, max_x+1):
      for j in range(min_y, max_y+1):
        if isinstance(griglia[i][j], tipo):
          vicini.append((i,j))
    return vicini
```



Classe Cellula – Definizione metodi (2/3)

```
class Cellula:

  . . .
Restituisce una lista contenente le coordinate delle cellule di
  ## tipologia tipo entro un raggio r dalla posizione corrente
  # @param griglia Griglia di cellule
  # @param r Raggio attorno alla posizione corrente
  # @param tipo Tipologia di cellula
  # @return Lista contenente le coordinate

  def getvicini(self, griglia, r, tipo):
    vicini = []
    L = len(griglia)
    min_x = max(0, self._x-r)
    max_x = min(L-1, self._x+r)
    min_y = max(0, self._y-r)
    max_y = min(L-1, self._y+r)
    for i in range(min_x, max_x+1):
      for j in range(min_y, max_y+1):
        if isinstance(griglia[i][j], tipo):
          vicini.append((i,j))
    return vicini
```



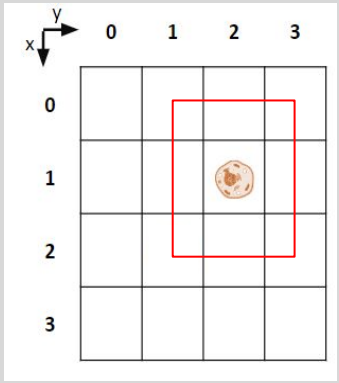
Classe Cellula – Definizione metodi (3/3)

```
class Cellula:

  . . .

  def move(self, griglia, r):
    if self._acted == 0:
      posizioni = self.getvicini(griglia, r, tipo = int)
      if len(posizioni)>0:
        (newx,newy) = random.choice(posizioni)
        oldx = self._x
        oldy = self._y
        self._x = newx
        self._y = newy
        griglia[newx][newy] = self
        griglia[oldx][oldy] = Cellula.EMPTY
        if (newx > oldx) or ((newx == oldx) and (newy > oldy)):
          self._acted = 1
    else:
      self._acted = 0  #resetto a 0 per il prossimo giro
```

`random.choice(posizioni)` restituisce un elemento a caso tra quelli presenti nella lista `posizioni`

Classe Cellula – Definizione metodi (3/3)

```
class Cellula:

  . . .

  def move(self, griglia, r):
    if self._acted == 0:
      posizioni = self.getvicini(griglia, r, tipo = int)
      if len(posizioni)>0:
        (newx,newy) = random.choice(posizioni)
        oldx = self._x
        oldy = self._y
        self._x = newx
        self._y = newy
        griglia[newx][newy] = self
        griglia[oldx][oldy] = Cellula.EMPTY
        if (newx > oldx) or ((newx == oldx) and (newy > oldy)):
          self._acted = 1
    else:
      self._acted = 0  #resetto a 0 per il prossimo giro
```



Scorrendo la griglia per righe, le cellule spostate a destra o in basso sono quelle da “segnalare” per evitare di spostarle nuovamente

Classe IScell: informazioni dal testo

IScell è una sottoclasse di Cellula, erediterà quindi gli attributi e i metodi in essa definiti.

Inoltre, dal testo si evince che la classe IScell definirà:

- un variabile **_numattacks** per tenere traccia del numero di attacchi effettuati dalla cellula del sistema immunitario
- una variabile di classe **IScell.contaIS** per indicare il numero totale di cellule del sistema immunitario nella griglia
- una variabile di classe **IScell.MAX_ATTACKS** per indicare il numero massimo di attacchi da effettuare prima di morire
- un metodo **act(grid)** che, in caso di raggiungimento del numero massimo di attacchi, faccia morire la cellula del sistema immunitario

Progettazione delle 3 classi

Per ognuna delle 3 classi da progettare (Cellula, Batterio e IScell) seguiremo lo schema in 4 passi:

1. **Definizione interfaccia pubblica**
2. Definizione dello stato dell'oggetto
3. Definizione del costruttore
4. Definizione dei metodi

Classe IScell – Definizione interfaccia pubblica

```
class IScell(Cellula):

  ## In caso di raggiungimento del numero massimo di attacchi,
  ## provoca la morte della cellula
  # @param griglia Griglia di cellule
  def act(self, griglia):
    # codice metodo

  ## Aggiorna il numero di attacchi effettuato dalla cellula
  def attack(self):
    # codice metodo
```

Progettazione delle 3 classi

Per ognuna delle 3 classi da progettare (Cellula, Batterio e IScell) seguiremo lo schema in 4 passi:

1. Definizione interfaccia pubblica
2. **Definizione dello stato dell'oggetto**
3. Definizione del costruttore
4. Definizione dei metodi

Classe IScell – Definizione stato dell'oggetto

```
class IScell(Cellula):

  contaIS = 0
  MAX_ATTACKS = 9

  ## In caso di raggiungimento del numero massimo di
  ## attacchi, provoca la morte della cellula
  # @param griglia Griglia di cellule
  def act(self, griglia):
    # codice metodo

  ## Aggiorna il numero di attacchi effettuato
  ## dalla cellula
  def attack(self):
    # codice metodo
```

Classe IScell deve definire:

- variabile _numattacks (numero di attacchi effettuati)
- variabile di classe IScell.contaIS (numero totale di cellule del sistema immunitario)
- variabile di classe IScell.MAX_ATTACKS (numero massimo di attacchi da effettuare prima di morire)

Progettazione delle 3 classi

Per ognuna delle 3 classi da progettare (Cellula, Batterio e IScell) seguiremo lo schema in 4 passi:

1. Definizione interfaccia pubblica
2. Definizione dello stato dell'oggetto
3. **Definizione del costruttore**
4. Definizione dei metodi

Classe IScell – Definizione costruttore

```
class IScell(Cellula):

  contaIS = 0
  MAX_ATTACKS = 9

  def __init__(self, x, y, acted=0):
    super().__init__(x,y,acted)
    self._numattacks = 0
    IScell.contaIS = IScell.contaIS + 1

  ## In caso di raggiungimento del numero massimo di
  ## attacchi, provoca la morte della cellula
  # @param griglia Griglia di cellule
  def act(self, griglia):
    # codice metodo

  ## Aggiorna il numero di attacchi effettuato dalla
  def attack(self):
    # codice metodo
```

Classe IScell deve definire:

- variabile _numattacks (numero di attacchi effettuati)
- variabile di classe IScell.contaIS (numero totale di cellule del sistema immunitario)
- variabile di classe IScell.MAX_ATTACKS (numero massimo di attacchi da effettuare prima di morire)

Progettazione delle 3 classi

Per ognuna delle 3 classi da progettare (Cellula, Batterio e IScell) seguiremo lo schema in 4 passi:

1. Definizione interfaccia pubblica
2. Definizione dello stato dell'oggetto
3. Definizione del costruttore
4. **Definizione dei metodi**

Classe IScell – Definizione metodi

```
class IScell(Cellula):

  contaIS = 0
  MAX_ATTACKS = 9

  def __init__(self,x,y,acted=0):
    super().__init__(x,y,acted)
    self._numattacks = 0
    IScell.contaIS = IScell.contaIS + 1

  def act(self, griglia):
    if self._numattacks == IScell.MAX_ATTACKS:
      self.die(griglia)
      IScell.contaIS = IScell.contaIS - 1

  def attack(self):
    self._numattacks = self._numattacks + 1
```

Classe Batterio: informazioni dal testo

Batterio è una sottoclasse di Cellula, erediterà quindi gli attributi e i metodi in essa definiti.

Inoltre, dal testo si evince che la classe Batterio definirà:

- una variabile **_vol** per tenere traccia del volume della cellula batterica
- una variabile di classe **Batterio.contaBatteri** per indicare il numero totale di cellule batteriche nella griglia
- una variabile di classe **Batterio.PERC_GROWTH** per definire il tasso di crescita (**0.3**) del batterio
- una variabile di classe **Batterio.VOLDIV** per definire il volume oltre il quale si esegue la duplicazione (**2**)
- una variabile di classe **Batterio.PROB_DIE** per definire la probabilità di morte in caso di attacco di una cellula del sistema immunitario (**0.2**)
- un metodo **act(grid)** che:
  - esegue l'eventuale crescita/duplicazione
  - simula l'attacco di eventuali cellule del sistema immunitario nelle vicinanze

Progettazione delle 3 classi

Per ognuna delle 3 classi da progettare (Cellula, Batterio e IScell) seguiremo lo schema in 4 passi:

1. **Definizione interfaccia pubblica**
2. Definizione dello stato dell'oggetto
3. Definizione del costruttore
4. Definizione dei metodi

Classe Batterio: definizione interfaccia pubblica

```
class Batterio(Cellula):

  ## Simula la crescita (di volume) della cellula batterica
  def growth(self):
    # codice metodo

  ## Simula la duplicazione cellulare quando il volume supera una certa soglia
  # @param griglia Griglia di cellule
  def divide(self, griglia):
    # codice metodo

  ## Esegue l'eventuale crescita/duplicazione del batterio e simula
  ## l'attacco di eventuali cellule del sistema immunitario nelle vicinanze
  # @ griglia Griglia di cellule
  def act(self, griglia):
    # codice metodo
```

Progettazione delle 3 classi

Per ognuna delle 3 classi da progettare (Cellula, Batterio e IScell) seguiremo lo schema in 4 passi:

1. Definizione interfaccia pubblica
2. **Definizione dello stato dell'oggetto**
3. Definizione del costruttore
4. Definizione dei metodi

Classe Batterio: definizione stato dell'oggetto

```
class Batterio(Cellula):

  contaBatteri = 0
  PERC_GROWTH = 0.3
  VOLDIV = 2
  PROB_DIE = 0.2

  ## Simula la crescita (di volume) della ce
  def growth(self):
    # codice metodo

  ## Simula la duplicazione cellulare quando
  # @param griglia Griglia di cellule
  def divide(self, griglia):
    # codice metodo

  ## Esegue l'eventuale crescita/duplicazion
  ## l'attacco di eventuali cellule del sist
  # @ griglia Griglia di cellule
  def act(self, griglia):
    # codice metodo
```

Classe Batterio deve definire:

- variabile _vol (volume cellula)
- variabile di classe Batterio.contaBatteri (numero totale di cellule batteriche nella griglia)
- variabile di classe Batterio.PERC_GROWTH (tasso di crescita (0.3))
- variabile di classe Batterio.VOLDIV (volume oltre il quale si esegue la duplicazione (2))
- variabile di classe Batterio.PROB_DIE (probabilità di morte in caso di attacco di una cellula del sistema immunitario (0.2))

Progettazione delle 3 classi

Per ognuna delle 3 classi da progettare (Cellula, Batterio e IScell) seguiremo lo schema in 4 passi:

1. Definizione interfaccia pubblica
2. Definizione dello stato dell'oggetto
3. **Definizione del costruttore**
4. Definizione dei metodi

Classe Batterio: definizione costruttore

Classe Batterio deve definire:

- variabile _vol (volume cellula)
- variabile di classe Batterio.contaBatteri (numero totale di cellule batteriche nella griglia)
- variabile di classe Batterio.PERC_GROWTH (tasso di crescita (0.3))
- variabile di classe Batterio.VOLDIV (volume oltre il quale si esegue la duplicazione (2))
- variabile di classe Batterio.PROB_DIE (probabilità di morte in caso di attacco di una cellula del sistema immunitario (0.2))

```
class Batterio(Cellula):

  contaBatteri = 0
  PERC_GROWTH = 0.3
  VOLDIV = 2
  PROB_DIE = 0.2

  def __init__(self, x, y, acted = 0, vol = 1):
    super().__init__(x,y,acted)
    self._vol = vol
    Batterio.contaBatteri = Batterio.contaBatteri + 1

  ## Simula la crescita (di volume) della cellula batterica
  def growth(self):
    # codice metodo

  ## Simula la duplicazione cellulare quando il volume supera una certa soglia
  # @param griglia Griglia di cellule
  def divide(self, griglia):
    # codice metodo

  . . .
```

Progettazione delle 3 classi

Per ognuna delle 3 classi da progettare (Cellula, Batterio e IScell) seguiremo lo schema in 4 passi:

1. Definizione interfaccia pubblica
2. Definizione dello stato dell'oggetto
3. Definizione del costruttore
4. **Definizione dei metodi**

Classe Batterio: definizione metodi (1/2)

```
class Batterio(Cellula):

  . . .


  def growth(self):
    self._vol = self._vol*(1 + Batterio.PERC_GROWTH)


  def divide(self, griglia):
    self._vol = 1
    posizioni = self.getvicini(griglia,r = len(griglia), tipo = int)
    (x,y) = self.getpos()
    if len(posizioni)>0:
      (newx,newy) = random.choice(posizioni)
      if (newx > x) or ((newx == x) and (newy > y)):
        griglia[newx][newy] = Batterio(newx, newy, acted = 1)
      else:
        griglia[newx][newy] = Batterio(newx, newy)
```

Classe Batterio: definizione metodi (2/2)

```
class Batterio(Cellula):

  . . .

  def act(self, griglia):
    if self.getacted() == 0:
      if self._vol >= Batterio.VOLDIV:
        self.divide(griglia)
      else:
        self.growth()
      (xb,yb) = self.getpos()
      posizioni = self.getvicini(griglia, r = 1, tipo = IScell)
      Lp = len(posizioni)
      if Lp>0:
        for (x,y) in posizioni:
          griglia[x][y].attack()
          if random.random() <= Batterio.PROB_DIE:
            self.die(griglia)
            griglia[xb][yb] = IScell(xb,yb)
            Batterio.contaBatteri = Batterio.contaBatteri - 1
            break
    else:
      self.setacted(0)
```

La chiamata del metodo `self.getacted()` restituirà **1** solo per le nuove cellule batteriche generate dalla duplicazione cellulare durante la iterazione corrente, e che quindi non devono subire attacchi da parte delle cellule del sistema immunitario.