Projecto de Biocomputação 2019-2020

Departamento de Matemática, IST

Novembro de 2019

Ficheiros

Os ficheiros disponibilizados na página da cadeira são os seguintes:

- `cell_adt.py` - módulo que disponibiliza o tipo de dados *célula*;
- `map_adt.py` - módulo que disponibiliza o tipo de dados *mapa*;
- `event_adt.py` - módulo que disponibiliza o tipo de dados *evento*;
- `cap_adt.py` - módulo que disponibiliza o tipo de dados *cadeia de acontecimentos pendentes*;
- `animacao.ipynb` - notebook para visualizar os resultados da simulação;
- `resultados.txt` - ficheiro de resultados de um exemplo de simulação.

cell_adt.py

Este módulo disponibiliza o tipo de dados *célula*. O tipo de dados foi implementado através da definição da classe `Cell`. Cada objecto de tipo célula é criado indicando quantos níveis tróficos irão popular a célula. A classe disponibiliza os seguintes métodos:

- `atrCel(e,p)` - atribui uma nova percentagem `p` à espécie `e`, desde que não exceda a capacidade total da célula;
- `valCel(e)` - devolve a percentagem de ocupação da espécie `e` na célula;
- `totCel()` - devolve a percentagem total de ocupação da célula;
- `freeSpace()` - devolve o espaço livre na célula, isto é, a percentagem da célula que não está ocupada por nenhuma espécie;

- `listCel()` - devolve uma lista das percentagens de ocupação de cada espécie, por ordem, da espécie 1 até à espécie no último nível;
- `num_espCel()` - devolve o número de espécies que podem povoar a célula, que é fixado no momento de criação da célula;
- `show()` - imprime a lista das percentagens de ocupação de cada espécie no ecrã, da espécie 1 até à espécie no último nível.

Seguem-se alguns exemplos de utilização.

```
 In [1]: import cell_adt as cell
 In [2]: c1=cell.Cell(4)
 In [3]: c1.atrCel(2,0.3)
 In [4]: c1.atrCel(1,0.5)
 In [5]: c1.atrCel(4,0.1)
 In [6]: c1.valCel(1)
 Out[6]: 0.5
 In [7]: c1.valCel(2)
 Out[7]: 0.3
 In [8]: c1.valCel(3)
 Out[8]: 0
 In [9]: c1.valCel(4)
 Out[9]: 0.1
In [10]: c1.totCel()
Out[10]: 0.9
In [11]: c1.freeSpace()
Out[11]: 0.1
In [12]: c1.listCel()
Out[12]: [0.5, 0.3, 0, 0.1]
In [13]: c1.num_espCel()
Out[13]: 4
In [14]: c1.show()
          [0.5, 0.3, 0, 0.1]
In [15]: c2=cell.Cell(4)
         c2.atrCel(1,0.5)
         c2.atrCel(3,0.2)
         c3=cell.Cell(4)
         c3.atrCel(2,0.3)
         c3.atrCel(4,0.2)
```

```
c3.atrCel(3,0.1)
c4=cell.Cell(4)
c4.atrCel(4,0.6)
c5=cell.Cell(4)
c5.atrCel(1,0.1)
c5.atrCel(2,0.2)
c5.atrCel(3,0.3)
```

map_adt.py

Este módulo disponibiliza o tipo de dados *mapa*, e importa `cell_adt.py`. O tipo de dados foi implementado através da definição da classe `Map`. A criação de um objecto desta classe é feita fornecendo, aquando da criação, a dimensão do mapa e o número de níveis tróficos `N`. Inicialmente, todas as quadrículas do mapa são ocupadas por células vazias, com `N` níveis tróficos. A classe disponibiliza os seguintes métodos:

- `dimMap()` - devolve a dimensão do mapa;
- `NumSpeciesMap()` - devolve o número de espécies em cada célula do mapa;
- `atrMap(i,j,c)` - coloca a célula `c` na quadrícula `(i,j)` do mapa, independentemente do que estivesse nessa quadrícula anteriormente;
- `celMap(i,j)` - devolve a célula que se encontra na quadrícula `(i,j)` do mapa;
- `freeMap(i,j)` - devolve o espaço livre na célula que se encontra na quadrícula `(i,j)`;
- `neiMap(i,j,r)` - devolve a lista das percentagens totais de cada espécie na vizinhança de raio `r` da quadrícula `(i,j)`, por ordem, da espécie 1 até à espécie `N`;
- `list_posMap()` - devolve uma lista de listas, uma por cada nível trófico; para cada nível a lista respectiva contém todos os pares da forma $((x, y), p)$ onde (x, y) são as coordenadas de cada célula que contém espécies desse nível e p a respectiva percentagem.

Seguem-se alguns exemplos de utilização.

```
In [1]: import map_adt as map
In [2]: m1=map.Map(10,4)
In [3]: m1.dimMap()
```

```
 Out[3]: 10
 In [4]: m1.NumSpeciesMap()
 Out[4]: 4
 In [5]: m1.atrMap(5,5,c1)
 In [6]: m1.atrMap(6,7,c2)
 In [7]: m1.atrMap(4,6,c3)
 In [8]: m1.atrMap(7,3,c4)
 In [9]: m1.atrMap(5,4,c5)
In [10]: nc=m1.celMap(5,5)
In [11]: nc.show()
          [0.5, 0.3, 0, 0.1]
Out[11]: c1.show()
          [0.5, 0.3, 0, 0.1]
In [12]: m1.freeMap(5,4)
Out[12]: 0.4
In [13]: c5.freeSpace()
Out[13]: 0.4
In [14]: m1.neiMap(5,5,1)
Out[14]: [0.6, 0.8, 0.4, 0.3]
In [15]: m1.neiMap(5,5,2)
Out[15]: [1.1, 0.8, 0.6, 0.9]
In [16]: m1.list_posMap()
Out[16]: [[((5, 5), 0.5), ((6, 7), 0.5), ((5, 4), 0.1)],
          [((5, 5), 0.3), ((4, 6), 0.3), ((5, 4), 0.2)],
          [((6, 7), 0.2), ((4, 6), 0.1), ((5, 4), 0.3)],
          [((5, 5), 0.1), ((4, 6), 0.2), ((7, 3), 0.6)]]
```

event_adt.py

Este módulo disponibiliza o tipo de dados *evento*. O tipo de dados foi implementado através da definição da classe `Event`. A criação de objectos desta classe é feita fornecendo, aquando da criação, cinco parâmetros: as *coordenadas* da quadrícula à qual o evento está associado (caso exista), o *nível trófico* ao qual o evento está associado (caso exista), o *instante* do evento e o *tipo* do evento. A classe disponibiliza os seguintes métodos:

- `evt_pos()` - as coordenadas da quadrícula à qual o evento está associado;
- `evt_species()` - devolve o nível trófico ao qual o evento está associado
- `evt_time()` - devolve o instante com que o evento foi criado;

- `evt_kind()` - devolve o tipo do evento;
- `show()` - mostra as características do evento.

Seguem-se alguns exemplos de utilização.

```
 In [1]: import event_adt as event
 In [2]: e1=event.Event(5,5,1,0.5,"loc")
 In [3]: e1.evt_pos()
 Out[3]: (5, 5)
 In [4]: e1.evt_species()
 Out[4]: 1
 In [5]: e1.evt_time()
 Out[5]: 0.5
 In [6]: e1.evt_kind()
         'loc'
 In [7]: e1.show()
          (1, 0.5, 'loc', (5, 5))
 In [8]: e2=event.Event(5,5,4,1.5,"loc")
 In [9]: e3=event.Event(6,7,3,2.8,"loc")
In [10]: e4=event.Event(5,4,2,1.8,"loc")
In [11]: e5=event.Event(0,0,0,3.5,"mud")
In [12]: e6=event.Event(5,4,1,0.8,"loc")
```

cap_adt.py

Este módulo disponibiliza o tipo de dados *cadeia de acontecimentos pendentes*, e importa o módulo `event_adt.py`. O tipo de dados foi implementado através da definição da classe `CAP`. Cada objecto é criado inicialmente sem eventos. A classe disponibiliza os seguintes métodos:

- `addE(e)` - acrescenta o evento `e` à cap;
- `delE()` - apaga o evento com menor instante da cap, caso esta não esteja vazia;
- `nextE()` - devolve o evento com menor instante da cap, caso esta não esteja vazia;
- `size()` - devolve o comprimento da cap;
- `show()` - mostra o conteúdo da cap.

Seguem-se alguns exemplos de utilização.

```
 In [1]: import cap_adt as cap
 In [2]: c = cap.CAP()
 In [3]: c.addE(e1)
 In [4]: c.addE(e2)
 In [5]: c.addE(e3)
 In [6]: c.addE(e4)
 In [7]: c.addE(e5)
 In [8]: c.addE(e6)
 In [9]: c.show()
          (1, 0.5, 'loc', (5, 5))
          (1, 0.8, 'loc', (5, 4))
          (4, 1.5, 'loc', (5, 5))
          (2, 1.8, 'loc', (5, 4))
          (3, 2.8, 'loc', (6, 7))
          (0, 3.5, 'mud', (0, 0))
In [10]: c.nextE()
Out[10]: <event_adt.Event at 0x10bf84f28>
In [11]: c.nextE().time()
Out[11]: 0.5
In [12]: c.delE()
In [13]: c.nextE().time()
Out[13]: 0.8
In [14]: c.leng()
Out[14]: 5
```

animacao.ipynb

O notebook `animacao.ipynb` permite gerar uma animação do resultado de uma simulação. Para tal, é necessário disponibilizar um ficheiro `resultados.txt` que deverá ser colocado na mesma pasta que o notebook e seguir as instruções aí fornecidas.