# Flussi Pedonali nell'Area del Colosseo: Codici

Roberto D'Autilia

2017-03-21

**Abstract**

File:colosseo_3_nuweb.tex

## 1 Introduzione

In questo lavoro vogliamo studiare i flussi pedonali nell'area del Colosseo nell'assetto urbanistico attuale e in quello di progetto. I luoghi dai quali si muove la popolazione sono luscita della metro $M_1$, la Via Sacra $M_2$ e Via di San Gregorio $M_3$. Abbiamo due carte, la sitazione attuale (`00-FASE0.jpg`) e il progetto (`00-FASE1.jpg`).

La mappa `00-FASE0.jpg` deve essere scalata per essere adattata alla simulazione. La scala ottimale è $(0.27, 0.27)$.

Ora dobbiamo visualizzare le coordinate del mouse. Poi ricostruire tutte le aree chiuse. Le disegno in verde per non confonderle con quelle ancora non corrette.

**Colosseo** lo modellizziamo come due emicicli (poligoni chiusi in modo di avere un'entrata e un'uscita). Fatto questo cominciamo a simulare una popolazione di 1000 persone che entra nel Colosseo sulla prima entrata. Poiché il *drift* all'interno del Colosseo ci occorre anche un poligono area interna del Colosseo.

## 2 Codice

Vediamo come è fatto il codice. Le costanti sono

"../../test/jl/costanti.jl" 1 ≡

```
# INIZIALIZZO LE COSTANTI
const N=1000 ::Int64          # Il numero di pedoni
const dt = 1.0 ::Float64         # Il passo di integrazione
const numero_iterazioni = 500    # Il numero di iterazioni della simulazione
const diag = sqrt(2) ::Float64      # diagonale
const passo = 1.0 ::Float64         # La dimensione di un passo
const raggio = 0.5 ::Float64     # Il raggio di non sovrapposizione dei pedoni
const dimenpedone = 2.0 ::Float64   # La dimensione del pedone
const scalax = 1.5 ::Float64     # lunghezza del passo di un pedone nella direzione x
const scalay = 1.5 ::Float64     # lunghezza del passo di un pedone nella direzione y
```

◇

"../../test/jl/variabili_globali.jl" 2a ≡

```
# Variabili globali che sarebbe meglio eliminare
posizioni_prima = zeros(2,N)           # le posizioni dei pedoni al tempo t
#posizioni_dopo = rand(30.0:821.0,2,N)    # le posizioni dei pedoni al tempo t+dt
altre = [rand(0.0:0.0,1,N);rand(100.0:100.0,1,N)] # altre posizioni
posizioni_dopo = hcat([rand(100.0:100.0,1,N);rand(200.0:200.0,1,N)],altre) # le posizioni dei pedoni al tempo
velocita = zeros(2,N)                  # le posizioni dei pedoni al tempo t+dt
k = 0 ::Int64                        # Un iteratore
passo = 1.0 ::Float64                  # La dimensione di un passo
raggio = 0.5 ::Float64                 # Il raggio di non sovrapposizione dei pedoni
dimenpedone = 2.0 ::Float64                  # La dimensione del pedone
scalax = 1.5 ::Float64                 # lunghezza del passo di un pedone nella direzione x
scalay = 1.5 ::Float64                 # lunghezza del passo di un pedone nella direzione y

mmm = 0.0 ::Float64                    # variabile di appoggio, ricontrollare se ci serve
```
◇

Per costruire i poligoni degli ostacoli dei pedoni costruiamo una costante di tipo `dictionary`

"../../test/jl/edifici_coord.jl" 2b ≡

```
###### DICTIONARY POLIGONI DEGLI EDIFICI ##################
#colosseo=[0 0 100 0; 100 0 100 200; 100 200 50 50; 50 50 0 100; 0 100 0 0]
###### DICTIONARY POLIGONI DEGLI EDIFICI ##################
#colosseo=[0 0 100 0; 100 0 100 200; 100 200 50 50; 50 50 0 100; 0 100 0 0]
const edifici_coord = Dict{String, Array}(
"colosseo_coord" => [314 256; 329 242; 359 223; 382 212; 419 204; 454 200; 489 204; 523 211; 558 223; 597 245;
651 444; 631 482; 600 507; 562 525; 520 533; 485 531; 446 524; 412 510; 374 490; 344 462; 314 420; 300 384; 30
"colosseoa_coord" => [373 203;390 198; 410 192; 440 190; 464 190; 487 191; 507 194; 537 202; 567 213; 594 229;
698 411; 699 444; 691 459; 679 455; 687 422; 689 402; 688 387; 687 374; 684 358; 679 344; 673 330; 664 317; 65
590 242; 577 234; 561 224; 548 220; 532 212; 518 210; 502 206; 485 204; 471 202; 455 201; 441 201; 425 202; 40
"colosseob_coord" => [659 449; 652 464; 643 478; 636 488; 625 498; 616 506; 605 514; 595 519; 583 524; 559 534
461 536; 444 531; 447 525; 465 528; 477 531; 491 533; 503 533; 518 533; 532 533; 545 531; 559 528; 573 523; 58
638 475; 644 464; 649 451; 650 444], # L'emiciclo b
"colosseoc_coord" => [409 519; 395 511; 381 503; 371 496; 361 488; 349 478; 340 468; 330 459; 322 448; 313 432
293 360; 293 344; 261 342; 264 323; 269 305; 276 289; 281 277; 288 266; 323 288; 315 303; 311 312; 307 325; 30
307 400; 315 421; 329 443; 336 454; 347 466; 367 485; 388 498; 399 505; 412 510], # L'emiciclo c
"colosseod_coord" => [305 245; 321 230; 335 221; 355 223; 340 233; 326 242; 314 254], # L'emiciclo d
"g1_coord" => [139 36; 201 66; 226 71; 241 70; 270 81; 273 71; 201 57; 159 41],
"g2_coord" => [287 74; 379 93; 417 91; 431 96; 456 90; 465 98; 436 109; 432 125; 348 110; 348 102; 342 95; 284
"g3_coord" => [424 83; 429 89; 459 83; 476 100; 441 116; 440 126; 518 141; 529 147; 560 159; 597 172; 632 193;
"arcoCostantino_coord" => [121 482; 131 439; 201 454; 191 497],
"prato1_coord" => [222 147; 230 127; 333 144; 315 182],
"prato2_coord" => [179 256; 213 171; 287 199; 248 238; 229 275],
"venereroma_coord" => [25 100; 193 165; 121 358; 25 329],
"metasudans_coord" => [120 424; 175 271; 224 291; 212 339; 222 400; 212 446],
"palatino_coord" => [23 783; 23 350; 100 377; 78 472; 80 488; 90 565; 83 652; 52 783],
"sangregorio_coord" => [120 785; 130 760; 147 740; 177 709; 203 683; 235 659; 269 639; 307 625; 353 617; 413 6
"bordoest_coord" => [608 230; 670 260; 700 280; 724 306; 734 330; 738 352; 737 376; 724 429; 693 504; 670 535;
570 573; 513 579; 451 578; 391 575; 343 576; 303 583; 262 594; 273 590; 221 614; 193 634; 157 660; 171 620; 18
503 543; 583 549; 638 536; 679 509; 702 465; 684 449; 693 402; 691 351; 676 307; 663 282; 643 254; 612 228])
```
◇

Abbiamo poi una funzione che crea i poligoni degli edifici

"../../test/jl/disegna_poligono.jl" 3a ≡

```
######################### UNA FUNZIONE CHE CREA I POLIGONI DEGLI EDIFICI ###############
function disegna_poligono(polig_coord)
   the_shape = ConvexShape()
   set_pointcount(the_shape, size(polig_coord)[1])
   for i = 1:size(polig_coord)[1]
      set_point(the_shape, i-1, Vector2f(polig_coord[i,1], polig_coord[i,2]))
   end
   set_position(the_shape, Vector2f(0.0, 0.0))
   set_fillcolor(the_shape, SFML.transparent)
   set_outlinecolor(the_shape, SFML.green)
   set_outline_thickness(the_shape, 2)
   return the_shape
end
#############################################################################################
```

◇

Abbiamo poi una funzione che verifica che le coordinate del pedone non siano all'interno di alcun poligono chiuso

"../../test/jl/esterno.jl" 3b ≡

```
############ UNA FUNZIONE CHE VERIFICA CHE UN PUNTO NON SIA INTERNO AD ALCUN POLIGONO ###########
function esterno(lax, lay, polig_coord)
w = map(ciclo_poligono, values(polig_coord))
a = sum(map(x->inpoly(lax,lay,x),w))
   if a>0
      return 1
   else
      return 0
   end
end
#############################################################################################
```

◇

Dobbiamo ora definire lo stato dei pedoni. Fino a questo momento lo stato dei pedoni era rappresentato dalle sole coordinate $(x, y)$. Ora vogliamo invece definire un tipo (o una struttura) con le coordinate spaziali e le coordinate dell'obiettivo. Nell'esempio seguente introduciamo il tipo Pedone e costruiamo un array di pedoni inizializzati a PEDONE_DEFAULT

"../../test/jl/pedone_test.jl" 4a ≡

```
type Pedone
      lax ::Float64
      lay::Float64
      lavx::Float64
      lavy::Float64
      ladestx::Float64
      ladesty::Float64

end
const PEDONE_DEFAULT = Pedone(0.1,0.1,0.1,0.1,0.1,0.1)

Pedone() = Pedone(PEDONE_DEFAULT)
\#ESEMPIO DI UN Array di 100 pedoni

popolazione = Array(Pedone,100)
for i in 1:100
   popolazione[i] = Pedone()
end
◇
```

File defined by 4ab.

## 2.1 la funzione posizioni

Definiamo una funzione che prende lo stato della popolazione di $N$ individui, che è un array di Statopedone e restituisce la matrice $2 \times N$ delle posizioni. Questa funzione sarà utile per calcolare la sovrapposizione delle posizioni con i metodi di NearestNeighbors

"../../test/jl/pedone_test.jl" 4b ≡

```
 function posizioni(stato::Statopedone)
       lex = []
       ley = []
       for i=1:N
           push!(lex,stato[i].lax)
       end
       for i=1:N
           push!(ley,stato[i].lay)
       end
       return transpose([lex ley])
       end
posizioni (generic function with 2 methods)
◇
```

File defined by 4ab.