El codi del *plug-in* VisDa pas a pas

Javier Melenchón

PID_00201054

UOC
Universitat Oberta de Catalunya

Índex

1. Introducció

En aquest capítol explicarem el codi de dibuix o renderització (*rendering*) de l'aplicació. N'oferirem dues versions: una versió que es pot executar de manera independent i una altra que es troba integrada amb el *plug-in* proposat. Els objectius són triples: mostrar la construcció del codi de dibuix, conèixer les possibilitats bàsiques de processos de manera pràctica, i conèixer les parts principals d'un codi amb interacció i animacions.

L'explicació es fa sobre la versió independent i, al final, comentarem com es pot integrar amb el *plug-in*. D'aquesta manera aconseguim que el lector pugui anar provant el codi mentre el construïm. Construirem el codi de manera incremental començant per un exemple molt senzill fins a arribar a tenir tot el codi sencer amb totes les funcionalitats i característiques.

Durant aquesta construcció també mostrarem diferents característiques bàsiques del Processing que paga la pena conèixer.

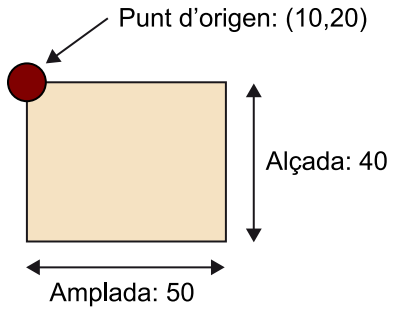
Recomanem al lector que provi cada tros de codi proposat per a comprovar-ne el funcionament. El desenvolupament s'ha fet amb la versió 1.5.1 del Processing, la més estable del moment. Al final de cada tros de codi s'adjunta un identificador entre parèntesis i precedit per dues contrabarres: aquest identificador indica l'*sketch* del Processing associat al codi, perquè l'estudiant el pugui obrir i executar. Podreu accedir a les diferents carpetes amb els codis descarregant el fitxer "recursos_visda.zip".

2. Dibuixar i farcir formes

Dibuixar en el Processing és molt simple. Només cal escriure una línia per a poder dibuixar, per exemple, un quadrat, o un cercle:

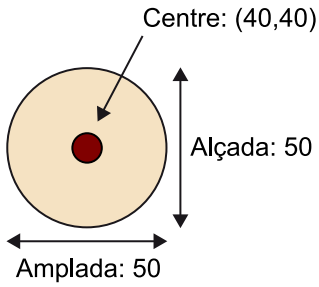
```
rect(10,20,50,40);
//(_1_010)
```

Figura 1. Dibuix d'un quadrat




```
ellipse(40,40,50,50);
//(_1_020)
```

Figura 2. Dibuix d'un cercle



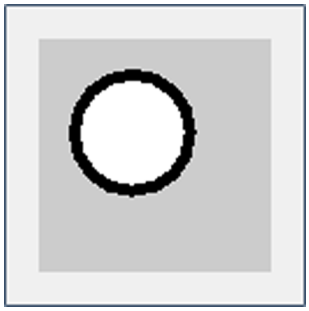
Recordeu que una el·lipse amb una amplada i una alçada iguals és un cercle.

2.1. Contorn de la forma

Podem canviar les diferents propietats de les formes anteriors amb la mateixa simplicitat. Per exemple, per a canviar el gruix del contorn del cercle anterior, només hem d'especificar la línia següent abans de dibuixar-lo.

```
strokeWeight(5);
ellipse(40,40,50,50);
//(_1_030)
```

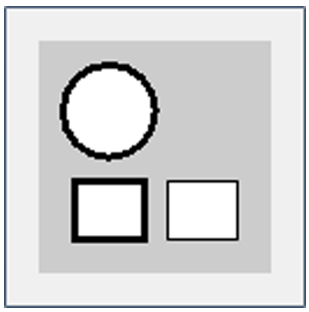
Figura 3. Cercle amb contorn gruixut



Hem de tenir en compte que, quan es canvia el gruix, tot el que es dibuixi a continuació es pintarà amb el gruix especificat. Si volem dibuixar una altra figura amb un gruix anterior, l'hem de tornar a especificar.

```
strokeWeight(3);
ellipse(30,30,40,40);
rect(15,60,30,25);

strokeWeight(1);
rect(55,60,30,25);
//(_1_040)
```

Figura 4. Diferents formes amb diferent gruix de contorn



Així, per no escriure tantes línies, intentarem dibuixar totes les figures amb característiques (en aquest exemple, de gruix de contorn) similars de manera seguida.

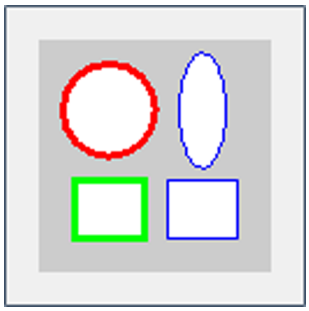
2.2. Colors

Una altra característica que es pot canviar és el color, tant del farcit com de la línia:

```
strokeWeight(3);
stroke(#FF0000);
ellipse(30,30,40,40);
stroke(#00FF00);
rect(15,60,30,25);
```

```
strokeWeight(1);
stroke(#0000FF);
rect(55,60,30,25);
ellipse(70,30,20,50);
//(_1_050)
```

Figura 5. Figures amb contorns de colors i gruixos diferents

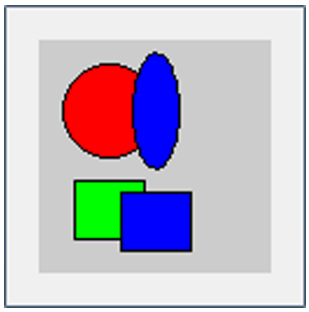


El color l'hem especificat amb format hexadecimal, el típic format que solem trobar en pàgines web. En l'últim exemple, com que tenim dues figures que volem pintar amb un contorn prim i de color blau, les hem pintat seguides, per a especificar el gruix i el color un sol cop.

També podem especificar el color del farcit. Per no fer el codi massa llarg, posarem tots els contorns iguals. Si no s'especifica cap contorn, es pren el negre i el gruix 1 per defecte:

```
fill(#FF0000);
ellipse(30,30,40,40);
fill(#00FF00);
rect(15,60,30,25);
fill(#0000FF);
rect(35,65,30,25);
ellipse(50,30,20,50);
//(_1_060)
```

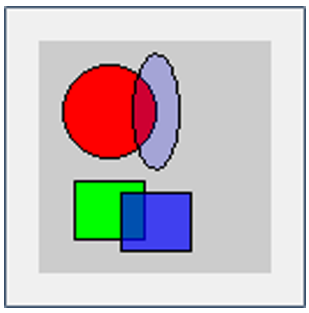
Figura 6. Figures amb farcits de diferents colors



Els objectes dibuixats després que altres es mostren a sobre. Si volem aplicar transparències als colors, només hem d'afegir un segon valor a la funció `fill` entre 0 (totalment transparent) i 255 (totalment opac):

```
fill(#FF0000);
ellipse(30,30,40,40);
fill(#00FF00);
rect(15,60,30,25);
fill(#0000FF,175);
rect(35,65,30,25);
fill(#0000FF,50);
ellipse(50,30,20,50);
//(_1_070)
```

Figura 7. Figures amb farcits de diferents colors i diferents transparències

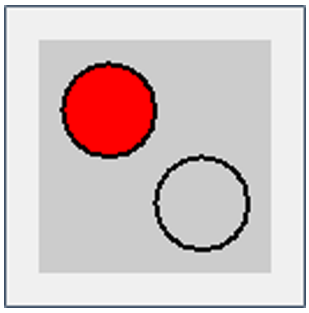


En aquest cas hem afegit una altra instrucció de `fill`, ja que volíem que el nivell de transparència per a l'el·lipse fos més alt que per al rectangle. Es pot veure com la zona superposada del cercle vermell de fons es veu més que la del rectangle verd.

També és possible no farcir els objectes per a obtenir, per exemple, una circumferència:

```
strokeWeight(2);
fill(#FF0000);
ellipse(30, 30,40,40);


noFill();
ellipse(70,70,40,40);
//(_1_080)
```

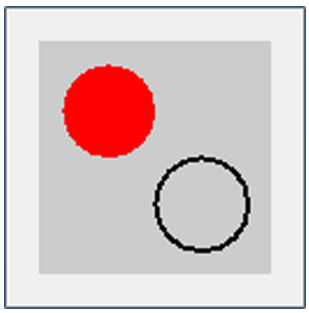
Figura 8. Un cercle vermell amb contorn negre i una circumferència negra



De manera similar, es pot dibuixar un cercle sense contorn:

```
strokeWeight(2);
noFill();
ellipse(70,70,40,40);


noStroke();
fill(#FF0000);
ellipse(30,30,40,40);
//(_1_090)
```

Figura 9. Un cercle vermell i una circumferència negra



En aquest cas hem canviat l'ordre de dibuix perquè, si haguéssim posat l'ordre `noStroke()` al principi, ens hauria obligat a posar una altra ordre `stroke(#000000)`, per a forçar el dibuix d'un contorn negre a la circumferència. Així escrivim menys i aprofitem els valors que el Processing ens posa per defecte.

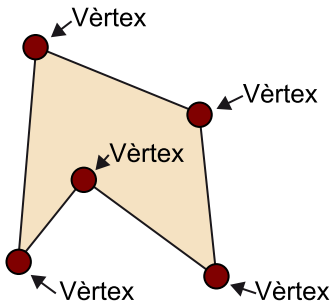
2.3. Objectes complexos

A continuació dibuixarem un polígon. Al contrari que amb els cercles, el·lipses i rectangles, necessitarem més d'una línia per a especificar un polígon. Bàsicament n'hem de definir els vèrtexs, especificant quan en comença i quan n'acaba la llista:

```
beginShape();
vertex(20,10);
```

```
vertex(70,30);
vertex(75,80);
vertex(35,50);
vertex(15,75);
endShape(CLOSE);
//(_1_100)
```

Figura 10. Dibuix d'un polígon

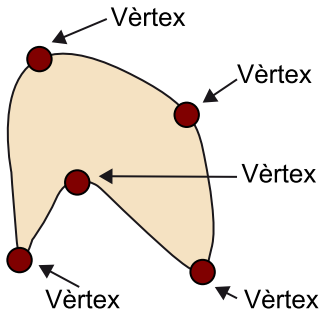


Com en les figures anteriors, podem modificar el contorn i el farcit de la mateixa manera, especificant-lo abans de començar el dibuix del polígon. Proveu a esborrar la paraula `CLOSE` del codi anterior i penseu com obtenir el mateix dibuix sense aquest mot (solució: repetint el primer vèrtex al final de la llista).

A continuació dibuixarem una forma corba. Aquests tipus de formes es poden especificar de diverses maneres. Aquí proposem utilitzar el `curveVertex`. Per definir una d'aquestes formes i que quedi tancada, només hem de posar la llista de punts (o vèrtexs) pels quals volem que passi, repetint els tres primers al final de la llista. Igual que amb el polígon, cal delimitar la llista de vèrtexs amb les mateixes dues instruccions:

```
beginShape();
curveVertex(20,10);
curveVertex(70,30);
curveVertex(75,80);
curveVertex(35,50);
curveVertex(15,75);
curveVertex(20,10);
curveVertex(70,30);
curveVertex(75,80);
endShape();
//(_1_110)
```

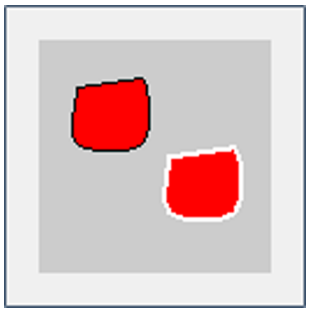
Figura 11. Dibuix d'una forma corba que passa per un conjunt de punts.



Finalment, per acabar aquesta primera secció, dibuixarem l'objecte que identificarà cada ítem d'informació del nostre *plug-in* de visualització, la nostra persona:

```
fill(#FF0000,220);
strokeWeight(0.5);
beginShape();
curveVertex(16, 16);
curveVertex(16, 20);
curveVertex(16, 44);
curveVertex(44, 44);
curveVertex(44, 16);
curveVertex(20, 16);
endShape(CLOSE);

strokeWeight(2);
stroke(#FFFFFF);
beginShape();
curveVertex(56, 46);
curveVertex(56, 50);
curveVertex(56, 74);
curveVertex(84, 74);
curveVertex(84, 46);
curveVertex(60, 46);
endShape(CLOSE);
//(_1_120)
```

Figura 12. Dibuix de la persona, sense seleccionar (amunt) i seleccionada (avall)



Hem dibuixat la persona d'una categoria concreta i amb les seves dues possibles aparences, sense seleccionar (amb contorn negre prim) i seleccionada (amb contorn blanc gruixut). Noteu que el farcit té una petita quantitat de transparència.

3. Escriure text

En aquesta secció veurem com s'escriu (o, millor dit, es dibuixa) text i es canvien algunes de les seves característiques.

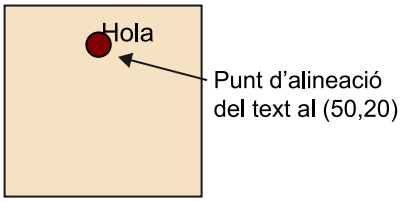
Primer de tot hem de començar seleccionant la font. Això es pot fer de diverses maneres, aquí en proposem una que aprofita les fonts instal·lades en el sistema. Seleccionarem una font prou comuna, l'helvètica, amb una mida inicial de 16 punts (píxels) d'alçada.

```
textFont(createFont("Helvetica",16));
```

Un cop seleccionada la font, podem escriure textos i situar-los en la zona de dibuix, de la mateixa manera que les formes que hem vist en la secció 1.

```
textFont(createFont("Helvetica",16));
text("Hola",50,20);
//(_2_010)
```

Figura 13. Text "Hola" dibuixat des del punt (50,20)



3.1. Mida i color

També podem jugar amb la mida i el color del text. El funcionament és el mateix que tenen les característiques de gruix i color en les formes, només que els textos no tenen contorn. Per a especificar la mida del text, podem fer servir `textSize`.

```
textFont(createFont("Helvetica",16));
fill(#FF0000);
text("Hola",50,20);
textSize(20);
text("Test",15,55);
fill(#222299);
textSize(10);
text("Casa",65,80);
//(_2_020)
```

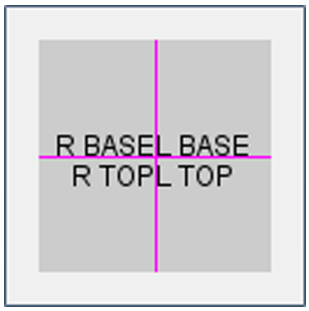
Figura 14. Diversos textos amb mides i colors diferents

3.2. Alineació

Una propietat del text és l'alineació. Donat un punt de localització del text, cal definir on queda el text: sobre el punt?, sota el punt?, a l'esquerra del punt?, a la dreta? Per defecte s'alinea amunt i a l'esquerra, la qual cosa vol dir que el punt de referència es troba en la cantonada inferior esquerra de la primera lletra del text.

```
textFont(createFont("Helvetica",12));
fill(#000000);
text("L BASE",50,50);
textAlign(LEFT,TOP);
text("L TOP",50,50);
textAlign(RIGHT);
text("R BASE",50,50);
textAlign(RIGHT,TOP);
text("R TOP",50,50);
stroke(#FF00FF);
line(50,0,50,100);
line(0,50,100,50);
//(_2_030)
```

Figura 15. Textos amb diferents alineacions: L (esquerra o *left*), R (dreta o *right*), TOP (superior o *top*), BASE (línia base, valor per defecte)



En la figura 15 es poden veure quatre exemples d'alineacions. Per a augmentar la claredat s'han dibuixat dues línies de color magenta: el punt (50.50) correspon a la intersecció de totes dues línies i també és el punt en què se situen tots

els textos. Mirant el codi, es pot veure quina alineació correspon a cada text. Per exemple, l'alineació (`LEFT`) (la més comuna) correspon al text "L BASE" i l'alineació (`RIGHT,TOP`), al text "R TOP".

3.3. Col·locació del text

A continuació, escriurem el text corresponent a la primera persona del nostre *plug-in*. Com que volem que el text se situï en la part inferior, ens cal conèixer l'alçada total de la zona de dibuix. El valor de l'alçada es troba amb la paraula clau `height`. La línia inferior la dibuixarem a aquesta alçada, i la línia superior a aquesta alçada menys el que ocupa una línia, perquè no se superposin. El que ocupa una línia és precisament la mida del text.

```
textFont(createFont("Helvetica",12));
textAlign(LEFT);
fill(#000000);
textSize(12);
text("Xavier Queralt",0,height);
textSize(10);
text("Entrevistes",0,height-12);
//(_2_040)
```

Figura 16. Dues línies de text situades en la part de baix de la zona de dibuix

El text anterior queda massa ajustat, tant a les vores de la zona de dibuix com entre aquestes. Podem donar una mica d'espai afegint marges a l'hora d'especificar l'alçada de cada línia de text. Deixarem un marge amb les vores igual a la mida de la font gran (valor de 12) i un espai entre línies de mitja mida de font (valor de 6).

```
textFont(createFont("Helvetica",12));
textAlign(LEFT);
fill(#000000);
textSize(12);
text("Xavier Queralt",12,height-12);
textSize(10);
text("Entrevistes",12,height-30);
```

```
//(_2_050)
```

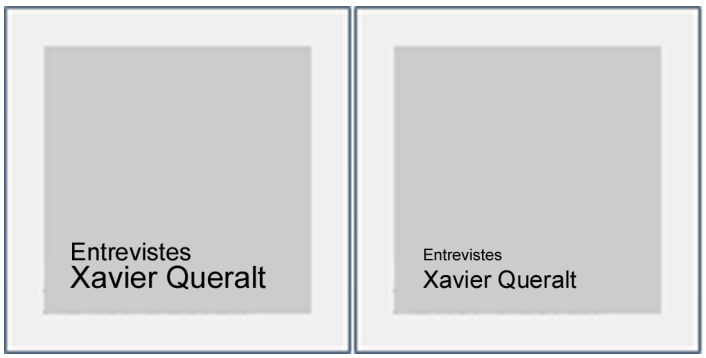
Figura 17. Dues línies de text amb marges i espaiat

Veiem que el codi anterior té molts números, però en realitat tots fan referència a la mateixa quantitat. En comptes de posar-los tantes vegades, associarem un nom al nombre. Això ho farem creant una variable, assignant-hi el valor i després posant el nom de la variable, fent així el codi més entenedor.

```
int medidaFuente = 12;
textFont(createFont("Helvetica",medidaFuente));
textAlign(LEFT);
fill(#000000);
textSize(medidaFuente);
text("Xavier Queralt",medidaFuente,height-medidaFuente);
textSize(medidaFuente*0.8);
text("Entrevistes",medidaFuente,height-medidaFuente*2.5);
//(_2_051)
```

Noteu que multipliquem per 0,8 per obtenir 10 i per 2,5 per obtenir el nombre 30. Amb això aconseguim que podem canviar la mida de tot plegat només canviant un sol valor, el de medidaFuente. Proveu a canviar el valor per 10, per exemple.

Figura 18. El mateix dibuix, però amb mides de fonts base diferents. Fixeu-vos com els marges i els espaiats són proporcionals a la mida dels textos.



3.4. Dibuixos i text simultanis

A continuació afegirem línies verticals acompanyades del número d'any, de manera similar a com ho volem construir en el *plug-in* de visualització. No obstant això, la zona de dibuix se'ns comença a quedar petita; i ho aprofitarem per a introduir l'ordre `size`, que ens permetrà establir les mides de la zona de dibuix que vulguem.

```
int medidaFuente = 16;
size(200,200);
textFont(createFont("Helvetica",medidaFuente));
textAlign(LEFT);
fill(#000000);
textSize(medidaFuente);
text("Xavier Queralt",medidaFuente,height-medidaFuente);
textSize(medidaFuente*0.8);
text("Entrevistes",medidaFuente,height-medidaFuente*2.5);

fill(#555555);
strokeWeight(1);
stroke(#000000,105);
textAlign(LEFT,TOP);
textSize(medidaFuente/2);

line(10,10,10,height-4*medidaFuente);
text("2001",10+medidaFuente/4,10);
line(120,10,120,height-4*medidaFuente);
text("2002",120+medidaFuente/4,10);
//(_2_060)
```

Figura 19. Dibuix de dues línies anuals i una etiqueta

Com podem veure, hem afegit dos blocs de codi, un primer amb la configuració de la línia i el text, i un segon que dibuixa les línies i escriu els dos números d'anys.

Però, i si volguéssim posar uns quants anys més? Hauríem de replicar tot el codi? No. Per a això podem fer servir funcions. Fins ara, hem treballat amb el mode *static* del Processing, que admet instruccions tal com les hem anat posant en els exemples. Passarem a introduir el mode *active*, que ens dóna més flexibilitat, ja que ens permet definir funcions.

```
int medidaFuente = 16;


void setup(){
  size(200,200);
  textFont(createFont("Helvetica",medidaFuente));
  textAlign(LEFT);
  fill(#000000);
  textSize(medidaFuente);
  text("Xavier Queralt",medidaFuente,height-medidaFuente);
  textSize(medidaFuente*0.8);
  text("Entrevistas",medidaFuente,height-medidaFuente*2.5);
  dibujarLineaAnual(2001,10);
  dibujarLineaAnual(2002,40);
  dibujarLineaAnual(2003,70);
  dibujarLineaAnual(2004,100);
  dibujarLineaAnual(2005,130);
  dibujarLineaAnual(2006,160);
  dibujarLineaAnual(2007,190);
}

void dibujarLineaAnual(int valor, int posicion){
  fill(#555555);
  strokeWeight(1);
  stroke(#000000,105);
  textAlign(LEFT,TOP);
  textSize(medidaFuente/2);
  line(posicion,10,posicion,height-4*medidaFuente);
  text(valor,posicion+medidaFuente/4,10);
}
//(_2_070)
```

Figura 20. Dibuix de set línies anuals i una etiqueta

Aquest codi obté un resultat similar al de la figura 19, però dibuixant moltes més línies. Cada línia que afegim es dibuixa amb una línia de codi, ja que hem definit una funció que es diu `dibujarLineaAnual`, que rep dos valors i els utilitza a dins seu per a dibuixar. D'aquesta manera reutilitzem codi i no escrivim tant. Si encara volem escriure menys, podem utilitzar un bucle `(for)` en comptes d'escriure les set línies amb `dibujarLineaAnual`, canviant:

```
...
  dibujarLineaAnual(2001,10);
  dibujarLineaAnual(2002,40);
  dibujarLineaAnual(2003,70);
  dibujarLineaAnual(2004,100);
  dibujarLineaAnual(2005,130);
  dibujarLineaAnual(2006,160);
  dibujarLineaAnual(2007,190);
...
```

per:

```
...
  for (int i=0;i<7;i++){
    dibujarLineaAnual(2001+i,10+30*i);
  }
...
//(_2_071)
```

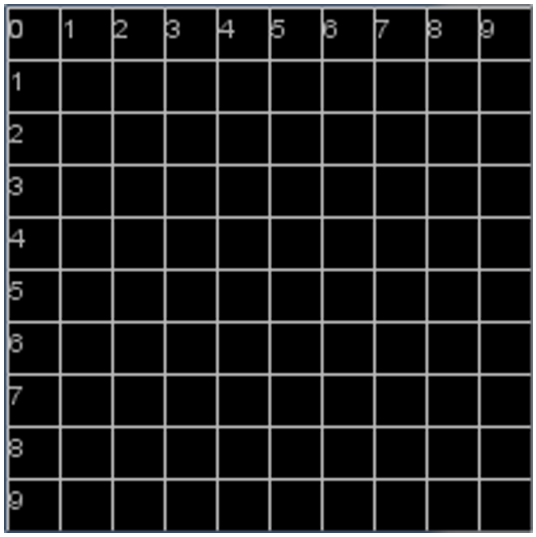
En aquests materials textuals utilitzem els punts suspensius (...) per a no haver de presentar de nou tot el codi superior i/o inferior a la nostra explicació. No és cap instrucció del Processing i si l'introduïm com a codi obtindrem un error.

4. Transformacions

Abans de començar a veure les transformacions, crearem una quadrícula que ens ajudarà a entendre'n millor el comportament dins del Processing. Utilitzarem el coneixement que ja tenim sobre dibuix de línies i text, però a més, afegirem una nova instrucció per poder posar un color de fons i deixar enrere el gris que apareix per defecte. Una possible quadrícula es pot crear com segueix:

```
int medidaFuente = 10;

void setup(){
  size(200,200);
  background(#000000);
  textFont(createFont("Helvetica",medidaFuente));
  stroke(#BBBBBB);
  fill(#AAAAAA);

  textAlign(LEFT,TOP);
  for (int i=0;i<11;i++){
      dibujarLineaVertical(i*20);
  }
  textAlign(LEFT,TOP);
  for (int i=0;i<11;i++){
      dibujarLineaHorizontal(i*20);
  }
}

void dibujarLineaVertical(int x){
  line(x,0,x,height);
  text(x/20,x+medidaFuente/8,medidaFuente/8);
}

void dibujarLineaHorizontal(int y){
  line(0,y,width,y);
  text(y/20,medidaFuente/8,y+medidaFuente/8);
}
//(_3_010)
```

Figura 21. Quadrícula de 10 ×10

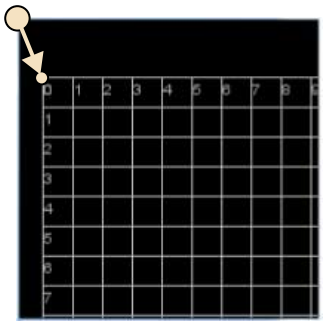


4.1. Translació

En l'exemple hem construït una quadrícula dibuixant deu línies verticals i deu línies horitzontals equiespaiades. Si volem desplaçar la quadrícula, només hem d'utilitzar la funció translate abans de fer el dibuix:

```
...
void setup(){
  size(200,200);
  background(#000000);
  textFont(createFont("Helvetica",medidaFuente));
  stroke(#BBBBBB);
  fill(#AAAAAA);

  translate(15,38);

  textAlign(LEFT,TOP);
  for (int i=0;i<11;i++){
      dibujarLineaVertical(i*20);
  }
  textAlign(LEFT,TOP);
  for (int i=0;i<11;i++){
      dibujarLineaHorizontal(i*20);
  }
}
...
//(_3_020)
```

Figura 22. Quadrícula desplaçada o traslladada 15 píxels a la dreta i 38 avall



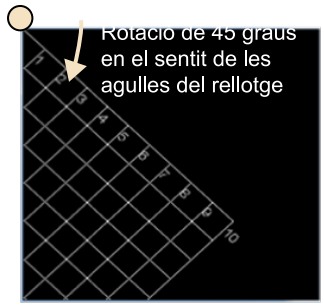
Podem provar diferents valors de desplaçaments o translació, fins i tot valors negatius. Una translació és un tipus de transformació que podem dur a terme al Processing. Però en podem fer de dos tipus més: rotació i escala.

4.2. Rotació

Vegem l'exemple següent sobre com es pot fer una rotació:

```
...
void setup(){
  size(200,200);
  background(#000000);
  textFont(createFont("Helvetica",medidaFuente));
  stroke(#BBBBBB);
  fill(#AAAAAA);

  rotate(radians(45));

  textAlign(LEFT,TOP);
  for (int i=0;i<11;i++){
      dibujarLineaVertical(i*20);
  }
  textAlign(LEFT,TOP);
  for (int i=0;i<11;i++){
      dibujarLineaHorizontal(i*20);
  }
}
...
//(_3_030)
```

Figura 23. Quadrícula que ha fet una rotació de 45 graus en el sentit de les agulles del rellotge



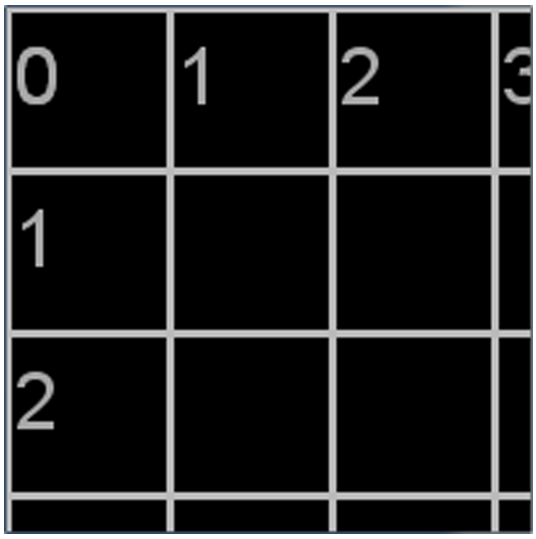
El sentit de les rotacions és sempre el de les agulles del rellotge. Podem fer rotacions en el sentit contrari especificant els graus amb valor negatiu.

4.3. Escala

L'escala és el tercer i últim tipus de transformació que veurem:

```
...
void setup(){
  size(200,200);
  background(#000000);
  textFont(createFont("Helvetica",medidaFuente));
  stroke(#BBBBBB);
  fill(#AAAAAA);

  scale(3.1);

  textAlign(LEFT,TOP);
  for (int i=0;i<11;i++){
      dibujarLineaVertical(i*20);
  }
  textAlign(LEFT,TOP);
  for (int i=0;i<11;i++){
      dibujarLineaHorizontal(i*20);
  }
}
...
//(_3_040)
```

Figura 24. Quadrícula escalada en un factor de 3,1



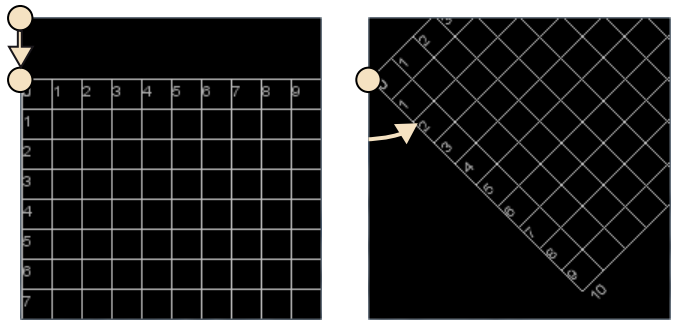
Com veiem, l'escala es una possible manera d'emular *zooms* quan treballem en dues dimensions. És important que notem que les transformacions de rotació i escala fan la seva funció sobre un punt de referència. Quan girem un objecte, necessitem pivotar sobre un punt, i quan fem una cosa gran o petita la fem centrant-nos en un punt concret. En el cas del Processing, el punt de referència és el (0,0), en els exemples, la cantonada superior esquerra. Si ens hi fixem, tant la rotació com l'escalat s'han dut a terme pivotant sobre aquest punt. Però, què passa si ho volem fer sobre un altre punt? A continuació veurem el comportament del Processing en dur a terme diferents transformacions seguides, i això ens permetrà pivotar sobre el punt que vulguem.

4.4. Acumulació de transformacions: l'ordre sí que importa

A continuació farem una comparació. Farem una translació i una rotació i en visualitzarem el resultat en fer-les en dos ordres diferents:

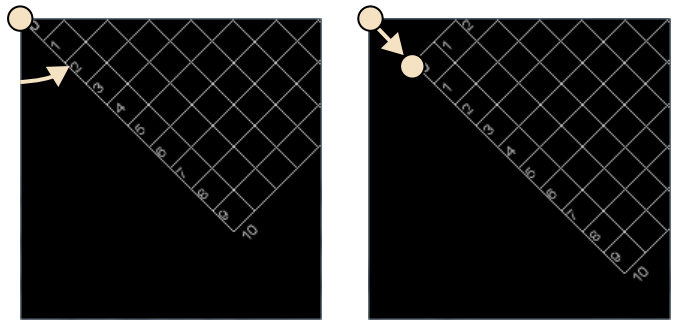
```
...
  translate(0,40);
  rotate(radians(-45));
...
//(_3_051), (_3_053)
```

Figura 25. Translació de 40 píxels (dues caselles de la quadrícula) seguida de rotació de 45 graus en sentit contrari a les agulles del rellotge



```
...
  rotate(radians(-45));
  translate(0,40);
...
//(_3_052), (_3_054)
```

Figura 26. Rotació de 45 graus en sentit contrari a les agulles del rellotge seguida d'una translació de 40 píxels (dues caselles de la quadrícula)



Una rotació sempre canvia l'orientació dels eixos (també coneguda com *el sistema de referència*) i la translació es fa en relació amb aquests eixos. Amb aquest exemple podem veure que quan fem una translació després d'una rotació, com que la rotació fa canviar els eixos, el desplaçament produït es fa sobre els eixos girats i això provoca que el desplaçament de la figura 26 no sigui en vertical, sinó en diagonal. En el cas de fer primer la translació, aquesta el que fa és moure el punt de pivotatge, per això la rotació subsegüent es fa al voltant del (`0,40`).

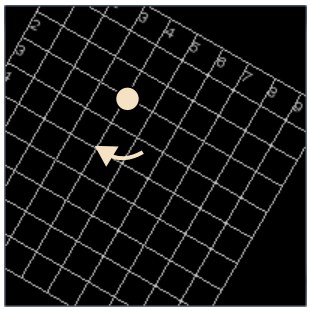
4.5. Canvi del punt de pivotatge en rotacions i escalats

Podem combinar diverses transformacions per a fer rotacions al voltant del punt que volem. Per exemple, per a efectuar un gir de 30 graus al voltant del punt (3,4) de la quadrícula necessitem tres transformacions seguides com:

```
...
  translate(80,60);
  rotate(radians(30));
  translate(-80,-60);
...
```

```
//(_3_060)
```

Figura 27. Rotació de 30 graus en el sentit de les agulles del rellotge pivotant sobre el punt (80,60), que és el punt (4,3) de la quadrícula.

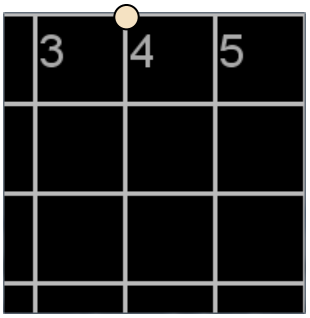


La rotació es fa entremig i les dues translacions tenen els mateixos desplaçaments, amb la diferència que en la segona translació es canvia el signe de les magnituds. És (80,60) perquè cada unitat de quadrícula que hem dibuixat ocupa 20 píxels.

Es pot procedir de manera similar amb l'escala:

```
...
  translate(80,0);
  scale(3);
  translate(-80,-0);
...
//(_3_070)
```

Figura 28. Escalat triple pivotant sobre el punt (80,0), que és el punt (4,0) de la quadrícula.



En aquest darrer cas hem escollit un punt d'alçada 0 per a poder veure les etiquetes de la quadrícula.

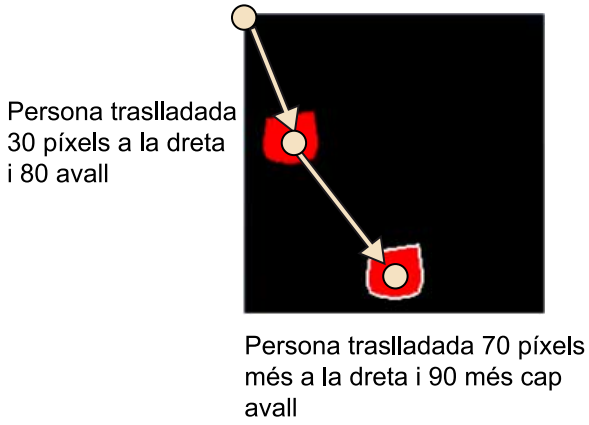
4.6. Ús de transformacions

Ara que coneixem les transformacions, podríem pensar a dibuixar una persona al voltant de l'origen de coordenades, el punt `(0,0)`, i després situar-la on vulguem mitjançant una transformació de translació. Definirem la funció de

dibuixar la persona i així escriurem menys codi si en dibuixem unes quantes. També definirem la variable `medidaPersona` per tenir identificada en una zona del codi la mida de les persones. La persona la podrem dibuixar seleccionada (amb un contorn blanc) o no seleccionada (amb un contorn negre).

```
int medidaPersona = 32;

void setup(){
  size(200,200);
  background(#000000);
  dibujarPersona(30,80,0);
  dibujarPersona(70,90,1);
}

void dibujarPersona(int tx, int ty, int seleccionada){
  translate(tx,ty);

  strokeWeight(2);
  fill(#FF0000);
  if (seleccionada==1){
     stroke(#FFFFFF);
  } else {
     stroke(#000000,50);
  }

  beginShape();
  curveVertex(-medidaPersona/2, -medidaPersona/2);
  curveVertex(-medidaPersona/2, -5*medidaPersona/14);
  curveVertex(-medidaPersona/2, medidaPersona/2);
  curveVertex(medidaPersona/2, medidaPersona/2);
  curveVertex(medidaPersona/2, -medidaPersona/2);
  curveVertex(5*medidaPersona/14, -medidaPersona/2);
  endShape(CLOSE);
}
//(_3_080)
```

Figura 29. Dibuix de dues persones, cadascuna traslladada a partir del sistema de referència anterior



Com podem veure en el codi de l'exemple de la figura 29, com que cada figura té la seva translació, en traslladar-la amb un `translate`, també estem modificant el sistema de referència. Quan situem la segona persona, la translació l'hem de fer tenint en compte la que hem fet a la primera persona. Això pot arribar a ser més complicat del necessari a mesura que afegim persones. El que volem és que cada persona tingui la translació corresponent, però que no es modifiqui el sistema de referència. Veurem la manera d'obtenir aquest comportament en la secció següent.

5. Sistemes de referència

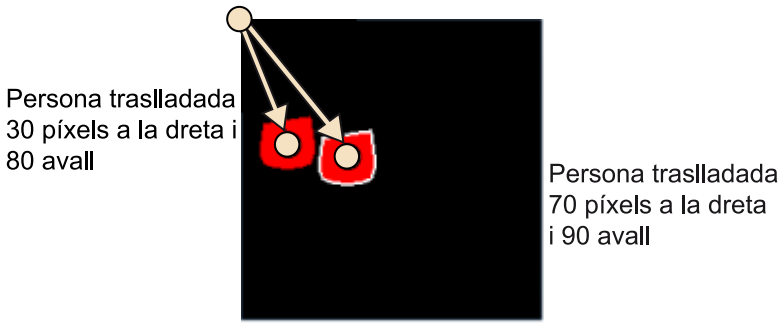
La secció anterior va finalitzar amb la situació en què cada cop que fèiem una transformació, el sistema de referència quedava modificat. Això ens pot ser d'utilitat a vegades, i d'altres ens pot donar més complicacions. Quan utilitzem funcions que ens dibuixen objectes, les pensem i codifiquem una sola vegada. Volem que tingui el funcionament dissenyat allà on la vulguem posar. En el cas de la secció anterior, això no es complia. Coses com aquestes afegeixen complexitat a l'hora de codificar un programa.

Si una transformació en canvia el sistema de referència, com podem tornar al sistema de referència anterior sense desfer la transformació? En el Processing s'aconsegueix utilitzant dues funcions: `pushMatrix()` i `popMatrix()`. No entrarem en el que signifiquen, sinó simplement en una manera efectiva d'utilitzar-les en la nostra aplicació. La idea és que `pushMatrix` guarda el sistema de referència vigent i `popMatrix` substitueix el sistema de referència vigent per l'últim que hem guardat amb `pushMatrix`. Si les utilitzem en la funció de dibuixar persona que produïa la figura 29, quedaria:

```
...
void dibujarPersona(int tx, int ty, int seleccionada){
  pushMatrix();
  translate(tx,ty);

  strokeWeight(2);
  fill(#FF0000);
  if (seleccionada==1){
     stroke(#FFFFFF);
  } else {
     stroke(#000000,50);
  }

  beginShape();
  curveVertex(-medidaPersona/2, -medidaPersona/2);
  curveVertex(-medidaPersona/2, -5*medidaPersona/14);
  curveVertex(-medidaPersona/2, medidaPersona/2);
  curveVertex(medidaPersona/2, medidaPersona/2);
  curveVertex(medidaPersona/2, -medidaPersona/2);
  curveVertex(5*medidaPersona/14, -medidaPersona/2);
  endShape(CLOSE);

  popMatrix();
}
...
```

```
//(_4_010)
```

Figura 30. Dues persones en dues posicions diferents respecte al mateix sistema de referència



Notem la diferència entre la figura 29 i 30? El fet de restaurar el sistema de referència quan tot just s'acaba de dibuixar la persona (havíem desat l'original abans de començar a dibuixar) permet que la funció dibuixi persones en la posició que diguem sempre respecte al mateix sistema de referència, que en aquest cas és el (0,0).

A continuació redibuixarem la figura 19 utilitzant `pushMatrix` i `popMatrix`. A més, utilitzarem funcions, com fem des de fa alguns exemples.

```
int medidaPersona = 16;
int medidaFuente = 16;
int espacioAnual = medidaPersona*4;

void setup(){
  size(200,200);
  background(#000000);
  textFont(createFont("Helvetica",medidaFuente));
  pushMatrix();
  translate(medidaFuente/2,medidaFuente/2);
  dibujarLineasAnuales(2001,2011);
  popMatrix();
  escribeTexto("Entrevistes","Xavier Queralt");
}

void dibujarLineasAnuales(int inicio, int fin){
  for (int i=inicio;i<=fin;i++){
     dibujarLineaAnual(i, (i-inicio)*espacioAnual);
  }
}

void dibujarLineaAnual(int valor, int posicion){
  pushMatrix();

  translate(posicion,0);
```

```
    fill(#AAAAAA);
    strokeWeight(1);
    stroke(#FFFFFF,105);
    textAlign(LEFT,TOP);
    textSize(medidaFuente/2);
    line(0,0,0,height-4*medidaFuente);
    text(valor,medidaFuente/4,0);

    popMatrix();
  }

  void escribeTexto(String categoria, String nombre){
    textAlign(LEFT);
    fill(#FFFFFF);
    textSize(medidaFuente);
    text(nombre,medidaFuente,height-medidaFuente);
    textSize(medidaFuente*0.8);
    text(categoria,medidaFuente,height-2.5*medidaFuente);
  }
  //(_4_020)
```

Figura 31. Línies dels anys i etiqueta inferior d'exemple

En aquest codi no fem una cosa gaire diferent de l'anterior (excepte la separació entre les línies que ara és més gran). En comptes de dibuixar dues persones, el que fem és dibuixar una sèrie de línies verticals amb una etiqueta de l'any i dues línies de text a sota. Fixeu-vos en l'ús de `pushMatrix` i `popMatrix` en la funció `setup`. El dibuix de les línies es troba entre aquests i després ve el dibuix del text. Totes les línies es dibuixen sota el sistema de referència fixat per `translate(medidaFuente/2,medidaFuente/2);` però el text es dibuixa sota el sistema de referència (0,0) original, ja que és després de `pop-`

`Matrix`, que reemplaça el sistema de coordenades posat pel translate, per la qual cosa s'havia desat amb el `pushMatrix`, que era el sistema de referència (0,0) original.

La funció que dibuixa les línies conté el mateix bucle que pintava la figura 20 (adaptat una mica per a permetre l'especificació d'un any inicial i un altre de final). La funció que pinta cada línia vertical amb la seva etiqueta és la mateixa que l'associada a la figura 20, però si ens hi fixem, veurem que hem delegat la responsabilitat de situar horitzonalment la línia a una transformació (`translate(posicion,0));` com que volem que la transformació no ens canviï el sistema de referència quan acabi la funció, utilitzem de nou `pushMatrix` i `popMatrix`.

Com a regla general, aconsellem l'ús de `pushMatrix` i `popMatrix` sempre que dibuixeu un objecte independent, posant-lo just abans de les transformacions i just després dels dibuixos.

D'altra banda, comencem a tenir un codi de més de quaranta línies i hi podem distingir una part de dibuix i una altra d'escriptura de text. Crearem dues pestanyes noves en l'IDE del Processing per cada una d'aquestes dues parts, ordenant així el codi (el teniu en l'`sketch_4_021`).

6. Incorporació de dades

Fins ara hem dibuixat persones a partir de dades concretes. En aquest punt tractarem de la part més abstracta del codi, la que en si mateixa no produeix cap resultat visual directament, però és imprescindible per al nostre codi de dibuix: el tractament de les dades a visualitzar.

En aquest exemple treballarem amb un conjunt de dades predefinides. En apartats posteriors es veurà com es pot adaptar el codi per a poder admetre dades entregades per codi PHP i l'accés a una base de dades.

Així, crearem una nova pestanya en l'IDE del Processing que contindrà la variable `String[][] datos` amb un contingut predefinit (vegeu l'`sketch_5_010`). Quan accedim a aquesta variable ho farem així:

```
datos[i][j]
```

on `i` simbolitza el número de registre o número de fitxa, i `j` indica el camp de la fitxa. En el contingut predefinit hi ha 314 fitxes de sis camps cada una. Hem de tenir en compte que, en el Processing, les indexacions comencen per 0, per tant hi haurà les fitxes des de la 0 fins a la 313 i els camps des del 0 fins al 5. La nostra persona serà cadascuna de les fitxes continguda a `datos`.

6.1. El temps UNIX

El quart camp de les persones de les nostres dades conté la data en el format de temps UNIX (o temps POSIX). Aquest format indica el nombre de segons transcorreguts des de les 00.00 de l'1 de gener de 1970. És una manera com una altra de tenir la data fins a una precisió de segons en un sol nombre. La data també apareix en els tercers camps, en format alfanumèric. No obstant això, degut a serà més fàcil i eficient treballar amb un únic valor que amb una cadena de caràcters, utilitzarem el temps UNIX per a manejar internament les dades i poder-les visualitzar en la línia de temps.

6.2. Ordenació de les dades per temps

Atès que la visualització de dades que volem fer dibuixa les persones ordenades en el temps, sembla lògic pensar que si les dades estan ordenades per temps les podrem dibuixar més fàcilment. D'aquesta manera, atès que no podem assegurar que les dades que manegem tinguin cap ordenació, les ordenarem nosaltres mateixos. Per a fer-ho, utilitzarem l'algorisme QuickSort, que acompanya aquests materials:

```
datos = ordenar(datos,0,datos.length-1);
```

```
//(_5_020)
```

Aquesta línia la podem posar en la funció `setup`, abans d'escriure ni dibuixar res. El que fa és rebre la variable `datos` i retornar-la ordenada. Podeu trobar el codi en els materials, on veureu que hi ha una pestanya on apareix el codi de l'algorisme d'ordenació. El resultat el desarem en la mateixa variable, i la modificarem per a les noves dades ordenades.

També val la pena marcar l'aparició de `length`. En una llista, ens diu el seu nombre de posicions. Atès que `datos` és una llista (tècnicament, *array*) de fitxes (o persones), denotat pel primer `[]`, podem saber el nombre de persones amb la propietat `length`. El mateix podem fer per a una persona concreta, per exemple `datos[0].length` ens diu el nombre de camps que té la primera persona.

6.3. Informació a partir de les dades

Ara que tenim les dades ordenades, buscarem el marge d'anys. Precisament, com que les tenim ordenades de la més antiga a la més recent, només haurem d'agafar la primera i l'última i veure els seus anys.

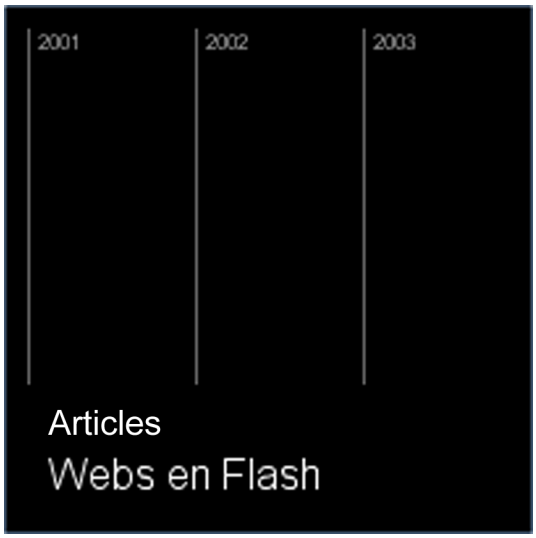
```
void margenAnual(){
  int epochPrimero, epochUltimo;
  epochPrimero = int(datos[0][3]);
  epochUltimo = int(datos[datos.length-1][3]);
  primero = epochPrimero/31558464+1970;
  ultimo = epochUltimo/31558464+1970;
}
//(_5_030)
```

Aquest codi també el posarem en una pestanya nova. El nombre `31558464` és el nombre mitjà de segons que conté un any. `primero` i `ultimo` són dues variables globals que haurem definit en la pestanya del codi de la funció `setup`, que quedarà com segueix, després d'afegir l'ordenació de les dades (recordem que `datos` es troba en una pestanya pròpia, atesa l'extensió que té):

```
int medidaPersona = 16;
int medidaFuente = 16;
int espacioAnual = medidaPersona*4;
int primero;
int ultimo;

void setup(){
  size(200,200);
  background(#000000);
  textFont(createFont("Helvetica",medidaFuente));
  datos = ordenar(datos,0,datos.length-1);
```

```
  margenAnual();
  pushMatrix();
  translate(medidaFuente/2,medidaFuente/2);
  dibujarLineasAnuales(primero,ultimo+1);
  popMatrix();
  escribeTexto(datos[0][4],datos[0][0]);
}
//(_5_030)
```

Figura 32. La línia de temps i el text de la primera persona



6.4. Col·locació de les persones

El pas següent és dibuixar les persones en el seu ordre de temps. Però abans necessitem saber la posició horitzontal i vertical de les persones. La posició horitzontal ha de ser proporcional al temps, i la vertical ha d'anar en funció de la categoria (camp sisè de la persona). Sabem que només hi ha quatre categories diferents. Per a conèixer-ne les posicions horitzontal i vertical:

```
void calcularPosicionesPersonas(){
  int incrementoPorApilamiento;
  posx = new int[datos.length];
  posy = new int[datos.length];
  incrementoPorApilamiento = (height-6*medidaFuente-medidaPersona)/3;
  for (int k=0;k<datos.length;k++){
     posx[k] = int((espacioAnual/31558464.0)*(float(datos[k][3])-(primero-1970)*31558464.0));
     posy[k] = int(datos[k][5])*incrementoPorApilamiento;
  }
}
//(_5_040)
```

`posx` i `posy` són dos *arrays* (o llistes) de valors. `posx` conté les posicions horitzontals de les persones i posy, les verticals. Aquestes dues variables les afegim a la llista de variables globals de la nostra aplicació. `incrementoPorApilamiento` és la distància vertical en píxels que hi ha entre categories; a l'hora de calcular-la hem de saber l'espai vertical disponible, que és igual a la longitud de la línia vertical menys els marges que deixem (dues vegades el que ocupa la font de marge vertical i horitzontal i el que ocupa una persona, ja que les persones es col·loquen segons el seu centre i tenen una alçada que no és 0 – ocupen un espai). La posició horitzontal es calcula com l'excedent en segons des de l'any `primero` i és proporcional a l'`espacioAnual`. La posició vertical s'obté multiplicant l'índex de la categoria (aprofitant que és entre 0 i 3) per l'`incrementoPorApilamiento`.

6.5. Dibuix de la població

Ja tenim tota la informació necessària per a poder dibuixar totes les persones. Per a fer-ho, utilitzarem el codi de `dibujarPersona` de la figura 32, però substituint `fill(#FF0000)` per:

```
...
  switch(categoria){
   case 0:
      fill(#FF0000);
      break;
   case 1:
      fill(#00FF00);
      break;
   case 2:
      fill(#0000FF);
      break;
   case 3:
      fill(#FF00FF);
      break;
  }
...
//(_5_050)
```

i afegint una variable a la capçalera, que sigui `categoria`:

```
...
void dibujarPersona(int tx, int ty, int categoria, int seleccionada){
...
//(_5_050)
```

El codi per a dibuixar la població serà el següent i s'inserirà en la funció `setup` just després de `dibujarLineasAnuales`:

```
void dibujarPoblacion(){
  pushMatrix();
  translate(0,medidaFuente+medidaPersona/2);
  for (int i=0;i<datos.length;i++){
     dibujarPersona(posx[i],posy[i],int(datos[i][5]),0);
  }
  popMatrix();
}
//(_5_050)
```

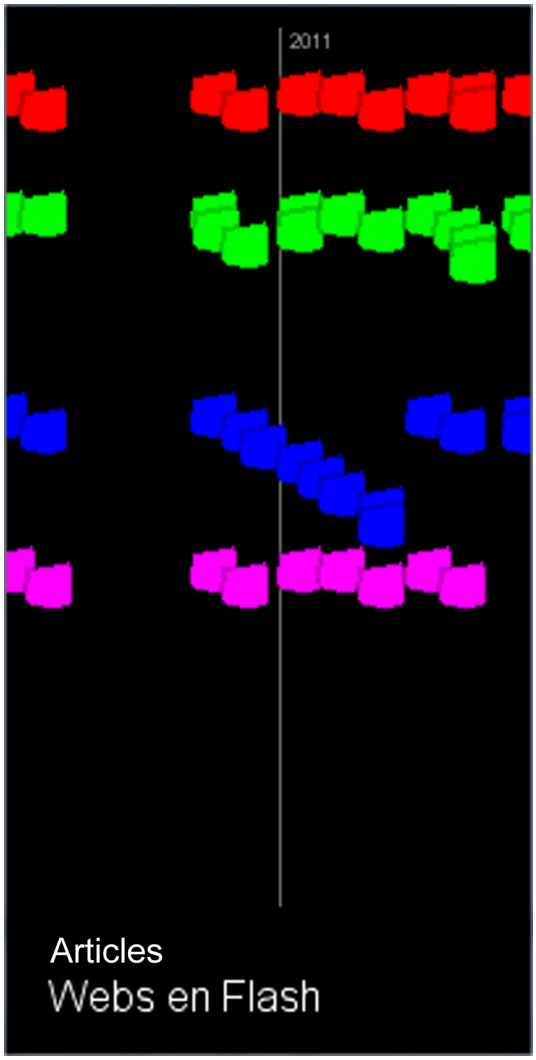
Figura 33. Dibuix de totes les persones, les línies anuals i el text de la primera persona

En la figura 33 hem establert un `espacioAnual` més petit per poder veure totes les persones, però queda clar que està massa condensat i l'haurem d'espaiar també verticalment. Per a fer-ho, es proposa modificar el codi de `calcularPosicionesPersonas`, que detalla una mica més la posició vertical de les persones en funció de si una persona queda superposada a la immediatament anterior de la mateixa categoria:

```
void calcularPosicionesPersonas(){
  float incrementoPorApilamiento;
  int[] maxposy = new int[4];
  int i,j;
  posx = new int[datos.length];
  posy = new int[datos.length];
  for (int k=0;k<datos.length;k++){
     posx[k] = int((espacioAnual/31558464.0)*(float(datos[k][3])-(primero-1970)*31558464.0));
  }

  for (int categoria=0;categoria<4;categoria++){
     i = 0;
     maxposy[categoria] = 0;
     while(int(datos[i][5])!=categoria && i<datos.length){
```

```
        i++;
    }
    for (j = i+1; j<datos.length;j++){
       if (int(datos[j][5])==categoria){
         if ((posx[j]-posx[i])<medidaPersona){
           posy[j] = posy[i]+1;
           if(posy[j]>maxposy[categoria]){
             maxposy[categoria] = posy[j];
           }
         }
         i = j;
       }
      }
    }
    maxposy[1] = maxposy[0]+maxposy[1];
    maxposy[2] = maxposy[1]+maxposy[2];
    maxposy[3] = maxposy[2]+maxposy[3];
    incrementoPorApilamiento = float(height-5*medidaFuente-4*medidaPersona)/(maxposy[3]-1.0);

    int offset;
    for (i=0;i<datos.length;i++){
      if (int(datos[i][5])>0){
        offset = int(maxposy[int(datos[i][5])-1]*incrementoPorApilamiento+(int(datos[i][5]))
        *medidaPersona);
      } else {
        offset = 0;
      }
      posy[i] = int(posy[i]*incrementoPorApilamiento+offset);
    }
  }
  //(_5_060)
```

Figura 34. Les persones que es troben prop de l'any 2011



A part de fer l'espai de visualització el doble d'alt, s'han espaiat els anys dotze vegades la mida de la persona. En la figura 34 es pot veure el resultat aplicant:

```
translate(medidaFuente/2-9.5*espacioAnual,medidaFuente/2);
```

en comptes de:

```
translate(medidaFuente/2,medidaFuente/2);
```

per a poder veure una zona en què hi ha totes quatre categories. Noteu com quan les persones se superposen horitzontalment entre elles, es van desplaçant cap avall, per a augmentar la seva zona visible.

7. Animacions

Volem donar un toc més viu a la visualització de les dades que proposem. Per fer-ho, recorrerem a les animacions, fent que les persones es moguin una mica, eliminant l'efecte estàtic que presenta la visualització que portem fins a aquest punt.

Primer de tot dibuixarem una persona que es mou d'esquerra a dreta de la zona de la pantalla i torna a entrar quan surt. Per a fer-ho, necessitem col·locar el codi de dibuix dins de la funció `draw` i establir la velocitat d'actualització amb `frameRate`:

```
int medidaPersona = 16;
int x;
float velocidadX;

void setup(){
  size(200,200);
  background(#000000);
  x = -medidaPersona/2;
  velocidadX = 2;
 frameRate(25);
}

void draw(){
  background(#000000);
  if (x>(width+medidaPersona/2)){
     x = -medidaPersona/2;
  } else {
     x=int(x+velocidadX);
  }
  dibujarPersona(x,100,0,0);
}
//(_6_010)
```

La funció `draw` es crida un nombre de vegades per segon, especificat per `frameRate`. En aquest exemple, cada cop que es crida la funció `draw` es fa el següent:

- Esborrar la pantalla amb `background(#000000)`.

- Trobar la nova posició; si no ha sortit per la dreta, la posició s'actualitza sumant-hi la velocitat; si no, es restableix al valor inicial (especificat inicialment a `setup`).

- Es dibuixa la persona amb la posició actual.

Així, com veiem, podem aconseguir una animació redibuixant constantment l'escena amb objectes que canvien de posició. Podem fixar-nos que, en aquest exemple, la velocitat de la persona depèn de les variables `velocidadX` i del valor que hem especificat a `frameRate`. Com més `frameRate` i com més `velocidadX`, més rapidesa. Recordem que per a aconseguir una escena fluida necessitem un mínim de 25 imatges per segon.

Ara canviarem el mode en què la persona es mou fent-la oscil·lar verticalment:

```
void draw(){
  background(#000000);
  x = int(x + velocidadX);
  dibujarPersona(100,int(100+10*cos(radians(x))),0,0);
}
//(_6_020)
```

En aquest cas ens estalviem la distinció entre sumar i restablir la `x`, ja que una oscil·lació es pot emular amb una funció sinusoïdal i un angle, que pot tenir un valor il·limitat i que, en aquest cas, emula la `x`. Com a curiositat, si posem una velocitat de 14,4 i 25 imatges per segon, obtindrem una oscil·lació per segon en l'animació.

Proveu de variar la velocitat i les imatges per segon per experimentar diferents velocitats. Proveu de disminuir les imatges per segon a menys de 25 per deixar d'experimentar una animació fluida.

7.1. Animació del conjunt de persones

Agafarem el codi que produeix la figura 34 i afegirem un efecte d'animació d'oscil·lació a les persones. Per fer-ho, col·locarem el codi de dibuix dins de la funció draw, establirem les imatges per segon a 25 i modificarem una mica la funció de dibuix de la població:

```
void setup(){
  size(200,400);
  background(#000000);
  textFont(createFont("Helvetica",medidaFuente));
  datos = ordenar(datos,0,datos.length-1);
  margenAnual();
  calcularPosicionesPersonas();
  angulo = 0.0;
```

```
  velocidadOscilacion = 7.2;
  amplitudOscilacion = 3;
  frameRate(25);
}

void draw(){
  angulo = angulo + velocidadOscilacion;
  background(#000000);
  pushMatrix();
  translate(medidaFuente/2-9.5*espacioAnual,medidaFuente/2);
  dibujarLineasAnuales(primero,ultimo+1);
  dibujarPoblacion();
  popMatrix();
  escribeTexto(datos[0][4],datos[0][0]);
}

// En la pestanya dibuixat
void dibujarPoblacion(){
  pushMatrix();
  translate(0,medidaFuente+medidaPersona/2);
  for (int i=0;i<datos.length;i++){
     pushMatrix();
     translate(0,amplitudOscilacion*cos(radians(angulo+i+ int(datos[i][5])*80)));
     dibujarPersona(posx[i],posy[i],int(datos[i][5]),0);
     popMatrix();
  }
  popMatrix();
}
//(_6_030)
```

Hem afegit tres variables: `angulo`, `velocidadOscilacion` i `amplitudOscilacion`. La tercera ens marcarà si el recorregut de l'oscil·lació de la persona és gran o curt. Notem que tot el codi de dibuix ha passat a la funció de `draw`. Per afegir l'efecte d'oscil·lació, a la funció de `dibujarPoblacion` hem afegit una transformació de translació abans de dibuixar la persona que depèn del valor d'`angulo` actual, de la mateixa persona `(i)` i de la seva categoria (`int(datos[i][5])*80`); així aconseguim que les diferents persones no oscil·lin alhora i les diferents categories, tampoc.

8. Interacció amb el ratolí

Ja només ens queda afegir la interacció de l'aplicació. El dispositiu per a interaccionar serà el ratolí. El Processing està específicament preparat i conté una sèrie de variables que emmagatzemen la posició i l'estat dels botons del ratolí, i també funcions que es criden cada vegada que hi ha un esdeveniment (moure's i prémer un botó, entre d'altres).

8.1. Les coordenades del ratolí

A continuació dibuixarem una persona a la posició en què es troba el ratolí dins de la zona de dibuix. Introduirem les variables on el Processing guarda la posició del ratolí, anomenades `mouseX` i `mouseY`, i utilitzarem el codi de `dibujarPersona` que ja tenim:

```
int medidaPersona = 16;

void setup(){
  size(200,200);
  background(#000000);
  frameRate(25);
}

void draw(){
  background(#000000);
  dibujarPersona(mouseX,mouseY,0,0);
}
//(_7_010)
```

`mouseX` i `mouseY` contenen el valor de la posició horitzontal i vertical respectivament del ratolí en cada moment. Si el ratolí surt fora de la zona de dibuix, contenen l'últim valor que tenia mentre era a dins. Proveu a eliminar la línia de `background(#000000)` i experimenteu quin comportament obté el codi.

8.2. Botons del ratolí

El Processing no solament té variables amb informació sobre el ratolí. També té funcions que es criden quan fem segons quines accions (o esdeveniments) amb el ratolí. Per exemple, les funcions `mousePressed()` i `mouseReleased()` s'activen quan es prem o es deixa anar un botó del ratolí (el que sigui). En l'exemple anterior, farem que canviï de categoria (i, per tant, de color) quan tinguem algun botó del ratolí premut, i que retorni a la categoria inicial quan el botó es deixi anar:

```
int medidaPersona = 16;
int activado = 0;

void setup(){
  size(200,200);
  background(#000000);
  frameRate(25);
}

void draw(){
  background(#000000);
  dibujarPersona(mouseX,mouseY,activado,0);
}

void mousePressed(){
  activado = 1;
}

void mouseReleased(){
  activado = 0;
}
//(_7_020)
```

El codi anterior també es pot dissenyar utilitzant la variable `mousePressed` (en comptes de la funció). Aquesta variable, de manera similar a `mouseX` i `mouseY`, val `true` si es troba algun botó premut i false en cas contrari. El codi queda més curt que en el cas anterior, però en funció del cas ens anirà millor controlar els clics del ratolí amb les funcions, i en d'altres, amb la variable:

```
int medidaPersona = 16;

void setup(){
  size(200,200);
  background(#000000);
  frameRate(25);
}

void draw(){
  background(#000000);
  if (mousePressed==true){
     dibujarPersona(mouseX,mouseY,1,0);
  } else {
     dibujarPersona(mouseX,mouseY,0,0);
  }
}
//(_7_030)
```

8.3. *Rollover*

Detectar quan el ratolí es troba sobre determinats objectes o zones de la zona de dibuix és un comportament necessari en moltes aplicacions. Per a fer-ho, cal comprovar on és el ratolí cada vegada que redibuixem la zona, és a dir, cada vegada que es crida la funció `draw`. A continuació dibuixarem una persona i comprovarem si el ratolí és a sobre d'ella: quan sigui així, la dibuixarem seleccionada (amb un contorn de color diferent), i desseleccionada en cas contrari:

```
int medidaPersona = 16;
int posicionX = 100;
int posicionY = 100;

void setup(){
  size(200,200);
  background(#000000);
  frameRate(25);
}

void draw(){
  background(#000000);
  if (detectarPersona()==true){
     dibujarPersona(posicionX,posicionY,0,1);
  } else {
     dibujarPersona(posicionX,posicionY,0,0);
  }
}

boolean detectarPersona(){
  int limiteSuperior = posicionY-medidaPersona/2;
  int limiteInferior = posicionY+medidaPersona/2;
  int limiteIzquierda = posicionX-medidaPersona/2;
  int limiteDerecha = posicionX+medidaPersona/2;
  if (mouseX>=limiteIzquierda && mouseX<=limiteDerecha && mouseY>=limiteSuperior &&
  mouseY<=limiteInferior) {
     return true;
  } else {
     return false;
  }
}
//(_7_040)
```

El procediment clau és veure si el ratolí és dins dels quatre límits (superior, inferior, esquerre i dret) que defineixen la caixa imaginària que conté la persona (caixa coneguda com a *bounding box*). Això es fa amb el condicional (`if`) que es troba a `detectarPersona`.

8.4. Ús de la posició del ratolí

Al final de la secció 4.3.6 teníem el dibuix de tot el nostre món. Però per a visualitzar-ho tot havíem de fer els espais entre anys (`espacioAnual`) molt petits o fer la finestra de dibuix molt ampla. Hi ha una altra alternativa per a poder-ho veure tot que es basa en la navegació: el que tenim dibuixat és una escena concreta del món; si naveguem pel món, podrem canviar l'escena i serem capaços de veure'l sencer sense necessitat de fer la zona de dibuix més gran ni disminuir l'espai entre els anys. El comportament que tindrem consistirà a desplaçar la línia de temps a l'esquerra si posem el ratolí en l'extrem esquerre de la zona de dibuix i desplaçar-lo a la dreta si el posem en la zona de l'extrem dret.

Partim del codi final que teníem al final de la secció 4.3.6 i hi afegirem una funció que ens detecti si el ratolí és en els extrems de la zona de dibuix i ens retorni com d'extrem es troba. Aquesta funció la situarem dins d'una nova pestanya que anomenarem *interacció*. La idea serà variar la translació fixa següent, que es troba dins de `draw`:

```
translate(medidaFuente/2-9.5*espacioAnual,medidaFuente/2);
```

utilitzant una variable `panHorizontal`, que canviarà segons el que ens vagi retornant la funció anterior (de detecció del ratolí):

```
translate(-panHorizontal,medidaFuente/2);
//(_7_050)
```

El motiu del signe negatiu és que si, per exemple, volem crear la sensació de desplaçar-nos cap a la dreta, haurem de dibuixar el món cap a l'esquerra. La variable `panHorizontal` estarà inicialitzada en setup al valor que tingui la primera persona, restant-li un marge igual a la mida de dues persones:

```
panHorizontal = posx[0]-medidaPersona*2;
//(_7_050)
```

Aquesta variable s'actualitzarà a la funció `draw` fent servir la funció de detecció del ratolí, abans de dibuixar ni fer cap transformació:

```
//(_7_050)
```

La funció de `detectarMouse` retornarà la modificació de la translació horitzontal, de manera que com més a prop del marge siguem, més modificació provocarem:

```
float detectarMouse(){
  if (mouseX<width*0.2 && panHorizontal>posx[0]-medidaPersona*2){
     return mouseX-width*0.2;
```

```
    }
    if (mouseX>width*0.8 && panHorizontal<posx[posx.length-1]-width+2*medidaFuente){
       return mouseX-width*0.8;
    }
    return 0;
  }
  //(_7_050)
```

Hem agafat uns marges del 20% de l'amplada de la zona de dibuix. Si no es troben dins dels marges es retornarà 0, la qual cosa provocarà una translació horitzontal nul·la. Si ens trobem dins dels marges, com més a l'extrem es trobi la posició del ratolí, més gran serà el valor retornat. Si es troba a l'esquerra, el valor serà negatiu (i farà decréixer el valor de la translació horitzontal provocada per `panHorizontal`), i si es troba a la dreta, positiu (i farà créixer el valor de la translació horitzontal).

8.5. Dibuix selectiu

Amb el codi que portem fins a aquest punt estem dibuixant tot el món, però només visualitzem una finestra concreta. La pregunta és: cal dibuixar-ho tot si només en veurem una part? La resposta és que no. El que no es veu no cal dibuixar-ho. Així fem el nostre codi més eficient i no consumim recursos de la màquina inútilment (dibuixar una cosa que no es veu no té gaire sentit). Per a fer això, hem de modificar la funció de `dibujarPoblación`, fent que només dibuixi les persones que surtin en la zona de dibuix. Tenim informació de l'amplada de la zona de dibuix (`width`) i també tenim la informació de la translació horitzontal (`panHorizontal`). Atès que tenim les dades ordenades, podem començar dibuixant la persona que es trobi amb un valor de translació igual a `panHorizontal` (menys un petit marge) i anar dibuixant les següents fins a arribar a la que es trobi a `panHorizontal+width`, més un marge (o fins que arribi a l'última persona):

```
void dibujarPoblacion(){
  int i = 0;
  pushMatrix();
  translate(0,medidaFuente+medidaPersona/2);
  while(posx[i]<panHorizontal-medidaPersona){
     i++;
  }
  while(posx[i]<panHorizontal+width+medidaPersona){
     pushMatrix();
     translate(0,amplitudOscilacion*cos(radians(angulo+i+ int(datos[i][5])*80)));
     dibujarPersona(posx[i],posy[i],int(datos[i][5]),0);
     popMatrix();
     i++;
     if (i>=datos.length){
       break;
```

```
      }
    }
    popMatrix();
  }
  //(_7_060)
```

Aquest és un tipus de canvi en el codi que no té cap impacte visual directe, sinó indirecte. El fet que alliberem la màquina de fer càlculs, vol dir que ho podem aprofitar per a dibuixar més objectes o efectes visuals que enriqueixin l'aplicació.

8.6. Detecció de la persona amb *rollover*

A continuació codificarem el comportament de *rollover* per saber si el ratolí és a sobre d'alguna persona en concret. En aquest cas la marcarem com a seleccionada i la pintarem amb un contorn blanc, per saber que hi som a sobre. Per dur-ho a terme farem els afegits següents: en primer lloc, comprovar per a totes les persones que dibuixem (seguint amb la filosofia d'abans, no cal mirar si som a sobre de persones que no surten en la zona de dibuix) que som a sobre d'alguna d'elles; en segon lloc, si som a sobre d'alguna, dibuixar-la amb el paràmetre de `seleccionada` a 1. Igual que en el codi que vam realitzar per detectar si érem a sobre d'una persona, la comprovació la farem cada vegada que redibuixem la zona de dibuix, ja que la posició del ratolí pot canviar en qualsevol moment.

Abans de res, augmentarem la zona visible a 400 × 400, per tenir més espai i poder observar millor l'efecte de selecció de la persona. Seguidament, definirem la funció `detectarPersona(i)`, que ens retornarà 1 si el ratolí es troba sobre la persona número `i` o 0 en cas contrari:

```
int personaActiva(int i){
  float limiteSuperior = posy[i]-medidaPersona/2+3*medidaFuente/2+
  medidaPersona/2+amplitudOscilacion*cos(radians(angulo+i+ int(datos[i][5])*80));
  float limiteInferior = posy[i]+medidaPersona/2+3*medidaFuente/2+
  medidaPersona/2+amplitudOscilacion*cos(radians(angulo+i+ int(datos[i][5])*80));
  float limiteIzquierda = posx[i]-panHorizontal-medidaPersona/2;
  float limiteDerecha = posx[i]-panHorizontal+medidaPersona/2;
  if (mouseX>=limiteIzquierda && mouseX<=limiteDerecha && mouseY>=limiteSuperior &&
  mouseY<=limiteInferior) {
     return 1;
  } else {
     return 0;
  }
}
//(_7_070)
```

Podem veure que el codi és molt similar al que hem utilitzat per a l'exemple anterior de *rollover*, només que la `posicionX` i `posicionY` les hem fixat acumulant totes les transformacions de translació que hem dut a terme fins al punt de la crida a aquesta funció, és a dir, hem acumulat, per horitzontal i vertical, les translacions següents:

```
translate(-panHorizontal,medidaFuente/2);
translate(0,medidaFuente+medidaPersona/2);
translate(0,amplitudOscilacion*cos(radians(angulo+i+ int(datos[i][5])*80)));
```

Així, podem utilitzar el mateix codi, només que `posicionX` és igual a l'acumulació següent:

```
-panHorizontal+0+0 = -panHorizontal
```

i `posicionY` esdevé:

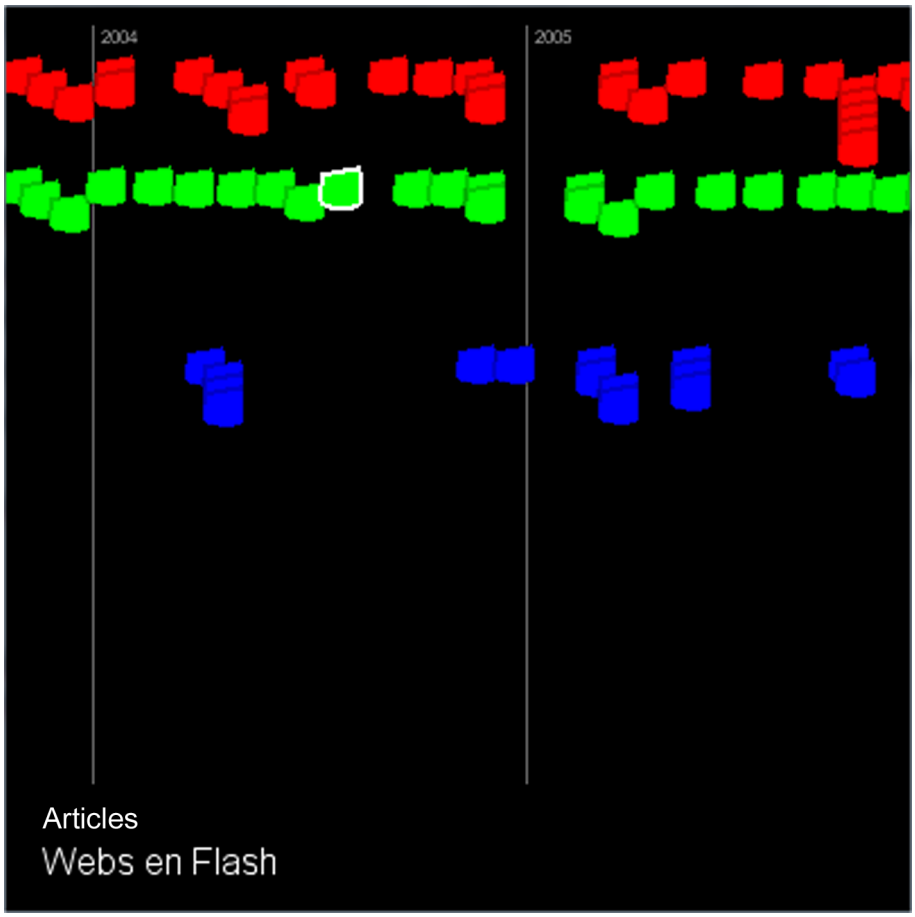
```
medidaFuente/2+medidaFuente+medidaPersona/2+
amplitudOscilacion*cos(radians(angulo+i+int(datos[i][5])*80)) =
3*medidaFuente/2+medidaPersona/2+
amplitudOscilacion*cos(radians(angulo+i+int(datos[i][5])*80))
```

Només hi ha una petita diferència addicional: en aquest cas retornem un enter (1 o 0), no un booleà (`true` o `false`), perquè el farem servir en una altra funció que necessita rebre un enter i no un booleà.

Un cop que tenim la funció per a detectar si el ratolí és a sobre d'una persona, hem d'utilitzar-la. Quan notem que hi som a sobre, hem de dibuixar la persona seleccionada. Atès que la funció `dibujarPoblacion` pinta per pantalla totes les persones, podem utilitzar-la perquè, quan pinti la persona, la pinti seleccionada. Només hem de saber, abans de pintar-la, si està seleccionada, utilitzant el resultat de `detectarPersona` com a quart paràmetre de `dibujarPersona`, per mitjà de la variable local activa:

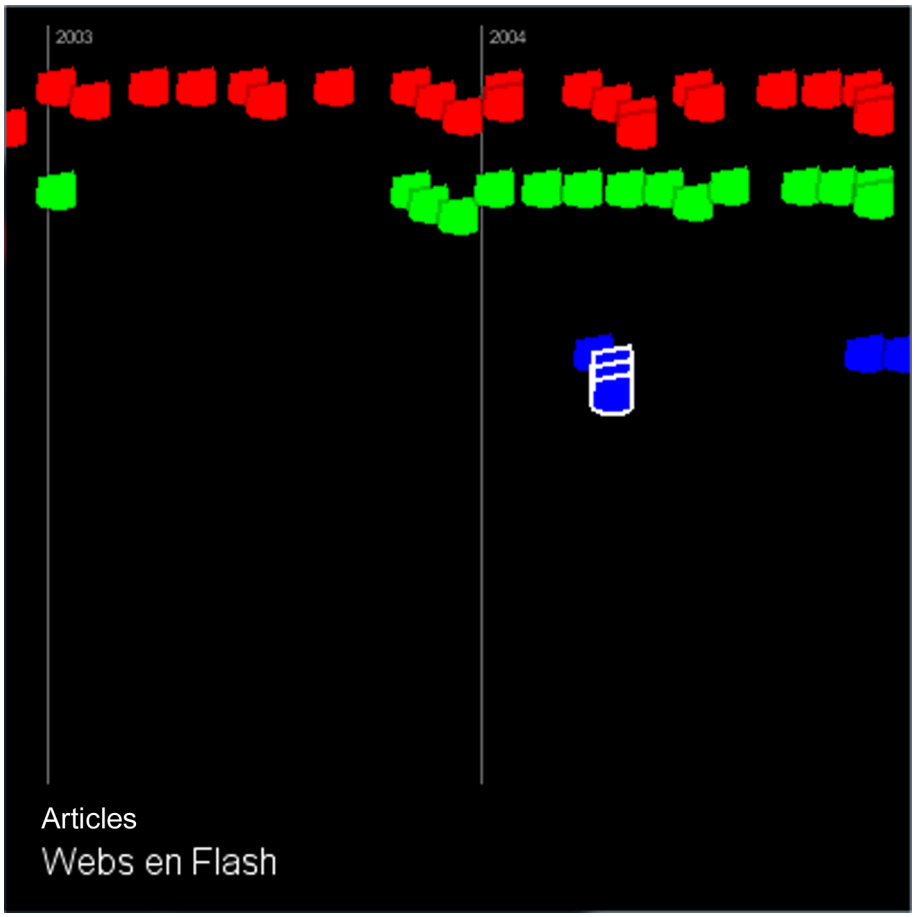
```
void dibujarPoblacion(){
  int i = 0;
  int activa = 0;
...
     activa = personaActiva(i);
     dibujarPersona(posx[i],posy[i],int(datos[i][5]),activa);
...
//(_7_070)
```

Figura 35. Selecció d'una persona en passar el ratolí per sobre



No obstant això, el codi que hem proposat té un problema, i és que, quan les persones se superposen, en pot arribar a seleccionar més d'una, com es pot veure en la figura 36.

Figura 36. Selecció de diverses persones simultàniament



Per evitar aquest efecte podem fer que només se seleccioni la primera persona que es trobi en la posició del ratolí, ignorant la resta. Això es pot aconseguir afegint una variable addicional a `dibujarPoblacion`:

```
...
  int activa = 0;
  int encontrada = 0;
...
    if (encontrada == 0){
      activa = personaActiva(i);
      encontrada = activa;
    } else {
      activa = 0;
    }
    dibujarPersona(posx[i],posy[i],int(datos[i][5]),activa);
...
//(_7_080)
```

Aquest codi té un efecte addicional: deixa de comprovar si la persona està activa un cop en troba una, de manera que estalvia també temps de procés de la màquina.

8.7. Escriptura del text en funció de la persona detectada

Un cop hem detectat la persona, no solament podem mostrar-la amb un contorn diferenciat. Per exemple, podem mostrar el text associat a la zona de text inferior utilitzant la funció d'escriptura de text `escribeTexto` associada a la figura 33. De fet, com que tenim més espai, ara que hem ampliat la zona de dibuix, també hi escriurem la data emmagatzemada a datos. Modificarem la funció convenientment i li tornarem a posar un nom, per seguir el mateix format de verb en infinitiu de la resta de funcions:

```
void escribirTexto(String categoria, String nombre, String fecha){
  textAlign(LEFT);
  fill(#FFFFFF);
  textSize(medidaFuente);
  text(nombre,medidaFuente,height-medidaFuente);
  textSize(medidaFuente*0.8);
  text(categoria + "::" + fecha,medidaFuente,height-2.5*medidaFuente);
}
```

Aquesta funció la situarem dins de la funció `draw`, al final. Però per a cridar-la necessitem saber quina persona és la que actualment està seleccionada. Atès que un cop que trobem la persona seleccionada deixem de buscar-ne més, podem incloure la línia següent dins de `personaActiva`:

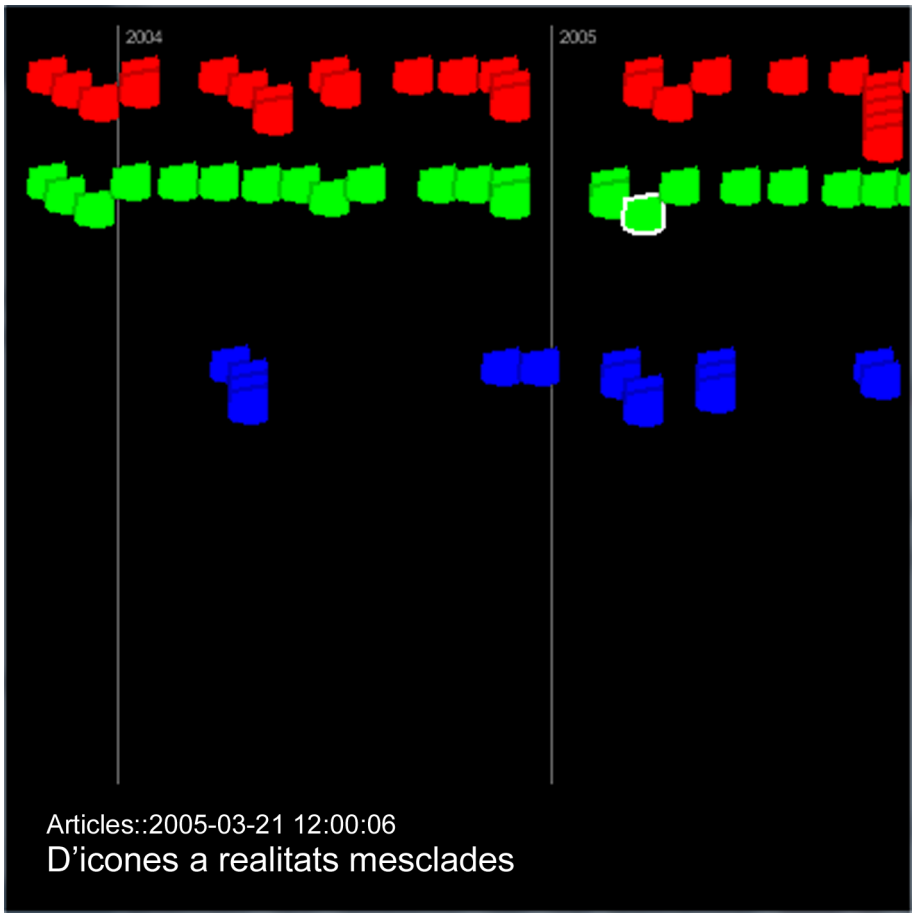
```
...
  if (mouseX>=limiteIzquierda && mouseX<=limiteDerecha && mouseY>=limiteSuperior &&
  mouseY<=limiteInferior) {
    persona = i;
    return 1;
  } else {
...
```

en què `persona` és una variable global que conté l'última persona per sobre de la qual ha passat el ratolí. Atès que a l'inici no hem passat encara per sobre de cap persona, indicarem aquest fet inicialitzant-la amb un valor de –1 dins de la funció `setup`.

Ara ja podem inserir la funció d'`escribirTexto` al final de la funció `draw`, de la manera següent:

```
...
  if (persona>-1){
    escribirTexto(datos[persona][4],datos[persona][0], datos[persona][2]);
  }
}
//(_7_090)
```

Figura 37. Selecció de diverses persones simultàniament



Notem que quan no hem passat encara per cap persona (quan persona té un valor de –1) no escrivim cap text, ja que no entrem dins de l'`if`.

8.8. Enllaç a pàgines web

També podem fer que l'aplicació ens obri la pàgina web que representa la persona si cliquem sobre una persona. Per a fer-ho, utilitzarem la funció de `mouseReleased()`, que s'activa quan deixem anar qualsevol botó del ratolí (per deixar-lo anar, abans l'hem de prémer). Ho fem així perquè si l'activem quan premem el botó, s'activa la crida i obrim la pàgina web amb el botó premut, la qual cosa pot generar esdeveniments no desitjats en la pàgina web en què entrem (no solem navegar amb el botó del ratolí premut).

```
void mouseReleased(){
  if (persona>-1){
    if (personaActiva(persona)==1){
      link(datos[persona][1]);
    }
  }
}
//(_7_100)
```

Quan entrem en la funció `mouseReleased()` (perquè hem aixecat el dit del botó), el primer que fem és comprovar que hem passat per sobre d'alguna persona. Si és així, comprovem que encara hi som a sobre. En cas afirmatiu, obrim l'adreça URL que hi ha emmagatzemada en el segon camp de la persona activa. Fixem-nos que no pot passar que siguem a sobre d'una persona que no sigui la identificada per persona: si cliquem a sobre d'una persona, abans de prémer, hi som a sobre i això fa que es dibuixi seleccionada i quedi com l'última persona activa, com hem vist abans.

9. Utilització del codi amb Processing.js

Aquesta secció vol deixar el codi llest per a poder utilitzar-lo amb el Processing.js i amb el WordPress, segons hem vist anteriorment en aquest mòdul.

En la pestanya datos, deixarem totes les variables que ens vindran imposades des del WordPress, de manera que només hàgim de suprimir-les i posar el codi PHP corresponent. Això implica que la funció `dibujarPersona` canvia les constants de color de categories per les variables `colorCategoria0`, `colorCategoria1`, `colorCategoria2` i `colorCategoria3`; les mides de la zona de dibuix també queden com a variables `ancho` i `alto`, i el color de fons utilitzant a `background` (a `setup` i `draw`) també queda com a variable `colorFondo`. Aquestes noves set variables, juntament amb datos, queden en la pestanya datos. En l'`sketch_8_010` tenim aquestes modificacions.

D'altra banda, el codi que tenim fins ara és un codi organitzat en pestanyes en l'IDE del Processing. De fet, aquesta organització en pestanyes és fictícia: per al Processing tot el codi va seguit i posa el contingut de les pestanyes un a sota de l'altre. Si volem copiar tot el codi a l'editor que hi ha en el WordPress, només hem de fer copiar i enganxar del contingut de cada pestanya i anar posant-lo un a sota de l'anterior. Recordem que la pestanya datos no s'ha de copiar i s'han d'igualar les vuit variables que conté amb les que contenen les dades de la base de dades del WordPress.