Tutoriel - Programmer avec Processing (4)

TOUTES LES FICHES

par Yves Durasnel

PUBLIC	PARTICIPANTS	ANIMATEURS	NIVEAU	PRÉPARATION	ACTIVITÉ

Description

Programmer une jeu (jeu pong) avec Processing

Objectifs

–

Matériel

1 vidéo projecteur et 1 ordinateur maître
Accès internet non indispensable, Processing doit alors être téléchargé et/ou installé au préalable
1 ordinateur par participant ou groupe de 2

Compétences travaillées

Concentration
Précision
Curiosité

Contenus utilisés

–

Pré-requis

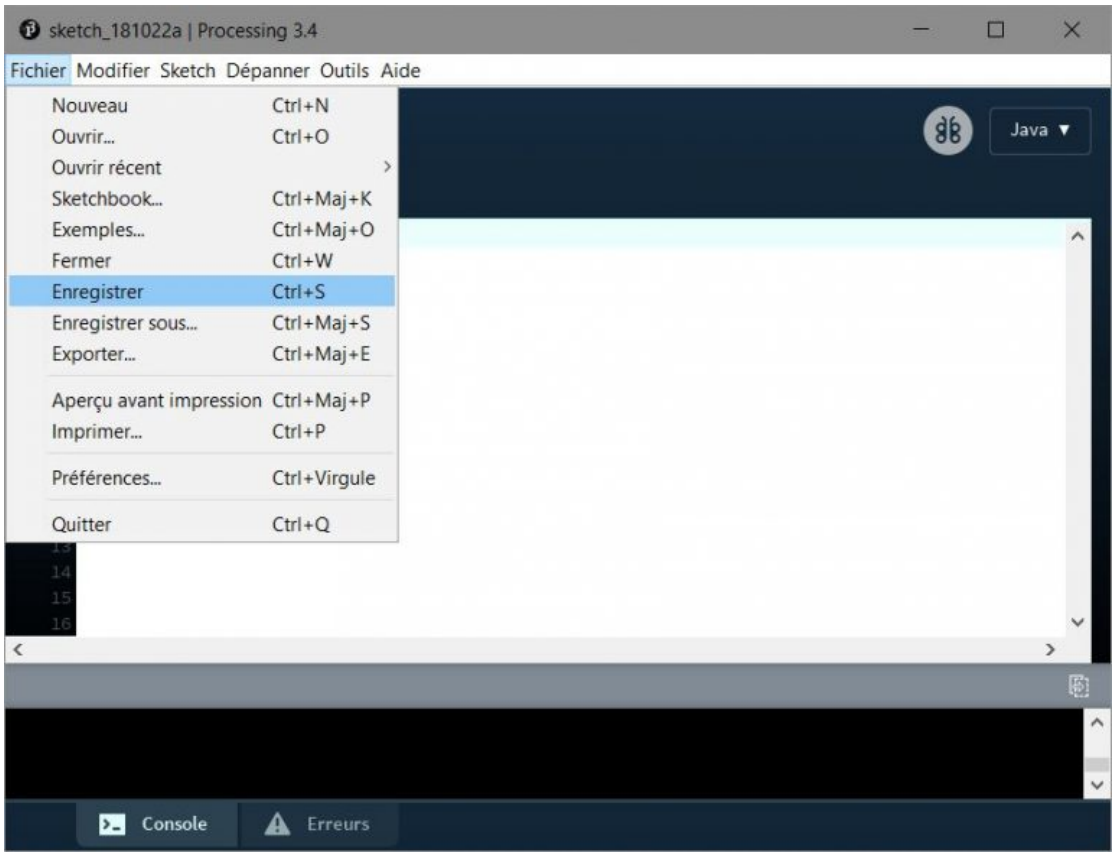
Avoir suivi les séquences précédentes “Programmer avec Processing (1) à (3)”

WORKFLOW

1 Mise en place de l’espace de travail

Pas à pas nous allons programmer un jeu de “pong”

Commençons par ouvrir une nouvelle fenêtre de programmation et à sauvegarder l’environnement de travail sous le nom de “Pong”.



A l’ouverture d’un nouveau programme – ou sketch suivant la dénomination adoptée par Processing – un nom est attribué aléatoirement, ici sketch_181022a.

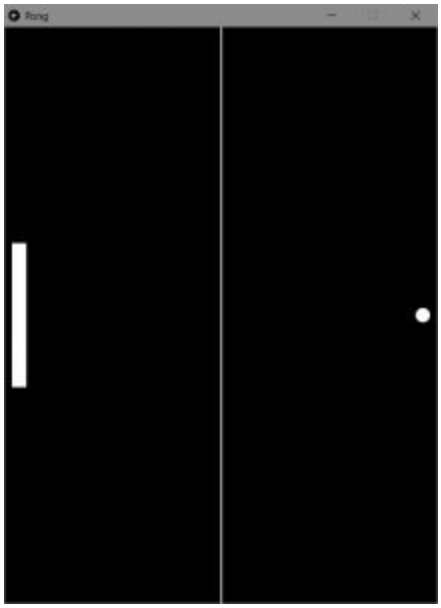
Après sauvegarde la fenêtre indique le nom que nous avons choisi, ici Pong, le fichier créé dans le répertoire de même nom étant Pong.pde.

2 Principe du jeu

Nous allons dans un premier temps définir le principe du jeu :

- sur un terrain rectangulaire on envoie une balle dont la trajectoire est aléatoire au lancé,
- une raquette se déplace le long d'un mur manœuvrée par la joueur avec la souris,
- la balle rebondit sur la raquette et sur les murs clôturant le terrain,
- la partie est perdue quand la balle touche le mur derrière la raquette.

L'espace de jeu



A droite la balle à sa position de départ. Le terrain est rectangulaire et une ligne blanche en détermine le milieu. A gauche la raquette qui est déplacée verticalement en suivant le mouvement de la souris (le curseur de la souris doit rester à l'intérieur du terrain pour permettre le déplacement).

3 Analyse du programme

Les variables globales

Pong

```
1 // Jeu de pong
2
3 float x,y;   // Coordonnées de la balle
4 float dx,dy; // Déplacement de la balle
5 int h,v;     // Coordonnées de la raquette
```

Au début du programme sont déclarées les variables qui contiennent :

- les coordonnées x et y de la balle qui varient pas à pas en fonction de la distance unitaire des déplacements horizontaux dx et verticaux dy,
- les coordonnées h et v de la raquette qui varient en fonction du déplacement de la souris.

Fonction setup()

```
6
7  void setup() { // Initialisation exécutée 1 fois au démarrage du programme
8    size(600, 800);
9    smooth();
10   x = width-20;
11   y = height/2;
12   dx = random(-8, -4);
13   dy = random(4, 8);
14   h = 20;
15   v = height/2;
16 }
```

Cette fonction d'initialisation est exécutée une seule fois au démarrage du programme. On y fixe :

- la taille de l'espace de jeu size() dont les arguments sont la largeur (600 pixels écran) et la hauteur (800 pixels écran) du plateau,
- le rendu à l'écran smooth() (suppression de l'effet d'escalier),
- la valeur des variables au début de l'action.

Les coordonnées x, y et v sont calculées à partir de la taille du plateau dont la largeur (variable système width) et la hauteur (variable système height) sont fixées par la fonction size().

La fonction random() permet de générer des nombres aléatoire entre les bornes données en arguments. Ainsi la direction et la vitesse de déplacement de la balle sont fixées aléatoirement au lancement du programme.

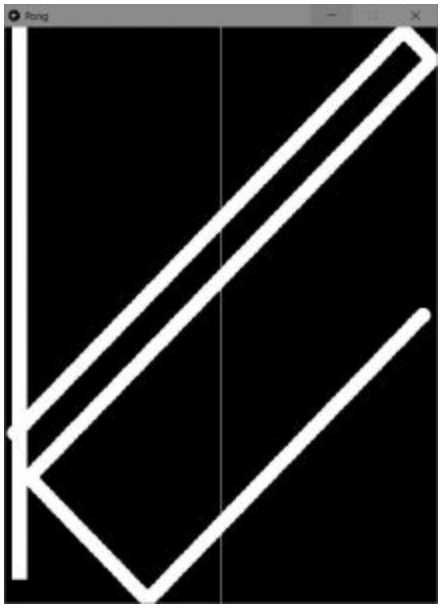
Fonction draw()

```
17
18 void draw () { // Affichage du plateau exécuté à chaque raffraichissement d'écran
19   clean();
20   show();
21   move();
22 }
```

La fonction draw() fait appel à 3 sous-fonctions. A chaque rafraîchissement de l'écran les 3 opérations sont effectuées dans l'ordre :

- clean() qui efface l'écran et ré-affiche les éléments fixes,
- show() qui affiche les éléments mobiles (balle et raquette),
- move() qui calcule la nouvelle position des éléments mobiles entre 2 rafraîchissements.

Notez que si l'écran n'était pas effacé et l'ensemble des éléments ré-affichés à chaque rafraîchissement, la trace de l'ensemble des mouvements de la balle et de la raquette resteraient à l'écran, comme illustré sur la capture d'écran.



Notez aussi que le déplacement des éléments mobiles est saccadé, il est fonction de la fréquence de balayage de votre écran. Heureusement, au delà de 24 images par seconde le phénomène de persistance rétinienne donne l'impression d'une image fluide, ce qui est le cas pour cette animation.

Fonction clean()

```
23
24 void clean() { // Affichage des éléments fixes
25   background(0);
26   stroke(255);
27   strokeWeight(1);
28   line(width/2, 0, width/2, height);
29 }
```

La fonction background() affiche un fond d'écran noir, la couleur étant déterminée par l'argument entre parenthèses. Quand une seule valeur est donnée (ici 0) elle est étendue sous forme de trois valeurs identiques (ici 0,0,0) qui correspondent à la couleur exprimée en codification RVB (voir par exemple le site w3schools pour les explications).

Les fonctions stroke() et strokeWeight() fixent les caractéristiques de la ligne line() qui a donc une couleur blanche et une épaisseur d'un pixel. La position de la ligne est calculée à partir de taille du plateau.

Fonction show()

```
30
31 void show() { // Affichage des éléments mobiles
32   fill(255);
33   rectMode(CENTER);
34   rect(h, v, 20, 200);
35   ellipseMode(CENTER);
36   ellipse(x, y, 20, 20);
37 }
```

La fonction fill() fixe la couleur de la raquette et de la balle, blanches toutes deux.

Les fonctions rectMode() et ellipseMode() règlent le système de coordonnées des figures dont le point d'origine est fixé au centre par l'argument CENTER.

La fonction ellipse() décrit ici un cercle et rect() un rectangle. Leurs dimensions sont fixes (les deux derniers arguments sont numériques) mais leurs positions sont variables, données par les deux premiers arguments qui définissent leurs coordonnées.

Fonction move()

Cette fonction est un peu plus complexe que les précédentes, elle traite le déplacement de la raquette et de la balle ainsi que la réaction de la balle lors des collisions avec les murs ou la raquette.

```
39 void move() { // Déplacement des éléments mobiles
40   x = x + dx;
41   y = y + dy;
42   v = mouseY;
43   if (x > width-10) { // La balle rebondit sur le mur à droite
44     dx = -dx;
45   }
46   if ((y > height-10) || (y < 10)) { // Rebondit sur les murs en haut et en bas
47     dy = -dy;
48   }
49   if ((x < 40) && (y < v + 100) && (y > v - 100)) { // Rebondit sur la raquette
50     dx = -dx;
51   }
52   if (x < 10) { // La balle touche le mur à gauche - Fin de partie
53     noLoop();
54     println("GAME OVER");
55   }
56 }
```

Déplacement de la raquette

Les coordonnées de la raquette sont définies dans les variables v et h.

La coordonnées sur l'axe horizontal est constante, le déplacement de la raquette est contraint sur une ligne verticale, elle est fixée à la valeur h = 20 au début du programme et ne change pas.

La coordonnée sur l'axe vertical est variable, la position verticale du curseur de la

souris est connue par la variable système mouseY, elle est attribuée à la raquette v = mouseY.

Déplacement de la balle

La vitesse de la balle est constante, par contre sa direction change à chaque collision avec un obstacle. x et y sont les coordonnées de la balle, dx et dy donnent la direction (la valeur de chaque déplacement suivant les deux axes) et la vitesse (plus leur valeur est importante plus la vitesse est rapide). Les valeurs dx et dy sont fixées aléatoirement au début du programme.

Lors d'une collision la balle rebondit, ce qui correspond à l'inversion de sa direction. C'est ainsi que par exemple si la balle heurte le mur à droite sa direction de gauche à droite est inversée de droite à gauche. Cela se traduit par l'inversion du signe de la variable dx qui se note dx = -dx.
Le phénomène se produit sur la mur à droite, le test sur la coordonnée x de la balle donne une distance au mur inférieure à 10 pixels, le test est noté if (x > width-10).

Sur les mur en haut et en bas l'inversion porte sur la variable dy (inversion dans le sens vertical). La distance minimum aux murs est toujours de 10 pixels, un seul test est nécessaire pour les deux situations grâce à l'utilisation de l'opérateur || dit OU logique.
Le test if ((y > height-10) || (y < 10)) détermine si la coordonnée y de la balle est bien dans l'intervalle entre les deux murs.

Pour le test sur la raquette nous utilisons l'opérateur && dit ET logique car plusieurs conditions doivent être remplies pour déterminer que la collision se produit.
La coordonnée horizontale doit déterminer que la balle est à 40 pixels du mur gauche (espace de 10 + épaisseur de la raquette 20 + rayon de la balle qui a son origine au centre 10).
La coordonnées verticale doit être comprise sur la longueur de la raquette soit 100 pixels de part et d'autre de son centre.
Le test if ((x < 40) && (y < v + 100) && (y > v – 100)) s'assure que les trois conditions sont réunies simultanément.

Enfin dans le cas où la balle touche le mur gauche la partie est terminée.
La dernière condition if (x < 10) teste ce cas et provoque l'arrêt du jeu, la fonction noLoop() interrompant l'activité de la fonction draw(). Enfin le message "GAME OVER" est affiché sur la console grâce à la fonction println().

Vue de la console lorsque la partie est perdue.

4 Le programme complet

```
// Jeu de pong
float x,y;   // Coordonnées de la balle
float dx,dy; // Déplacement de la balle
int h,v;     // Coordonnées de la raquette
void setup() { // Initialisation exécutée 1 fois au démarrage du programme
size(600, 800);
smooth();
x = width-20;
y = height/2;
dx = random(-8, -4);
dy = random(4, 8);
h = 20;
v = height/2;
}
void draw () { // Affichage du plateau exécuté à chaque raffraîchissement d'écran
clean();
show();
move();
}
void clean() { // Affichage des éléments fixes
background(0);
stroke(255);
strokeWeight(1);
line(width/2, 0, width/2, height);
}
void show() { // Affichage des éléments mobiles
fill(255);
rectMode(CENTER);
rect(h, v, 20, 200);
ellipseMode(CENTER);
ellipse(x, y, 20, 20);
}
void move() { // Déplacement des éléments mobiles
x = x + dx;
y = y + dy;
v = mouseY;
if (x > width-10) { // La balle rebondit sur le mur à droite
dx = -dx;
}
if ((y > height-10) || (y < 10)) { // Rebondit sur les murs en haut et en bas
dy = -dy;
}
if ((x < 40) && (y < v + 100) && (y > v – 100)) { // Rebondit sur la raquette
dx = -dx;
}
```

```
if (x < 10) { // La balle touche le mur à gauche – Fin de partie
noLoop();
println(“GAME OVER”);
}
}
```

5 En suite

Ce programme est volontairement simple, il peut servir de base pour aller plus loin en ajoutant de nouvelles fonctionnalités, par exemple :

- afficher au démarrage un écran d’accueil avec la règle du jeu,
- déterminer le score de l’utilisateur et ajouter un écran de fin avec affichage du score,
- varier l’angle de rebond et la vitesse de la balle lors du renvoi par la raquette,
- en fonction de la durée rendre la difficulté croissante (vitesse, taille de la raquette…),
- créer un mode deux joueurs…

Bon courage pour la suite, et à vous de progresser en vous amusant !

Yves Durasnel
Club sympaTIC
Octobre 2018