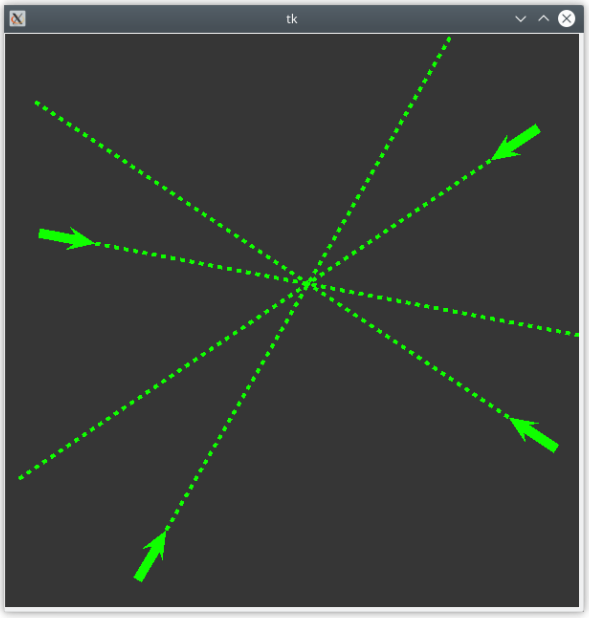
Pascal ORTIZ



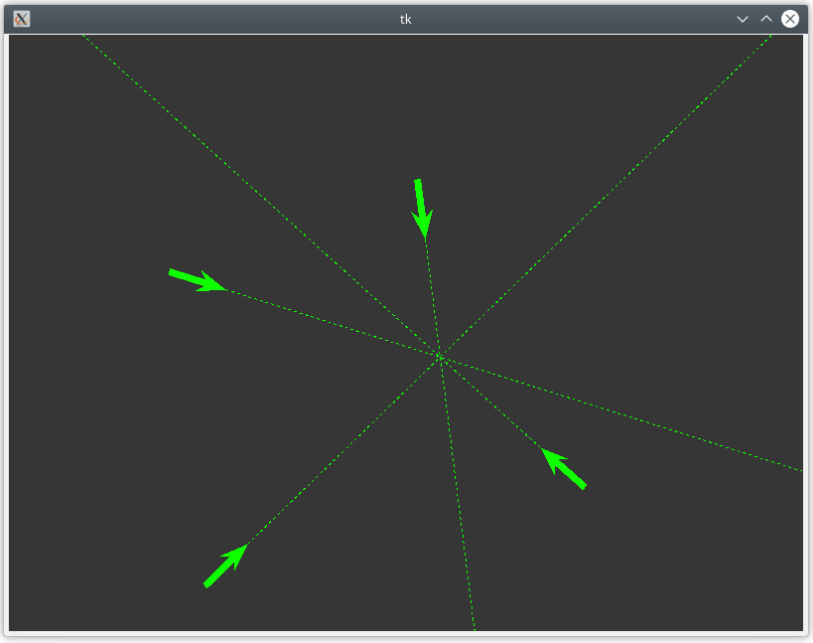
Convergence (démo Phaser)

Version du 13 mai 2021

Table des matières

Présentation de l'activité

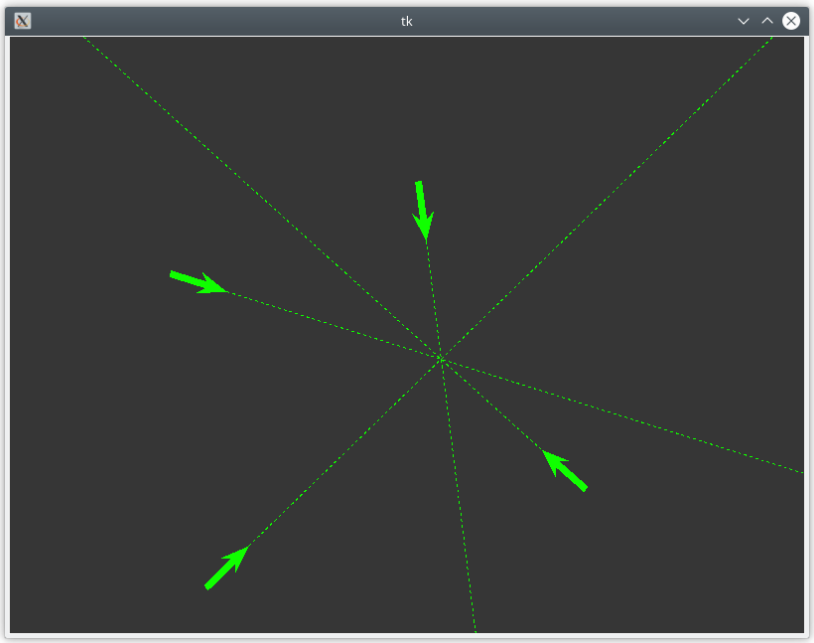
Cette activité propose de programmer en Tkinter un clone de la démo Multi Angle To Pointer de la bibliothèque de jeux Javascript Phaser. Sous Tkinter, l'animation aura la forme suivante :



Lorsque le curseur de la souris se déplace, les quatres flèches présentes sur le plateau s'orientent vers le curseur ce qui dessine 4 lignes pointillées mobiles et convergentes.

Le calcul des coordonnées des points inconnus

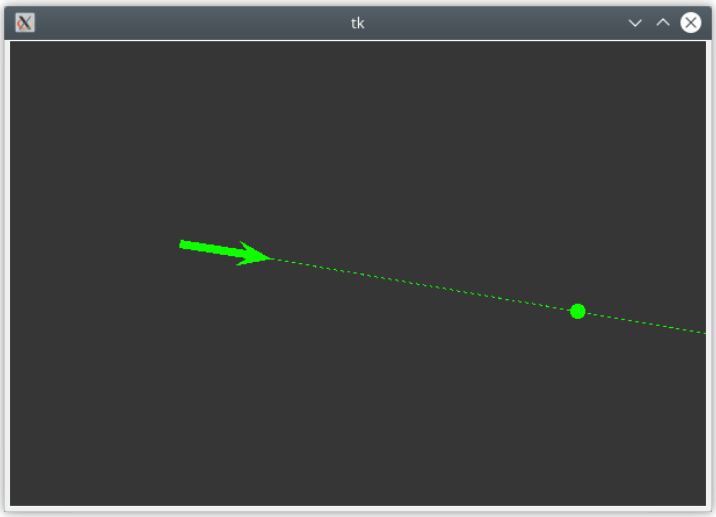
L'essentiel de l'activité consistera à réaliser une représentation **statique** de l'animation :



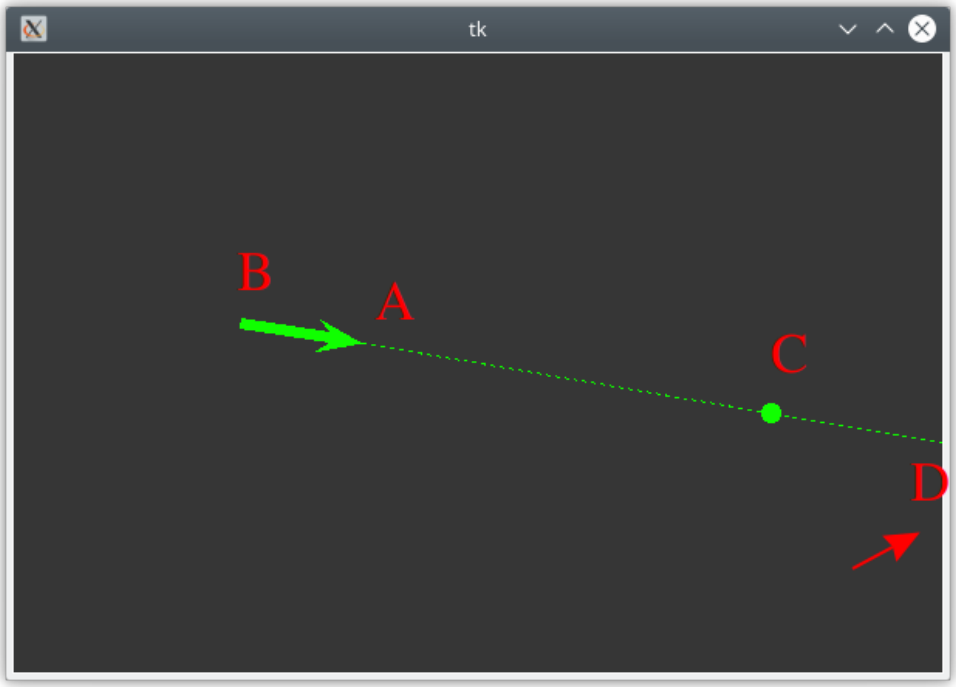
Si cette partie est convenablement codée, il n'y aura que très peu de modifications à apporter pour obtenir l'animation.

Il est **essentiel** de bien **observer** l'animation (si nécessaire, se rendre sur la page de démo de Phaser. On remarque les points suivants :

— les extrémités des flèches sont immobiles pendant toute l'animation ;
— les flèches ont même longueur, disons `L` ;
— il suffit de savoir réaliser le dessin avec **une seule** flèche pour pouvoir produire un dessin complet : il suffira d'écrire une boucle `for` pour dessiner autant de flèches que souhaité :



— le mouvement de chaque flèche est circulaire, le centre étant la pointe de la flèche, disons `A` ; l'origine de la flèche sera notée `B` :

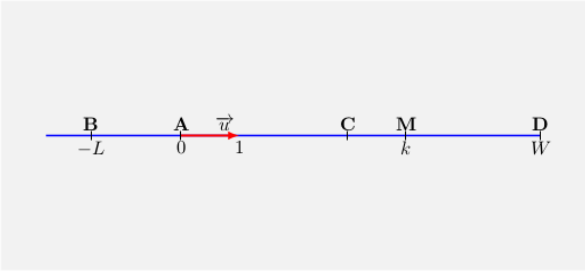


— la ligne pointillée passe par la pointe `A` de la flèche et un autre point dont les coordonnées sont connues (à terme, c'est le curseur de la souris mais cette situation ne sera prise en compte que tout à la fin de l'activité). Le point sera appelé `C`.

— la ligne pointillée se prolonge depuis `A` et au-delà de `C`, jusqu'en un point, disons `D`. Pour fixer les idées, on supposera que la distance `AD` vaudra la largeur `W` du plateau ;

— les points `B` et `D` se déduisent de la connaissance de `A`, `C` et des constantes `L` et `W`. Cet aspect est essentiel car il fait comprendre que la réalisation du programme dépend essentiellement de l'écriture d'une **seule** fonction (appelée ci-dessous `line`).

Soit $\overrightarrow{u}$ un vecteur de longueur 1 et de même direction et même sens que le vecteur $\overrightarrow{AC}$:



Il faut imaginer que $(\mathtt{A}, \overrightarrow{u})$ est un repère d'origine `A` de la droite `AC`. Alors

$$\overrightarrow{AB} = -\mathtt{L}\overrightarrow{u}, \overrightarrow{AD} = \mathtt{W}\overrightarrow{u}.$$

On voit donc que pour déterminer B et D, il suffit de connaître $\overrightarrow{u}$. Or, $\overrightarrow{u} = \dfrac{\overrightarrow{AC}}{AC}$.
Plus généralement, pour déterminer les coordonnées d'un point M tel que

$$\overrightarrow{AM} = \mathtt{k}\,\overrightarrow{u}$$

il suffit de passer aux coordonnées dans l'égalité ci-dessus, ce qui donne, en posant $\mathtt{A} = (\mathtt{x_A}, \mathtt{y_A})$ et $\overrightarrow{u} = (\mathtt{x_u}, \mathtt{y_u})$:

$$\begin{cases} x_M &= k \times x_u + x_A \\ y_M &= k \times y_u + y_A \end{cases}$$

On en déduit le code d'une fonction `point(A, C, k)` qui calcule les coordonnées de M :

```
1  def point(A, C, k):
2      xA, yA=A
3      xC, yC=C
4      xAC, yAC=(xC-xA, yC-yA)
5      AC=(xAC**2+yAC**2)**0.5
6      xu, yu=(xAC/AC, yAC/AC)
7      xM=xA+k*xu
8      yM=yA+k*yu
9      return (xM, yM)
10
11 A=(150, 100)
12 C=(450, 330)
13 k=400
14
15 print(point(A, C, k))
```

```
16 (467.4425445167664, 343.37261746285424)
```

— Lignes 2-3 : on extrait les coordonnées
— Ligne 4 : on calcule les coordonnées $(\mathtt{x}, \mathtt{y})$ du vecteur $\overrightarrow{AC}$
— Ligne 5 : on applique la formule $\mathtt{AC} = \sqrt{\mathtt{x}^2 + \mathtt{y}^2}$
— Ligne 6 : on applique la formule $\overrightarrow{u} = \dfrac{\overrightarrow{AC}}{\mathtt{AC}}$
— Lignes 7-8 : on applique la formule de calcul des coordonnées de M vue plus haut.

Alternative

On peut légèrement simplifier le code en utilisant la classe `Vec2D` du module `turtle` qui implémente des vecteurs du plan. Voici le code :

```
1  from turtle import Vec2D as vect
2
3  def point_alt(A, C, k):
4      v=vect(*C)-vect(*A)
5      u=1/abs(v)*v
6      M=vect(*A)+k*u
```

```
    return M

A=(150, 100)
C=(450, 330)
k=400

print(point_alt(A, C, k))
```

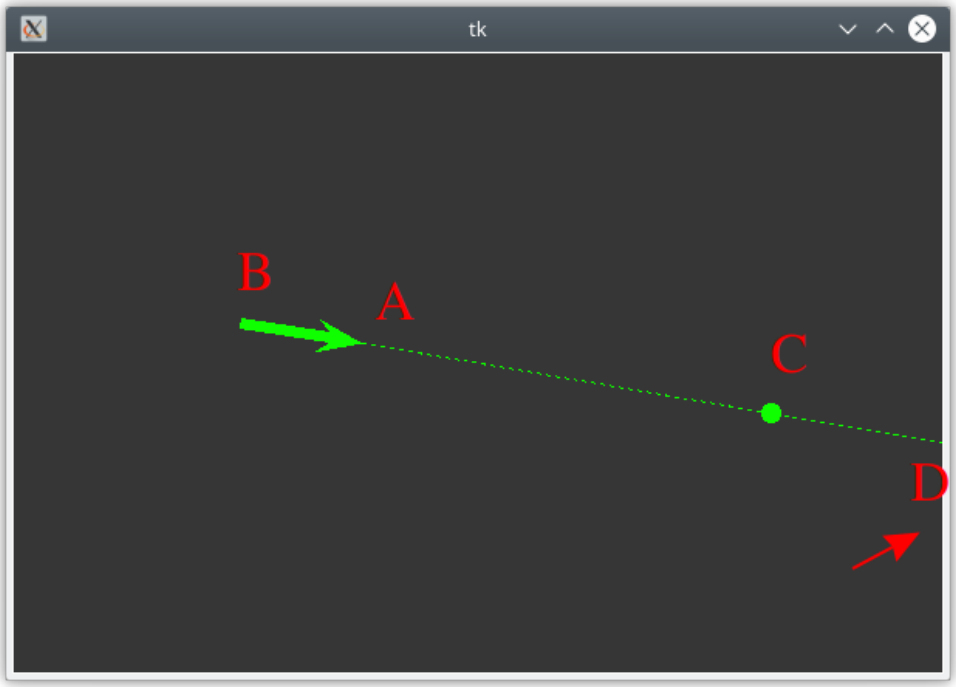
```
(467.44,343.37)
```

Tracé d'une seule flèche et des pointillés

Avec la fonction précédente, et partant

- de la longueur `L` de la flèche
- de son extrémité `A`
- d'un point `C`

on va pouvoir tracer la flèche $\overrightarrow{BA}$ et la ligne pointillée `AD` :



Pour faire le tracé avec Tkinter, on a besoin de

- savoir représenter des couleurs avec leur code RGB hexadécimal,
- tracer une droite avec la méthode `Canvas.create_line`
- tracer des pointillés avec l'option `dash`
- tracer une flèche et la customiser

— savoir représenter un point avec un disque.

La couleur verte des flèches dans l'animation Phaser est de code `10ff00`, la couleur grise du fond est de code `363636` (valeurs mesurées avec Gimp). Pour marquer le point `C`, on utilisera une fonction `dot` qui dessine un petit disque.

Voici le code complet :

visee.py

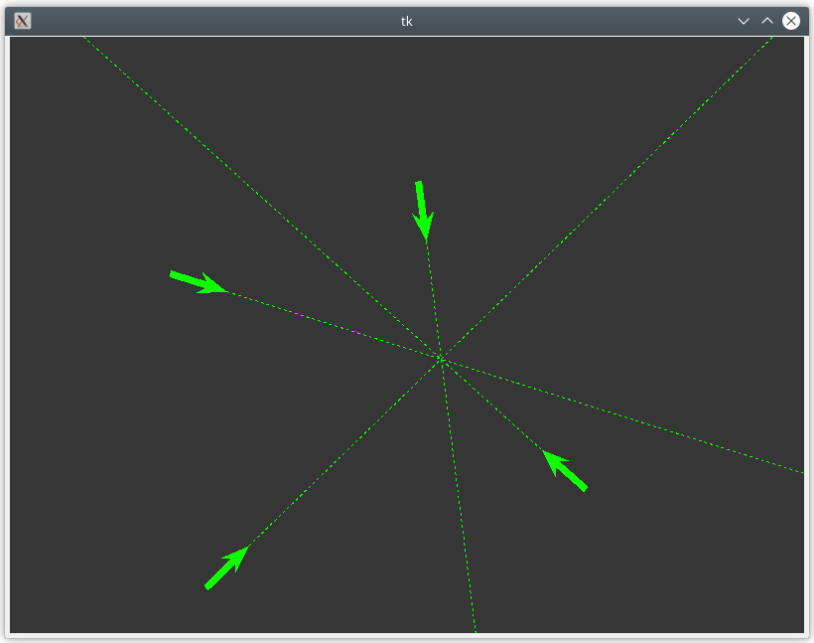
```
from tkinter import Tk, Canvas

WIDTH=600
HEIGHT=400

GREEN="#10ff00"
GRAY="#363636"


def point(A, C, k):
    xA, yA=A
    xC, yC=C
    xAC, yAC=(xC-xA, yC-yA)
    AC=(xAC**2+yAC**2)**0.5
    xu, yu=(xAC/AC, yAC/AC)
    xM=xA+k*xu
    yM=yA+k*yu
    return (xM, yM)

root=Tk()
cnv=Canvas(root, width=WIDTH, height=HEIGHT, bg=GRAY)
cnv.pack()


def dot(cnv, C, R=6, color='red'):
    xC, yC=C
    A=(xC-R, yC-R)
    B=(xC+R, yC+R)
    return cnv.create_oval(A,B, fill=color, outline=color)

def line(A, C):
    B=point(A, C, -L)
    D=point(A, C, WIDTH)

    cnv.create_line(B,A, width=7, fill=GREEN, arrow='last',
                    arrowshape=(18,30, 8))
    cnv.create_line(A,D, fill=GREEN, dash=3)


A=(150, 100)
C=(450, 330)
```

```
42 L=80
43
44 line(A,C)
45 dot(cnv, C, color="#10ff00")
46
47 root.mainloop()
```

— Bien sûr pour connaître l'origine de la flèche et l'extrémité des pointillés, on a besoin de la fonction `point` (ligne 10).
— La longueur de la flèche est fixée arbitrairement à 80.
— La fonction qui réalise le tracé est la fonction `line` (ligne 31).
— Le premier appel (ligne 35) dessine la flèche avec l'option `arrow` en lui donnant une forme particulière grâce à l'option `arrowshape`.
— L'option `arrow='last'` indique que l'extrémité de la flèche porte sur le 2e point.
— Pour la droite (ligne 36), les pointillés sont obtenus avec l'option `dash`.

Plusieurs flèches

On va maintenant placer 4 lignes convergentes :



Pour ne pas avoir à décider des points extrémités des flèches, on les choisit au hasard. Pour cela, il suffit de choisir une abscisse et une ordonnée avec la fonction `randrange` :

```
1 from random import randrange
2
3 WIDTH=600
4 HEIGHT=400
5
```

```
A=(randrange(WIDTH), randrange(HEIGHT))
```

Ensuite, il faut choisir 4 points. On utilise donc une boucle `for` et on choisit à chaque fois au hasard le point `A`. Il faudra aussi préalablement choisir le point `C` vers lequel vont converger les lignes. D'où le code partiel suivant :

```
C=(randrange(WIDTH), randrange(HEIGHT))

for _ in range(4):
    A=(randrange(WIDTH), randrange(HEIGHT))
    line(A,C)
```

et le code complet suivant :

```
from tkinter import *
from random import randrange

WIDTH=800
HEIGHT=600

GREEN="#10ff00"
GRAY="#363636"


def point(A, C, k):
    xA, yA=A
    xC, yC=C
    xAC, yAC=(xC-xA, yC-yA)
    AC=(xAC**2+yAC**2)**0.5
    xu, yu=(xAC/AC, yAC/AC)
    xM=xA+k*xu
    yM=yA+k*yu
    return (xM, yM)

root=Tk()
cnv=Canvas(root, width=WIDTH, height=HEIGHT, bg=GRAY)
cnv.pack()


def dot(cnv, C, R=6, color='red'):
    xC, yC=C
    A=(xC-R, yC-R)
    B=(xC+R, yC+R)
    return cnv.create_oval(A,B, fill=color, outline=color)

def line(A, C):
    B=point(A, C, -L)
    D=point(A, C, WIDTH)

    cnv.create_line(B,A, width=7, fill=GREEN, arrow='last',
```

```
                    arrowshape=(18,30, 8))
    cnv.create_line(A,D, fill=GREEN, dash=3)


L=60

C=(randrange(WIDTH), randrange(HEIGHT))

for _ in range(4):
    A=(randrange(WIDTH), randrange(HEIGHT))
    line(A,C)

root.mainloop()
```

Comme on peut le constater, ce code a juste nécessité de modifier très légèrement notre tout premier code de dessin.

L'animation

En prérequis pour utiliser Tkinter, il faut connaître les événements de la souris et l'effacement d'items sur le canevas.

Pour coder l'animation, il suffit de remplacer le point `C` ci-dessus par le curseur mobile de la souris. Pour cela, d'abord, on définit aléatoirement 4 extrémités de flèches (les points `A`). Puis, on lie avec la méthode `bind` le déplacement de la souris (événement `"<Motion>"`) sur le canevas à une fonction, disons `anim` qui appelera la fonction `line` de tracé de lignes. Avant de dessiner l'animation, la fonction `anim` devra effacer tous les items présents sur le canevas. D'où le code partiel suivant :

```
1 def anim(event):
2     cnv.delete('all')
3     C=(event.x, event.y)
4     for A in centres:
5         line(A, C)
6
7 centres=[(randrange(L, WIDTH-L), randrange(L, HEIGHT-L))
8         for _ in range(4)]
9 cnv.bind("<Motion>",anim)
```

- Ligne 7 : on prédéfinit les extrémités des flèches dans la liste `centres`. Pour que ces points ne soint pas trop près du bord, on restreint la plage de choix aléatoire (`L` est la longueur de la flèche).
- Lignes 1-5 : La fonction `anim` efface **tout** le canvas (ligne 2) ; noter l'argument `all`.
- Ligne 9 : tout déplacement de la souris va appeler la fonction `anim`.
- Ligne 3 : la position du curseur est donnée par les attributs `event.x` et `event.y` où `event` représente l'évènement `Motion` de la souris (ligne 9).
- Lignes 4-5 : à chaque mouvement de la souris, et pour chaque centre (ligne 4), la fonction `line` (ligne 5) de tracé de la flèche et de la ligne pointillée est appelée.

D'où le code complet :

```
from tkinter import *
from random import randrange

WIDTH=800
HEIGHT=600

GREEN="#10ff00"
GRAY="#363636"


def point(A, C, k):
    xA, yA=A
    xC, yC=C
    xAC, yAC=(xC-xA, yC-yA)
    AC=(xAC**2+yAC**2)**0.5
    xu, yu=(xAC/AC, yAC/AC)
    xM=xA+k*xu
    yM=yA+k*yu
    return (xM, yM)

root=Tk()
cnv=Canvas(root, width=WIDTH, height=HEIGHT, bg=GRAY)
cnv.pack()


def dot(cnv, C, R=6, color='red'):
    xC, yC=C
    A=(xC-R, yC-R)
    B=(xC+R, yC+R)
    return cnv.create_oval(A,B, fill=color, outline=color)

def line(A, C):
    B=point(A, C, -L)
    D=point(A, C, WIDTH)

    cnv.create_line(B,A, width=7, fill=GREEN, arrow='last',
                    arrowshape=(18,30, 8))
    cnv.create_line(A,D, fill=GREEN, dash=3)


def anim(event):
    cnv.delete('all')
    C=(event.x, event.y)
    for A in centres:
        line(A, C)

L=60

```

```
centres=[(randrange(L, WIDTH-L), randrange(L, HEIGHT-L))
         for _ in range(4)]
cnv.bind("<Motion>",anim)

root.mainloop()
```