
TIPE:

ÉTUDE DES MOUVEMENTS DE FOULE PAR MODÉLISATIONS PHYSIQUE ET INFORMATIQUE

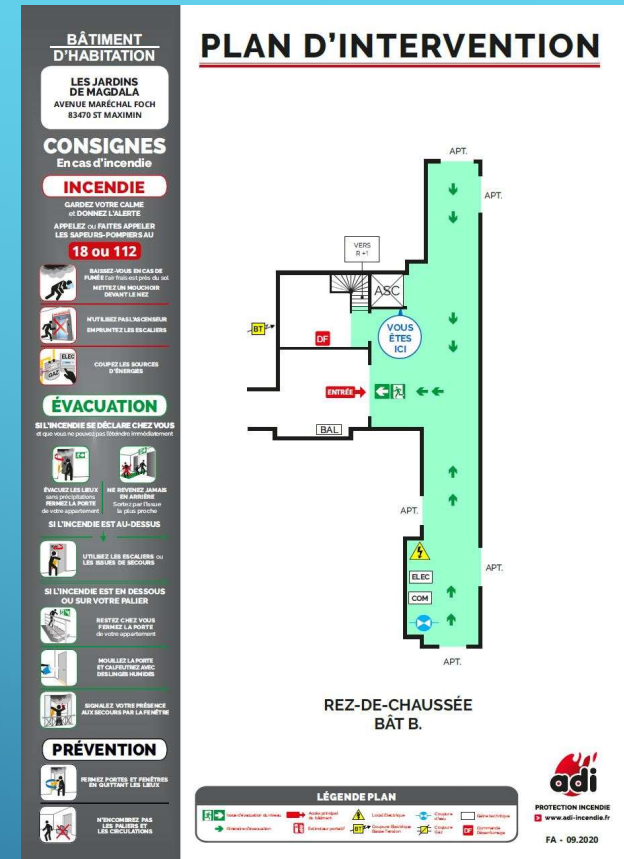
LAPORTE Paul

Numéro d'inscription : 52520



A wide-angle, high-angle photograph of the Kaaba in Mecca at night. The Kaaba, a large black cube with gold bands, is the central focus, surrounded by a massive, dense crowd of pilgrims. The pilgrims are mostly wearing white ihram clothing. The courtyard is illuminated by bright, circular lights, and the surrounding architecture features multiple levels of arches and balconies, also lit up. The scene captures the scale and atmosphere of the Hajj pilgrimage.

2/32



PLAN

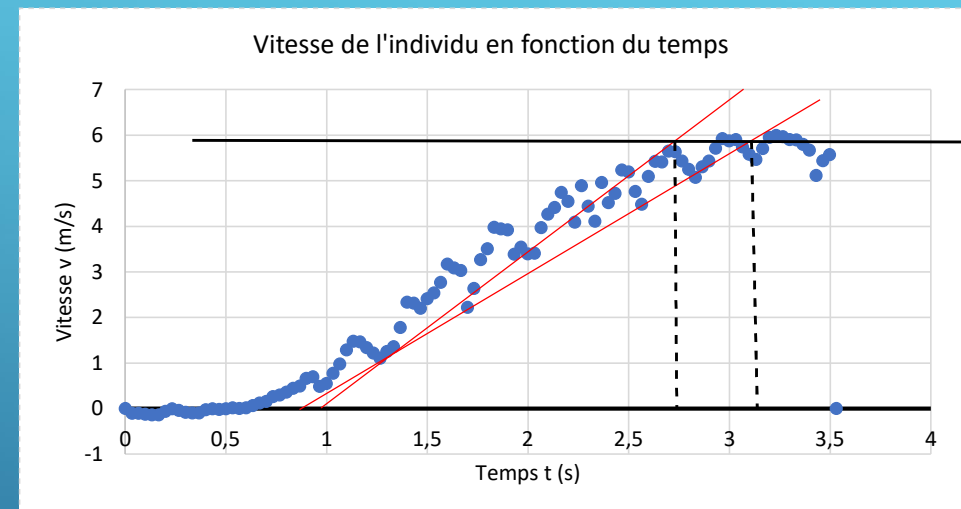
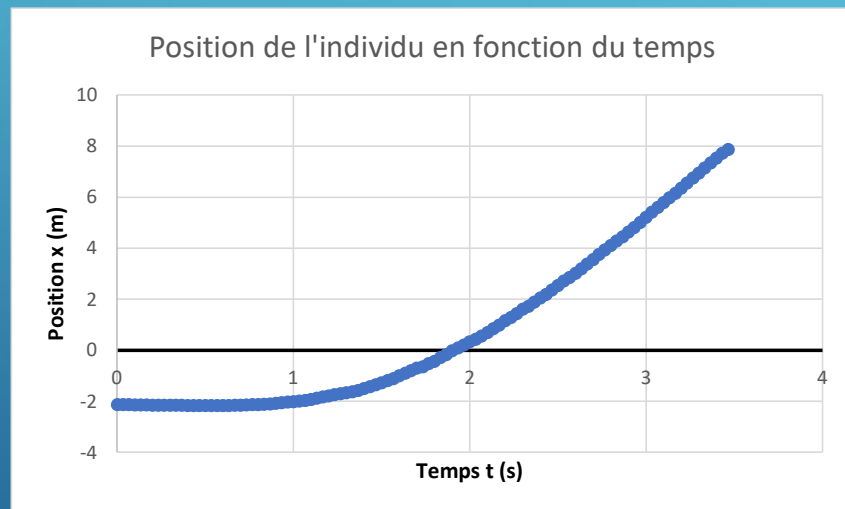
- ▶ I – Recherche d'un modèle adapté
- ▶ II – Application du modèle à un exemple
- ▶ III – Modélisation informatique
- ▶ Annexes

Recherche d'un
modèle adapté

Application du modèle
à un exemple

Modélisation
informatique

- Position et vitesse d'une personne qui commence à courir :



$$\tau_{max} = 2,0s$$

$$\tau_{min} = 1,8s$$

$$\tau = \frac{\tau_{max} + \tau_{min}}{2} = 1,9s$$

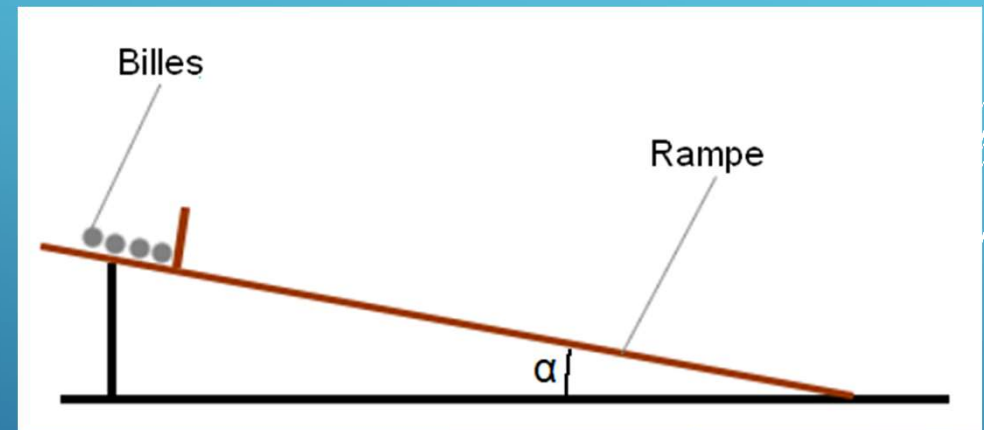
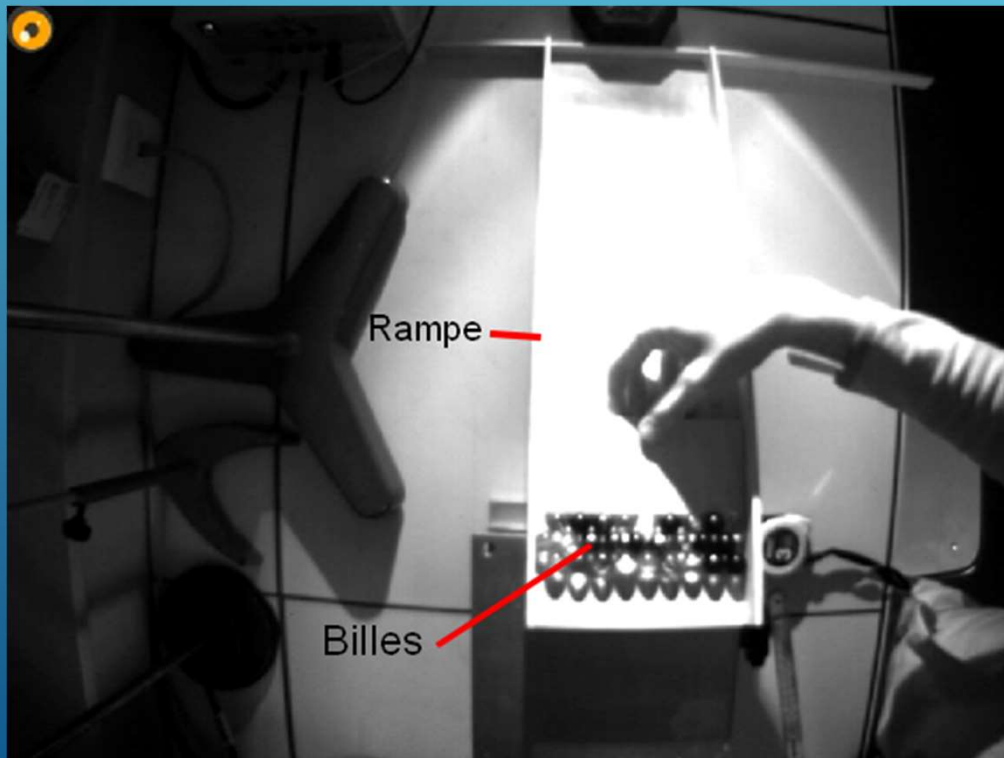
$$u_{\tau} = \frac{1}{\sqrt{2}} \sqrt{(\tau_{max} - \tau)^2 + (\tau_{min} - \tau)^2} = 0,1s \quad 4/32$$

Recherche d'un
modèle adapté

Application du modèle
à un exemple

Modélisation
informatique

► Expérience proposée :



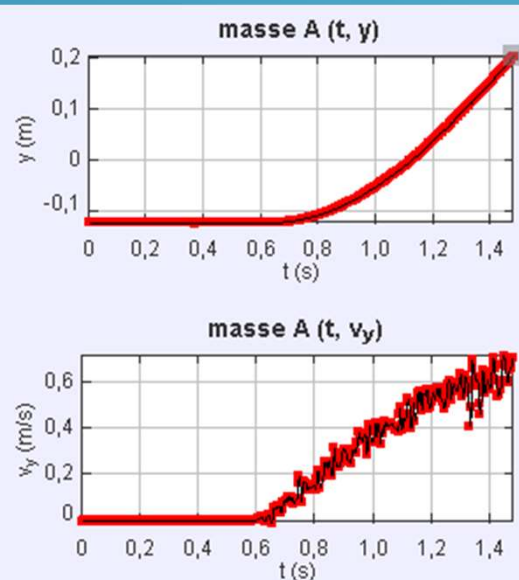
Recherche d'un
modèle adapté

Application du modèle
à un exemple

Modélisation
informatique

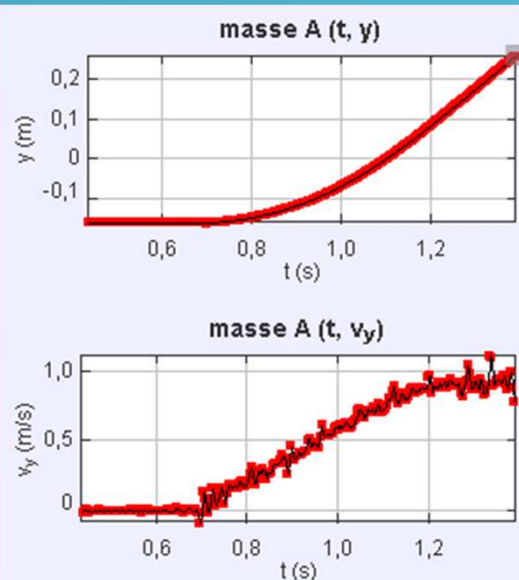
- Position et vitesse des billes en fonction du temps, pour différentes inclinaisons de la rampe :

0,14rad / 8°



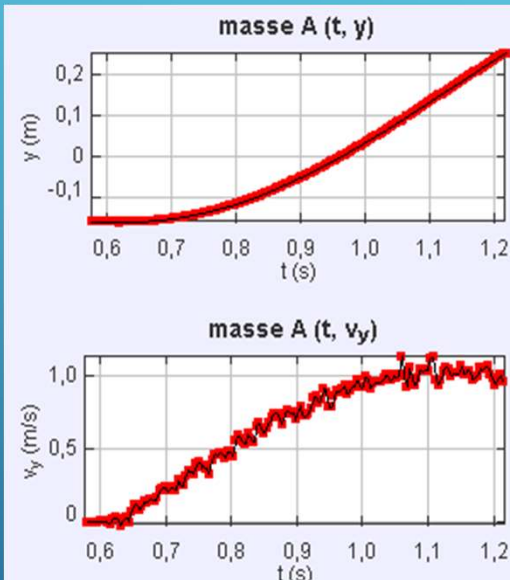
$\tau = 0,8s$

0,24rad / 14°



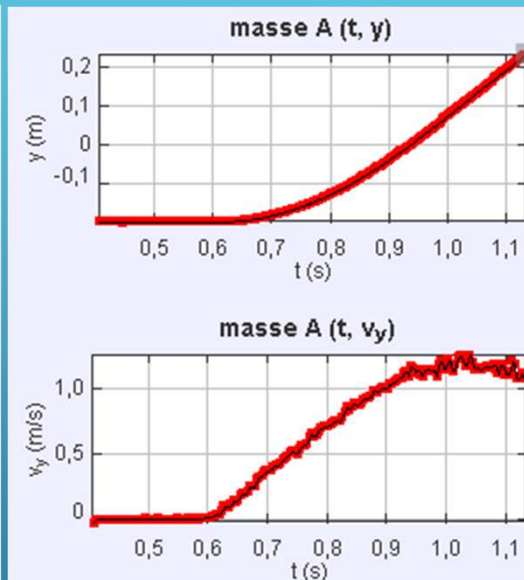
$\tau = 0,6s$

0,33rad / 19°



$\tau = 0,4s$

0,43rad / 25°



$\tau = 0,3s$

Recherche d'un
modèle adapté

Application du modèle
à un exemple

Modélisation
informatique

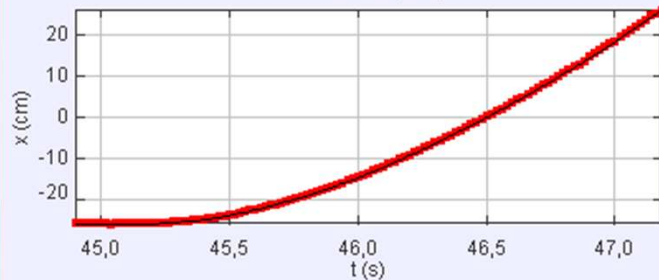
Angle	Réf.	1°	2°	4°	5°	6°	7°	8°	14°	19°	25°
Constante de temps	1,9s	1,8s	Non Valide	1,2s	1s	0,8s	0,75s	0,8s	0,6s	0,4s	0,3s

Mesures de constantes de temps en fonction de l'angle α :

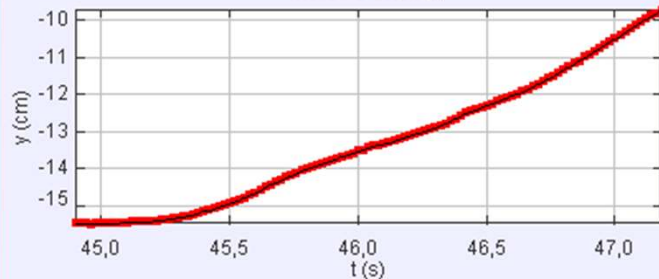
0,04rad / 2°

0,02rad / 1°

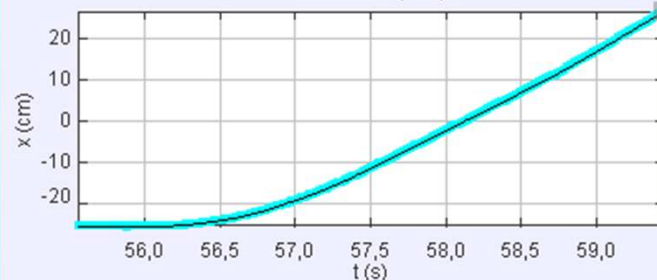
masse E (t, x)



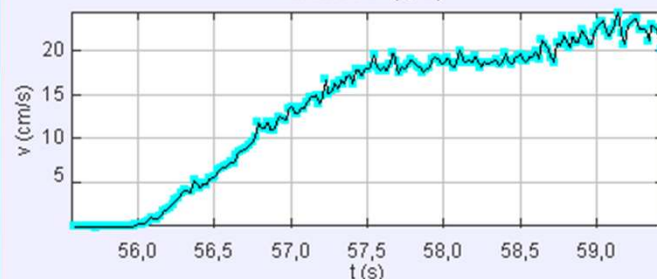
masse E (t, y)



masse F (t, x)



masse F (t, v)

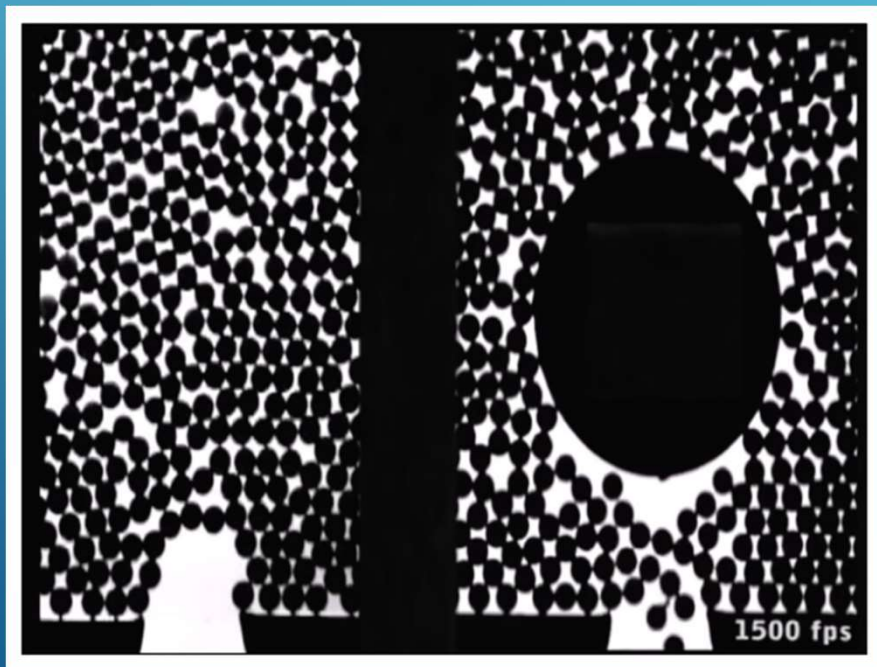


Recherche d'un
modèle adapté

Application du modèle
à un exemple

Modélisation
informatique

► Division du flux :



Recherche d'un
modèle adapté

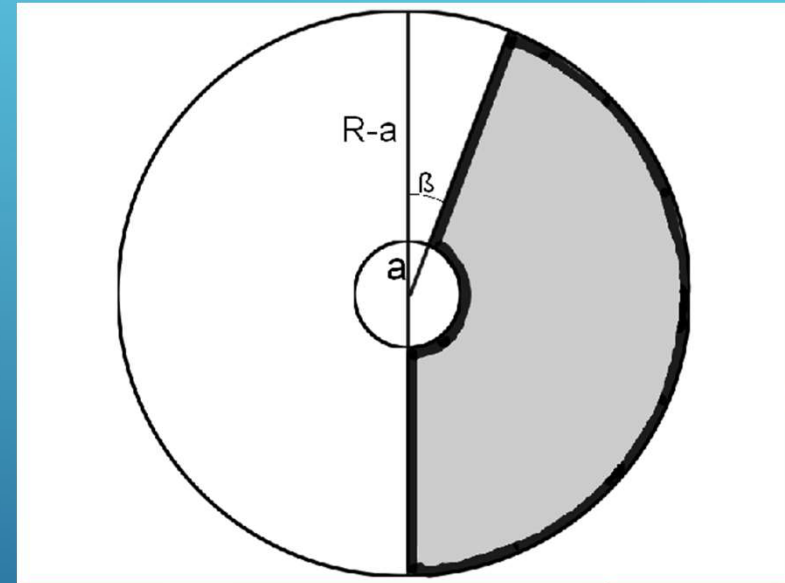
Application du modèle
à un exemple

Modélisation
informatique

► Expérience proposée :



- Rampe en polystyrène extrudé
- Arcs et support en bois



Recherche d'un
modèle adapté

Application du modèle
à un exemple

Modélisation
informatique

► Résultats :

Tests :	Résultats :																				Pourcentage d'arches :
Sans obstacle	A	A	A	X	X	X	A	X	X	A	A	X	A	X	A	A	X	X	X	A	50 %
Avec obstacle caché	A	X	A	X	X	A	X	X	A	X	A	X	X	A	A	X	X	X	A	X	40 %
Avec obstacle apparent	A	X	X	A	X	X	A	X	X	X	X	X	X	A	A	A	A	A	X	A	45 %

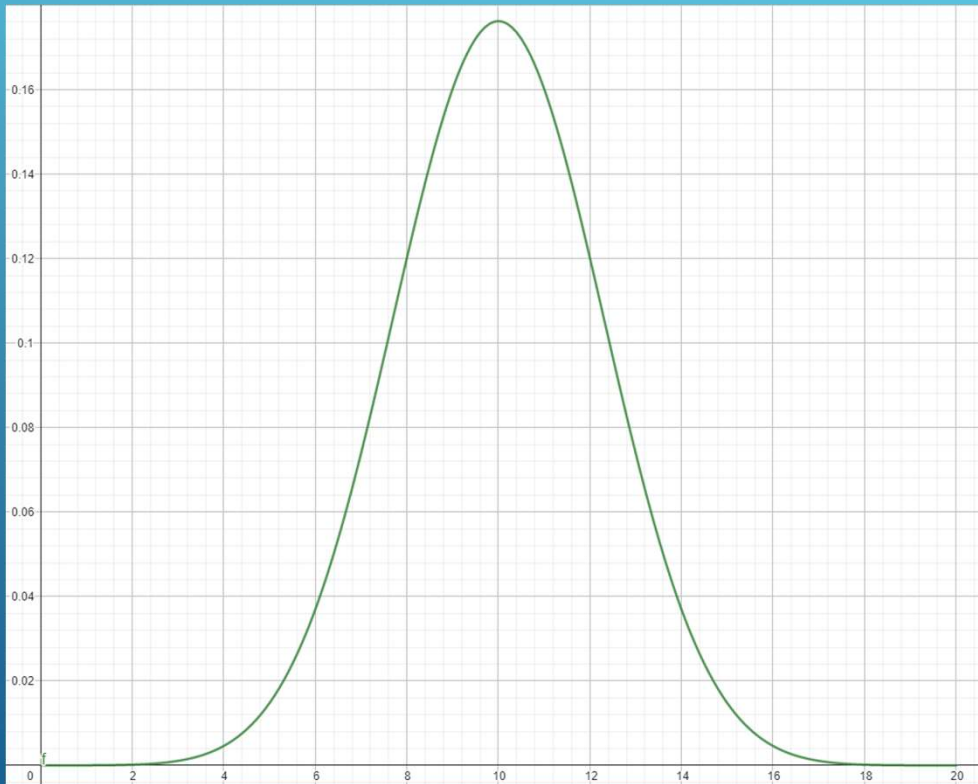
A : Formation d'arche(s)

X : Passage fluide

Recherche d'un
modèle adapté

Application du modèle
à un exemple

Modélisation
informatique



► Échec de la modélisation

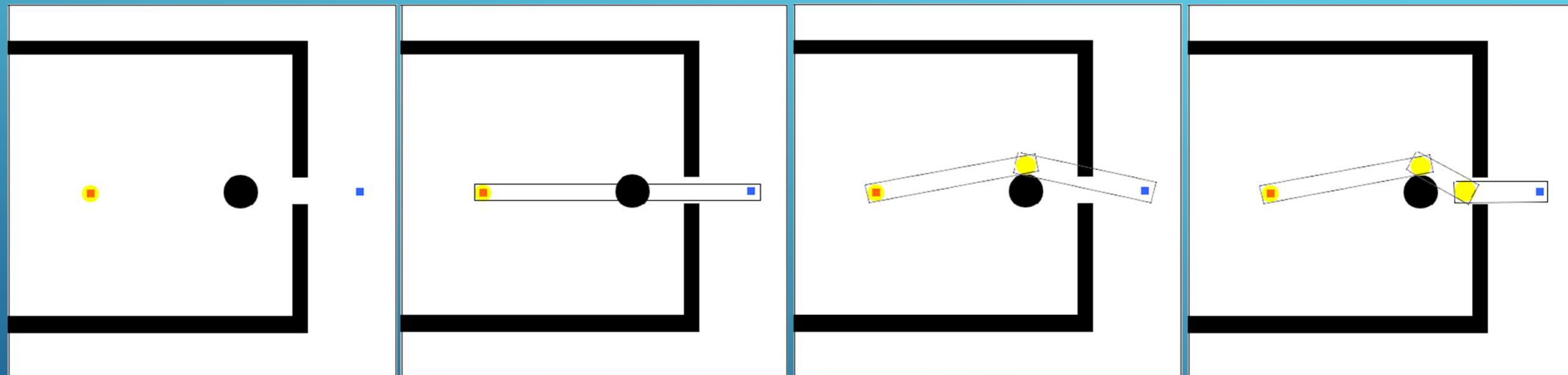
Gaussienne de la modélisation pour une probabilité de 0,5 de former une arche :

Recherche d'un
modèle adapté

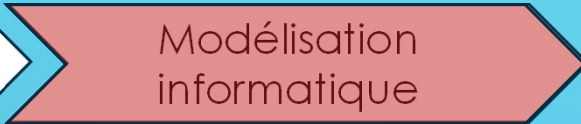
Application du modèle
à un exemple

Modélisation
informatique

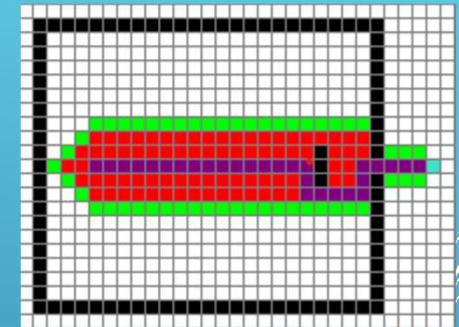
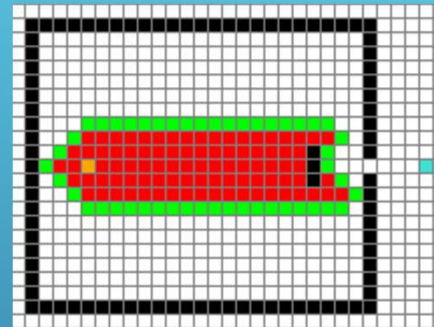
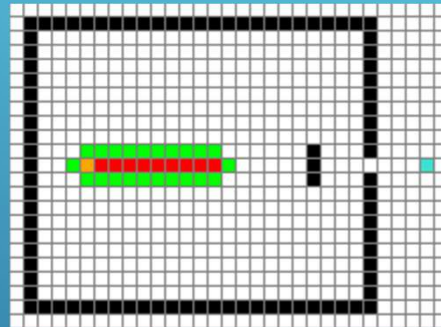
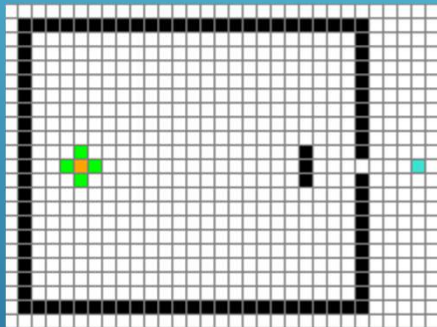
► Algorithme personnel :



- : Début
- : Fin
- : Obstacle
- : Nœuds de la trajectoire



- ▶ L'algorithme A^*



1 : Début

Fin : Fin

■ : Obstacle

Case : Case considérée

Case 1 : Case déjà vérifiée

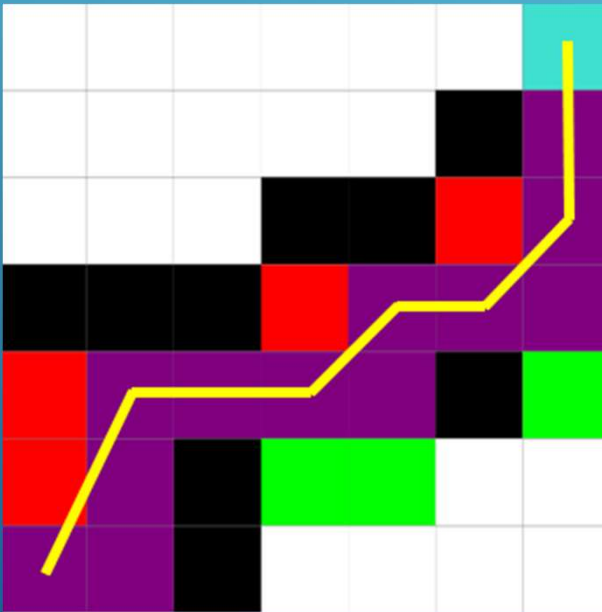
Chemin obtenu

Recherche d'un
modèle adapté

Application du modèle
à un exemple

Modélisation
informatique

► L'algorithme A*



Listes des coordonnées successives, initialement :

[(6, 1), (6, 2), (6, 3), (5, 3), (4, 3), (4, 4), (3, 4), (2, 4), (1, 4), (1, 5), (1, 6), (0, 6)]

Listes des coordonnées successives, après algorithme A* :

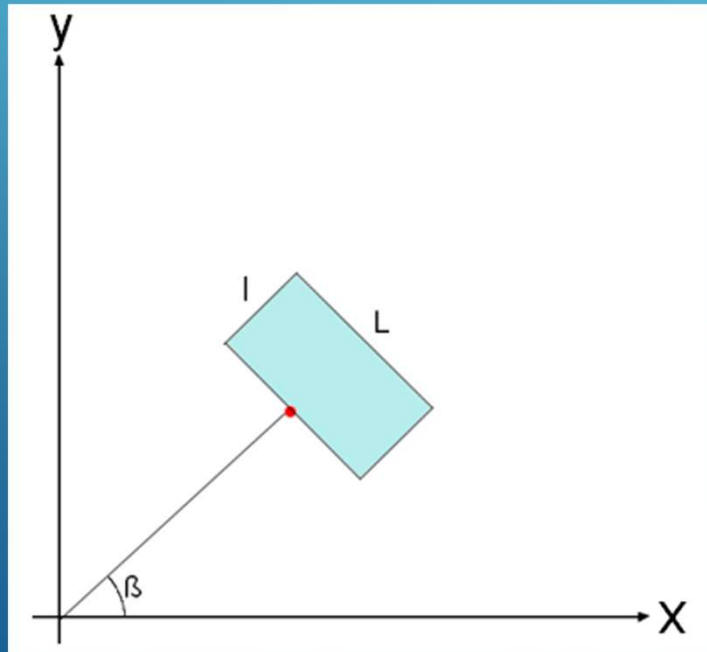
[(6, 1), (6, 2), (5, 3), (4, 3), (3, 4), (1, 4), (0, 6)]

Recherche d'un
modèle adapté

Application du modèle
à un exemple

Modélisation
informatique

- La prise en compte des autres individus



$$\text{Max} \left(\frac{2}{L} \left| x_M - x_0 - \frac{l}{2} \sin(\beta) \right|, \frac{2}{l} \left| y_M - y_0 - \frac{l}{2} \cos(\beta) \right| \right) \leq 1$$

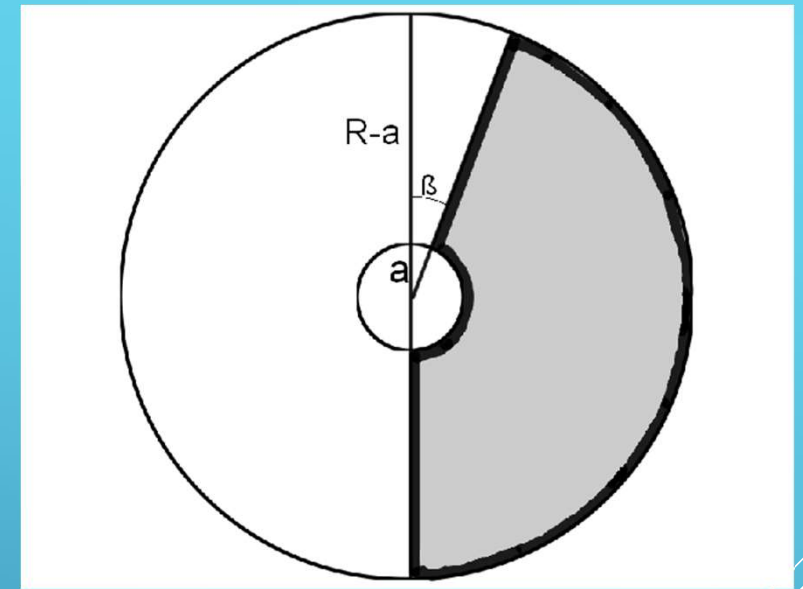
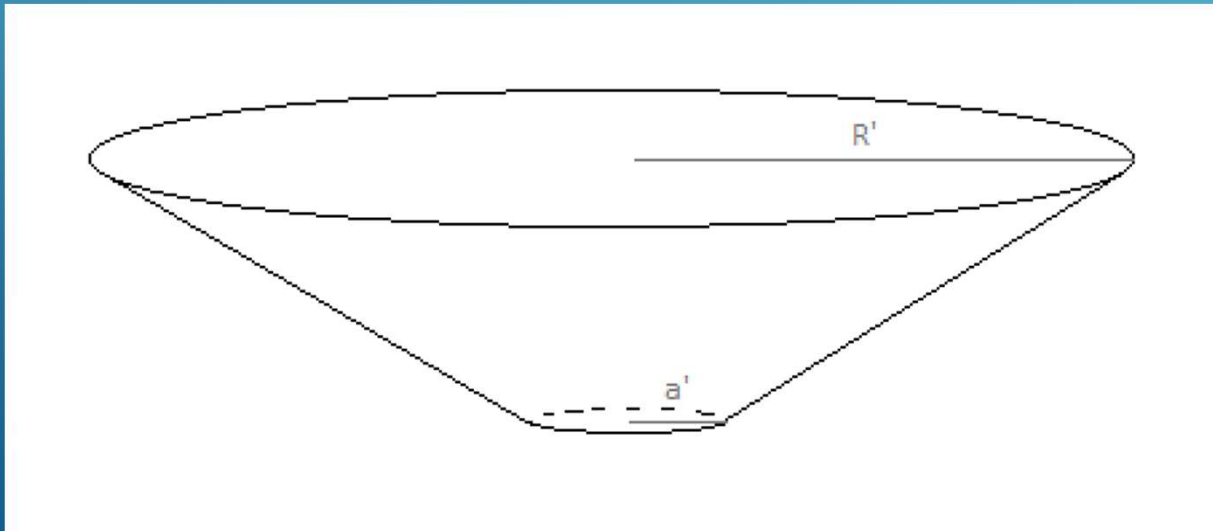
CONCLUSION

Several thin, white, parallel diagonal lines are drawn across the bottom right corner of the slide, creating a sense of motion or a stylized graphic element.

16/32

ANNEXES :

- Fabrication de l'entonnoir:



Données initiales :

$$\pi R' = (\pi - \beta)R$$

$$\pi a' = (\pi - \beta)a$$

$$R' = a' + (R - a)\cos(\alpha)$$

Système résolu :

$$R' = R \times \cos(\alpha)$$

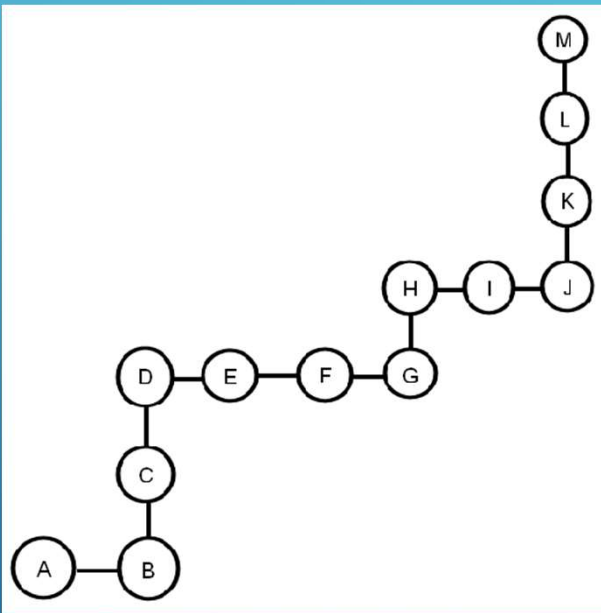
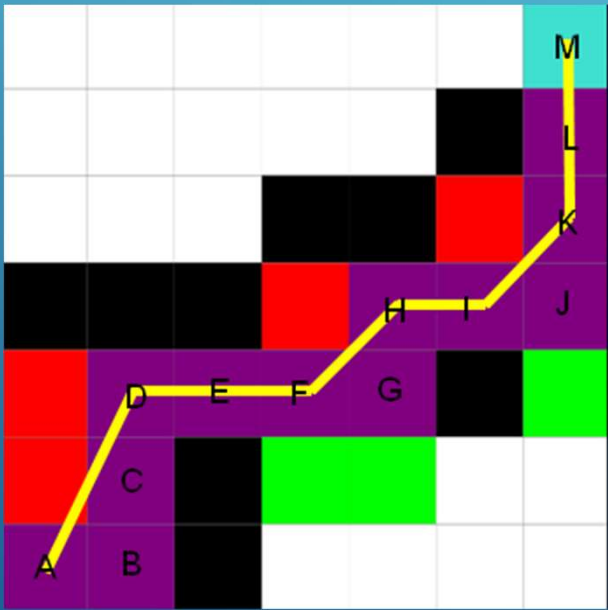
$$a' = a \times \cos(\alpha)$$

$$\beta = \pi \times (1 - \cos(\alpha))$$

ANNEXES :

► Algorithme A*

	A	B	C	D	E	F	G	H	I	J	K	L	M
F-score	8,5	8,8	8,5	8,6									
G-score	0	1	1,4	2,2									
H-score	8,5	7,8	7,1	6,4									
Dernier	X	A	A	A									



ANNEXES :

► Code de l'algorithme personnel

```
##### MODULES #####
import matplotlib.image as mpimg # permet la conversion de l'image
from math import sqrt

##### DEFINITION DES VARIABLES DU PROBLEME #####
path = "C:/Users/paull/OneDrive/Bureau/Sans Titre.png" # Chemin d'accès de l'image servant de fond
r = 1 # facteur permettant de réduire l'image (voir plus tard avec 'divide_image')
rayon = 3 # rayon des individus assimilés à des cercles
leng = 5
larg = 5

##### FONCTIONS UTILES #####
def som(L):
    """
    Cette fonction fonctionne de la même manière que la fonction 'sum'.
    Elle peut cependant additionner des listes et pas seulement des int ou float.
    """
    if isinstance(L[0], int) or isinstance(L[0], float):
        return sum(L)
    if isinstance(L[0], list):
        l = []
        for elt in L:
            l += elt
        return l

def moy(L):
    """
    Effectue une moyenne rapide de la liste (ou tuple) passée en argument.
    """
    return som(L)/len(L)
```

```

def affich_tabl(tabl):
    """
    Fonction pratique pour afficher un tableau ligne par ligne et avoir plus de visibilité.
    """
    for elt in tabl:
        print(elt)

def pyth(a, b):
    print("a: ", a, "b: ", b)
    """
    Applique le théorème de Pythagore pour déterminer la distance entre a et b
    """
    return sqrt(sum([(a[elt] - b[elt]) ** 2 for elt in range(len(a))]))

def add_list_lin(L1, L2, y1, y2):
    """
    Fait une moyenne pondérée de deux vecteurs
    """
    return [L1[elt] * y1 + L2[elt] * y2 for elt in range(len(L1))]

##### IMPORT DU FOND ET MISE EN PLACE DU CADRE #####
def RGV_to_int(l):
    """
    Une fois importée, l'image est au format RGV (liste de listes de listes).
    Les transforme en liste de listes de nombres
    """
    for elt1 in range(len(l)):
        for elt2 in range(len(l[0])):
            l[elt1][elt2] = round(moy(l[elt1][elt2]), 1)

```

```
def find_dots(image1):  
    """  
    Les coordonnées initiales des personnes sont renseignées directement sur l'image: on les récupère.  
    """  
    l = []  
    exit1 = None  
    for i1, i2 in enumerate(image1):  
        for j1, j2 in enumerate(i2):  
            if j2 == 0.4:  
                image1[i1][j1] = 1.0  
                exit1 = [i1, j1]  
            if j2 == 0.5:  
                image1[i1][j1] = 1.0  
                l.append([i1, j1])  
    return exit1, l
```

```

def divide_image(image1, factor):
    """
    Divise le temps de calcul par r**2 environ
    """
    L = list()
    elt1, elt2 = factor - 1, factor - 1
    width = len(image1)
    lenght = len(image1[0])
    while elt1 < width:
        l = []
        while elt2 < lenght:
            m = [image1[elt1 - k][elt2 - factor + 1:elt2 + 1] for k in range(factor)]
            l.append(round(moy(som(m)), 1))
            elt2 += factor
        if elt2-factor+1 != lenght:
            m = [image1[elt1 - k][elt2 - factor + 1:] for k in range(r)]
            l.append(round(moy(som(m)), 1))
        L.append(l)
        elt1 += factor
        elt2 = factor - 1
    if elt1-factor+1 != width:
        elt1 -= factor
        l = []
        while elt2 < lenght:
            m = [image1[elt1 + k][elt2 - factor + 1:elt2 + 1] for k in range(1, width - elt1)]
            l.append(round(moy(som(m)), 1))
            elt2 += factor
        if elt2-factor+1 != lenght:
            m = [image1[elt1 + k][elt2 - factor + 1:] for k in range(1, width - elt1)]
            l.append(round(moy(som(m)), 1))
        L.append(l)
    return L
return L

```

```

##### INITIALISATION ET MISE A JOUR DES POSITIONS #####
img = (mpimg.imread(path)).tolist()
RGV_to_int(img)
dots = find_dots(img)
exit = dots[0]
coords = dict()
for i, j in enumerate(dots[1]):
    coords[i] = [j, [j]]

##### CALCUL DU CHEMIN SANS PRISE EN COMPTE DES INTERACTIONS #####

def verifie_cercle(image1, pos, radius, **kwargs):
    center = kwargs.get('center', pos)
    checked = kwargs.get('checked', [])
    if image1[int("{:.0f}".format(pos[0]))][int("{:.0f}".format(pos[1]))] == 0.0:
        return True
    checked.append(pos)
    for k in [[pos[0]-2, pos[1]], [pos[0]+2, pos[1]], [pos[0], pos[1]-2], [pos[0], pos[1]+2]]:
        if k not in checked and pyth(k, center) < radius:
            if verifie_cercle(image1, k, radius, center=center, checked=checked):
                return True
    return False

```



```

def verifie_chemin2(image1, chemin):
    valid = True
    L = []
    pos = None
    for t in range(len(chemin[0]) - 1):
        if chemin[1][t]:
            L.append(True)
        if chemin[1][t] is False:
            L.append(False)
            valid = False
        else:
            dist = pyth(chemin[0][t], chemin[0][t + 1])
            k = 2 * rayon / dist
            while k <= 1:
                checking_pos = add_list_lin(chemin[0][t], chemin[0][t + 1], k, 1 - k)
                int_pos = [int(elt) for elt in checking_pos]
                if verifie_cercle(image1, int_pos, rayon):
                    valid = False
                    L.append(False)
                    pos = checking_pos
                    k = 2
            k += 2*rayon/dist
    return "True" if valid else [chemin[0], L, pos]

def longueur_chemin(chemin):
    return sum([pyth(chemin[elt], chemin[elt+1]) for elt in range(len(chemin)-1)])

```

```

def calcule_chemin(table, debut, fin, **kwargs):
    chemins = kwargs.get("chemins", [((debut, fin), [None])])
    if not chemins:
        return "erreur, plus de chemin restant"
    result = verifie_chemin2(table, chemins[0])
    if result == "True":
        return chemins[0][0]
    pos = result[2]
    chemins.pop(0)
    for x, y in ([pos[0]+2*r, pos[1]], [pos[0]-2*r, pos[1]], [pos[0], pos[1]+2*r], [pos[0], pos[1]-2*r]):
        if (x, y) not in result[0] and not verifie_cercle(table, (x, y), rayon):
            k = result[1].index(False)
            new = (list(result[0][:k+1]) + [(x, y)] + list(result[0][k+1:]), list(result[1][:k]) + [None] + list(result[1][k:]))
            lenght = longueur_chemin(new[0])
            if not chemins:
                chemins.append(new)
            else:
                i = 0
                while longueur_chemin(chemins[i][0]) < lenght:
                    i += 1
                chemins = chemins[:i] + [new] + chemins[i:]
    return calcule_chemin(table, debut, fin, chemins=chemins)

### RECHERCHE DU CHEMIN ###

if __name__ == "__main__":
    image = divise_image(img, r)
    print(calcule_chemin(image, dots[1][0], exit))

```

ANNEXES :

► Code de l'algorithme A*

```
import pygame
from queue import PriorityQueue

WIDTH = 800
WIN = pygame.display.set_mode((WIDTH, WIDTH))
pygame.display.set_caption("A* Path Finding Algorithm")

RED = (255, 0, 0)
GREEN = (0, 255, 0)
BLUE = (0, 255, 0)
YELLOW = (255, 255, 0)
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
PURPLE = (128, 0, 128)
ORANGE = (255, 165, 0)
GREY = (128, 128, 128)
TURQUOISE = (64, 224, 208)

class Person:
    def __init__(self):
        self.pos = []
        self.angle = 0

class Spot:
    def __init__(self, row, col, width, total_rows):
        self.row = row
        self.col = col
        self.x = row * width
        self.y = col * width
        self.color = WHITE
        self.neighbors = []
        self.width = width
        self.total_rows = total_rows
```

```
def get_pos(self):
    return self.row, self.col

def is_closed(self):
    return self.color == RED

def is_open(self):
    return self.color == GREEN

def is_barrier(self):
    return self.color == BLACK

def is_start(self):
    return self.color == ORANGE

def is_end(self):
    return self.color == TURQUOISE

def reset(self):
    self.color = WHITE

def make_start(self):
    self.color = ORANGE

def make_closed(self):
    self.color = RED

def make_open(self):
    self.color = GREEN

def make_barrier(self):
    self.color = BLACK

def make_end(self):
    self.color = TURQUOISE

def make_path(self):
    self.color = PURPLE

def draw(self, win):
    pygame.draw.rect(win, self.color, (self.x, self.y, self.width, self.width))
```

```

def update_neighbors(self, grid):
    self.neighbors = []
    if self.row < self.total_rows - 1 and not grid[self.row + 1][self.col].is_barrier(): # DOWN
        self.neighbors.append(grid[self.row + 1][self.col])

    if self.row > 0 and not grid[self.row - 1][self.col].is_barrier(): # UP
        self.neighbors.append(grid[self.row - 1][self.col])

    if self.col < self.total_cols - 1 and not grid[self.row][self.col + 1].is_barrier(): # RIGHT
        self.neighbors.append(grid[self.row][self.col + 1])

    if self.col > 0 and not grid[self.row][self.col - 1].is_barrier(): # LEFT
        self.neighbors.append(grid[self.row][self.col - 1])

def __lt__(self, other):
    return False

def h(p1, p2):
    x1, y1 = p1
    x2, y2 = p2
    return abs(x1 - x2) + abs(y1 - y2)

def reconstruct_path(came_from, current, draw1):
    L = []
    while current in came_from:
        current = came_from[current]
        current.make_path()
        draw1()
        L.append((current.row, current.col))
    return L

```



```

def algorithm(draw1, grid, start, end):
    count = 0
    open_set = PriorityQueue()
    open_set.put((0, count, start))
    came_from = {}
    g_score = {spot: float("inf") for row in grid for spot in row}
    g_score[start] = 0
    f_score = {spot: float("inf") for row in grid for spot in row}
    f_score[start] = h(start.get_pos(), end.get_pos())
    open_set_hash = {start}
    while not open_set.empty():
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
        current = open_set.get()[2]
        open_set_hash.remove(current)

        if current == end:
            end.make_end()
            return reconstruct_path(came_from, end, draw1)

        for neighbor in current.neighbors:
            temp_g_score = g_score[current] + 1

            if temp_g_score < g_score[neighbor]:
                came_from[neighbor] = current
                g_score[neighbor] = temp_g_score
                f_score[neighbor] = temp_g_score + h(neighbor.get_pos(), end.get_pos())
                if neighbor not in open_set_hash:
                    count += 1
                    open_set.put((f_score[neighbor], count, neighbor))
                    open_set_hash.add(neighbor)
                    neighbor.make_open()

        draw1()

        if current != start:
            current.make_closed()

    return False

```

```

def make_grid(rows, width):
    """
    Initialise le tableau et les spots
    """
    grid = []
    gap = width // rows
    for i in range(rows):
        grid.append([])
        for j in range(rows):
            spot = Spot(i, j, gap, rows)
            grid[i].append(spot)

    return grid

def draw_grid(win, rows, width):
    gap = width // rows
    for i in range(rows):
        pygame.draw.line(win, GREY, (0, i * gap), (width, i * gap))
    for j in range(rows):
        pygame.draw.line(win, GREY, (j * gap, 0), (j * gap, width))

def draw(win, grid, rows, width):
    """
    - Remplit de blanc
    - Colorie les spots
    - Refait le quadrillage
    """
    win.fill(WHITE)

    for row in grid:
        for spot in row:
            spot.draw(win)

    draw_grid(win, rows, width)
    pygame.display.update()

```

```

def get_clicked_pos(pos, rows, width): # x,y ==> row,col
    gap = width // rows
    y, x = pos

    row = y // gap
    col = x // gap

    return row, col

def main(win, width):
    ROWS = 100
    grid = make_grid(ROWS, width)

    start = None
    end = None

    run = True

    while run:
        draw(win, grid, ROWS, width)
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                run = False

            if pygame.mouse.get_pressed()[0]: # LEFT
                pos = pygame.mouse.get_pos()
                row, col = get_clicked_pos(pos, ROWS, width)
                spot = grid[row][col]
                if not start and spot != end:
                    start = spot
                    start.make_start()

                elif not end and spot != start:
                    end = spot
                    end.make_end()

```

```

        elif spot != end and spot != start:
            spot.make_barrier()

    elif pygame.mouse.get_pressed()[2]: # RIGHT
        pos = pygame.mouse.get_pos()
        row, col = get_clicked_pos(pos, ROWS, width)
        spot = grid[row][col]
        spot.reset()
        if spot == start:
            start = None
        elif spot == end:
            end = None

    if event.type == pygame.KEYDOWN:
        if event.key == pygame.K_SPACE and start and end:
            for row in grid:
                for spot in row:
                    spot.update_neighbors(grid)
            a = algorithm(lambda: draw(win, grid, ROWS, width), grid, start, end)
            run = False

        if event.key == pygame.K_c:
            start = None
            end = None
            grid = make_grid(ROWS, width)

    pygame.quit()
    print(a)
    return a

```

```
main(WIN, WIDTH)
```