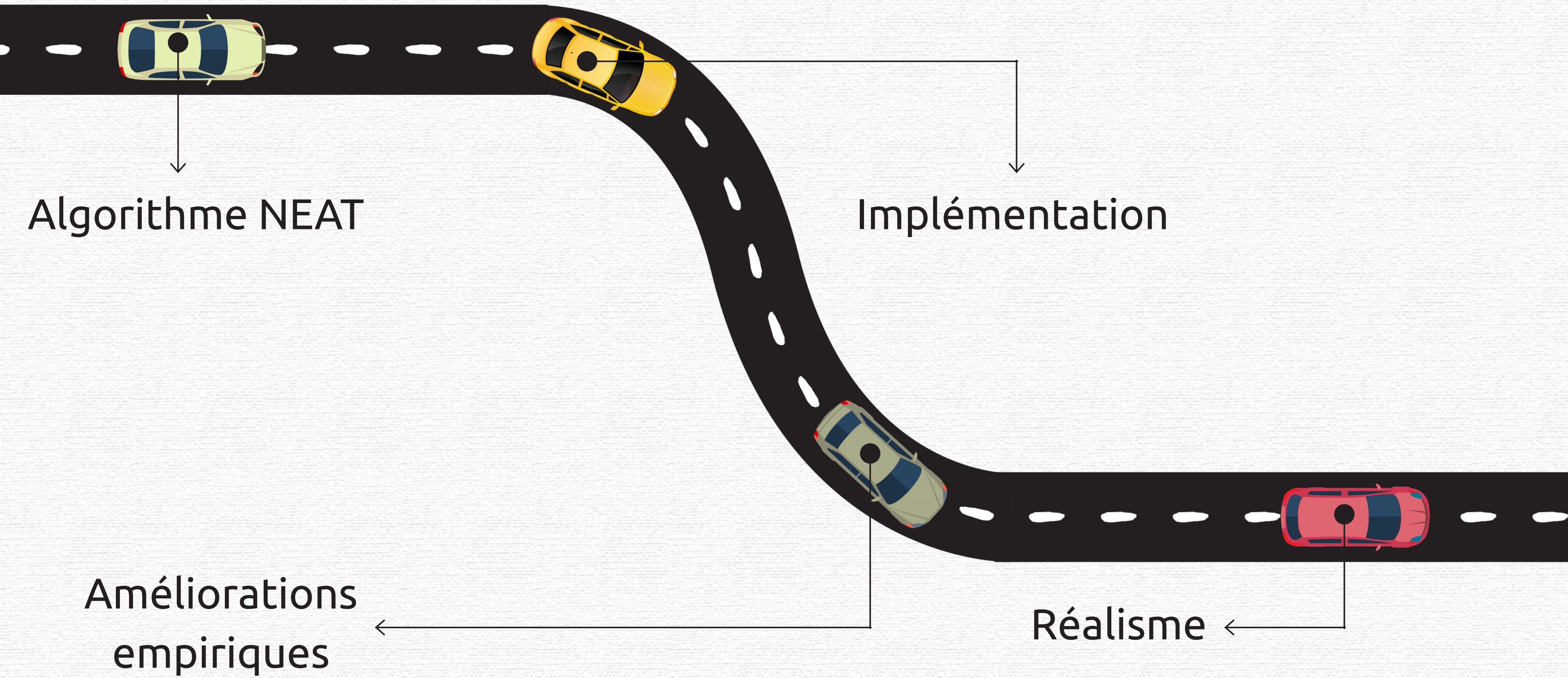
Algorithme NEAT : application aux jeux de course

Arthur EVAIN, MPI 2024

Candidat n°14102

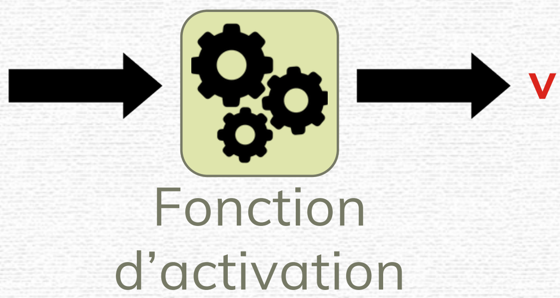
Plan

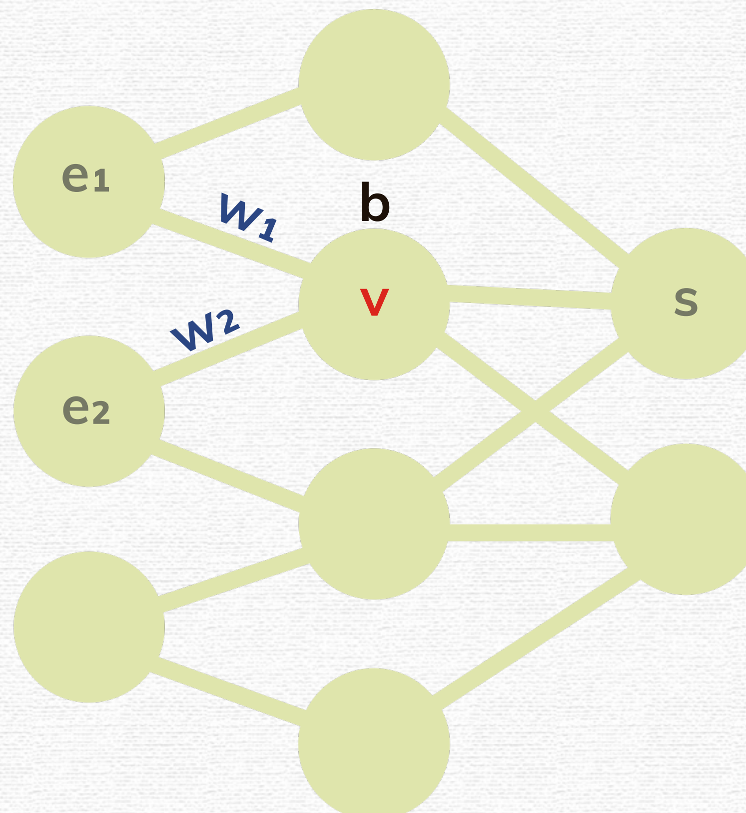


Dans quelle mesure l'algorithme NEAT permet-il de simuler le comportement d'un pilote de course humain ?

Réseaux de neurones

$w_1 \cdot e_1 + w_2 \cdot e_2 + b$





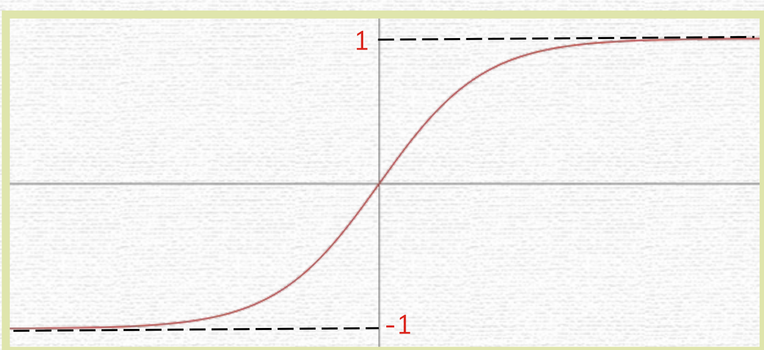
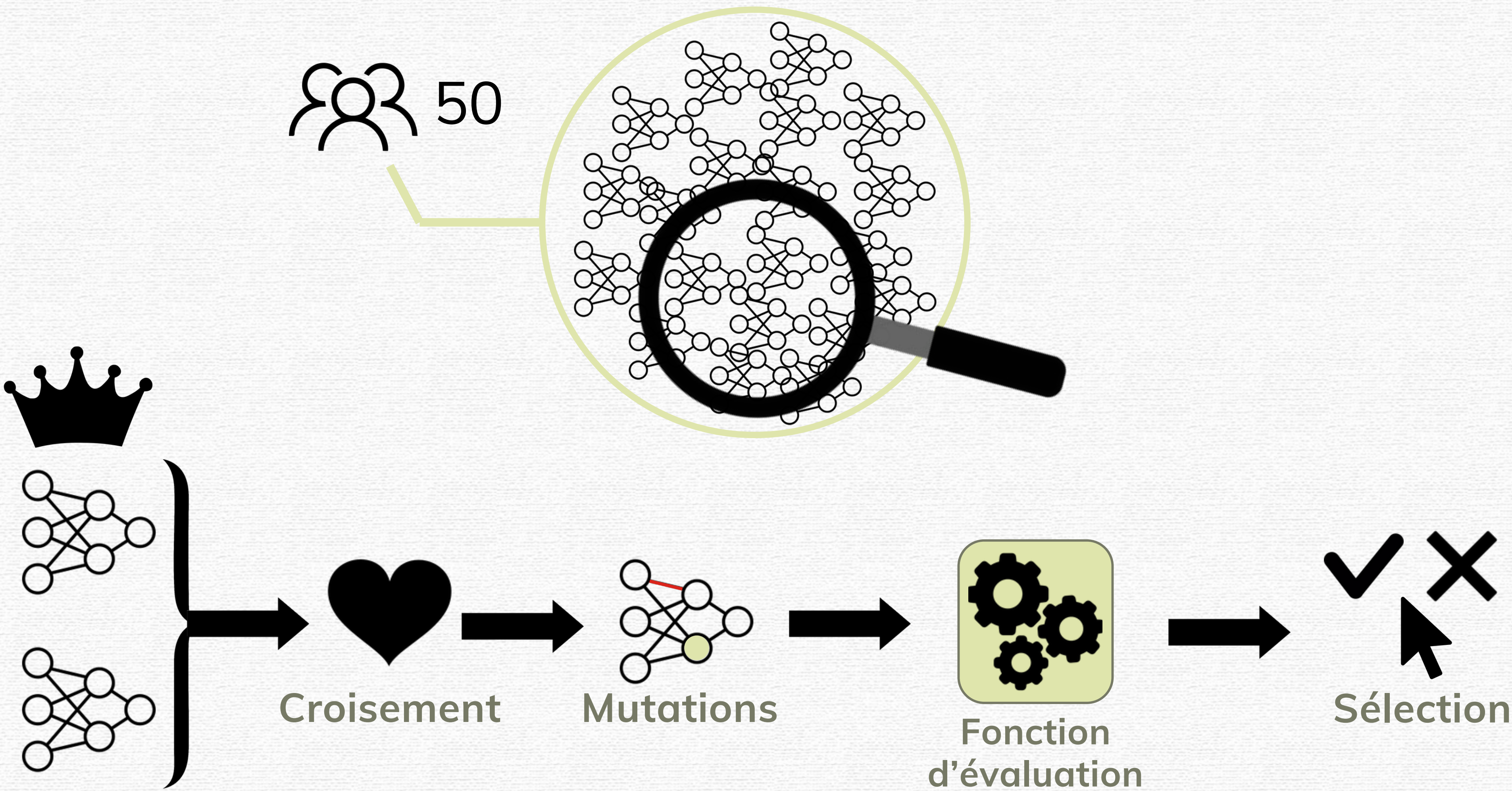
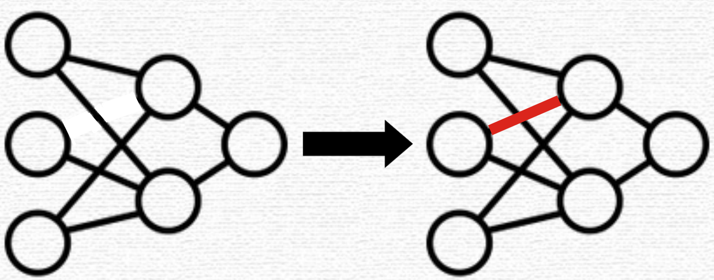


Figure 1 : **La fonction tangente hyperbolique**

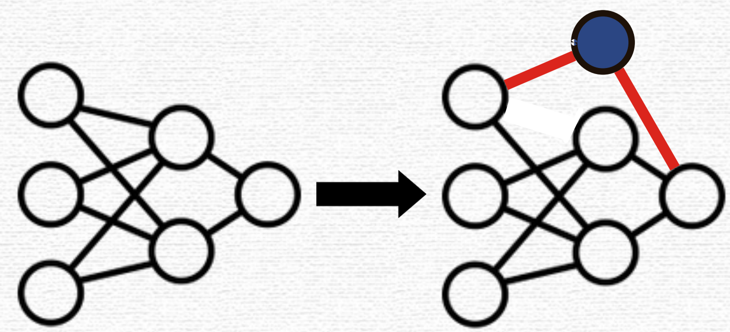
Algorithmes génétiques



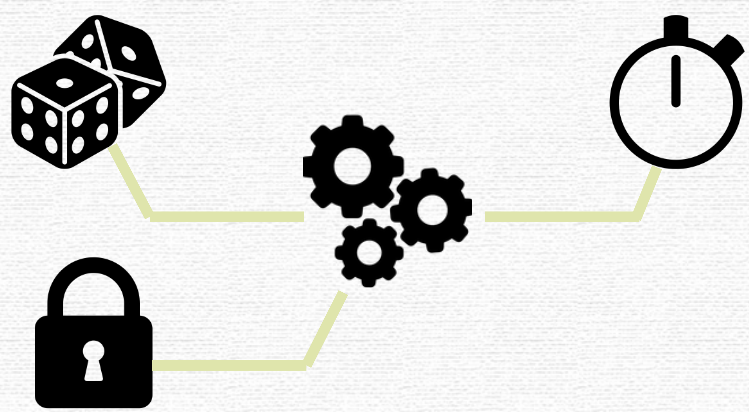
Algorithme NEAT



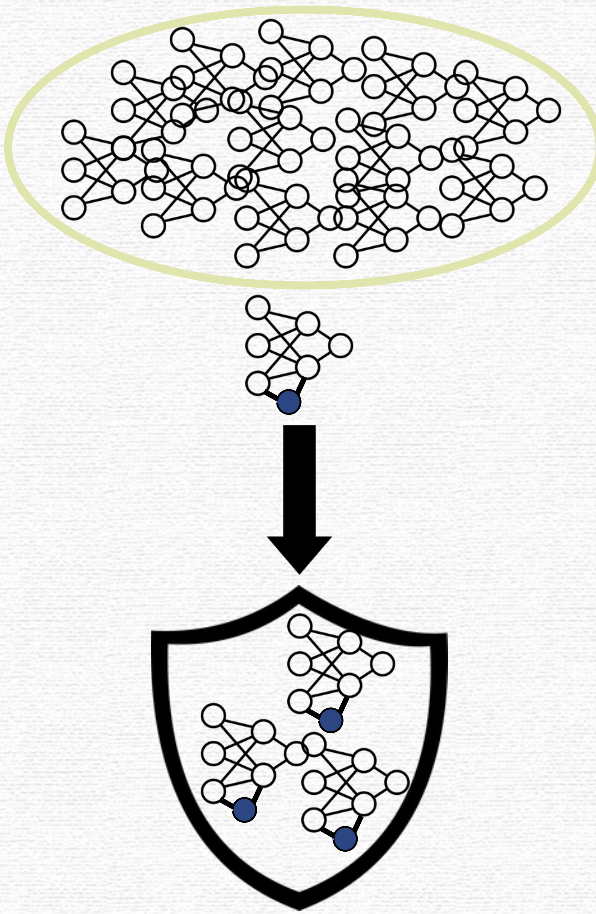
Ajout de connexion



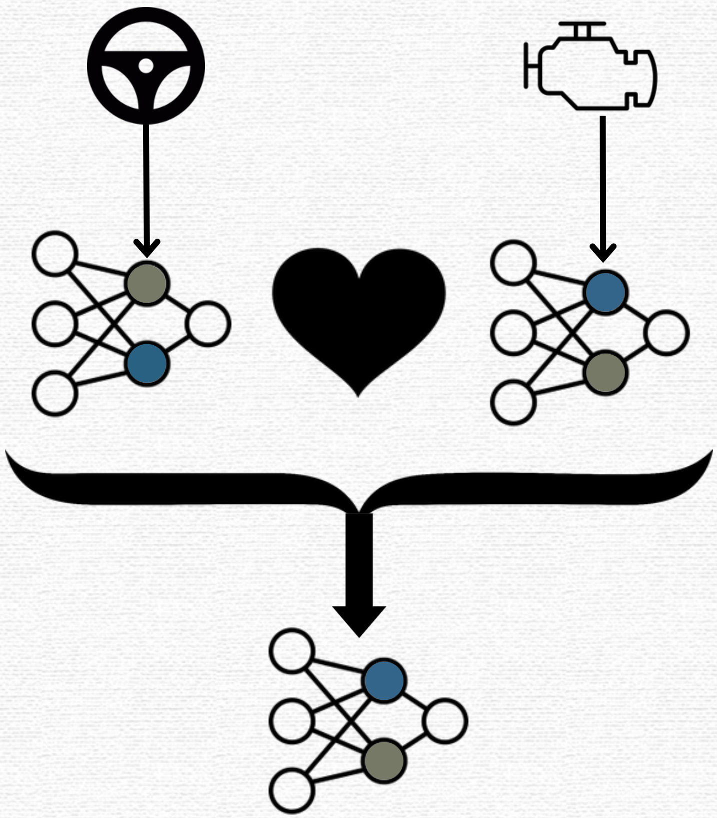
Ajout de noeud



Subtilités

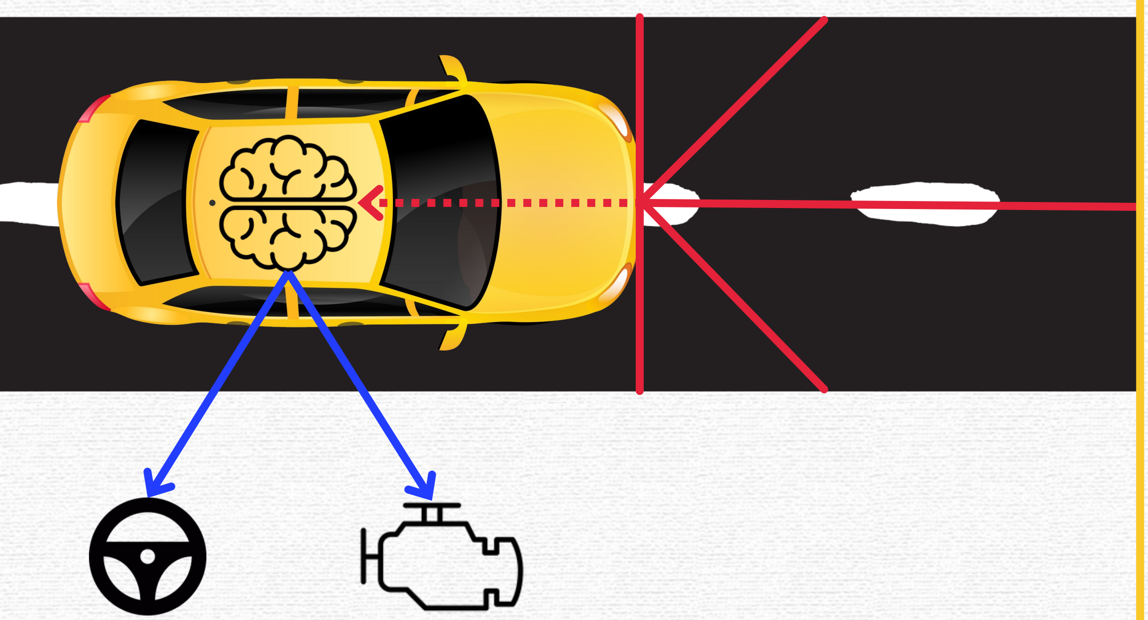


Spéciation



Marquage historique

Implémentation



Proc calc_distance(direction)
  (x0, y0) ← position du pixel avant de la voiture
  curseur ← couleur de la piste en (x0, y0)
  bord ← couleur de l'extérieur de la piste
  d ← 0

  Tant que curseur ≠ bord **faire**
    d ← d + 1
    Si le curseur est hors de l'image **alors**
    | retourner d
    Sinon
      x ← x0 + d · sin(angle + direction)
      y ← y0 + d · cos(angle + direction)
      curseur ← couleur de la piste en (x, y)
    Fin si
  Fin tant que
  retourner d

Algorithme 1 : **Calcul d'une valeur d'entrée**

Fonction d'évaluation

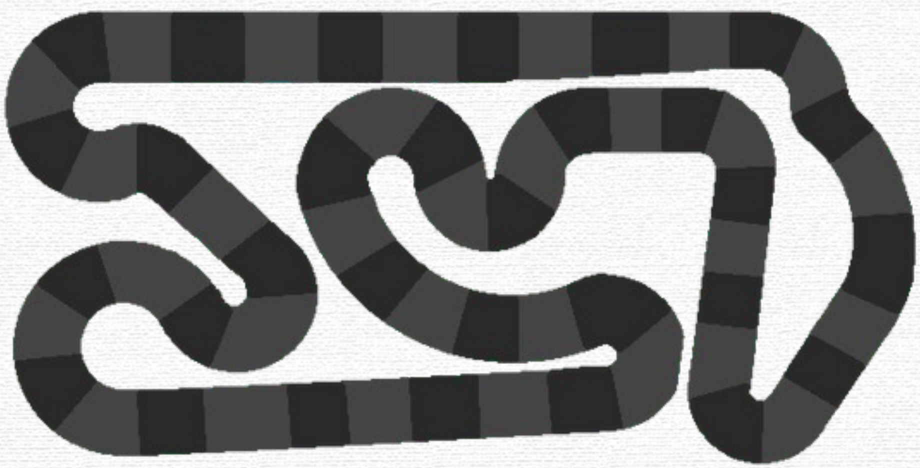


Figure 2 : **Circuit d'entraînement**

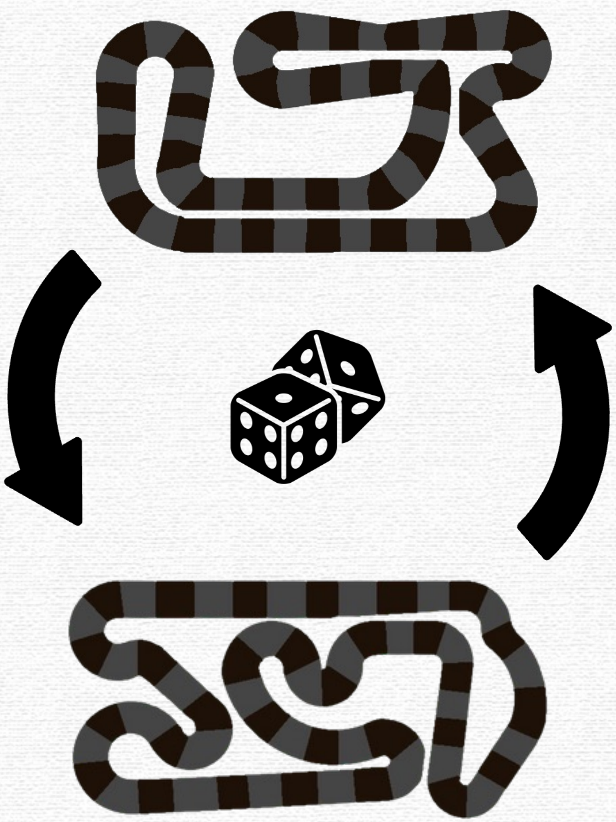
Proc gain
couleur ← couleur à l'avant de la voiture

Si couleur foncée et couleur claire **alors**
action ← 0
couleur ← couleur foncée
retourner récompense
Sinon si couleur claire et couleur foncée **alors**
action ← 0
couleur ← couleur claire
retourner récompense
Fin si

action ← action + 1
retourner 0

Algorithme 2 : **Fonction d'évaluation**

Premiers résultats



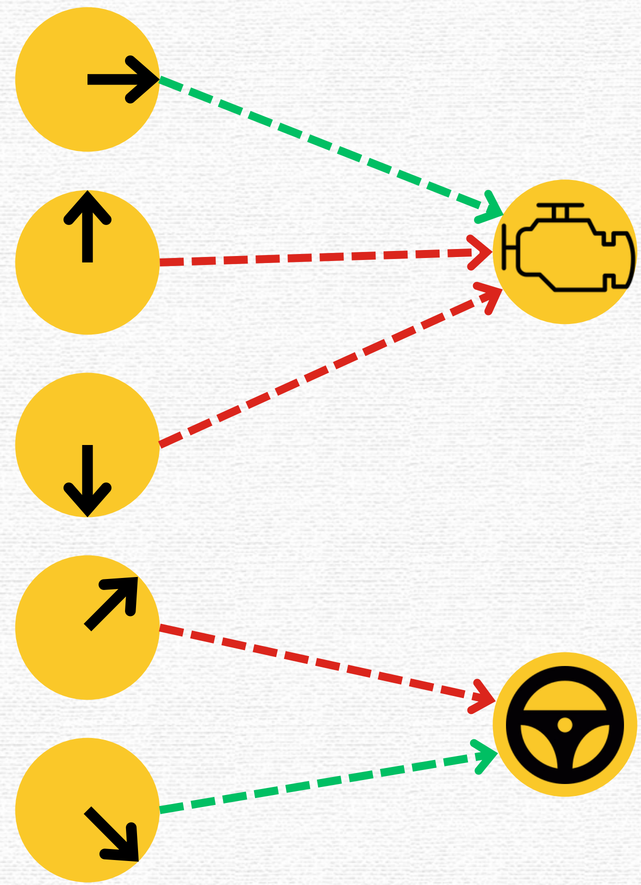


Figure 3 : **Réseau obtenu**

Améliorations empiriques

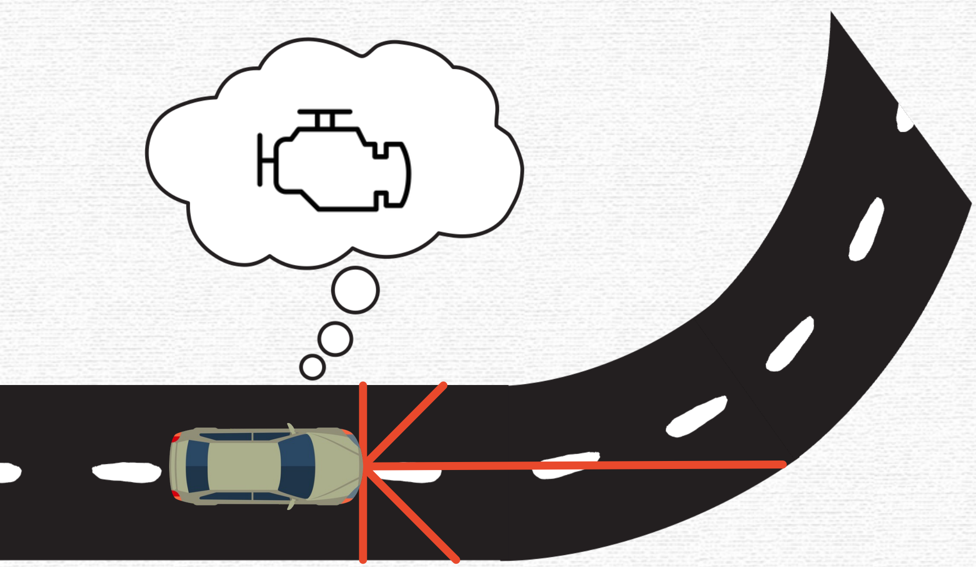


Figure 4 : **Un déficit de vision certain**

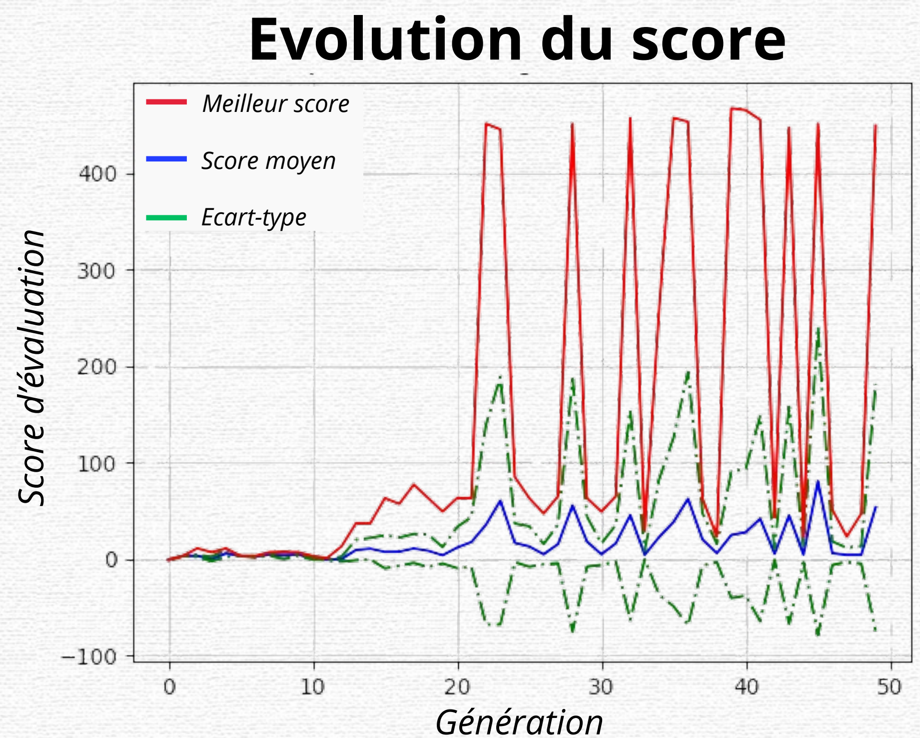


Figure 5 : **Des performances d'entraînement à demi-teinte**

Solutions

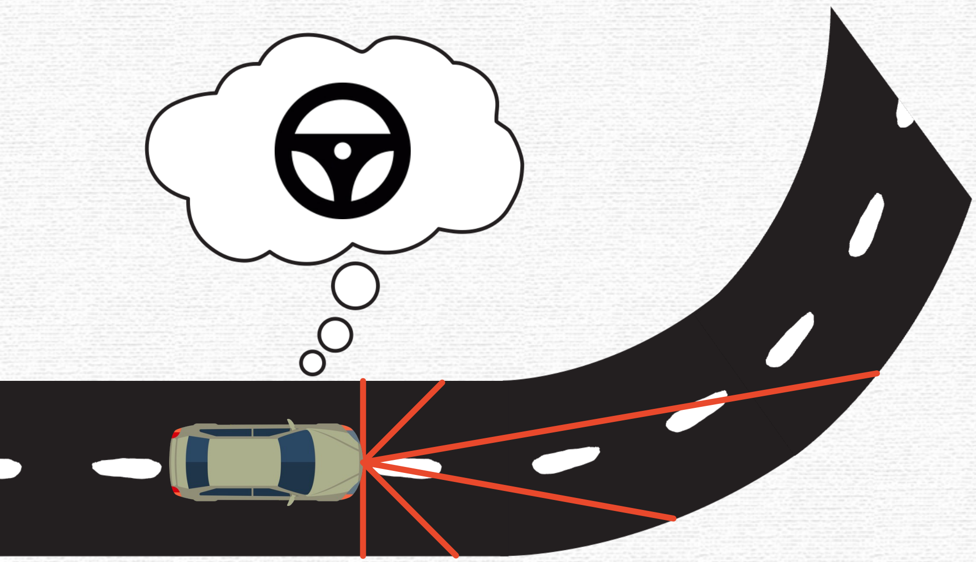


Figure 6 : **Une vision plus adaptée**

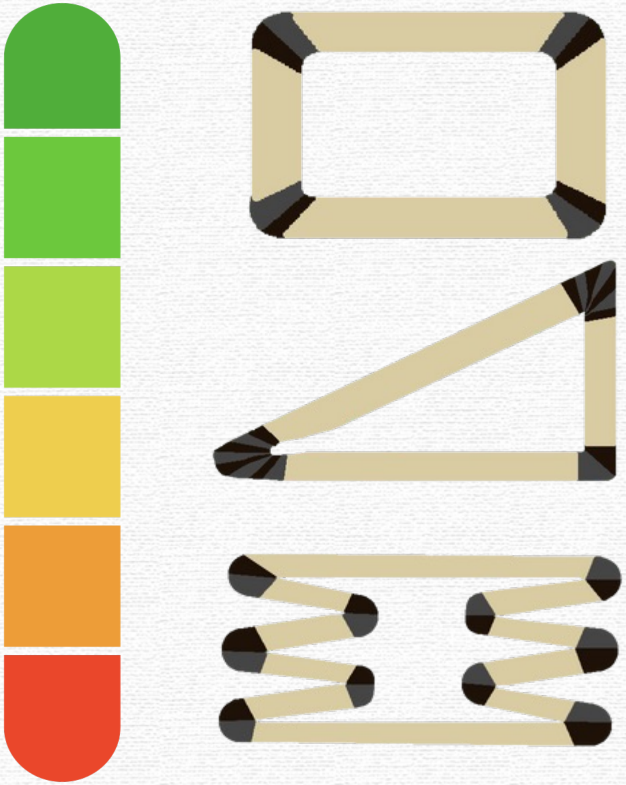
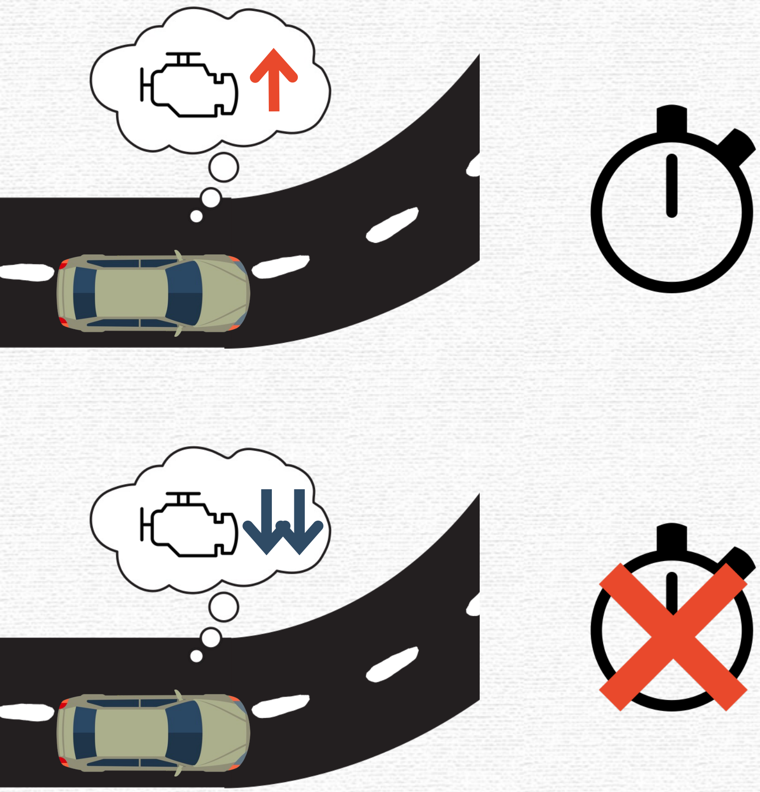


Figure 7 : **Des formes variées pour un entraînement progressif**

Résultats



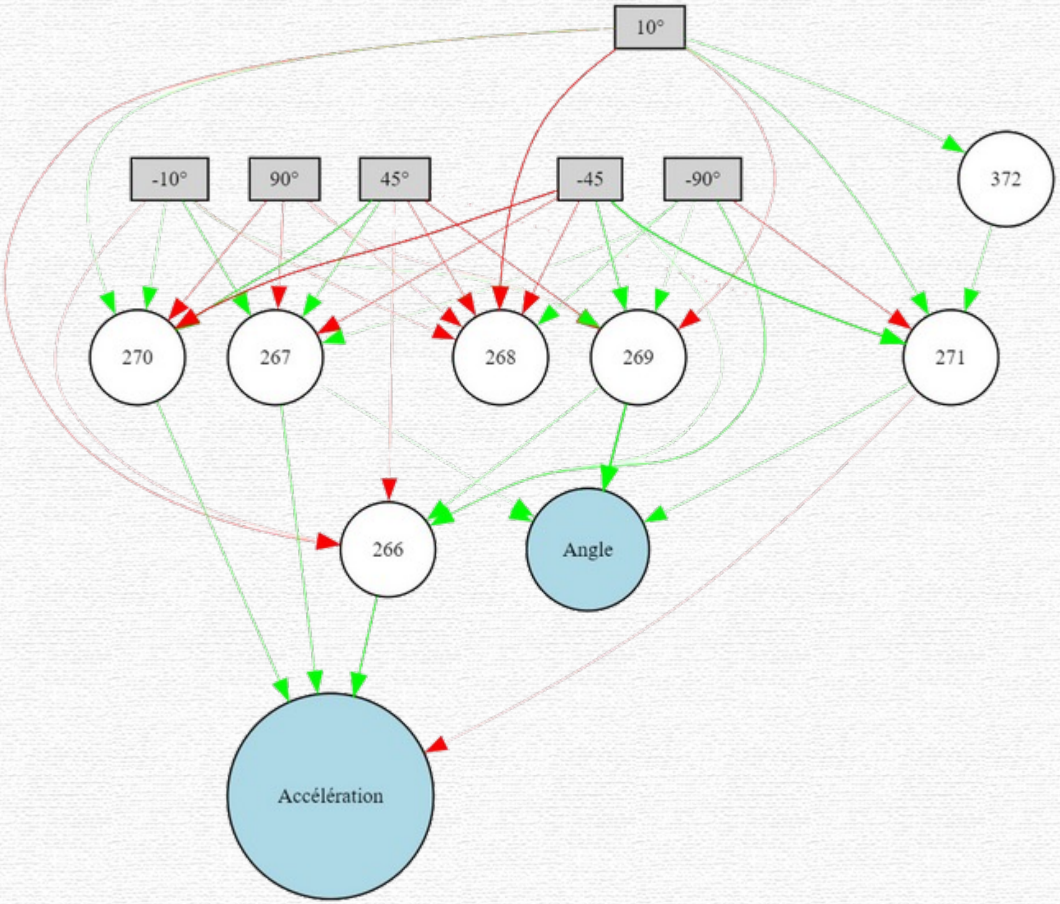
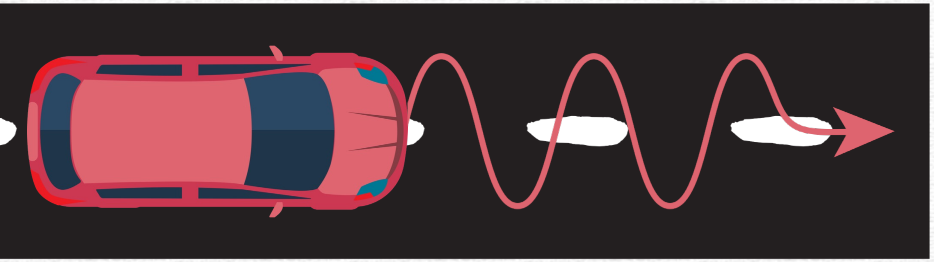
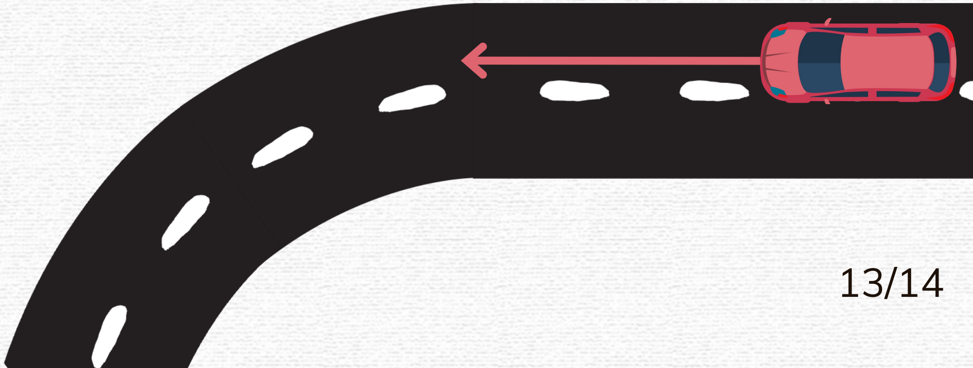


Figure 8 : **Une exploitation difficile des rayons**

Réalisme

Minimisation du rayon de courbure





Conclusion

Merci de m’avoir écouté !

Annexe

Simulation

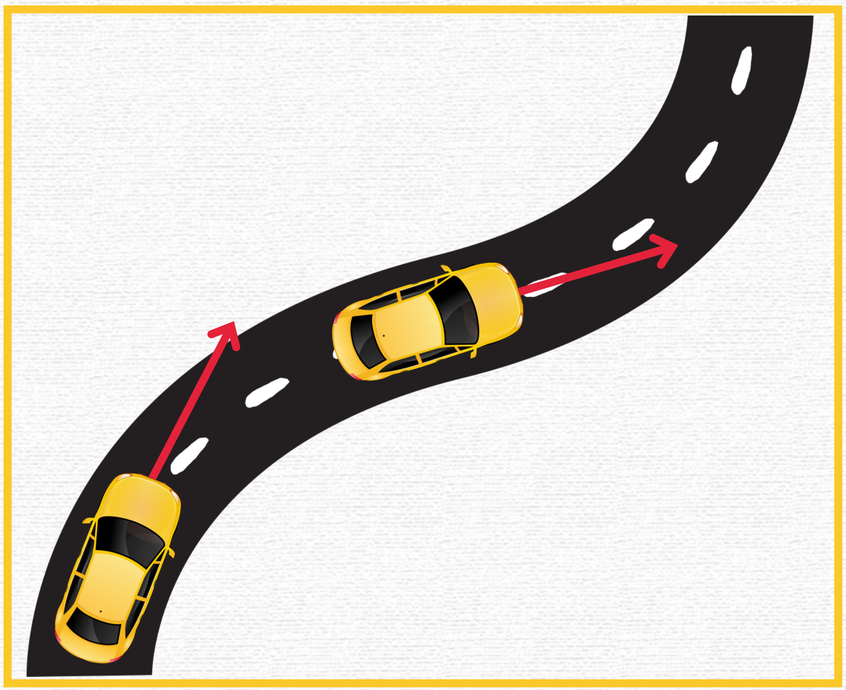


Figure : **Modèle physique**

Proc accélérer(s1)
  vitesse ← vitesse + sgn(s1) · min(|s1|, 0.1)
  Si vitesse > 2 **alors**
   | vitesse ← 2
  Si vitesse < 0.5 **alors**
   | vitesse ← 0.5

Proc tourner(s2)
  angle ← angle + s2 - sgn(s2) · vitesse
  Si angle < 0 **alors**
   | angle ← angle + 360
  Si angle > 360 **alors**
   | angle ← angle - 360

Proc bouger( )
  x ← x + sin(angle) · vitesse
  y ← y + cos(angle) · vitesse

Algorithme : **Classe Voiture**

Code

Initialisation

```
import os.path
import random
import pickle
import neat
import pygame
from PIL import Image
from car import Car
import math
import visualize

pygame.font.init()

filename1 = "PIERRE_CUP_ZONED2.png"
filename2 = "PIERRE_CUP_ZONED3.png"
filename3 = "PIERRE_CUP_ZONED4.png"
filename4 = "PIERRE_CUP_ZONED5.png"
filename5 = "PIERRE_CUP_ZONED6.png"
trackpx1 = Image.open(filename1)
trackpx2 = Image.open(filename2)
trackpx3 = Image.open(filename3)
trackpx4 = Image.open(filename4)
trackpx5 = Image.open(filename5)
window = pygame.display.set_mode((trackpx1.size[0], trackpx1.size[1]))

font = pygame.font.SysFont("comicsans", 15)
gen = 0
```

Affichage

```
def draw_screen(screen, cars, BG_IMG, generation, seconds, cars_len, id):
  screen.blit(BG_IMG, (0, 0))
  for car in cars:
    car.draw(screen)

  gen = font.render("Gen: " + str(generation), True, (255, 255, 255))
  screen.blit(gen, (10, 10))
  timer = font.render("Time: " + str(seconds), True, (255, 255, 255))
  screen.blit(timer, (trackpx1.size[0] - timer.get_width(), 10))
  cars_alive = font.render("Cars alive: " + str(cars_len), True,
                           (255, 255, 255))
  screen.blit(cars_alive,
              (10, trackpx1.size[1] - 10 - cars_alive.get_height()))
  screen.blit(cars_alive, (10, 500))
  pygame.display.update()
```

Classe Track

```python
class Track:

  def __init__(self):
    self.size = trackpx1.size[0], trackpx1.size[1]
    self.track = [
        Image.open("PIERRE_CUP_ZONED2.png"),
        Image.open("PIERRE_CUP_ZONED3.png"),
        Image.open("PIERRE_CUP_ZONED4.png"),
        Image.open("PIERRE_CUP_ZONED5.png"),
        Image.open("PIERRE_CUP_ZONED6.png")
    ]
    self.gen = 0

  def collide(self, car, id):
    front_x, back_x, front_y, back_y = car.get_ends()
    rf, gf, bf, *a = self.track[id].getpixel((front_x, front_y))
    rb, gb, bb, *a = self.track[id].getpixel((back_x, back_y))
    if (rf, gf, bf) == (109, 129, 98) or (rb, gb, bb) == (109, 129, 98):
      return True
    return False

  def draw(self, win):
    win.blit(self.track, (0, 0))
```

Fonction d’évaluation

```
def eval_genomes(genomes, config):
  global gen
  gen += 1
  nets = []
  cars = []
  ge = []

  #CHOIX DU CIRCUIT
  track_id = random.randint(0, 4)
  pos_id = random.randint(0, 4)

  if track_id == 0:
    if pos_id == 0:
      default_x, default_y = 300, 220
    elif pos_id == 1:
      default_x, default_y = 390, 90
    elif pos_id == 2:
      default_x, default_y = 500, 35
    elif pos_id == 3:
      default_x, default_y = 475, 270
    elif pos_id == 4:
      default_x, default_y = 135, 45
  elif track_id == 1:
    if pos_id == 0:
      default_x, default_y = 425, 75
    elif pos_id == 1:
      default_x, default_y = 360, 25
    elif pos_id == 2:
      default_x, default_y = 80, 170
    elif pos_id == 3:
      default_x, default_y = 365, 255
    elif pos_id == 4:
      default_x, default_y = 255, 75
  elif track_id == 2:
    default_x, default_y = 320, 30
  elif track_id == 3:
    default_x, default_y = 385, 35
  elif track_id == 4:
    default_x, default_y = 280, 165

  BG_IMG = pygame.image.load(f"Circuit{track_id}.png")
```

```python
for genome_id, genome in genomes:
    net = neat.nn.FeedForwardNetwork.create(genome, config)
    nets.append(net)
    cars.append(Car(default_x, default_y, [trackpx1, trackpx2, trackpx3, trackpx4, trackpx5]))
    genome.fitness = 0
    ge.append(genome)

  clock = pygame.time.Clock()
  track = Track()
  clock.get_time()
  running = True
  begin_time = pygame.time.get_ticks()
  frames = 0

  while running:

    #PARAMETRE PYGAME
    clock.tick(60)
    millis = pygame.time.get_ticks()
    seconds = (millis - begin_time) // 1000
    frames += 1

    for event in pygame.event.get():
      if event.type == pygame.QUIT:
        running = False
        pygame.quit()
        quit()
        break
```

```python
#GAIN
for x, car in enumerate(cars):
  gain = car.gain_points(track_id)
  ge[x].fitness += gain

  if track.collide(car, track_id) or car.stay >= 50:
    ge[x].fitness -= 1
    cars.pop(x)
    nets.pop(x)
    ge.pop(x)
    continue

  if seconds > 60:
    ge[x].fitness += 1000
    cars.pop(x)
    nets.pop(x)
    ge.pop(x)
    continue

#ARRET
if len(cars) == 0:
  running = False

#MOUVEMENT
for x, car in enumerate(cars):
  inputs = car.generate_inputs(track_id)
  outputs = nets[x].activate(inputs)
  car.accelerate(outputs[1])
  car.turn(outputs[0])
  car.move()

draw_screen(window, cars, BG_IMG, gen, seconds, len(cars), track_id)
```

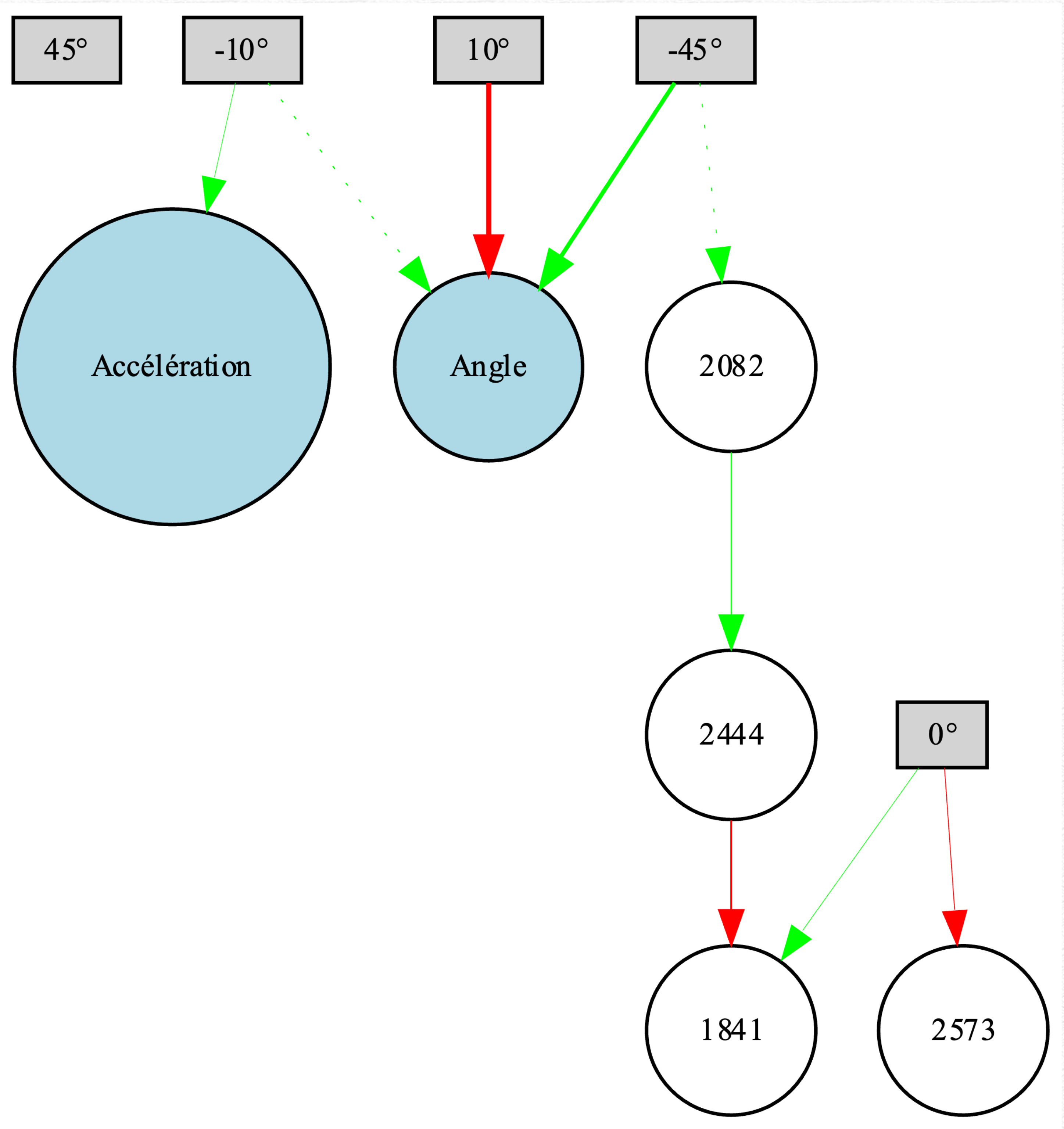
Main

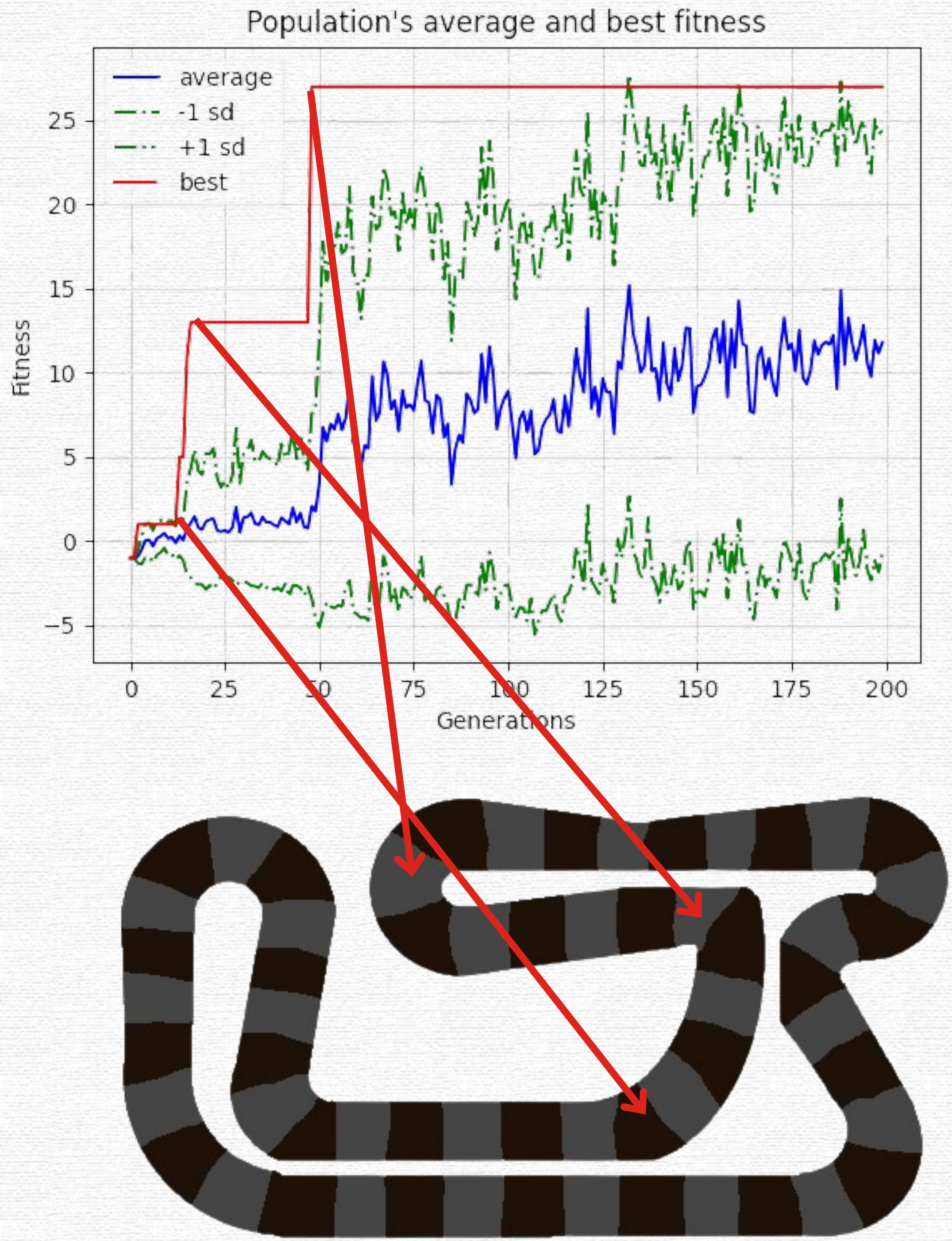
```
def run(config_path):
  config = neat.config.Config(neat.DefaultGenome, neat.DefaultReproduction,
                              neat.DefaultSpeciesSet, neat.DefaultStagnation,
                              config_path)

  p = neat.Population(config)
  p.add_reporter(neat.StdOutReporter(True))
  stats = neat.StatisticsReporter()
  p.add_reporter(stats)
  p.add_reporter(neat.Checkpointer(50, 3600))
  #p = neat.Checkpointer.restore_checkpoint('neat-checkpoint-%i' % 349)

  winner = p.run(eval_genomes, 300)

  with open("winner.pkl", "wb") as f:
    pickle.dump(winner, f)
    f.close()

  visualize.plot_stats(stats, ylog=False, view=True)
  visualize.plot_species(stats, view=True)
  node_names = {-5: '-45°', -4: '-10°', -3: '0°', -2: '10°', -1: '45°', 0: 'Angle', 1: 'Accélération'}
  visualize.draw_net(config, winner, True, node_names=node_names)
  #visualize.draw_net(config, winner, True, node_names=node_names, prune_unused=True)

  stats.save()


if __name__ == '__main__':
  local_dir = os.path.dirname(__file__)
  config_path = os.path.join(local_dir, "config.txt")
  run(config_path)
```

Classe Car

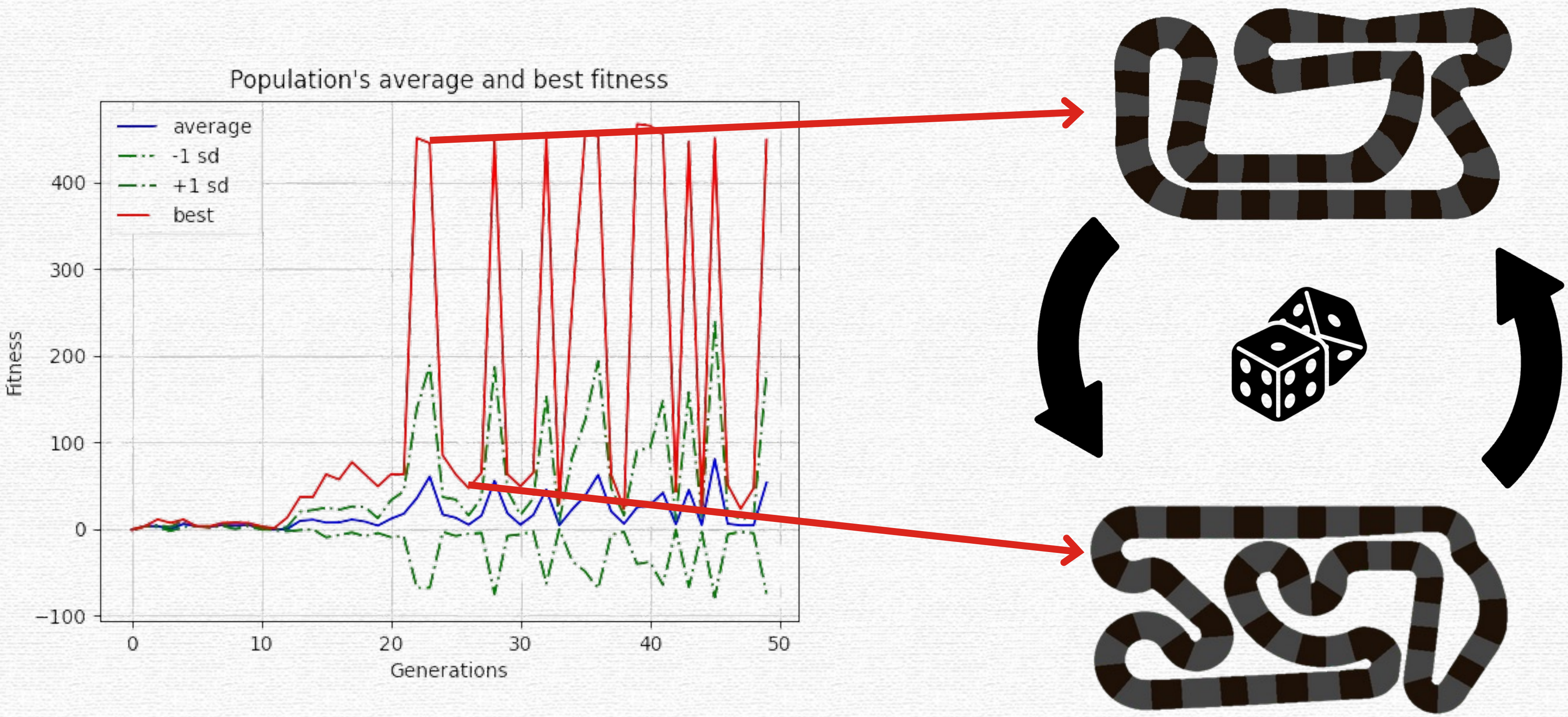
```python
import math
import pygame


class Car:

  def __init__(self, x, y, tracks):
    self.x = x
    self.y = y
    self.vel = 0
    self.angle = 90
    self.prev_angle = 90
    self.img = pygame.image.load("car.png")
    self.img.set_colorkey((0, 0, 0))
    self.front_x = self.x + 10 + 10 * math.sin(math.radians(self.angle))
    self.front_y = self.y + 15 + 15 * math.cos(math.radians(self.angle))
    self.tracks = tracks
    self.color = "BLACK"
    self.checkpoint_reward = 2
    self.stay = 0

  def normalize(self, value):
      if value < 0.5:
          return 2
      else:
          return 3

  def move(self):
    self.x = self.x + math.sin(math.radians(self.angle)) * self.vel
    self.y = self.y + math.cos(math.radians(self.angle)) * self.vel

  def accelerate(self, value):
    sgn = [i for i in [-1, 1] if value*math.fabs(value)*i >= 0][0]
    self.vel += sgn*self.normalize(math.fabs(value))/20
    if self.vel < 0.5:
        self.vel = 0.5
    if self.vel > 2:
        self.vel = 2

  def turn(self, value):
    sgn = [i for i in [-1, 1] if value*math.fabs(value)*i >= 0][0]
    self.prev_angle = self.angle
    self.angle += sgn*(self.normalize(math.fabs(value)) - self.vel)
    if self.angle < 0:
        self.angle += 360
    if self.angle > 360:
        self.angle -= 360
```

```python
  def draw(self, win):
      rotated_img = pygame.transform.rotate(self.img, self.angle)
      win.blit(rotated_img, (self.x - int(rotated_img.get_width()/2), self.y -
int(rotated_img.get_height()/2)))

  def get_ends(self):
    self.front_x = max(min(self.x + 10 * math.sin(math.radians(self.angle)), 599),0)
    self.back_x = max(min(self.x - 10 * math.sin(math.radians(self.angle)), 599),0)
    self.front_y = max(min(self.y + 15 * math.cos(math.radians(self.angle)), 299),0)
    self.back_y = max(min(self.y - 15 * math.cos(math.radians(self.angle)), 299),0)
    return self.front_x, self.back_x, self.front_y, self.back_y

  def calc_distance(self, angle, id):
    r, g, b, d = 0, 0, 0, 0
    while (r, g, b) != (109, 129, 98):  #pas rouge :')
      d += 1
      if self.front_x + d * math.sin(math.radians(
          self.angle + angle)) >= 599 or self.front_y + d * math.cos(
              math.radians(self.angle + angle)) >= 299:
        break

      if self.front_x + d * math.sin(math.radians(
          self.angle + angle)) < 0 or self.front_y + d * math.cos(
              math.radians(self.angle + angle)) < 0:
        break

      r, g, b, _ = self.tracks[id].getpixel(
          (self.front_x + d * math.sin(math.radians(self.angle + angle)),
           self.front_y + d * math.cos(math.radians(self.angle + angle))))

    return d

  def generate_inputs(self, id):
    inputs = []
    angles = [-45, -10, 0, 10, 45]
    for i in angles:
      d = self.calc_distance(i, id)
      inputs.append(d)
    return inputs

  def gain_points(self, id):
    r, g, b, _ = self.tracks[id].getpixel((self.front_x, self.front_y))

    if (r, g, b) == (69, 69, 69) and self.color == "BLACK":
      self.stay = 0
      self.color = "WHITE"
      return self.checkpoint_reward

    elif (r, g, b) == (29, 17, 8) and self.color == "WHITE":
      self.stay = 0
      self.color = "BLACK"
      return self.checkpoint_reward

    elif (r, g, b) == (217, 203, 162):
      self.stay = 0

    self.stay += 1
    return 0
```

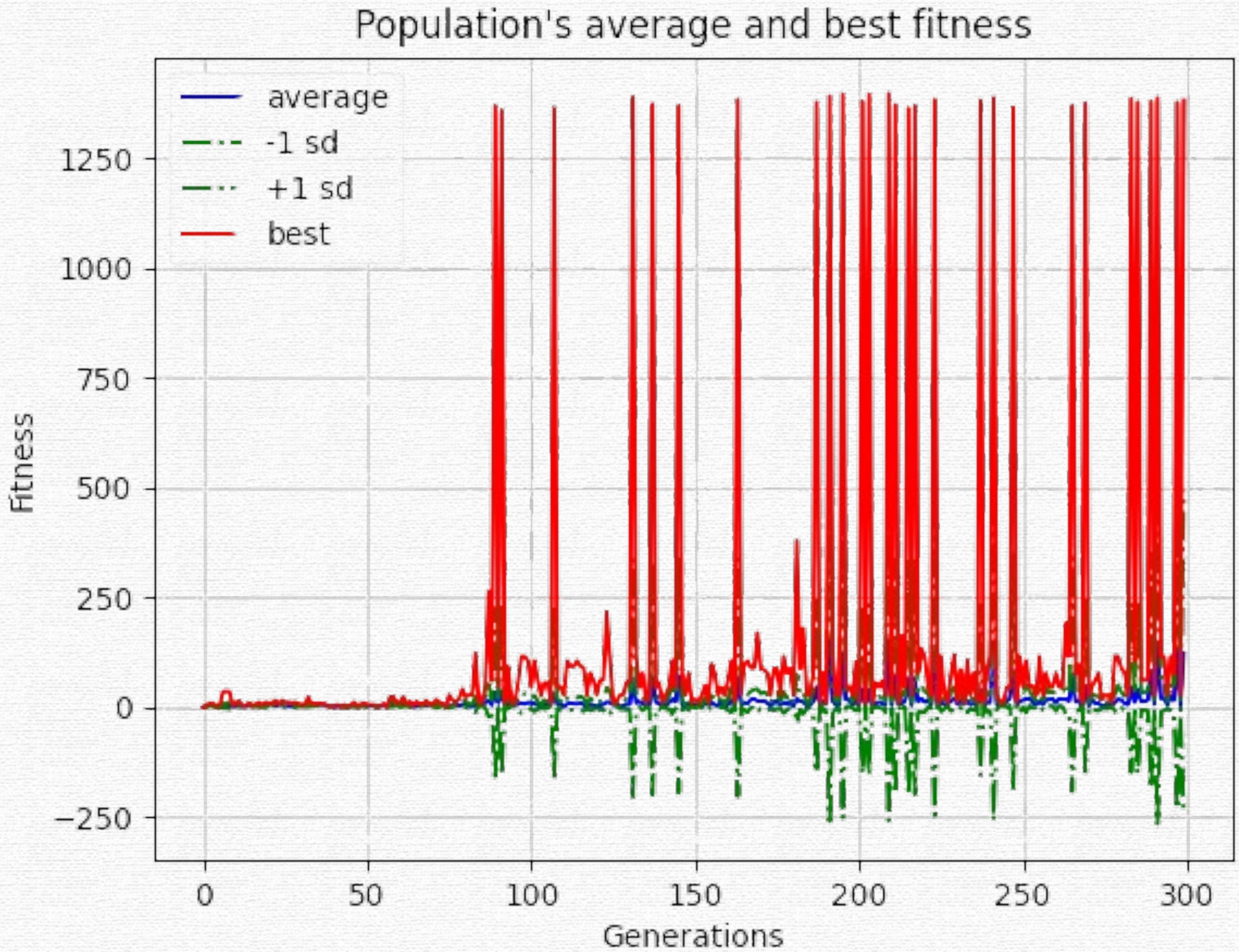

45°
-10°
10°
-45°
Accélération
Angle
2082
2444
0°
1841
2573



Population's average and best fitness
average
-1 sd
+1 sd
best
Fitness
Generations
−5
0
5
10
15
20
25
0
25
50
75
100
125
150
175
200



Population's average and best fitness
average
-1 sd
+1 sd
best
Fitness
Generations
400
300
200
100
0
−100
0
10
20
30
40
50


Population's average and best fitness
average
-1 sd
+1 sd
best
Fitness
1250
1000
750
500
250
0
−250
0
50
100
150
200
250
300
Generations

Paramètres génétiques

```
[NEAT]
fitness_criterion     = max
fitness_threshold     = 800
pop_size              = 40
reset_on_extinction   = False

[DefaultGenome]
# node activation options
activation_default      = tanh
activation_mutate_rate  = 0.0
activation_options      = tanh

# node aggregation options
aggregation_default     = sum
aggregation_mutate_rate = 0.0
aggregation_options     = sum

# node bias options
bias_init_mean          = 0.0
bias_init_stdev         = 1.0
bias_max_value          = 30.0
bias_min_value          = -30.0
bias_mutate_power       = 0.5
bias_mutate_rate        = 0.7
bias_replace_rate       = 0.1

# genome compatibility options
compatibility_disjoint_coefficient = 1.0
compatibility_weight_coefficient   = 0.5

# connection add/remove rates
conn_add_prob           = 0.5
conn_delete_prob        = 0.5

# connection enable options
enabled_default         = True
enabled_mutate_rate     = 0.01

feed_forward            = True
initial_connection      = partial_nodirect 0.8

# node add/remove rates
node_add_prob           = 0.2
node_delete_prob        = 0.2

# network parameters
num_hidden              = 3
num_inputs              = 6
num_outputs             = 2

# node response options
response_init_mean      = 1.0
response_init_stdev     = 0.0
response_max_value      = 30.0
response_min_value      = -30.0
response_mutate_power   = 0.0
response_mutate_rate    = 0.0
response_replace_rate   = 0.0

# connection weight options
weight_init_mean        = 0.0
weight_init_stdev       = 1.0
weight_max_value        = 30
weight_min_value        = -30
weight_mutate_power     = 0.5
weight_mutate_rate      = 0.8
weight_replace_rate     = 0.1

[DefaultSpeciesSet]
compatibility_threshold = 3.0

[DefaultStagnation]
species_fitness_func = max
max_stagnation       = 15
species_elitism      = 2

[DefaultReproduction]
elitism            = 2
survival_threshold = 0.2
```