Résumé et mots clés :

1) Résumé :

On veut montrer que nous pouvons nous inspirer de la nature pour créer une meilleure intelligence artificielle. On va démontrer cela en mettant en œuvre une intelligence artificielle distribuée fonctionnant grâce à l'algorithme des Boids qui créera un comportement émergent pour avoir des capacités de résolution de problèmes et de auto organisation. Des recherches scientifiques sur le domaine de l'intelligence artificielle ont commencé en 1956 et accélèrent rapidement de nos jours.

2) Mots clés :

Quelques mots clés à connaître sont :

- Intelligence artificielle : créer ou de simuler l'intelligence par une machine ou un programme
- Boids : un programme informatique de vie artificielle, développé par Craig W. Reynolds en 1986, qui simule le comportement d'une nuée d'oiseaux en vol. Ceci permet de modéliser un comportement émergent en appliquant trois règles simples : la cohésion, la séparation et l'alignement.
- algorithme émergent : une méthode de résolution de problème qui utilise un ensemble de règles simples pour faire émerger un comportement global plus complexe sans que ce dernier n'ait été explicitement détaillé.
- auto organisation : une organisation, un ordre ou une coordination se formant grâce à des interactions locales entre des plus petits éléments dans un locale, système, ou endroit originalement désorganisée. Le système s'organise lui même.
- algorithme génétique : un algorithme qui utilise la notion de sélection naturelle et l'applique à une population de solutions possibles pour obtenir une solution approchée à un problème d'optimisation.
- réseau neuronal : un modèle dans le domaine de l'intelligence artificielle qui s'inspire des neurones biologiques. Il est très utilisé pour tout qui concerne la reconnaissance de formes.
- neurone : en biologie un neurone est une cellule nerveuse qui est l'unité de base du système nerveux. Il est excitable et envoie un signal électrique et chimique lorsque il reçoit un stimulus.

Dans un réseau neuronal, c'est aussi l'unité de base qui décide d'envoyer un signal au fonction d'une information reçue et d'un poids interne.

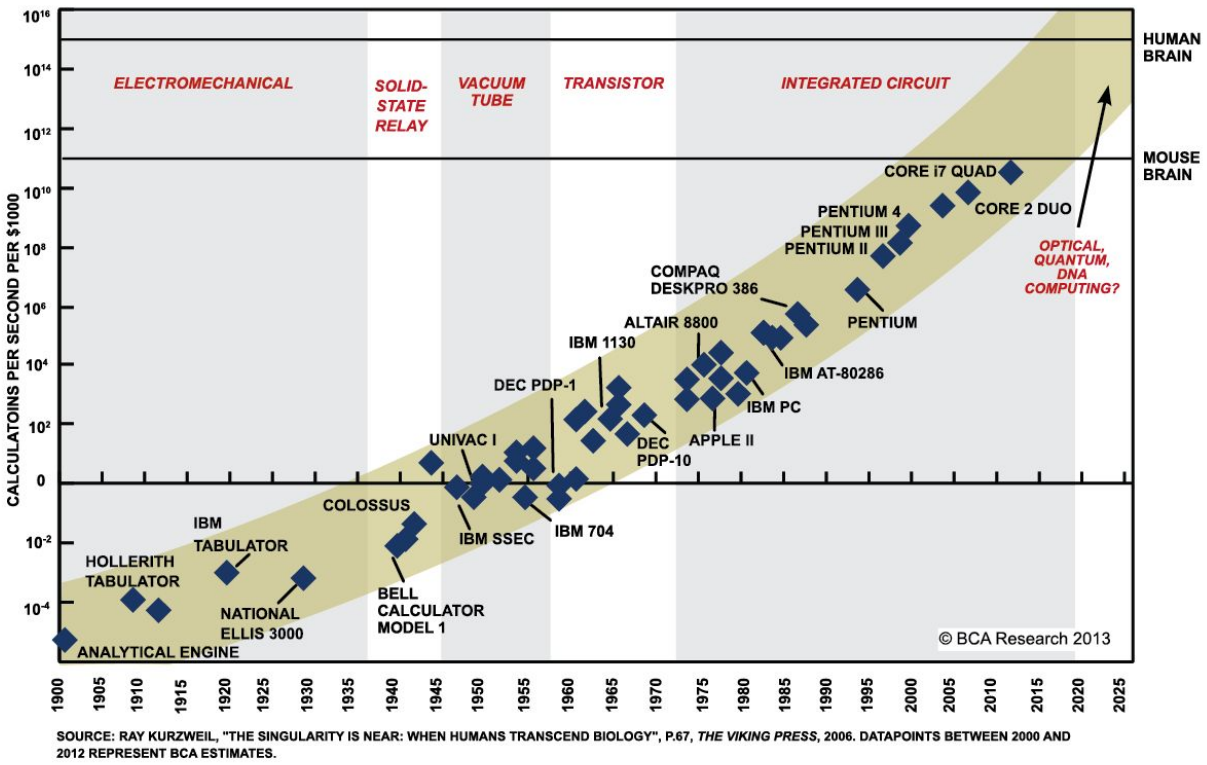
- agent : Une entité autonome qui analyse puis agit sur son environnement pour atteindre un but.
- intelligence distribuée : Un système composé de plusieurs agents. Ils peuvent travailler ensemble ou d'une manière indépendante mais essaye collectivement d'atteindre un but commun.

Table des matières

Introduction :

1) Contexte :

Le domaine de l'intelligence artificielle date depuis 1956 dont beaucoup de progrès ont été fait depuis les années 1990 grâce, principalement, à des grand progrès dans la puissance de calcul des ordinateurs, comme le témoigne le graphique ci dessous. Ce graphique nous montre que le nombre de calcul par seconde pour $1000 a augmenté de seulement 10^4 entre 1950 et 1990 mais entre 1990 et 2010, ce nombre a augmenté d'environ 10^7 calculs par seconde pour $1000. Cette augmentation peut être expliquée par la loi de Moore qui décrit que le nombre de transistors sur un processeur (et donc sa puissance) doublera tout les deux ans, or, la puissance de calcul augmentera d'une manière exponentielle.



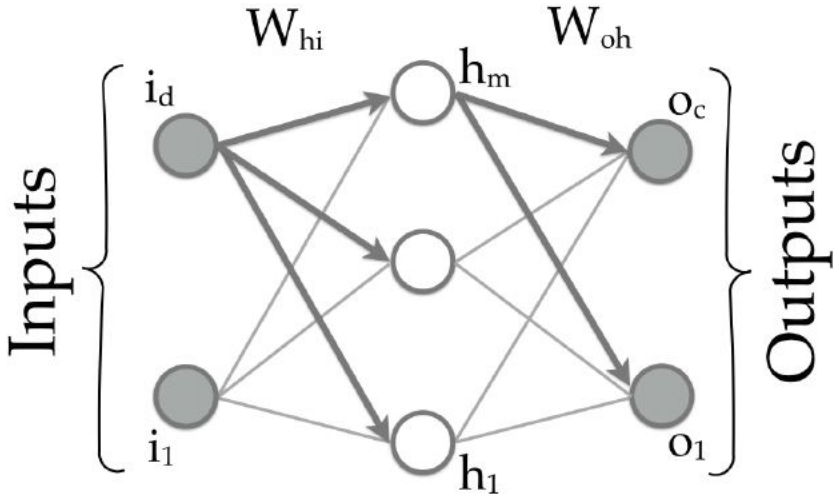
SOURCE: RAY KURZWEIL, "THE SINGULARITY IS NEAR: WHEN HUMANS TRANSCEND BIOLOGY", P.67, *THE VIKING PRESS*, 2006. DATAPOINTS BETWEEN 2000 AND 2012 REPRESENT BCA ESTIMATES.

Cette augmentation de puissance de calcul fait de l'intelligence artificielle un sujet d'actualité, car dorénavant il était simplement impossible de créer un système suffisamment complexe pour être considéré comme intelligent à cause du grand nombre de calculs nécessaires.

Même si l'intelligence artificielle est un domaine qui a pour longtemps intéressé les scientifiques et les ingénieurs, sa définition reste toujours disputée. En effet, même si l'idée simple de créer ou de simuler l'intelligence est largement acceptée, les caractéristiques de l'intelligence qu'un programme d'intelligence artificielle doit posséder varient. Ces caractéristiques sont généralement de pouvoir percevoir son environnement et d'être capable de prendre des décisions qui maximisent les chances d'un résultat favorable. Le débat tourne en particulier autour de si l'agent doit avoir une intelligence dite humaine qui fonctionne donc d'une manière similaire (avec un agent ou des agents rationnels). Pour notre étude, nous allons dire que l'intelligence artificielle ne doit pas forcement fonctionner d'une manière humaine. Actuellement, des systèmes capables de résoudre des problèmes d'une manière autonome existent, mais un système entièrement intelligent et conscient de lui même n'existe pas.

La mise en oeuvre de l'intelligence artificielle peut utiliser plusieurs méthodes. En effet, ces méthodes s'inspirent des structures et des mécanismes dans la nature pour la simple raison que la nature à développée pendant des millions d'années des techniques et des structures d'auto organisation et d'auto optimisation. Ces techniques et structures prisent ou inspirées de la nature permettent au programme d'utiliser des données pour opérer d'une manière autonome qui maximisent les chances de succès et même de se optimiser pour devenir plus efficace. Cette auto optimisation permet en effet au système ou à l'agent de prendre en compte les succès et les échecs, ce qui est l'équivalent "d'apprendre".

Une des techniques la plus couramment utilisée et qui se repend de jour en jour est le réseau neuronal. Celui ci est inspirée des neurones contenus dans le cerveau. Il fonctionne grâce à de différents neurones qui échangent de l'information entre eux.

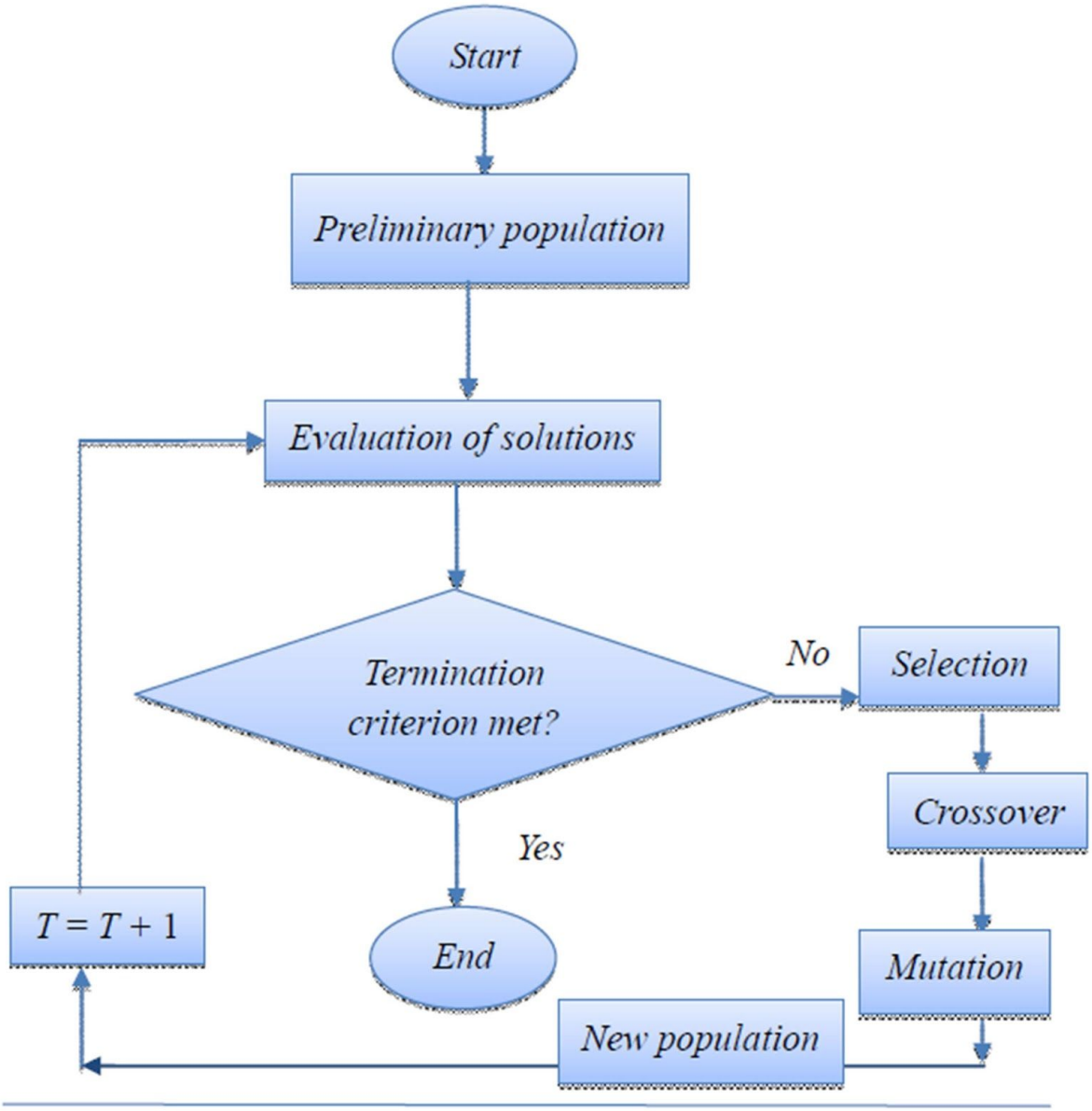


Comme nous le voyons dans le schéma, il y a trois couches de neurones différentes : il y a la couche des neurones “input” qui reçoivent de l’information de l'extérieur, la couche des “hidden neurones” et la couche des neurones outputs qui donnent le résultat final. Les neurones Inputs reçoivent l’information extérieure. Ensuite ils transmettent l’information aux neurones dites “Hidden” qui traitent l’information et la transmettent au prochain neurone. Dans le schéma, les flèches de différentes largeurs représentent vers quel neurone l’information continue et le “poids” de cette information. Le poids de l’information est sa signifiance. Si un neurone reçoit deux informations de poids différents, il va habituellement choisir celle avec le plus grand poids. Il peut aussi faire la moyenne des deux poids et choisir cette option. La façon dont le poids des connections est choisie est assez simple. Le programmeur donne aux programme une série d’informations et la réponse attendue. Le programme change le poids des connections jusqu'à ce que la réponse du programme est identique à celle donnée. Le poids est pratique car il permet aux neurones de prendre une décision face à deux informations reçues. Le poids des neurones est trouvé grâce à de nombreuses équations qui adaptent le poids selon les expériences passées du réseau. Un output et de l’information sont pre-designées par l’utilisateur et le système. Quand les neurones “hidden” terminent de traiter l’information, ils les envoient aux neurones “output”. Ces neurones “output” traitent l’information une dernière fois et choisissent la réponse ou décision choisie par le programme. Un exemple de ce programme est Alpha Go, un robot crée par Google. Il joue à Go, un jeu traditionnel Chinois, ou il faut capturer le plus de territoire possible. Il apprends en scannant de nombreux matches de Go et en jouant contre des autres joueurs. Alpha Go est le premier ordinateur a battre un joueur professionnel de Go et est le premier joueur mondial actuel de Go.

Une autre technique est l’algorithme génétique. Il s’appuie sur la sélection naturelle pour obtenir une solution approchée à un problème d'optimisation, c’est à dire qu’il cherche la meilleure solution pour des problèmes qui n’ont pas une solution exacte. L’algorithme génétique fonctionne en créant une population initiale dont les individus sont des candidats pour la solution idéale. Chaque individu possède des caractères qui représentent les chromosomes. Ces individus de la population initiale sont générés d’une manière aléatoire. Ensuite, “l’adaptation” au problème de chaque individu, c’est à dire le succès que chaque individu face à la résolution du problème est utilisée pour calculer une note pour chaque individu. On utilise ces notes pour faire un classement des individus au sein de la population. Ceux avec les meilleures notes sont sélectionnés. Ils sont ceux qui vont pouvoir se reproduire et transmettre leurs gènes. Cette transmission de gènes a lieu par le croisement des

chromosomes des individus sélectionnés. Puis, une faible proportion de cette population subit des mutations et la nouvelle population est crée. Ceci se répète comme montre l'algorigramme ci dessous jusqu'à qu'il y aie une solution avec une notes satisfaisante ou après un nombre de génération fixe. Comme "l'adaptabilité" des individus peut être calculée rapidement sans qu'un individu soit capable de reproduire il y a beaucoup plus de générations qui se créent beaucoup plus rapidement que dans la nature, ce qui fait que l'algorithme génétique s'appuie sur et accélère le processus d'évolution. L'algorithme génétique est couramment utilise pour plusieurs choses. Un exemple de la mise en oeuvre d'un algorithme génétique est l'antenne du navire spatial ST5 de NASA. La forme de l'antenne a été optimise pour avoir la meilleure signal. Par contre, l'algorithme génétique a aussi des limites, principalement le temps de calcul (qui peut être long) et le fait qu'il est impossible de déterminer si la

solution trouvée est la meilleure.



Une autre technique utilisée est l'intelligence distribuée. L'intelligence distribuée fonctionne grâce a des agents autonomes qui agissent selon des règles simples. Les interactions entre les agents est un algorithme émergent qui permet des actions globales complexes et imprévisibles. C'est les structures globales et complexes qui permettent la résolution de problèmes. Ce système fait souvent preuve d'auto organisation. En effet, l'intelligence artificielle distribuée a une différence majeure par rapport aux autres techniques, elle est décentralisée. Normalement, L'intelligence artificielle repose sur une entité centrale qui rassemble les informations, les traite et prend des décisions. Ici, il n'y a pas

d’agents qui réunit toutes les informations reçues. Chaque agent reçoit, traite et agit au fonction des information qu’il peut “observer”.

Les systèmes d’intelligences distribuées s’inspirent de plusieurs systèmes dans la nature, en particulier avec les insectes (les fourmis et les termites qui sont autonomes et travaillent ensembles pour créer des structures massives et impressionnantes comme les fourmilières et les termitières qui misent à l'échelle humaine font 1 400 m en hauteur), mais aussi les oiseaux et poissons (la nuée d’oiseau et le banc pour les poissons).

2) Hypothèse :

Nous hypothéquons qu'il est possible de s'inspirer de la nature pour créer une meilleure intelligence artificielle.

Nous pensons aussi que nous pouvons mettre en place une intelligence artificielle distribuée fonctionnant grâce à l’algorithme des Boids qui serait capable de s'auto organiser et de résoudre des problèmes.

3) Objectif de travail :

Le but de l’étude est d'explorer le domaine de l'intelligence artificielle, en particulier son rapport aux structures et systèmes présents dans la nature et d'utiliser ces connaissances pour créer un programme d'intelligence artificielle.

Matériel et méthodes :

1) Matériel :

Pour notre expérience, il nous faut :

- un ordinateur
- beaucoup de caféine, des boissons énergétiques et des écouteurs (pour programmer)
- python 2.7x (un langage de programmation)
- pygame (une bibliothèque pour les graphiques sous python).
- Le programme que nous avons écrit en python qui existe sous deux versions :
    - Une version relativement simple

- Une version plus complexe avec des obstacles et un but.

2) Le code de notre projet :

Pour notre expérience, nous avons décidés de mettre en place une intelligence artificielle distribuée. Nous avons écrit deux versions. La version simple fonctionne mais n'a pas des obstacles pas de but, ce que nous avons mis dans la version complète (qui est disponible sur https://github.com/FakeNameSE/Boids-with-obstacles-and-goals.git). Le code pour la version simplifiée est ici :

```
#!/usr/bin/env python
# Boid implementation in Python using PyGame
from __future__ import division # required in Python 2.7
import sys
import pygame
import random
import math
# === constants ===
SCREEN_SIZE = SCREEN_WIDTH, SCREEN_HEIGHT = 1350, 710
BLACK = (0, 0, 0)
RED   = (255, 0, 0)
MAX_BOID_VELOCITY = 8
NUM_BOIDS = 55
BORDER = 30
# === weights ===
COHESION_WEIGHT = 100
ALIGNMENT_WEIGHT = 40
SEPERATION_WEIGHT = 5
OBSTACLE_AVOIDANCE_WEIGHT = 10
GOAL_WEIGHT = 100
# === classes ===

class Boid(pygame.sprite.Sprite):
    def __init__(self, x, y):
        super(Boid, self).__init__()
        # Load image as sprite
        self.image = pygame.image.load("ressources/img/boid.png").convert()
        # Fetch the rectangle object that has the dimensions of the image
```

```
        self.rect = self.image.get_rect()
        # Coordinates
        self.rect.x = x
        self.rect.y = y
        self.velocityX = random.randint(1, 10) / 10.0
        self.velocityY = random.randint(1, 10) / 10.0

    def distance(self, boid):
        '''Return the distance from another boid'''
        distX = self.rect.x - boid.rect.x
        distY = self.rect.y - boid.rect.y
        return math.sqrt(distX * distX + distY * distY)

    def cohesion(self, boid_list):
        '''Move closer to a set of boid_list'''
        if len(boid_list) < 1:
                return
        # calculate the average distances from the other boid_list
        avgX = 0
        avgY = 0
        for boid in boid_list:
                if boid.rect.x == self.rect.x and boid.rect.y == self.rect.y:
                        continue
                avgX += (self.rect.x - boid.rect.x)
                avgY += (self.rect.y - boid.rect.y)
        avgX /= len(boid_list)
        avgY /= len(boid_list)
        # set our velocity towards the others
        distance = math.sqrt((avgX * avgX) + (avgY * avgY)) * -1.0
        self.velocityX -= (avgX / COHESION_WEIGHT)
        self.velocityY -= (avgY / COHESION_WEIGHT)

    def alignment(self, boid_list):
        '''Move with a set of boid_list'''
        if len(boid_list) < 1:
                return
        # calculate the average velocities of the other boid_list
        avgX = 0
        avgY = 0
        for boid in boid_list:
                avgX += boid.velocityX
```

```
            avgY += boid.velocityY
        avgX /= len(boid_list)
        avgY /= len(boid_list)
        # set our velocity towards the others
        self.velocityX += (avgX / ALIGNMENT_WEIGHT)
        self.velocityY += (avgY / ALIGNMENT_WEIGHT)

    def seperation(self, boid_list, minDistance):
        '''Move away from a set of boid_list. This avoids crowding'''
        if len(boid_list) < 1:
            return
        distanceX = 0
        distanceY = 0
        numClose = 0
        for boid in boid_list:
            distance = self.distance(boid)
            if  distance < minDistance:
                numClose += 1
                xdiff = (self.rect.x - boid.rect.x)
                ydiff = (self.rect.y - boid.rect.y)
                if xdiff >= 0:
                    xdiff = math.sqrt(minDistance) - xdiff
                elif xdiff < 0:
                    xdiff = -math.sqrt(minDistance) - xdiff
                if ydiff >= 0:
                    ydiff = math.sqrt(minDistance) - ydiff
                elif ydiff < 0:
                    ydiff = -math.sqrt(minDistance) - ydiff
                distanceX += xdiff
                distanceY += ydiff
        if numClose == 0:
            return
        self.velocityX -= distanceX / SEPERATION_WEIGHT
        self.velocityY -= distanceY / SEPERATION_WEIGHT

    def update(self):
        '''Perform actual movement based on our velocity'''
        # ensure they stay within the screen space
        # if we roubound we can lose some of our velocity
        if self.rect.x < BORDER and self.velocityX < 0:
            self.velocityX = -self.velocityX * random.random()
```

```
        if self.rect.x > SCREEN_WIDTH - BORDER and self.velocityX > 0:
                self.velocityX = -self.velocityX * random.random()
        if self.rect.y < BORDER and self.velocityY < 0:
                self.velocityY = -self.velocityY * random.random()
        if self.rect.y > SCREEN_HEIGHT - BORDER and self.velocityY > 0:
                self.velocityY = -self.velocityY * random.random()
        # Obey speed limit
        if abs(self.velocityX) > MAX_BOID_VELOCITY or abs(self.velocityY) > MAX_BOID_VELOCITY:
                scaleFactor = MAX_BOID_VELOCITY / max(abs(self.velocityX), abs(self.velocityY))
                self.velocityX *= scaleFactor #illuminati confirmed
                self.velocityY *= scaleFactor
        self.rect.x += self.velocityX
        self.rect.y += self.velocityY

# === main === (lower_case names)
# --- init ---
pygame.init()
screen = pygame.display.set_mode(SCREEN_SIZE)
# Set the title of the window
pygame.display.set_caption('Boids')

# --- objects ---
# lists
boid_list = pygame.sprite.Group()
# This is a list of every sprite.
all_sprites_list = pygame.sprite.Group()
# --- create boids and obstacles at random positions on the screen ---
# Place boids
for i in range(NUM_BOIDS):
    boid = Boid(random.randint(0, SCREEN_WIDTH), random.randint(0, SCREEN_HEIGHT))
    # Add the boid to the lists of objects
    boid_list.add(boid)
    all_sprites_list.add(boid)

# --- mainloop ---

clock = pygame.time.Clock()
running = True
while running:
    # --- events ---
```

```
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
                running = False
        elif event.type == pygame.KEYDOWN:
                if event.key == pygame.K_ESCAPE:
                        running = False
    # --- updates ---
    # Scan for boids and obstacles to pay attention to
    for boid in boid_list:
        closeboid = []
        for otherboid in boid_list:
                if otherboid == boid:
                        continue
                distance = boid.distance(otherboid)
                if distance < 200:
                        closeboid.append(otherboid)
        # Apply the rules of the boids
        boid.cohesion(closeboid)
        boid.alignment(closeboid)
        boid.seperation(closeboid, 20)
        boid.update()

    # --- draws ---

    # Background colour
    screen.fill(BLACK)
    # Draw all the spites
    all_sprites_list.draw(screen)
    # Go ahead and update the screen with what we've drawn.
    pygame.display.flip()
    pygame.time.delay(10)
    # Used to manage how fast the screen updates
    clock.tick(120)

# --- the end ---
pygame.quit()
sys.exit()
```

Ce programme fonctionne en utilisant l'algorithme Boids. Nous avons donc utilisés l'intelligence distribuée. L'algorithme Boids fonctionne en créant des agents qui simulent des poissons

ou des oiseaux. Comme ces animaux, les agents (qui dans notre programme sont des particules) sont relativement “stupide” car ils ne possèdent que quelques règles pour leurs comportements. Dans l’algorithme Boids, il n’y a que trois règles fondamentales :

1. Cohésion : les agents essayent de se diriger vers les autres agents.
2. Alignement : les agents essayent de se diriger dans la même direction approximative que les autres.
3. Séparation : les agents évitent de se serrer trop et de se percuter.

Ces règles sont appliquées en calculant pour chacun un vecteur de mouvement pour l’agent puis d’ajouter ces vecteurs. Le vecteur final est le mouvement de l’agent. Pour rendre les agents plus réalistes, leur vitesse est limitée à 8 pixels par cycle de calcul et ils ont un champ de vision limitée à 200 pixels (il n’y a pas d'échelle). Par contre, chaque règle a un poids attribué. Les vecteurs sont divises par leurs poids respectifs. De ce fait, la séparation est plus important que la cohésion.

Nous avons aussi écrit une version plus complexe. Ce programme fonctionne de la même manière, mais un but (atteindre la souris) et des obstacles (à éviter). Les agents ont donc deux règles de plus :

1. Éviter les obstacles : chaque agent calcul le vecteur vers l’obstacle le plus près et dévie du vecteur opposé par 10%.
2. Atteindre un but : chaque agent calcul le vecteur vers le but et dévie vers ce vecteur par 1%.

3) Méthodes :

1. D’abord, nous avons fait tourner le système simple qui montre le comportement complexe obtenu grâce à l'intelligence artificielle distribuée.
2. Ensuite, peu à peu, nous avons enlevé de différentes caractéristiques du programme pour voir si le comportement complexe global des agents est lié aux interactions entre les trois règles des agents :
    a. D’abord, nous avons commencé par enlever la règle “cohésion”,
    b. Ensuite, nous avons enlevé la règle “alignement”
    c. Finalement, nous avons enlevé la règle “séparation”.

3. Finalement, nous avons fait tourner la version complète du programme (qui contient des obstacles et un but). Ceci nous permet d'observer si notre système d'intelligence artificielle est capable de résoudre des problèmes.

Résultats :

En observant les simulations (lien : https://youtu.be/xTjGHyzoNjM) nous pouvons voir plusieurs choses. Dans le premier cas avec les trois règles, on observe que les agents créent un comportement global complexe et émergent. En effet, ils montrent l'auto organisation en créant des agrégations automatiquement qui se sont ensuite réunies pour former une agrégation d'agents. Ces agrégations se déplacent ensemble dans une direction aléatoire et dans une forme organisée, sans se percuter entre eux. Par contre, lorsque l'on enlève la règle de cohésion, les agents ne font pas d'agrégations et sont la majorité du temps seuls. Lorsque l'on enlève la règle d'alignement, ils s'agrègent mais ils ne sont pas organisés et ils sortent du groupe de façon spontanée. Lorsque l'on enlève la règle de séparation, les agents s'agrègent et se superposent donnant l'impression d'avoir seulement un agent. Finalement, lorsque nous faisons tourner le programme final, les agents se regroupent parfaitement, sans se superposer ou quitter le groupe, ils essayent d'atteindre le but (la souris) et ils trouvent des façons pour pouvoir contourner les obstacles. Parfois, ils se divisent même pour contourner des obstacles, ce qui montre le comportement complexe et émergent crée grâce aux interactions des différents agents et de leurs règles simples.

Nous avons trouvé qu'il est possible de s'inspirer de la nature pour créer une meilleure intelligence artificielle. Nous avons vu cela grâce à nos recherches sur les techniques d'intelligence artificielle et grâce à notre expérience. En effet, les systèmes que nous avons étudié se basent tous sur des structures inspirées de la nature.

En s'inspirant du comportement des oiseaux et des poissons, nous avons vu que c'est l'interaction des règles simples qui permet des comportements complexes émergent. En effet, notre système est un système d'intelligence artificielle distribuée avec des agents qui utilisent l'algorithme Boids. Ceci les permets d'éviter des obstacles et d'atteindre un but ensembles. Il est donc possible d'écrire un programme qui utilise l'intelligence artificielle distribuée et fonctionnant grâce à l'algorithme des Boids qui serait capable de s'auto organiser et de résoudre des problèmes.

Conclusion et discussion

D’autres questions qui ont émergées lors de nos études sont :

1. Comment nos recherches et nos programmes sont-ils applicables dans une situation réelle.
2. Comment ceci peut agrandir notre compréhension de systèmes biologiques complexes.
3. Comment peut-on combiner le programme avec d'autres techniques (algorithmes génétiques) pour améliorer le programme.
4. Quels autres structures naturelles pouvons-nous nous inspirer de pour créer une meilleure intelligence artificielle.

Remerciement

Premièrement, nous voulons remercier les musiciens des années 1980 aux années 1990's (en particulier MC Hammer) pour l'aide mentale que nous avions eu besoin dans le développement de notre projet. Nous voulons aussi remercier Conrad Parker et Craig Reynolds pour leur travail dans le domaine de l'intelligence artificielle et en particulier pour leur travail sur l''algorithme Boids, Jordan Grossman pour son aide avec l'affiche, M. Millet car sans lui il n'y aurait pas de groupe et finalement Al Gore pour avoir inventé l'internet.

Bibliographie

"Agrégation (comportement)." *Wikipedia*. Wikimedia Foundation, n.d. Web. 02 Apr. 2016.

"Artificial Intelligence." *Wikipedia*. Wikimedia Foundation, n.d. Web. 02 Mar. 2016.

"Artificial Life (Alife)." N.p., n.d. Web. 5 Mar. 2016.

"Artificial Neural Network." *Wikipedia*. Wikimedia Foundation, n.d. Web. 20 Mar. 2016.

"Association for the Advancement of Artificial Intelligence." *Association for the Advancement of Artificial Intelligence*. N.p., n.d. Web. 02 Apr. 2016.

"Distributed Intelligence." *TheFreeDictionary.com*. N.p., n.d. Web. 02 Apr. 2016.

"Moore's Law." *Moores Law*. N.p., n.d. Web. 25 Feb. 2015.

"Moore's Law." N.p., n.d. Web. 02 Apr. 2016.

"Neural Networks and Deep Learning." *Neural Networks and Deep Learning*. N.p., n.d. Web. 02 Apr. 2016.

N.p., n.d. Web. 25 Feb. 2015.

Parker, Conrad. "Boids Pseudocode." *Boids Pseudocode*. N.p., n.d. Web. 02 Dec. 2015.