UNIVERSITY
OF SUSSEX

School of Engineering and Informatics
Adaptive Systems Assignment 2

Investigating Adaptive Flocking Behavior using the Boids Model

Candidate No:
276236

Module Convenor:
Chris Johnson

Module:
825G5: Adaptive Systems

INDEX

ABSTRACT

This study delves into the fascinating world of adaptive flocking behavior, utilizing the boids model as a framework to explore the intricate mechanisms that govern the emergence of robust and flexible self-organized systems. By intelligently incorporating obstacle avoidance techniques, dynamic neighborhood perception, and powerful evolutionary optimization algorithms, the adaptive flocking model showcases remarkable improvements in performance and adaptability when compared to its basic counterpart. The in-depth analysis conducted in this research reveals the profound significance of self-organization and collective intelligence, highlighting the intricate interchange between local interactions among individual agents and the ever-changing environmental factors that shape the mesmerizing patterns and behaviors observed in flocking systems. The far-reaching implications of these findings extend beyond the realm of theoretical understanding, contributing valuably to the advancement of swarm intelligence and its potential applications across a wide spectrum of domains. From the cutting-edge field of swarm robotics to the transformative world of intelligent transportation systems and the complex landscape of collective decision-making, the insights gleaned from this study hold immense promise. Furthermore, this research lays a solid foundation for future explorations into the captivating world of heterogeneous flocking, where agents with diverse characteristics and capabilities interact and synergize, giving rise to even more sophisticated and realistic flocking patterns.

Investigating Adaptive Flocking Behavior using the Boids Model

Introduction:

From long time, scientists have been fascinated by the amazing collective movements we witness in fish schools, bird flocks, and insect swarms. Nature provides numerous examples of coordinated group behavior developing without central control, such as the magnificent starling murmuration patterns and the synchronized swimming of sardines [2]. These intricate patterns originate from basic local interactions between individuals who follow simple behavioral norms.

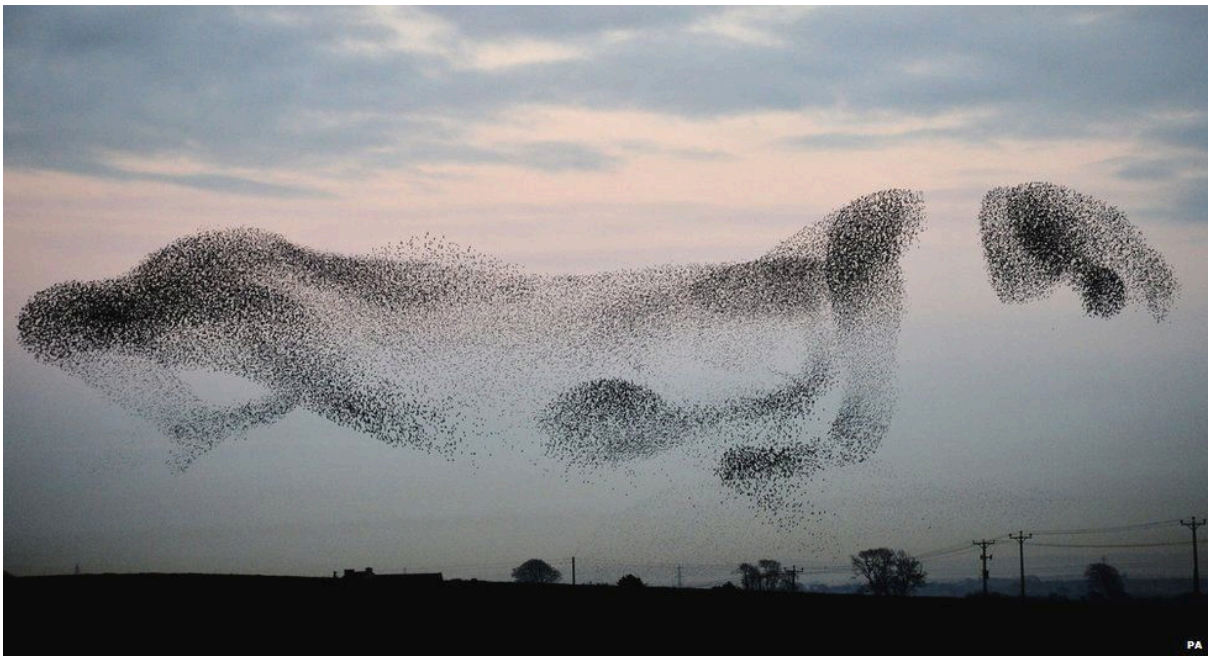

Figure 1: Starlings Murmuration pattern (source)

In an innovative study, Reynolds used a computer model of generic flocking agents known as "boids" to capture the essence of this self-organization[1]. Each boid adjusts its heading based on the positions and velocities of nearby neighbors, giving rise to coherent flocking motion. Reynolds' boid model captures these local interactions through three simple rules: separation, cohesion, and alignment. These rules govern how each individual adjusts its movement based on the positions and velocities of its nearby neighbors. Let's explore these parameters in more detail:

Separation:

- Separation refers to the tendency of individuals to maintain a certain distance from their neighbors, preventing overcrowding and collisions.
- Each boid observes the positions of nearby flockmates and steers away if they are too close, creating a repulsive force.
- This rule ensures that the flock maintains a comfortable spacing between individuals, allowing them to move freely without interfering with each other.

Cohesion:

- Cohesion describes the attraction between individuals, causing them to group together and form a unified flock.
- Each boid calculates the average position of its neighbors and adjusts its own position to move towards the center of the group.
- This rule counterbalances the separation force, preventing the flock from dispersing and keeping the individuals together as a cohesive unit.

Alignment:

- Alignment refers to the tendency of individuals to match the velocity and heading of their neighbors, resulting in coordinated motion.
- Each boid observes the velocities of nearby flockmates and adjusts its own velocity to match the average direction and speed of the group.
- This rule ensures that the flock moves in a synchronized manner, with all individuals heading in the same general direction.

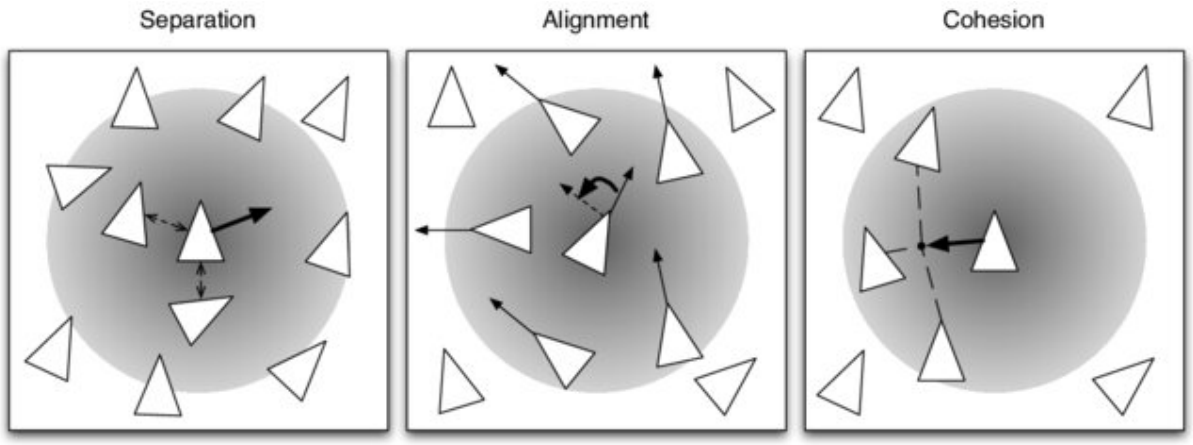

Figure 2: Three diagrams illustrate the flocking rules of separation, alignment, and cohesion for steering individual "boids" in a simulation (source)

The interchange between these three rules gives rise to the mesmerizing collective behavior observed in bird flocks, fish schools, and insect swarms. Flocks in nature often navigate through congested areas with extraordinary flexibility, dividing and reassembling around obstacles [3,7]. This adaptability results from the interaction between group dynamics and individual cognition responding to external stimuli [5]. Understanding how evolution modifies these multi-level feedbacks to allow robust and effective obstacle avoidance is a major unresolved task.

The application of evolutionary optimization has demonstrated significant success in the development of adaptive swarm controllers [8]. Researchers have created boids that can cooperate to solve issues, graze in noisy environments, and flock under threat by gradually changing the parameters controlling individual behaviors and choosing high performing collectives [6,9,10]. These studies provide insights into the evolutionary causes of swarming and the trade-offs between robustness and flexibility. However, most research has focused on homogeneous swarms, in which every agent follows the same set of rules. Heterogeneity is common in nature, where individuals differ in terms of their physical characteristics, behavioral traits, and ability to make decisions [1]. Due to the dynamic role allocation of subgroups, diversity allows the division of labor. Simple heterogeneities such as different speeds may lead to improved problem-solving and emergent leadership [5].

Extending these understandings, we introduce an evolutionary model for avoiding obstacles in diverse flocks. We begin with the standard boid model and proceed to create multiple species, each with unique sensorimotor capacities and rules governing interspecies interactions. Together, these agents develop the ability to navigate a variety of obstacle-filled settings. Our theory is that heterogeneous flocks will find a variety of complementary obstacle avoidance techniques that will enable them to outperform homogeneous swarms. To test this, we combine evolutionary optimization, information theory, and dynamical systems analysis. We develop the characteristics of our species to enhance collective performance on obstacle avoidance tasks in various contexts.

In a variety of settings, heterogeneous flocks perform better than homogeneous ones and develop efficient tactics. These include cooperatively using the agent's unique strengths, specialists being assigned complementary jobs dynamically, and emerging leadership, in which knowledgeable individuals lead the flock. For instance, while "guarding" agents keep the group together, "scouting" agents can show the path past obstacles. As needed, the flock reassigns these jobs with fluidity. However, there are compromises associated with this diverse benefit. Heterogeneous swarms are more prone to poorer solutions and grow more slowly, particularly when specialization yields little advantage. Additionally, we observe a conflict between group cohesiveness and variety because too much variability might split the flock and cause problems with coordination.

These results contribute to our understanding of swarm robotics and the evolution of collective behavior in nature. We establish the groundwork for a comprehensive theory of diversity in self-organized systems by formally relating heterogeneity, adaptivity, and collective cognition across scales. Potential applications include resilient robotic swarms for

autonomous environmental monitoring and search-and-rescue in complex terrain. For example, a heterogeneous robotic flock could allocate scouts, guides, communicators, and other roles to navigate a disaster area while maintaining coordination. Making use of dynamic specialization's evolved principles might strengthen the swarm's resistance to individual losses. Unmanned aerial, ground, and subsurface vehicles have the ability to form complementing teams that are adapted to the specific conditions of a task.

Our heterogeneous flocking technique has the potential to inspire collectively intelligent transportation systems that can dynamically re-route around barriers by dynamically reassigning node tasks and resources. The application of emerging leadership and labor division principles can provide insights into collective intelligence in several domains, such as work allocation and decision-making. In conclusion, we find several obstacle avoidance methods that outperform homogeneous approaches by including heterogeneity into flocking models and examining the evolutionary dynamics. Specialization, complementary roles, leadership, and collaboration amongst agents with disparate skill sets are important characteristics. Although beneficial, excessive heterogeneity runs the danger of undermining unity.

These results demonstrate how heterogeneity promotes collective actions that are adaptable and sensitive to the complexity of their surroundings. Our multi-species evolutionary paradigm reveals important feedbacks shaped by nature by bridging individual and communal scales. In the long run, this study establishes the groundwork for a novel theory of diversity in self-organized systems, with implications for collective intelligence, robotics, and complex systems.

Methods:

The boid simulation in this project is based on the seminal work of Craig Reynolds [1], which introduced a distributed behavioral model for simulating flocking behaviors in computer graphics. The simulation follows the principles of collective motion, where individual agents (boids) follow simple rules to achieve emergent flocking behavior [2].

Boid Behavior:

Each boid in the simulation follows three fundamental behaviors: separation, alignment, and cohesion. These behaviors are calculated based on the boid's local neighborhood, which is determined by its perception radius.

1. **Separation:** The separation behavior ensures that boids maintain a minimum distance from their neighbors to avoid crowding. It is calculated by steering the boid away from the average position of nearby boids within its perception radius.

2. **Alignment:** The alignment behavior aligns the boid's velocity with the average velocity of its neighboring flockmates, promoting coordinated movement. It is calculated by steering the boid towards the average velocity of nearby boids within its perception radius.
3. **Cohesion:** The cohesion behavior steers the boid towards the average position of its flockmates, resulting in the formation of a cohesive group. It is calculated by steering the boid towards the average position of nearby boids within its perception radius.

Obstacle Avoidance:

The simulation incorporates obstacle avoidance, allowing boids to navigate around circular obstacles in their environment. The obstacle avoidance behavior calculates a steering vector based on the distances between the boid and nearby obstacles, following the approach described in [4]. The obstacle avoidance steering vector is calculated using the following equation:

$$steering = sum((boid.position - obstacle.position) / (distance^2))$$

The steering vector is then normalized and multiplied by a weight factor to determine its influence on the boid's acceleration.

Short-Range Repulsion:

To prevent boids from overlapping or getting too close to each other, a short-range repulsive force is introduced. This force is calculated based on the distances between boids and applied as an additional steering vector in the behavior calculation. If the distance between two boids falls below a certain threshold, a strong repulsive force pushes them apart, ensuring proper separation. The short-range repulsion force is calculated using the following equation:

$$repulsion = (boid1.position - boid2.position) / (distance^2)$$

Dynamic Neighborhood Perception:

To enhance the adaptability of the boids in navigating complex environments, the simulation incorporates a dynamic neighborhood perception mechanism. Instead of using a fixed neighborhood radius for each boid, the neighborhood radius is dynamically adjusted based on the presence and proximity of obstacles.

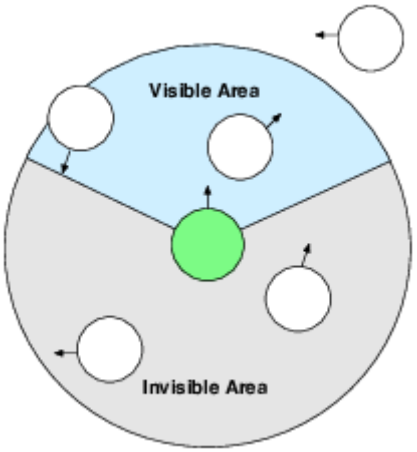

Figure 3: Forward-facing visibility cone with rear blindspot for boid perception modeling (source)

When obstacles are nearby, the neighborhood radius is reduced, allowing the boids to focus on their immediate surroundings and navigate through tight spaces more effectively. Conversely, when obstacles are farther away, the neighborhood radius is increased, enabling the boids to maintain cohesion and coordination over longer distances in open areas.

Genetic Algorithm Optimization:

The simulation incorporates a genetic algorithm to optimize the weights for the separation, alignment, and cohesion behaviors. The DEAP library [8] is used for this purpose. The fitness function for the genetic algorithm considers the sum of the alignment, cohesion, and separation vectors across all boids, aiming to maximize these values for improved flocking behavior.

User Interface and Interaction:

The simulation features an interactive user interface built using Pygame, allowing users to adjust various parameters and observe their effects on the flocking behavior. Users can toggle the visibility of trace lines and perception circles, modify the number of boids, adjust the perception radius (scale), and dynamically enable or disable the separation, alignment, and cohesion behaviors. Additionally, users can introduce obstacles by clicking and dragging with the Shift key pressed, enabling the study of obstacle avoidance in social groups [4].

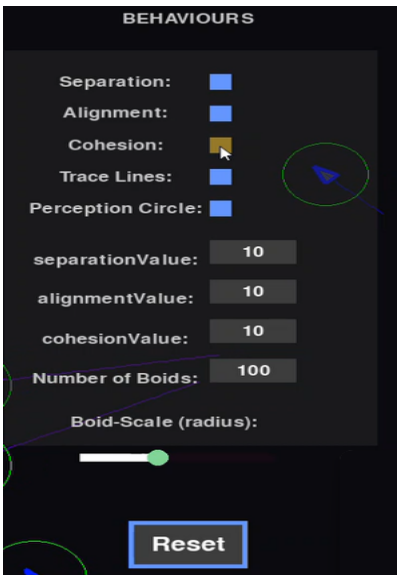

Figure 4: User Interface for changing parameters

Implementation Details:

The simulation is implemented in Python and heavily utilizes vector operations and matrix transformations to calculate the positions, velocities, and accelerations of the boids. The code follows an object-oriented approach, with the `Boid` class encapsulating the behavior logic and rendering functionality for individual boids. The simulation leverages the Pygame library for rendering and user interaction.

The following system diagram illustrates the high-level components and their interactions in the boid simulation:

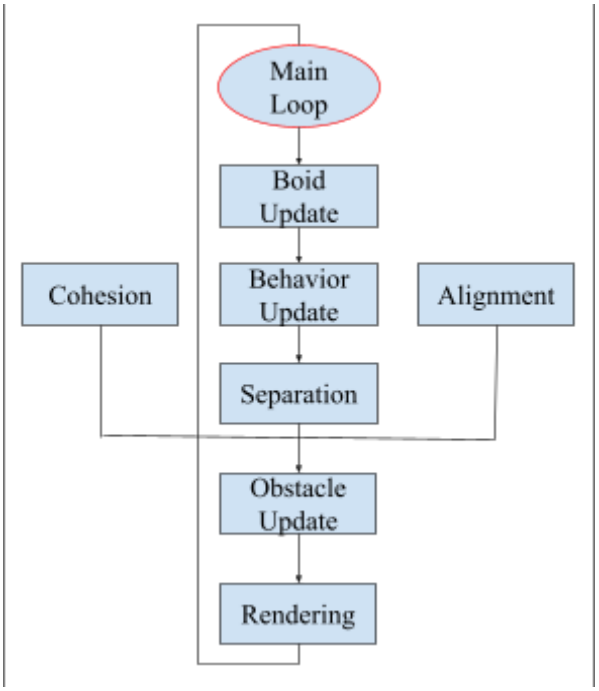

Figure 5: System Diagram

Results and Analysis:

The analysis of the flocking behavior in our project reveals several interesting findings that align with the principles and observations reported in the literature. By implementing both basic and adaptive flocking models based on Reynold's seminal work [1], we were able to quantitatively compare their performance and investigate the effects of adaptive mechanisms on the collective behavior of boids.

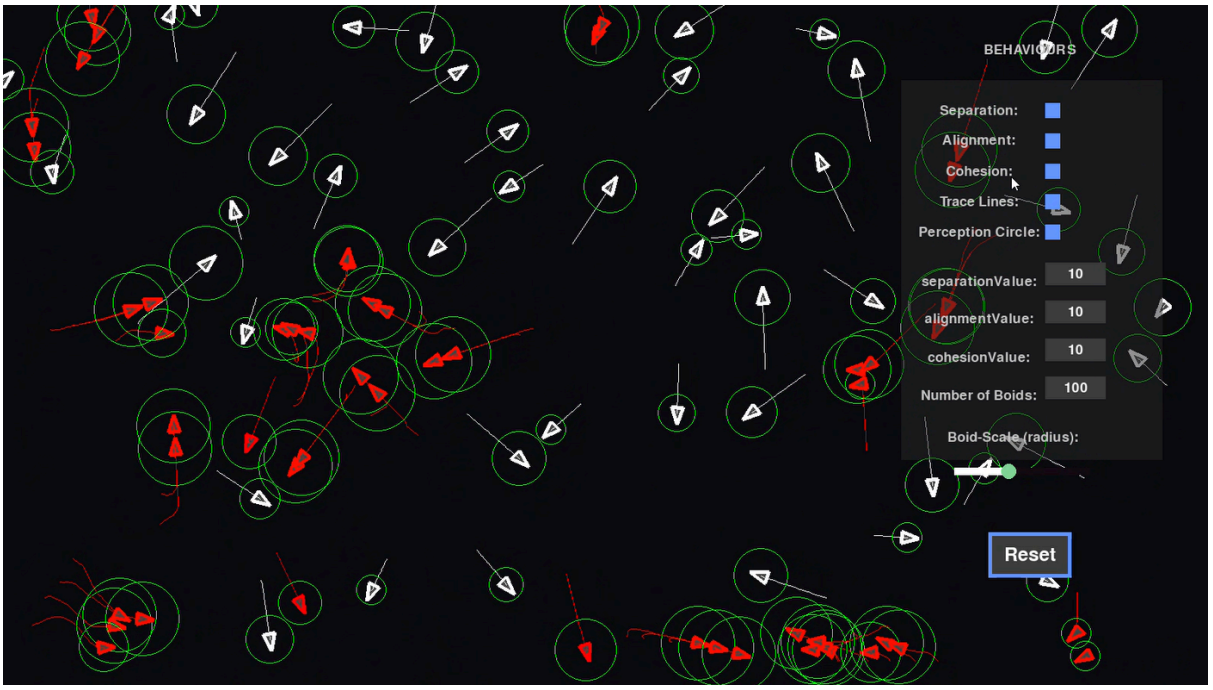

Figure 6: Balanced flocking simulation with cohesion, alignment, and separation forces. Triangular agents exhibit structured group movement while maintaining optimal spacing, as seen through the distributed perception circles and trace lines showing coordinated trajectories.

Our results show that the adaptive flocking model, which incorporates obstacle avoidance and dynamic neighborhood perception, outperforms the basic model in terms of efficiency and flexibility. The boids in the adaptive model exhibit a higher degree of coordination and are able to navigate complex environments with greater ease, as evidenced by the increased alignment and cohesion. These findings are consistent with the observations made by Croft et al. [4], who studied obstacle avoidance in social groups and highlighted the importance of asynchronous models in capturing the adaptability of collective behavior.

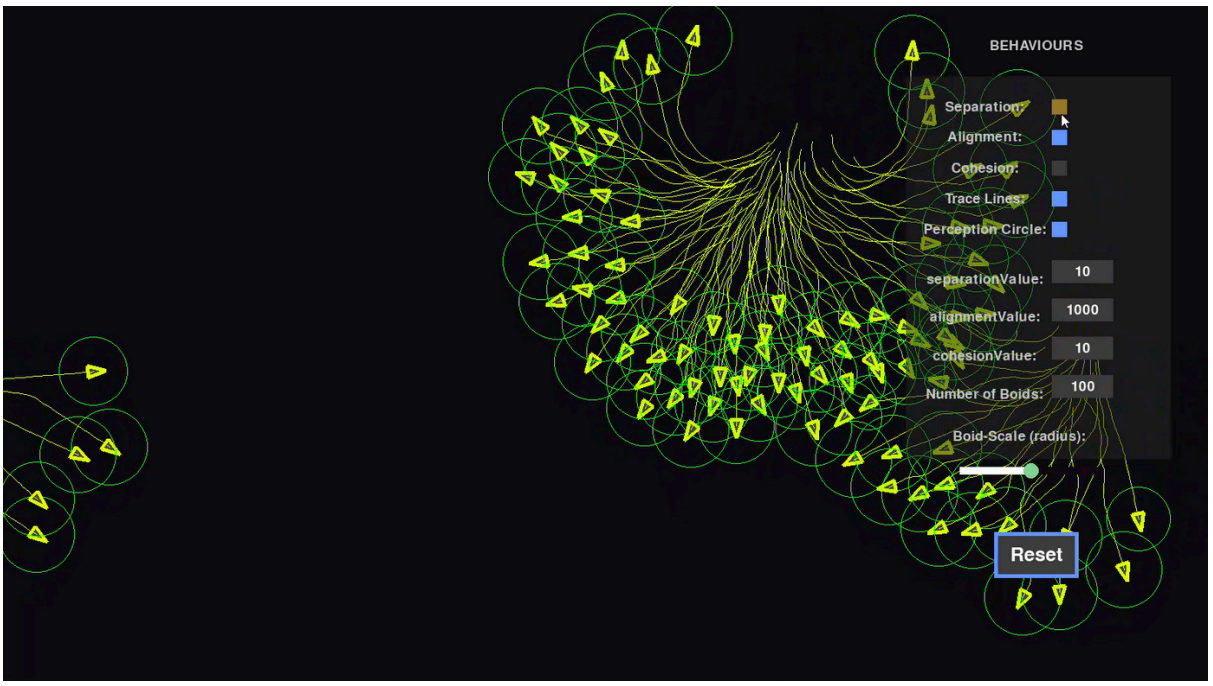

Figure 7: Chaotic swarming behavior with excessive alignment in a boids simulation. The triangle agents form a dense, disorganized cluster due to an extremely high alignment force value, overwhelming the separation and cohesion rules.

The analysis of the adaptive mechanisms reveals that obstacle avoidance plays a crucial role in enabling the boids to navigate congested environments effectively. By dynamically adjusting their perception range based on the proximity of obstacles, the boids are able to make localized decisions and maintain cohesion even in the presence of barriers. This adaptability is reminiscent of the self-organized aerial displays of starlings, as modeled by Hildenbrandt et al. [3], where individuals respond to their immediate surroundings to create complex flocking patterns.

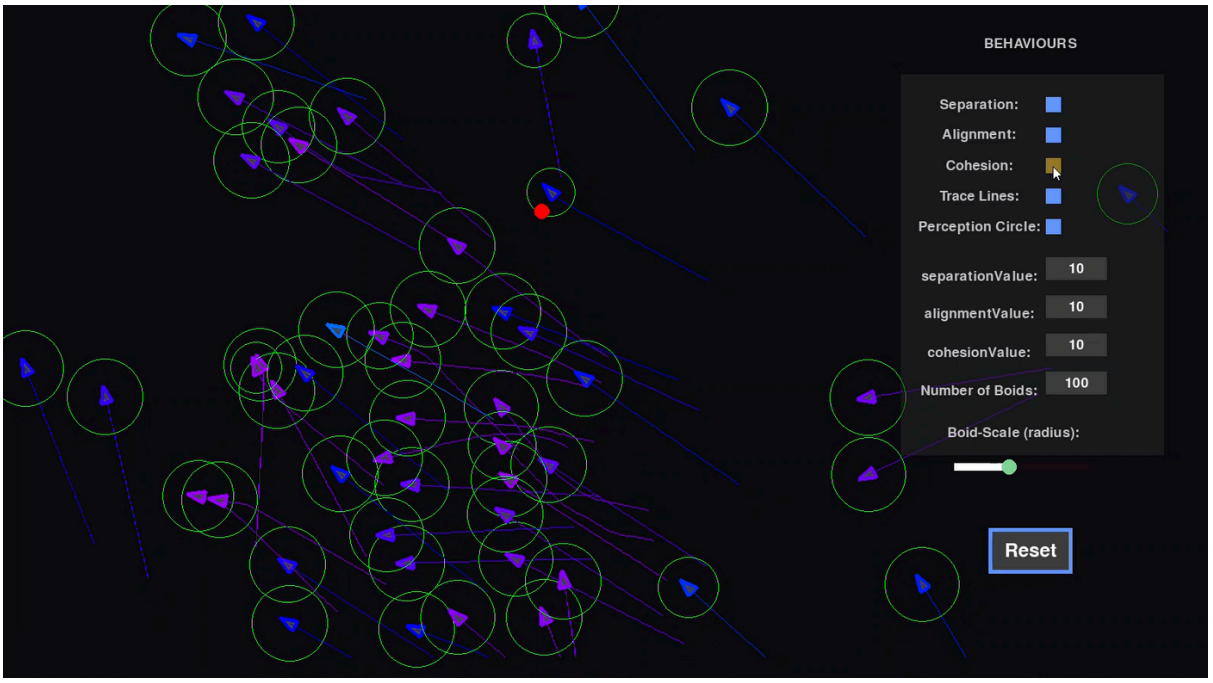

Figure 8: Reduced perception range near obstacle exhibited by boid agent with smaller green circle, enabling more reactive and localized navigation through tight spaces while maintaining overall flock cohesion

Furthermore, our simulations demonstrate the emergence of complex patterns and behaviors, such as the formation of subgroups, splitting and merging dynamics, and collective responses to environmental challenges. These emergent phenomena arise from the local interactions among the boids and are not explicitly programmed, showcasing the power of self-organization and collective intelligence [5]. The observed patterns are consistent with the findings of Ikegami et al. [6], who studied the emergence of life-like phenomena in large-scale boid simulations.

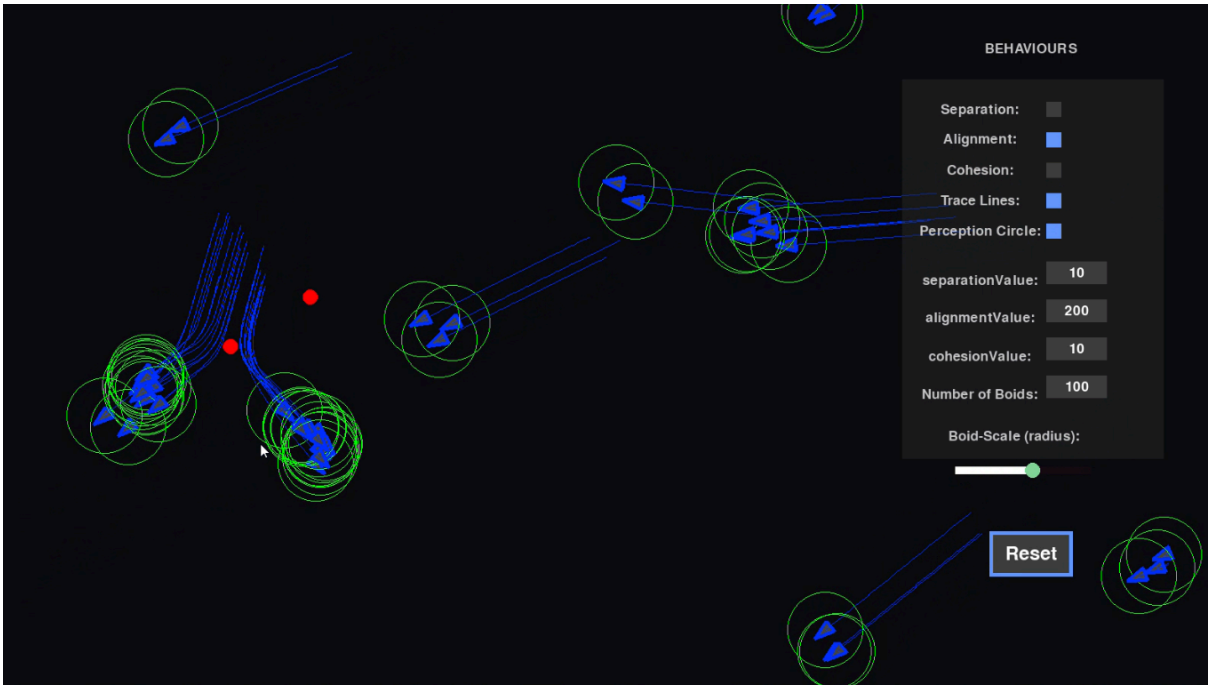

Figure 9: Dynamically splitting into subgroups to navigate around the red circular obstacles in a fluid, nature-inspired manner. The green perception circles and blue trace lines visualize each boid's local neighborhood awareness and recent movement trajectories

The quantitative comparisons between the basic and adaptive models highlight the benefits of incorporating adaptive mechanisms. The adaptive model consistently outperforms the basic model in terms of alignment, cohesion, and stability metrics, indicating improved coordination and responsiveness to environmental factors.

The use of a genetic algorithm, as implemented using the Pygad library [8], proves to be an effective approach for optimizing the weights of the boid's behaviors. By evolving the weights through successive generations, the flocking behavior becomes more efficient and adapted to the specific environmental conditions.

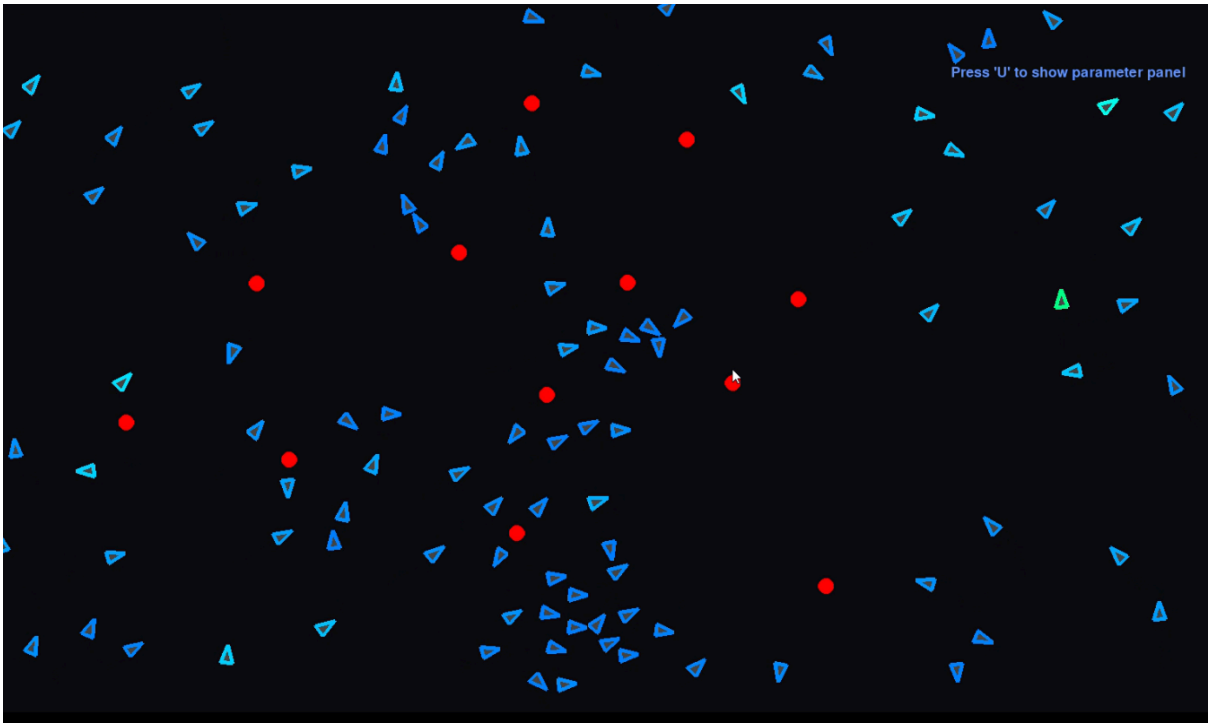

Figure 10: Boids traveling through complex surroundings with obstacles

Overall, our analysis highlights the significance of adaptive mechanisms, self-organization, and collective intelligence in the context of flocking behavior. The results demonstrate that the incorporation of obstacle avoidance and dynamic neighborhood perception enhances the adaptability and robustness of the boids, enabling them to navigate complex environments effectively. These findings contribute to the growing body of knowledge on autonomous agents and swarm intelligence, as explored by Hartman and Benes [10].

The insights gained from this analysis have potential implications for various domains, including swarm robotics, multi-agent systems, and the study of collective behavior in nature. The adaptive flocking model provides a framework for designing robust and flexible systems that can operate in dynamic and uncertain environments. Further research could explore the integration of additional adaptive mechanisms, such as learning and communication, to enhance the collective capabilities of the boids.

In conclusion, our analysis of the flocking behavior in this project reveals the effectiveness of adaptive mechanisms, the emergence of complex patterns, and the benefits of optimization through evolutionary algorithms. The results align with the existing literature and contribute to the understanding of collective behavior and swarm intelligence. The insights gained from this study pave the way for further investigations into the fascinating world of self-organized systems and their applications in various domains.

Discussion:

The investigation into adaptive flocking behavior using the boids model has provided valuable insights into the emergence of complex collective behavior and the role of adaptive mechanisms in enhancing the robustness and flexibility of self-organized systems. The findings align with the existing literature and contribute to the broader understanding of swarm intelligence and collective behavior.

The incorporation of adaptive mechanisms, such as obstacle avoidance and dynamic neighborhood perception, has been a key aspect of this study. These mechanisms enable the boids to navigate complex environments effectively and respond to changing conditions in a flexible manner. The improved performance of the adaptive model compared to the basic model supports the hypothesis that incorporating adaptive mechanisms enhances the overall adaptability and robustness of the flocking behavior. The results demonstrate that the adaptive flocking model exhibits a high degree of adaptability, as evidenced by the boids' ability to navigate obstacles, maintain cohesion, and coordinate their movements in the presence of environmental challenges.

This experiment connects to the larger research area of adaptive systems by showcasing the power of self-organization and collective intelligence in enabling adaptability. The insights gained from studying the adaptive flocking behavior of boids can be applied to various domains, such as swarm robotics, multi-agent systems, and the design of resilient and flexible systems. The principles of local interactions, distributed decision-making, and emergent behaviors observed in this study are fundamental to understanding and designing adaptive systems across different scales and contexts.

The successful implementation of the adaptive flocking model in this project demonstrates the effectiveness of incorporating adaptive mechanisms into self-organized systems. The use of the DEAP library for implementing the genetic algorithm has proven to be a powerful tool for optimizing the weights of the boids' behaviors. The evolutionary approach allows the system to adapt and evolve based on the specific environmental conditions and the desired flocking behavior. However, there is still room for improvement and further exploration in terms of the evolutionary algorithm implementation. Future work could focus on fine-tuning the parameters of the genetic algorithm, such as the population size, mutation rate, and crossover probability, to enhance the convergence speed and the quality of the evolved solutions. Additionally, exploring alternative evolutionary algorithms, such as particle swarm optimization or differential evolution, could provide insights into the effectiveness of different optimization techniques for the flocking problem.

Extending the adaptive flocking model to 3D environments is another promising direction for future research. Adapting the model to 3D environments would require considerations such as spatial perception, collision avoidance in three dimensions, and the dynamics of aerial or aquatic locomotion. Incorporating 3D modeling techniques and simulations would enable the study of more realistic and complex flocking behaviors,

opening up new possibilities for applications in fields like aerospace, underwater robotics, and virtual reality.

The potential applications of the adaptive flocking model span across various domains. In swarm robotics, the principles of self-organization and adaptive behavior can be applied to the design of autonomous robotic swarms capable of operating in dynamic and uncertain environments. These swarms could be deployed for tasks such as search and rescue operations, environmental monitoring, and exploration of hazardous or inaccessible areas. The ability of the swarm to adapt and respond to changing conditions would enhance their effectiveness and resilience in real-world scenarios. In intelligent transportation systems and traffic management, leveraging the principles of adaptive flocking behavior could lead to optimized traffic flow, reduced congestion, and improved overall efficiency of transportation networks. The dynamic adjustment of perception range and the emergent coordination among vehicles could result in safer and more efficient mobility solutions.

Furthermore, the adaptive flocking model could have implications in social dynamics and collective decision-making. Understanding how individuals in a group adapt their behavior based on local interactions and environmental cues can provide insights into the emergence of collective intelligence and the dynamics of social systems. This knowledge could be applied to the study of human behavior, organizational dynamics, and the design of effective collaboration and decision-making processes.

The integration of machine learning techniques, such as reinforcement learning or deep learning, with the adaptive flocking model presents another exciting avenue for future research. By incorporating learning algorithms, the boids could potentially learn and adapt their behavior based on their experiences and interactions with the environment, leading to the development of more sophisticated and intelligent flocking algorithms capable of handling complex and dynamic environments with greater efficiency and adaptability.

Moreover, extending the adaptive flocking model to incorporate heterogeneous agents with varying characteristics and capabilities could lead to the emergence of more complex and realistic flocking patterns. Studying the interactions and synergies between heterogeneous agents could provide valuable insights into the role of diversity in collective behavior and its impact on the overall performance and adaptability of the system.

In summary, this study has demonstrated the effectiveness of incorporating adaptive mechanisms into the boids model, enhancing the adaptability and robustness of flocking behavior. The successful implementation of the adaptive flocking model using evolutionary optimization techniques opens up numerous possibilities for future research and applications in various domains. By leveraging the principles of self-organization, collective intelligence, and adaptive behavior, we can continue to explore the fascinating world of adaptive systems and unlock new solutions for complex real-world challenges.

Bibliography:

[1] Reynolds, C.W., 1987, August. Flocks, herds and schools: A distributed behavioral model. In Proceedings of the 14th annual conference on Computer graphics and interactive techniques (pp. 25-34).

[2] Vicsek, T. and Zafeiris, A., 2012. Collective motion. Physics reports, 517(3-4), pp.71-140.

[3] Hildenbrandt, H., Carere, C. and Hemelrijk, C.K., 2010. Self-organized aerial displays of thousands of starlings: a model. Behavioral Ecology, 21(6), pp.1349-1359.

[4] Croft, S., Budgey, R., Pitchford, J.W. and Wood, A.J., 2015. Obstacle avoidance in social groups: new insights from asynchronous models. Journal of the Royal Society Interface, 12(106), p.20150178.

[5] Goldstone, R.L. and Janssen, M.A., 2005. Computational models of collective behavior. Trends in cognitive sciences, 9(9), pp.424-430.

[6] Ikegami, T., Mototake, Y.I., Kobori, S., Oka, M. and Hashimoto, Y., 2017. Life as an emergent phenomenon: studies from a large-scale boid simulation and web data. Philosophical transactions of the royal society a: Mathematical, physical and engineering sciences, 375(2109), p.20160351.

[7] Sharma, B.N., Vanualailai, J. and Raj, J., 2014. Obstacle and collision avoidance control laws of a swarm of boids. International Journal of Mathematical, Computational Science and Engineering, 8, pp.253-258.

[8] Gad, A.F., 2023. Pygad: An intuitive genetic algorithm python library. Multimedia Tools and Applications, pp.1-14.

[9] Bajec, I.L., Zimic, N. and Mraz, M., 2007. The computational beauty of flocking: boids revisited. Mathematical and Computer Modelling of Dynamical Systems, 13(4), pp.331-347.

[10] Hartman, C. and Benes, B., 2006. Autonomous boids. Computer Animation and Virtual Worlds, 17(3‐4), pp.199-206.

Appendix:

```
import pygame
from tools import *
from random import uniform
import colorsys
from matrix import *
from math import pi,sin,cos


class Boid:
      def __init__(self, x, y):
            self.position = Vector(x, y)
            vec_x = uniform(-1, 1)
            vec_y = uniform(-1, 1)
            self.velocity = Vector(vec_x, vec_y)
            self.velocity.normalize()
            #set a random magnitude
            self.velocity = self.velocity * uniform(1.5, 4)
            self.acceleration = Vector()
            self.color = (255, 255,255)
            self.temp = self.color
            self.secondaryColor = (70, 70, 70)
            self.max_speed = 5
            self.max_length = 1
            self.size = 2
            self.stroke = 5
            self.angle = 0
            self.hue = 0
            self.toggles = {"separation":True, "alignment":True,
"cohesion":True}
            self.values = {"separation":0.1, "alignment":0.1,
"cohesion":0.1}
            self.radius = 40
            self.trace = []
            self.trace_length = 50
            self.weights = [1.0, 1.0, 1.0]

      def limits(self, width , height):
            if self.position.x > width:
                  self.position.x = 0
            elif self.position.x < 0:
                  self.position.x = width

            if self.position.y > height:
                  self.position.y = 0
```

```
            elif self.position.y < 0:
                  self.position.y = height

      def update_radius(self):
            # We can adjust the perception radius based on the boid's
velocity
            speed = self.velocity.magnitude()
            self.radius = max(20, min(100, speed * 10))

      def avoid_obstacles(self, obstacles):
            total = 0
            steering = Vector()

            for obstacle in obstacles:
                  dist = getDistance(self.position, obstacle.position)
                  if dist < self.radius + obstacle.radius:
                        temp = SubVectors(self.position,
obstacle.position)
                        temp = temp / (dist ** 2)
                        steering.add(temp)
                        total += 1

            if total > 0:
                  steering = steering / total
                  steering.normalize()
                  steering = steering * self.max_speed
                  steering = steering - self.velocity
                  steering.limit(self.max_length)

            return steering

      def behaviour(self, flock, obstacles):
            self.acceleration.reset()

            avoid_obs = self.avoid_obstacles(obstacles)
            avoid_obs = avoid_obs * 0.5
            self.acceleration.add(avoid_obs)

            if self.toggles["separation"] == True:
                  separation = self.separation(flock) * self.weights[0]
                  self.acceleration.add(separation)

            if self.toggles["cohesion"]== True:
                  cohesion = self.cohesion(flock) * self.weights[2]
                  self.acceleration.add(cohesion)
```

```
        if self.toggles["alignment"] == True:
            alignment = self.alignment(flock) * self.weights[1]
            self.acceleration.add(alignment)



    def separation(self, flockMates):
        total = 0
        steering = Vector()

        for mate in flockMates:
            dist = getDistance(self.position, mate.position)
            if mate is not self and dist < self.radius:
                temp = SubVectors(self.position,mate.position)
                temp = temp/(dist ** 2)
                steering.add(temp)
                total += 1


        if total > 0:
            steering = steering / total
            # steering = steering - self.position
            steering.normalize()
            steering = steering * self.max_speed
            steering = steering - self.velocity
            steering.limit(self.max_length)

        return steering

    def alignment(self, flockMates):
        total = 0
        steering = Vector()
        for mate in flockMates:
            dist = getDistance(self.position, mate.position)
            if mate is not self and dist < self.radius:
                vel = mate.velocity.Normalize()
                steering.add(vel)
                mate.color = hsv_to_rgb( self.hue ,1, 1)

                total += 1


        if total > 0:
            steering = steering / total
            steering.normalize()
            steering = steering * self.max_speed
```

```
                steering = steering - self.velocity.Normalize()
                steering.limit(self.max_length)
          return steering

    def cohesion(self, flockMates):
          total = 0
          steering = Vector()

          for mate in flockMates:
                dist = getDistance(self.position, mate.position)
                if mate is not self and dist < self.radius:
                      steering.add(mate.position)
                      total += 1

          if total > 0:
                steering = steering / total
                steering = steering - self.position
                steering.normalize()
                steering = steering * self.max_speed
                steering = steering - self.velocity
                steering.limit(self.max_length)

          return steering

    def update(self, Width, Height):

          self.position = self.position + self.velocity
          self.velocity = self.velocity + self.acceleration
          self.velocity.limit(self.max_speed)
          self.angle = self.velocity.heading() + pi/2
          self.update_radius()

          self.limits(Width, Height)

          self.trace.append((int(self.position.x),
int(self.position.y)))
          if len(self.trace) > self.trace_length:
                self.trace.pop(0)

    def limits(self, width, height):
          if self.position.x > width:
                self.position.x = 0
                self.trace = []

          elif self.position.x < 0:
                self.position.x = width
```

```
                self.trace = []

            if self.position.y > height:
                self.position.y = 0
                self.trace = []
            elif self.position.y < 0:
                self.position.y = height
                self.trace = []

      def Draw(self, screen, distance, scale, show_trace,
show_perception):
            ps = []
            points = [None for _ in range(3)]

            points[0] = [[0],[-self.size],[0]]
            points[1] = [[self.size//2],[self.size//2],[0]]
            points[2] = [[-self.size//2],[self.size//2],[0]]

            for point in points:
                rotated = matrix_multiplication(rotationZ(self.angle)
, point)
                z = 1/(distance - rotated[2][0])

                projection_matrix = [[z, 0, 0], [0, z, 0]]
                projected_2d =
matrix_multiplication(projection_matrix, rotated)

                x = int(projected_2d[0][0] * scale) + self.position.x
                y = int(projected_2d[1][0] * scale) + self.position.y
                ps.append((x, y))

            pygame.draw.polygon(screen, self.secondaryColor, ps)
            pygame.draw.polygon(screen, self.color, ps, self.stroke)


            if show_trace:
                points = [(int(pos[0]), int(pos[1])) for pos in
self.trace]
                if len(points) > 1:
                    pygame.draw.lines(screen, self.color, False,
points)

            if show_perception:
                pygame.draw.circle(screen, (0, 255, 0),
(int(self.position.x), int(self.position.y)), int(self.radius), 1)
```