CSC 484 - Building Game AI: Homework 2 Writeup

Introduction

The purpose of this assignment was to explore and implement various steering behaviors for AI movement, including Velocity Matching, Arrive & Align, Wander, and Flocking. These algorithms play a critical role in creating realistic and dynamic AI behavior in games. The primary goal of this project was to evaluate the effectiveness of these algorithms and observe how parameter tuning affects their behavior. By experimenting with different methods and parameters, I aimed to understand the strengths and weaknesses of each behavior and their impact on AI movement.

Implementation and Analysis

Part 1: Variable Matching Steering Behaviors

Variable Matching Steering Behaviors are designed to allow an AI-controlled character (boid) to match different kinematic properties of a target, including position, orientation, velocity, and rotation. In this part of the assignment, I implemented a library containing four distinct variable matching behaviors: Position Matching, Orientation Matching, Velocity Matching, and Rotation Matching. These behaviors are crucial for creating cohesive and realistic AI movement in games, particularly in scenarios where agents need to move together smoothly, such as in swarms or formations.

This section was specifically intended to demonstrate the Velocity Matching behavior, where the character matches the velocity of a target. The character is designed to match the velocity of the mouse pointer, making it move in the same direction and at a similar speed. This behavior allows agents to synchronize their speed and direction, which is useful for maintaining group formations or ensuring that units travel together in a coordinated manner.

While implementing Velocity Matching, I encountered some challenges in ensuring that the character smoothly matched the target's velocity. For example, when the **timeToTarget** parameter was set too low, the character's velocity fluctuated too much, resulting in jittery movement. By adjusting the **timeToTarget** to a slightly higher value, I was able to achieve smoother transitions.

I experimented with different **timeToTarget** values and found that higher values resulted in smoother behavior but slower reaction times. Lower values made the AI respond more quickly but led to more abrupt movements. Overall, I concluded that balancing these parameters is essential for achieving realistic behavior.

Additionally, although this section was only meant to demonstrate Velocity Matching, I chose to apply some orientation logic to the sprite to visually indicate the direction the character was

facing. This provided a more intuitive representation of the AI's movement and enhanced the overall realism of the behavior.

Part 2: Arrive & Align

The Arrive and Align behaviors work together to produce more natural and realistic AI movement. The Arrive behavior allows an AI character (boid) to gradually slow down as it approaches a target, while the Align behavior smoothly rotates the character to face the direction of motion. These combined behaviors are essential for creating believable and fluid movement in AI agents, especially in scenarios where precise control and orientation are important. [Screenshot #1, #2]

To demonstrate these behaviors, I implemented two different methods for both Arrive and Align:

- **Method 1:**
    - This method focused on a faster approach with quick deceleration. It used a smaller target radius and a lower **timeToTarget** value for both Arrive and Align. As a result, this approach provided quicker response times and a faster convergence to the target, making it well-suited for scenarios where the AI needs to respond quickly to sudden changes in target position.
- **Method 2:**
    - This method emphasized a smoother and more gradual approach by using a larger target radius and a higher **timeToTarget** value. This approach resulted in more fluid movement and smoother rotations, but the boid took longer to reach the target. This method is better suited for situations where visually pleasing, lifelike behavior is more important than speed.

In addition, I implemented a breadcrumb system to visualize the path of the boid over time. A fixed-length queue of locations was sampled at regular intervals, and small circles were drawn at each location to illustrate the boid's motion history. This helped me analyze and compare the differences between the two methods more effectively.

I found that adjusting parameters such as **maxAcceleration**, **maxRotation**, **targetRadius**, and **slowRadius** had a significant impact on the behavior:

- In Method 1, the boid quickly reached the target but exhibited more abrupt deceleration and rotation.
- In Method 2, the boid moved more fluidly but took longer to converge on the target.

Ultimately, I concluded that the choice between these two methods depends on the specific needs of the AI's role in the game. Method 1 is more suitable for fast-paced gameplay where quick response times are essential, while Method 2 is better suited for scenarios that prioritize smooth and visually appealing movement.

Part 3: Wander

The Wander behavior allows a boid to move in a random yet smooth and continuous pattern, simulating natural or idle movement when the AI has no specific target. This behavior is useful for creating lifelike movement in games, particularly for NPCs that are exploring or patrolling. [Screenshot #3]

To implement the Wander behavior, I used a circle-based approach, where a wander target is calculated a fixed distance ahead of the boid. The target position is then adjusted incrementally based on a changing wander angle.

I implemented two different methods for changing the boid's orientation:

- **Method 1:** The boid instantly snaps to face the direction of movement. This method results in quick and responsive changes in direction but looks less natural due to the abruptness of the orientation changes.
- **Method 2:** The boid smoothly rotates toward the target direction using a smoothing factor. This approach results in more gradual changes in orientation, producing smoother and more visually appealing movement.

In Method 1, the boid's orientation responded quickly to changes in the wander angle, but the abruptness of the rotation made the movement feel less lifelike. In Method 2, the boid's orientation changes were smoother, resulting in more organic-looking movement that resembled natural exploration.

Additionally, I experimented with the **WANDER_CIRCLE_DISTANCE** and **WANDER_CIRCLE_RADIUS** parameters. Increasing the circle radius resulted in wider, sweeping movements, while decreasing it produced tighter, more frequent turns. The **WANDER_ANGLE_SMOOTHING** parameter also played a significant role, as higher values led to more gradual changes in direction, while lower values resulted in more erratic behavior.

I concluded that **Method 1** is more visually pleasing due to its smoother orientation changes, making it better suited for scenarios where the AI is simulating idle exploration or natural movement. **Method 2** might be more appropriate for fast-paced games where quick responses are necessary, but it lacks the lifelike quality produced by the smoother approach.

Part 4: Flocking

The **Flocking** behavior in AI involves creating a cohesive group of autonomous agents (boids) that move together while maintaining a natural spacing. This behavior is achieved using the standard Boids algorithm, which combines three key steering behaviors: **Separation**, **Alignment**, and **Cohesion**. [Screenshot #4]

- **Separation:** This behavior ensures that boids maintain a minimum distance from their nearby neighbors, preventing collisions and overcrowding. It is particularly useful when the flock becomes too dense, as it encourages the agents to spread out.
- **Alignment:** This behavior makes boids match the average direction of nearby boids. It helps the group move in a coordinated manner, creating smoother and more consistent overall movement.
- **Cohesion:** This behavior encourages boids to move toward the average position of their neighbors, keeping the group together as a single unit.

In my implementation, I blended these three behaviors using adjustable weights to fine-tune the overall flocking behavior. The key challenge was finding the right balance between these components to achieve visually pleasing and realistic group dynamics. I conducted experiments with different weights for Separation, Alignment, and Cohesion:

- **Increased Separation:** This resulted in a more dispersed formation, where individual boids maintained a greater distance from each other. While this prevented collisions, it made the flock appear more scattered and less cohesive.
- **Emphasized Cohesion:** This created a tighter formation, where boids moved closer together. However, it sometimes led to clumping, which looked less natural.
- **Increased Alignment:** This produced smoother, more coordinated movement, where boids followed a similar direction. It made the flock appear more unified, but without sufficient Separation, it sometimes caused boids to overlap.

After testing various combinations, I found that a balanced approach with slightly higher weight on Separation produced the most visually appealing results. This balance allowed the flock to move as a cohesive unit while maintaining enough spacing to look natural. When increasing the number of boids, I observed that higher weights for Separation became more important to prevent overcrowding. Conversely, with fewer boids, Cohesion played a more significant role in keeping the group together.

Flocking behavior is highly sensitive to the weights assigned to Separation, Alignment, and Cohesion. The key takeaway from this implementation is that finding the right balance between these components is essential for creating realistic and visually appealing group dynamics. Overall, I learned that blending these behaviors intelligently can result in lifelike and cohesive flocking behavior, making it a powerful tool for AI in games and simulations.

Reflection on Parameter Tuning

Parameter tuning was crucial for achieving the desired behavior in each algorithm. Adjusting values like **timeToTarget** in Velocity Matching improved smoothness, while tweaks to **maxAcceleration**, **maxRotation**, and **slowRadius** in Arrive & Align affected responsiveness and precision. In Wander, varying **WANDER_CIRCLE_DISTANCE** and **WANDER_ANGLE_SMOOTHING** impacted the randomness and smoothness of movement. For Flocking, balancing the weights of **Separation**, **Alignment**, and **Cohesion** was key to creating natural group dynamics. These experiments highlighted that even small parameter changes can significantly affect AI behavior.

Conclusion

This assignment deepened my understanding of AI steering behaviors, including Velocity Matching, Arrive & Align, Wander, and Flocking. Achieving realistic AI movement required careful tuning and blending of these algorithms. The experience reinforced the importance of balancing responsiveness and smoothness. In future work, I aim to explore more complex behaviors, such as obstacle avoidance or goal-oriented navigation, for more dynamic and lifelike AI.

Appendix

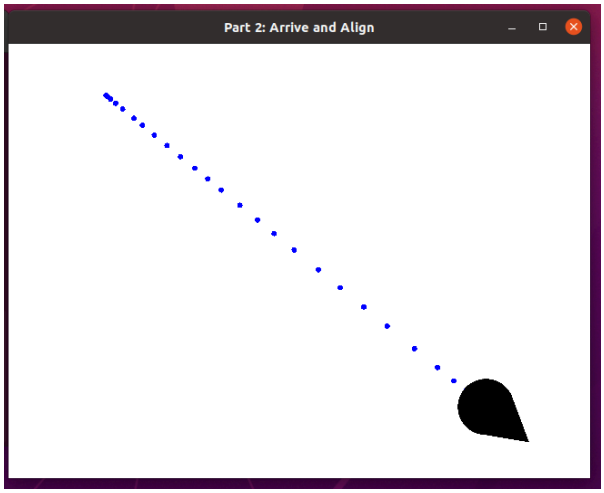
Acknowledgements

I referred to the SFML official tutorials to understand and implement graphical and windowing components for this project. Additionally, the course book "Artificial Intelligence for Games" by Ian Millington provided necessary insights into the implementation and behavior of various AI steering algorithms, including Velocity Matching, Arrive & Align, Wander, and Flocking. I also adapted the Breadcrumb class provided by TA Derek Martin, as well as the sample Makefile provided by Dr. David L. Roberts, to fit the needs of my project.

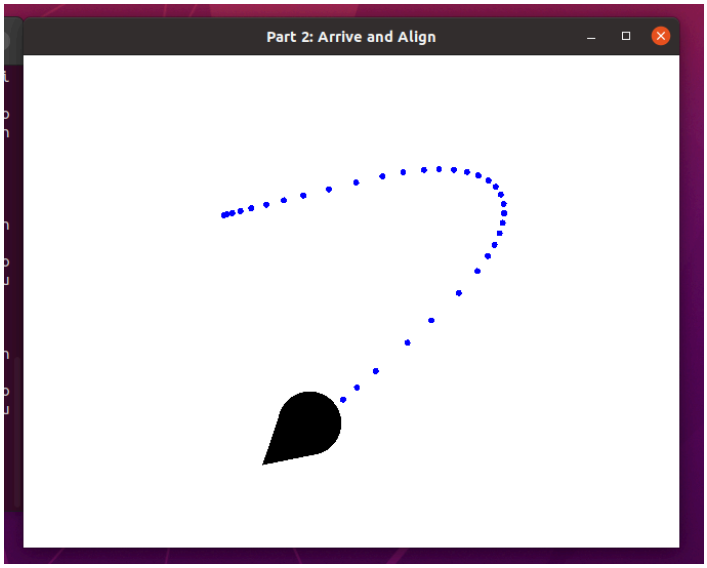
I used OpenAI's ChatGPT as a resource to provide assistance with understanding AI steering behaviors, structuring my code, and commenting on my implementation. ChatGPT provided explanations, insights, and guidance in the development of Arrive and Align, Wander, and Flocking behaviors.

Screenshots

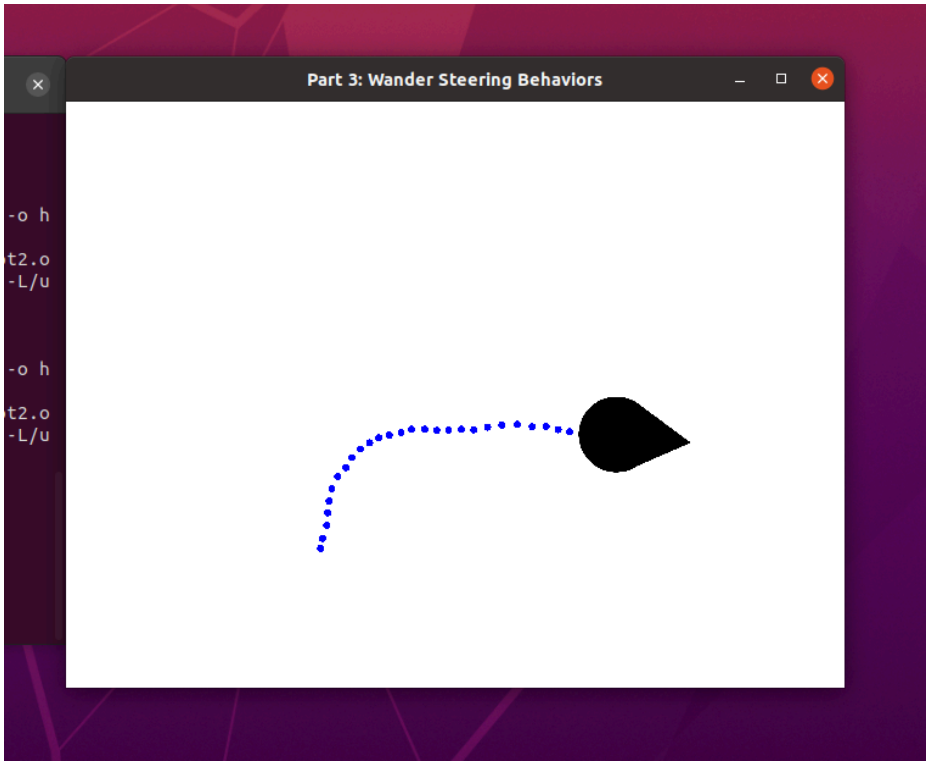
[Screenshot #1]



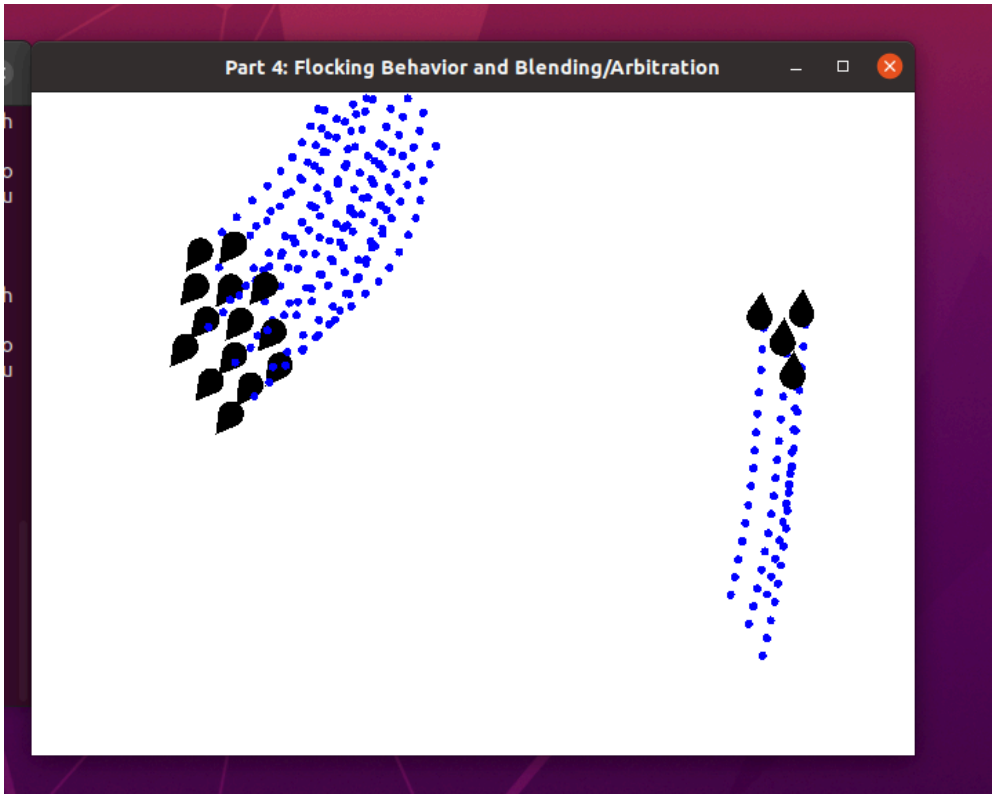
[Screenshot #2]



[Screenshot #3]



[Screenshot #4]


Part 4: Flocking Behavior and Blending/Arbitration