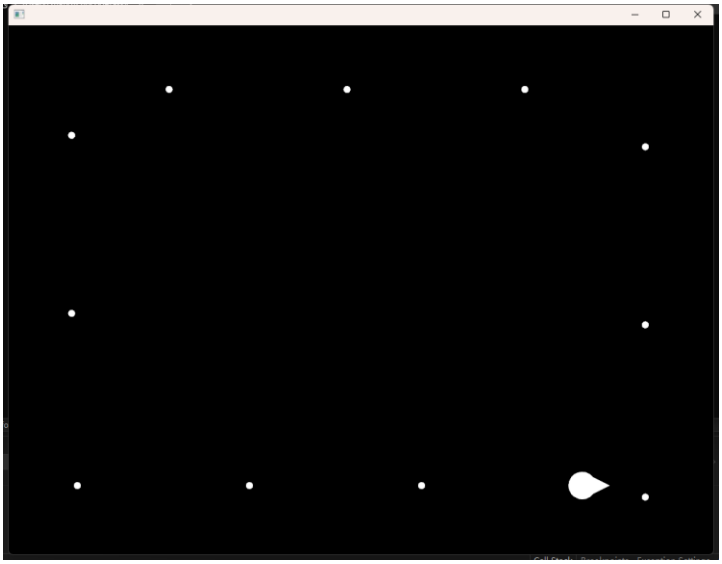
I am a student curious about the intricacies of games. In this project, I devote myself to AI technologies. I learned how to implement complex AI movement behaviors, which I was previously unfamiliar with. The project's difficulty lay not only in applying theoretical knowledge to the development of actual game AI but also in considering the many details, such as how to adjust various parameters for optimal results.

I found this assignment extremely beneficial; it deepened my understanding of the complexities of game AI. This project sparked my intense interest in more nuanced game AI algorithms, like those in racing game AIs. The work I did is closely related to the class content. I believe the outcomes of this project will be applied in my course projects, especially in AI design and debugging. Currently, I have not yet explored the application of machine learning in game AI, as I have seen about GT7's Sophy. This has ignited a strong desire in me to learn and implement machine learning techniques in game AI in the future.

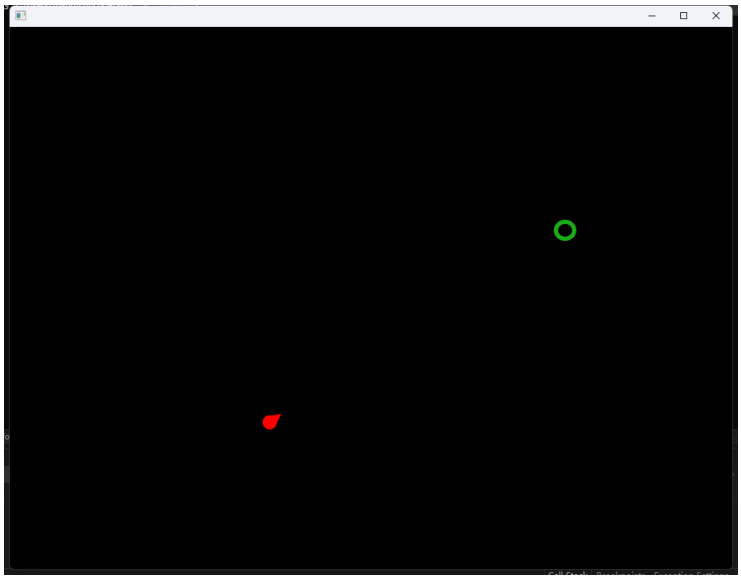
1. Kinematic Motion



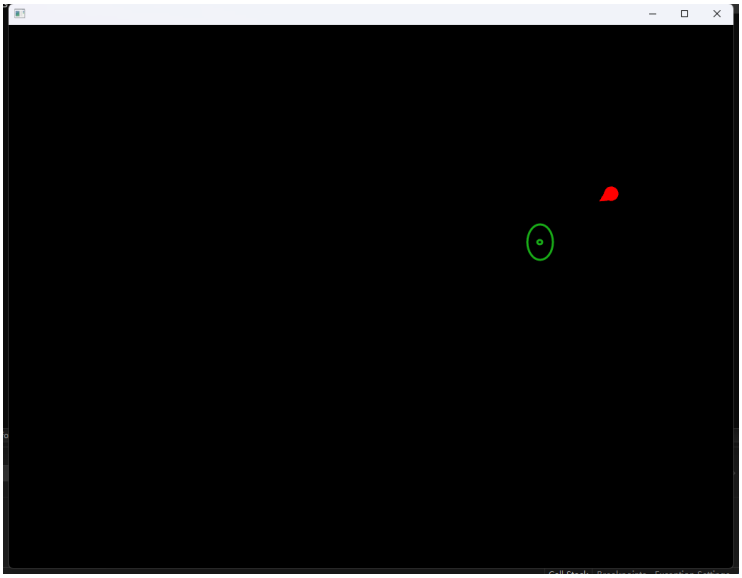
screenshot of shape and the breadcrumbs

2. Dynamic Steering Behaviours

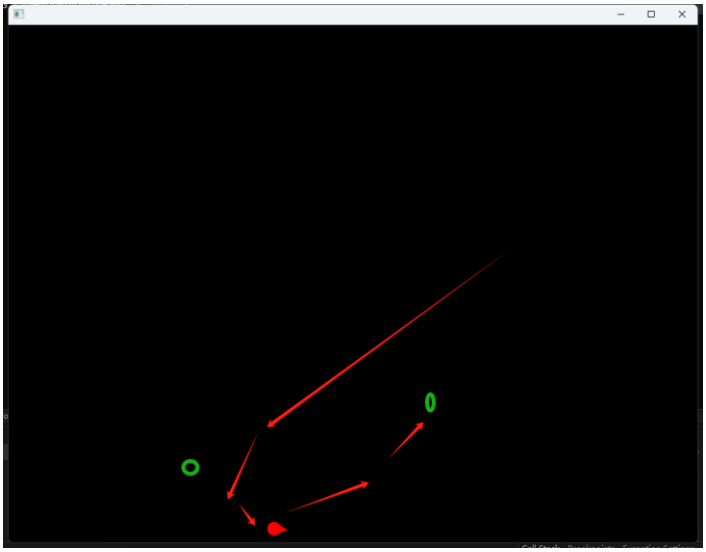
2.1 The first type of arrive: Arrive algorithm with smooth transition



The character moves towards the green area.



Insufficient deceleration leads to wandering behavior.

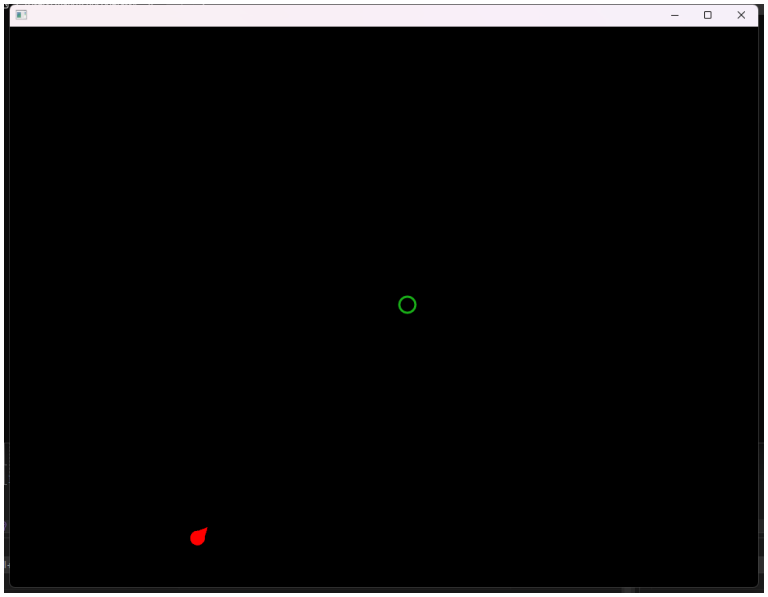


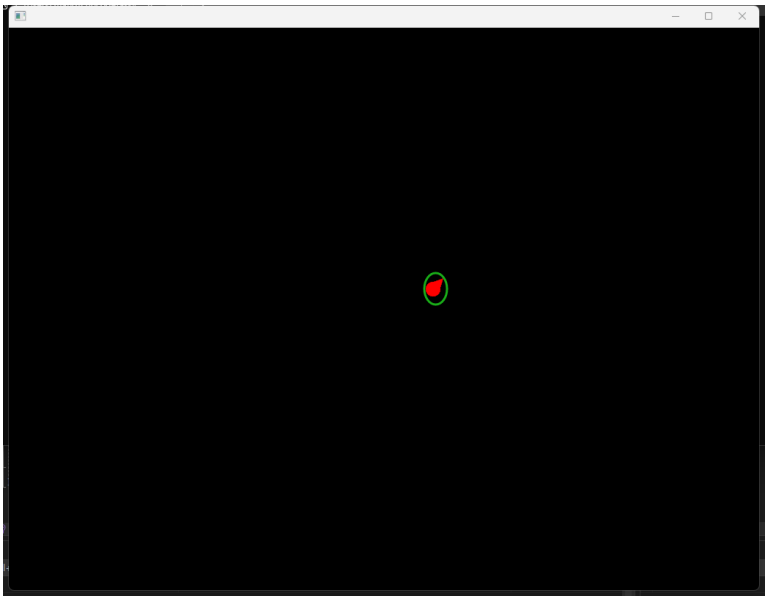
Changing the destination during movement results in curvilinear detour behavior.

Smooth Deceleration: Provides a more natural and realistic movement as the character gradually slows down when approaching the target, instead of stopping abruptly. Smooth deceleration can offer a more satisfying visual and gameplay experience.

Adjustability: By tweaking the slowRadius and timeToTarget parameters, the character's deceleration process and behavior can be finely controlled.

2.2 The Second Type: Without Smooth Transition





Which arrival method is better?

Regarding Visual Effects and Realism: If aiming for realistic and smooth character behavior, the arrive algorithm with smooth transition generally looks better because it simulates the deceleration process found in the real world.

3. Wander Steering Behaviors

3.1 Method 1: Adding a random angle change directly to the current direction.

- **Performance:** Requires minimal computational resources, suitable for use with many entities without impacting performance.
- **Randomness**: Provides a degree of random wandering behavior, suitable for scenarios where precise control over the wandering path is not necessary.

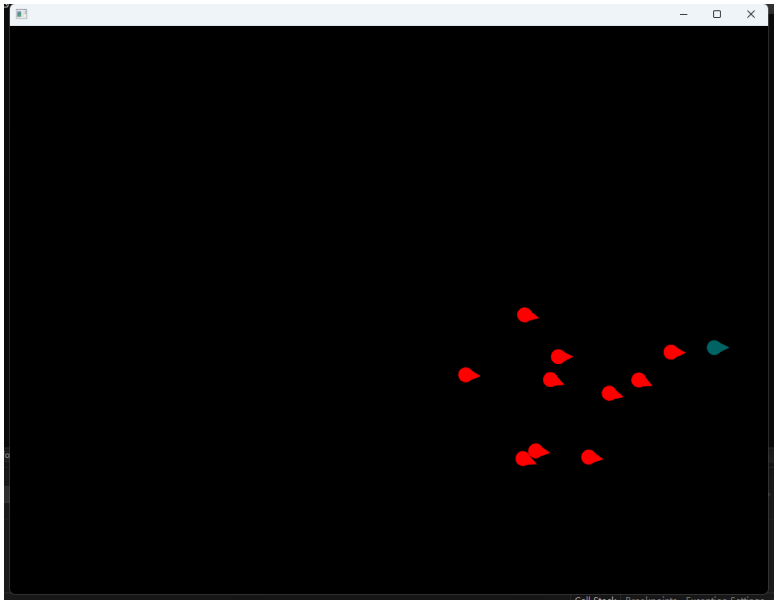
3.2 Method 2: Changing direction based on the character's current orientation plus a fixed offset and a randomly selected target point on a circle in front.

- **Natural Behavior**: The movement of the character is more natural and fluid, simulating the exploratory behavior of creatures in the real world.
- **Controllability**: By adjusting parameters (such as wanderOffset, wanderRadius, and wanderRate), the wandering behavior can be finely controlled to suit different gaming environments and needs.

Which method looks better? Why?

The second wandering algorithm appears better because it provides more natural movement behavior. This method can simulate behavior closer to the wandering of natural creatures. It's more suitable for applications where characters need to exhibit more natural wandering paths, such as creating lifelike game NPCs.

4. Flocking Behavior and Blending



Observations During Implementation

- **Behavior Integration**: Integrating the four behaviors of separation, collision avoidance, leader following, and cohesion into a unified system is complex. It's necessary to carefully adjust the weights and parameters for each behavior to ensure the flock's behavior is natural and effective.
- **Parameter Adjustment**: To achieve ideal flock dynamics, it's required to adjust the algorithm's parameters multiple times. For instance, the threshold for separation behavior, the acceleration for leader following, or the weight for cohesion behavior all need adjustments based on actual effects.
- **Weight Adjustment**: To balance the impact of different behaviors, different weights are applied to the outputs of each behavior. This is one of the key steps to implement effective flock behavior.
- **Impact of Follower Quantity**: As the number of followers increases, maintaining the flock behavior becomes more challenging. More followers mean more interactions, which could lead to unexpected patterns of flock behavior.
- **Multiple Wanderers**: If followers follow the nearest wanderer, this could lead to the flock splitting or forming multiple subgroups. Extra logic required to handle the splitting and merging of groups.