CSC 584 Building Game AI
Homework 2

hbharat2

Question 1:
The given task to implement Steering Behaviours in a virtual class had some cyclic dependency issues which were to be noted. The Original Kinematic structure has an update function which takes a Steering object to manipulate the 'pos' and 'vel', the Steering Behaviour takes in Kinematic variables to update them. This created a cyclic dependency which could not be solved by empty structure declarations and thus the usage of forward declaration in the form of another Class was used. Velocity matching was done after initially getting the sprite to arrive at the location of the mouse pointer. This ensured that the sprite followed the mouse pointer and there was only a slight delay as it took time to sample the mouse location and appropriately feed it and render the new window images.

Question 2:
The given task is to implement arrive and align at mouse click. The 2 methods implemented were derived by changing the parameters to get either **Snappy and aggressive movements** (Appendix Screenshot 2a) or **smooth and fluid movements** (Appendix Screenshot 2c). The parameters used for simulating such behaviours have been mentioned in the Appendix along with their screenshots.
In the Snappy and aggressive simulation the character moves quickly to the target, aggressively decelerates upon nearing the target and gives a sharp braking effect. The alignment is almost instant, causing snappy rotations. This simulation method is highly responsive and precise thus can be used in games like chasing and aggressive pursuit in case of cops activated in GTA or enemies activated within a radius that then start chasing you.
In the Smooth and fluid simulation the character glides smoothly to the target, starts decelerating early, resulting in a natural stop. The gradual deceleration feels realistic but it;s very sluggish and slow when simulating. It takes a lot of time for the sprite to reach the intended location and thus does not serve any real word use case other than willingly having such behaviour incorporated into a character in any game. With respect to just accuracy of arrive and align algorithms this simulation might work better as it stops within a reasonable range within the target and has fluid and graceful movement which might be appealing to the human.
A mix of snappy movement but gradual deceleration(Appendix Screenshot 2b) might sometimes be an interesting behaviour to replicate in games. This behaviour involves setting a very high max Acceleration thus scaling to great speeds very quickly but not having a very high maxVelocity. This combined with a bigger radius of Deceleration will give us this effect.
Key Find1 : The ratio between Max Velocity and Max Acceleration (Appendix Parameters table 1) is critical in determining how swiftly an object reaches its top speed. A high ratio shows a full ramp-up to full speed whereas a lower ratio makes the object reach its destination quickly appearing responsive.
Key Find 2: Time to target Velocity and Time to target Rotation are crucial as low values lead to rapid adjustments making the behaviour seem very unstable/ higher values of such variables are more gradual in their movement but result in a lot of delay when used. This is thus use case specific as to whether we want an aggressive chaser or a calm one.
Key Find 3: Radius of Satisfaction and Radius of Deceleration are key parameters in that their ratio defines a character's stopping distance. Increasing both proportionally maintains the approach behaviour. A small radius of satisfaction with a larger radius of declaration gives the most appealing arrive. The Snappy and Aggrexxive movement requires a larger radius of deceleration but a smaller radius of Satisfaction according to experiments and the vice versa is true for the slow and gradual movements.

Question 3:
The given task is to implement the Wander algorithm. The algorithm deployed on the codebase has been borrowed from the professor's description of the wander algorithm taught in class. We take in a **random binomial** to change the orientation and position of the sprite. The 2 different implementations of the wander algorithm can be seen in the Appendix. Implementation 1 (Appendix Screenshot 3a) does not involve any smoothing and thus takes random sharp turns whereas implementation 2 (Appendix Screenshot 3b) has a **smoothness factor** multiplied with the wanderOrientation.
This factor after experimenting was set to 0.9 to show a stark difference between random wandering and smooth wandering. By multiplying the wanderOrientation by 0.9, we are effectively applying a form of low-pass filter that removes high-frequency oscillations. An arbitrary decision as to which method looks better cannot be made due to its use case. From a cognitive perspective, humans tend to perceive smooth, continuous motion as more natural and realistic, especially when simulating organic entities like animals or pedestrians. A smooth wander would be more appropriate in an open world game like GTA - Grand Theft Auto where humans usually walk in straight lines on pavements and very rarely change course. Changing their course ever so slightly puts them in a slight tilt and thus this wandering behaviour best mimics humans. The random wander movement is more suited for Indie style 2D games where the AI has to take up the role of a sentry and needs a random path to follow so as to be unpredictable.
One of the benefits of the smoothing factor approach is that it introduces a parameter (`0.9` in this case) that can be dynamically adjusted to control the degree of smoothness. This gives Game developers more freedom.

Question 4:

The given task is to implement the Boids flocking algorithm. The algorithm deployed in the codebase has been derived from the textbooks guidelines and ideas. A few issues and key findings that have been observed and can be outlined are listed below. There is a triangular relationship which begs the balance between the free parameters to blend the forces of separation, alignment and cohesion. This triangular relationship causes a lot of minute issues which are handled in the findings below. The Flocking behaviour derives some of the same findings from part 2 where the ratio of velocity to acceleration is a key factor here.
Finding I:
Speed Consistency is very important.Since we are applying a blending between all the parameters once they start getting into a flock they tend to slow their speeds down. The velocities were averaged after they joined a flock. For this reason a maintainSpeed() function was created to ensure they don't come to a halt. This ensures all the velocity vectors are normalized but the orientation is preserved. Boids move more naturally when their momentum is preserved. If speed is inconsistent, flock movement looks jittery or sluggish. Speed consistency amplifies alignment and cohesion by maintaining momentum, ensuring fluid motion.
Finding II:
The radius of influence controls how far each boid looks to detect neighbors. This radius profoundly affects flock dynamics. Small Radius: Boids collide more frequently, leading to jittery movements. Large Radius: Boids repel each other too strongly, creating scattered and disorganized flocks. Setting the separation radius just above the boid size created smooth movement while avoiding collisions.
Finding III:
A cohesion radius will define how far a boid can detect others from a flock. Small Radius results in localized, dynamic groups with some free-flying boids whereas a larger radius leads to global flocking, where all boids aggregate into a single group. Therefore in my implementation i have

reduced the cohesion radius and created multiple small groups, enhancing the realism of the simulation.

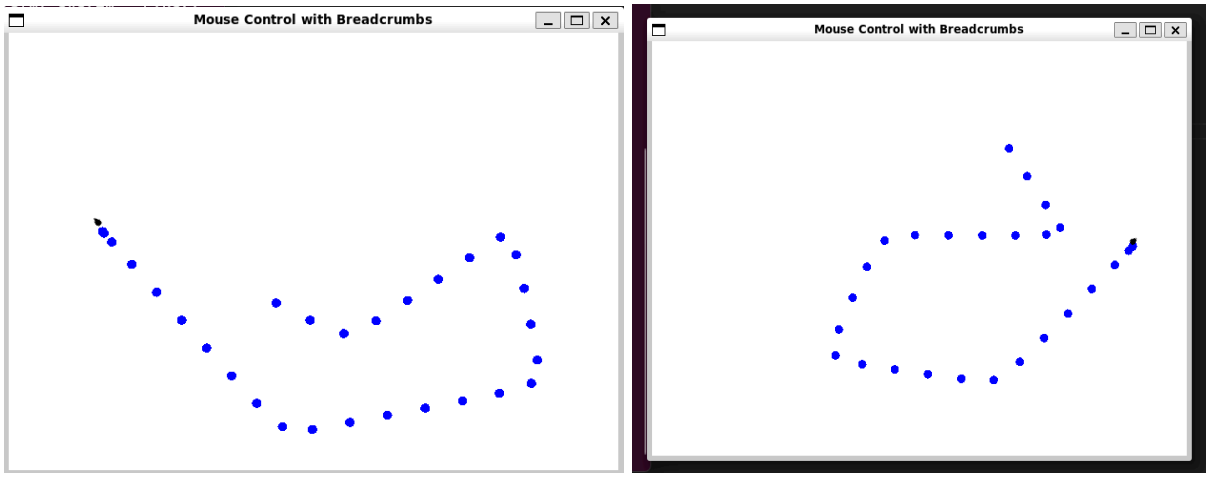
Finding IV:

50 Boids: Flocks formed quickly with minimal clustering. Independent flyers enhanced realism.
100 Boids: Multiple flocks emerged, showing effective separation and localized grouping. (Has been chosen finally as the best representative for flocking behaviour)
200 Boids: Stable performance with dynamic flocks, but required fine-tuning to avoid crowding.

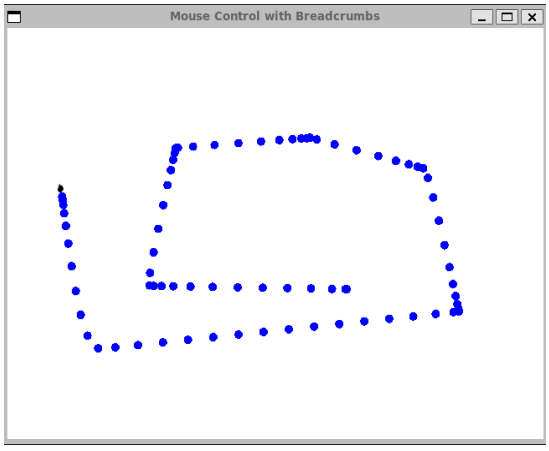
As seen in the Screenshots (Appendix Screenshot 4c) when simulated with 10 boids and reducing the radius, the algorithm seems to get the boids to coincide and collide more frequently. This does not seem natural and this screenshot was prior to the implementation of the maintainSpeed() function. To mitigate the coinciding of boids I had to increase the separation radius and increase the repelling force. For this reason the findings indicated above have been implemented to get the desired gradual flocking like behaviour.
Snappy flocking behaviour where aggressive turns are allowed could be implemented by increasing the cohesion radius

APPENDIX

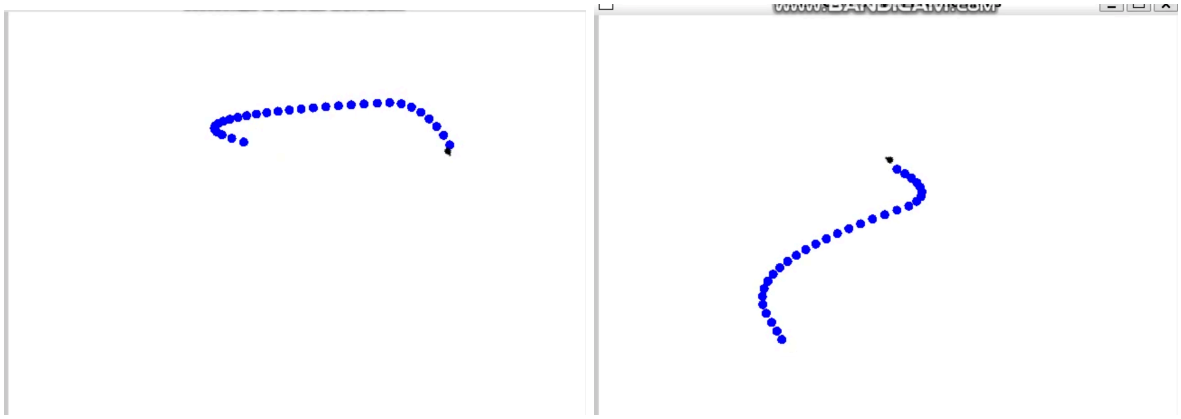
Screenshot 2a:



Screenshot 2b:



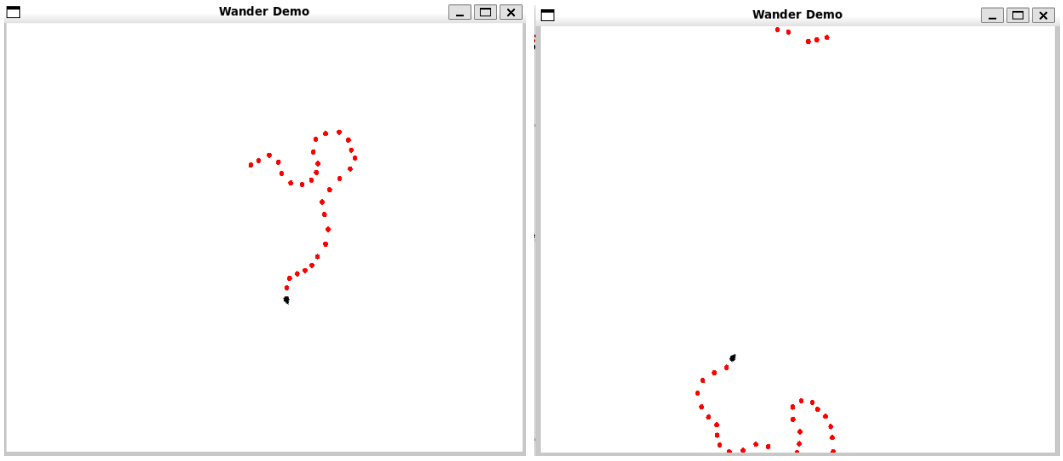
Screenshot 2c:



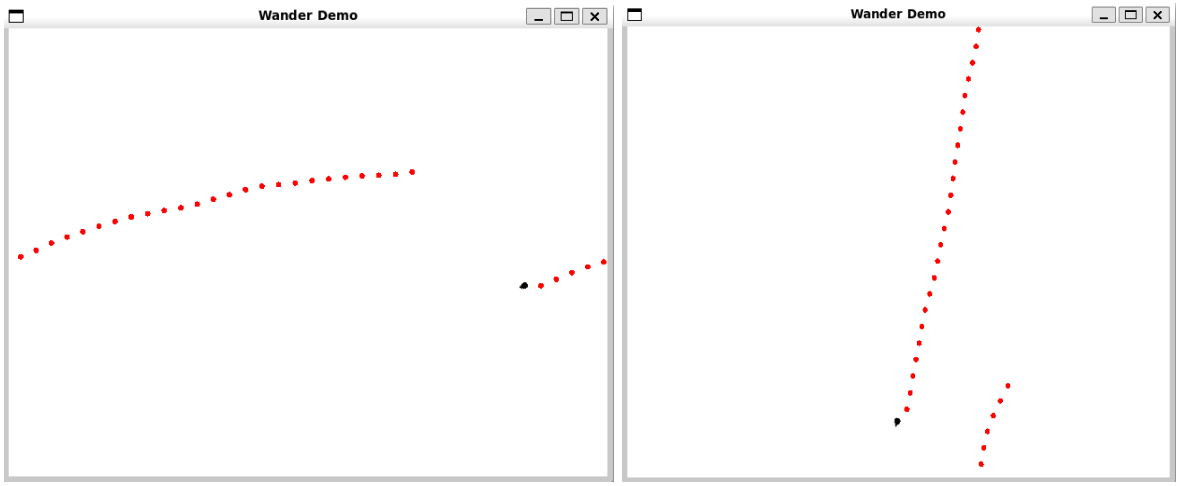
Parameters table 1:

Parameter	Snappy and Aggressive	Smooth & Fluid
Max Velocity	400.0f	150.0f
Max Acceleration	300.0f	80.0f
Vel/ Acceleration	1.33	1.88
Time to target	0.05f	0.5f
Radius of Satisfaction	5.0f	20.0f
Radius of Deceleration	50.0f	150.0f
Movement type	Sharp and snappy	Smooth and Gliding
Success in Accuracy	High	High
Success in Responsiveness	Very High	Moderate
Success in Aesthetics	Moderate	Very High

Screenshot 3a:



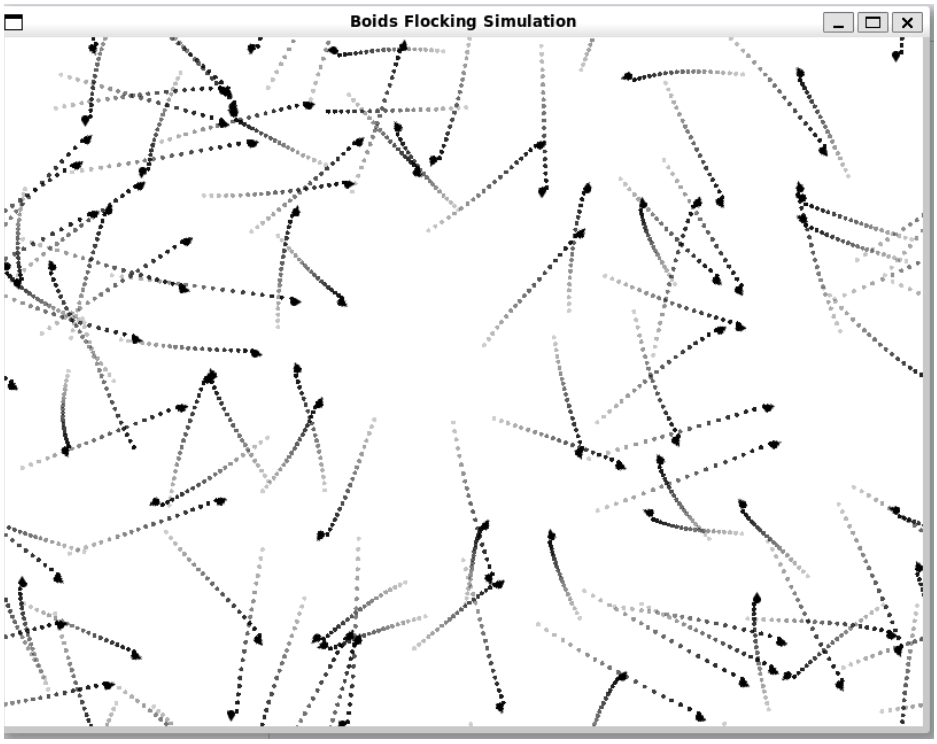
Screenshot 3b:



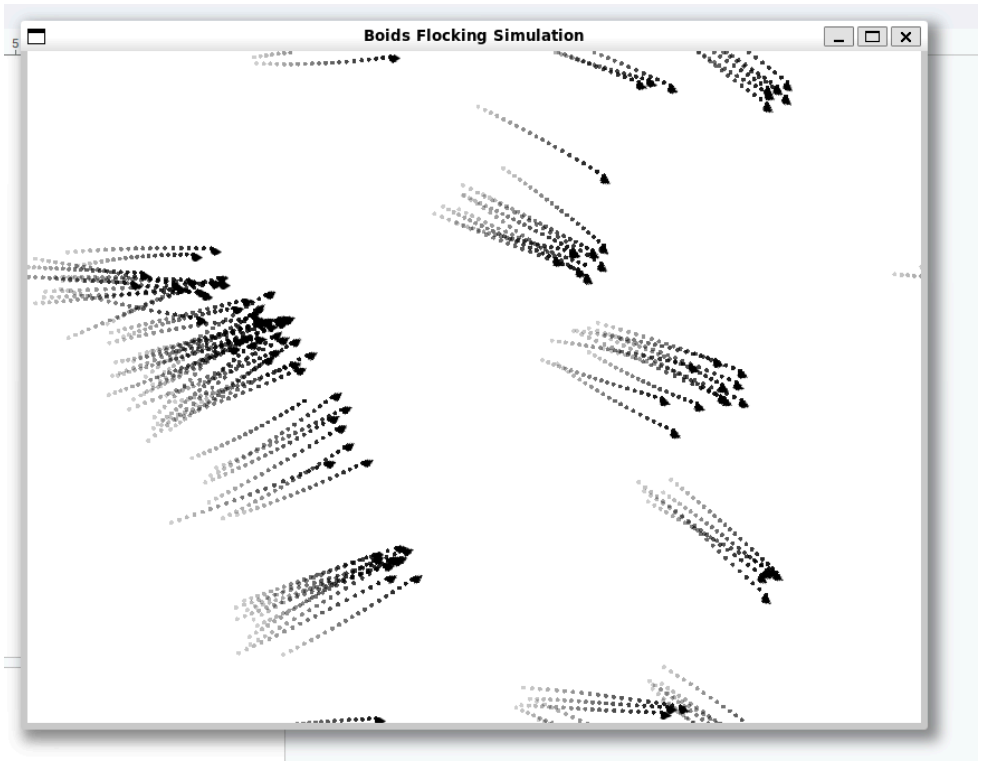
Parameter table 2:

Parameter	Value
WANDER_RADIUS	50.0f
WANDER_OFFSET	100.0f
WANDER_RATE	0.5f

Screenshot 4a:



Screenshot 4b:



Screenshot 4c:

Parameter Table 3:

Parameter	Value
Max Velocity	200.0f
Max Acceleration	100.0f
Vel/ Acceleration	2.0
Time to target	0.2f
Radius of Satisfaction	15.0f
Radius of Deceleration	100.0f
Movement type	Initial Fast to slow calm flocking
Success in Accuracy	High
Success in Responsiveness	Moderate to ensure human like behaviour
Success in Aesthetics	High