Modeling and Analysis of Variable Matching Movement Algorithms

Theo Grant
University of Utah EAE
332 1400 E, Salt Lake City, UT 84112
847-687-3096
theo.grant9@gmail.com

ABSTRACT
In this study we discuss patterns and consequences associated with a handful of variable matching movement algorithms. These include Flocking, Wandering, Seeking and a handful of others. This stands to argue that objective superiority of one algorithm architecture over another is impossible to assert without proper contextualization of its use. Additionally, analyzes a handful of aspects of the parameters of previously mentioned movement algorithms.

Keywords
Movement algorithms, Variable Matching Algorithms, Dynamic Algorithms

INTRODUCTION
As an introduction to the field of Game AI, this study focused on movement algorithms. For the sake of simplicity and focus on understanding the movement behaviors, all assessment took place in a 2 dimensional (2D) plane, with looping boundaries to maintain a controlled space to analyze each behavior. Much of what this study will focus on is the effects and weights that a particular algorithm parameter can have on its output. This has also helped contextualize the fine tuning associated with general Game AI and give us a framework of movement behaviors for future studies focused around decision making and path finding.

ENVIRONMENT
For this study, a simple 2D square plane was created with the OpenFrameworks library. This was to ensure simplicity in the vector math of these algorithms as it was not the focus of this particular study. In order to support movement algorithms, an object described as a Boid was created. This object is a circle with an arrow used to indicate its orientation. A Boid contains a variety of information. A Rigidbody object is used to contain raw dynamic motion information. This includes: position, linear velocity, linear acceleration, orientation, angular velocity, and angular acceleration. The Boid also contains a handful of parameters used for limiting movement including max speed, max linear acceleration, max angular velocity, and max angular acceleration. These are some in conjunction with other limiting parameters that will augment the variety of movement

Transactions of the Digital Games Research Association Vol 1 No 1, ISSN…., Feb 2022

algorithms created in this study. Motion is handled from the highest derivative of motion downwards. That is, acceleration is added to velocity, then velocity is added to position and a Boids update() function. Before each addition to the lower derivative, a check is made to ensure that the acceleration of velocity is not exceeding the max value indicated in the rigidbody. This is also true for angular motion. In order to keep all Boids contained and observable, this study utilizes a toroidal plane. This can have some peculiar effect on some of the behaviors including Flee, Flock, and Separation. Finally, each Boid is given a random color on creation unless specified. They also drop Breadcrumbs, simple dots created in order to display its previous pathing, after a certain number of Update() calls. These Breadcrumbs will correspond in color to their owning Boid.

SEEKING BEHAVIORS

Seeking behaviors are simple behaviors that attempt to move towards a given position. These on their own are independent of angular motion completely. In this study a total of 5 seeking behaviors were created: Kinematic Seek, Kinematic Flee, Dynamic Seek, Dynamic Arrive(Position Matching), and Dynamic Flee. For all but Dynamic Arrive, each movement behavior only requires 3 parameters: their rigidbody, their target rigid body, and a max acceleration or speed for dynamic and kinematic respectively. Because of the limited number of parameters affecting each movement it was easy to explore exactly how each affects the movement of a given Boid. This is where a majority of analysis was made.

The behavior of Dynamic Seek specifically works by finding the direction of the target rigidbody, and applying its max acceleration in the direction. This has a few effects like the Boid will never stop at its target. Instead, it will gain as much velocity as it can before passing through the target Boid (assuming the target is stationary) only to have its acceleration point in the opposite direction. It will then slow till it changes direction and return at max acceleration towards the target creating an oscillating motion. (see Figure 1)

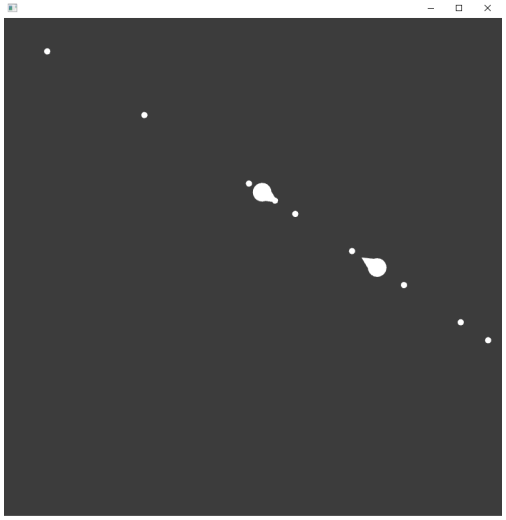

Figure 1: The Boid on the left is a stationary target for the Boid on the first which started in the top left corner.

The breadcrumbs show the path the left boid took as it oscillates back and forth.

The above example only works under a certain acceleration threshold. Give the looping nature of our environment, if the Boid has too much velocity as it passes its target, it will loop to the other side at the same Y position. This creates a situation where the Boid is always seeking to its right, therefore its X component of the linear acceleration is always positive. This continues to loop from right to left, either appearing just under or about the target Boids Y position creating a fish shaped path (see Figure 2).

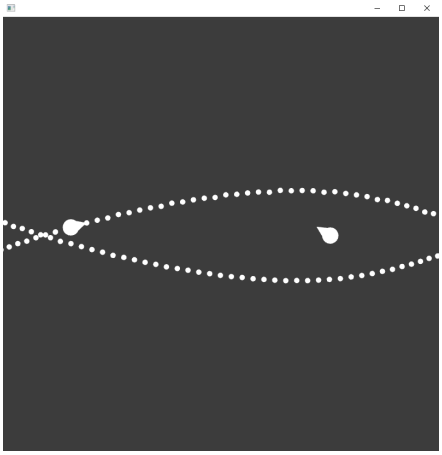

Figure 2: Fish pathing

This shape can be tweaked and controlled quite accurately by tuning the two parameters that affect motion in this behavior: Max Speed and Max Acceleration of the boid. The ratio of max speed to max acceleration actually increases the vertical thickness of the fish shape. This pointed out a relationship between the two parameters (see Figure 3).

```
m_maxSpeed * x;
m_maxAccel * x ^ 2;
```

Figure 3: This relationship (wheres X is some positive integer values) will maintain the exact shape of the fish

Though time restricted exhaustive analysis of this pathing shape, What I believe we are seeing is actually the sinusoidal wave of the oscillation behavior described before, just stretched over the entire X axis.

Dynamic Arrive differs greatly from Dynamic Seek in the sense that it attempts to slow itself as it gets closer to its target. While it still can oscillate past the target Boid, it usually can be avoided by properly changing the parameters of the behavior(i.e. max speed, max acceleration, slow radius, etc.). It's typical pathing can be seen in Figure 4

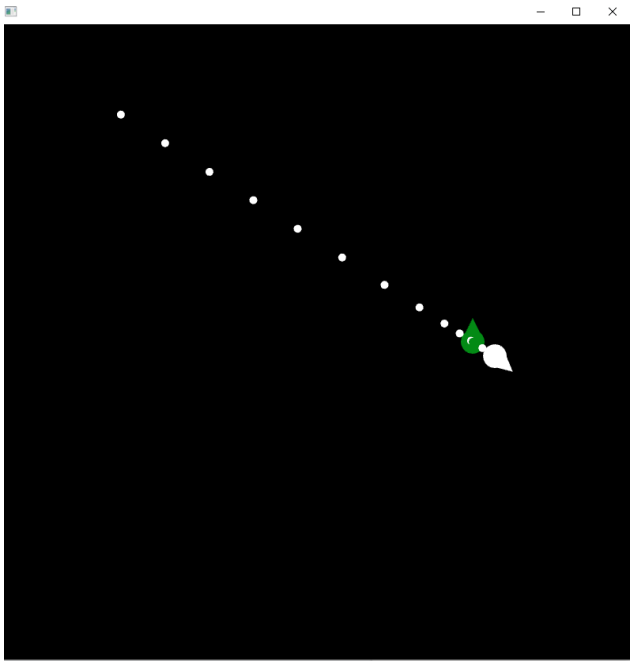

Figure 4: As you can see from the decreased distance between breadcrumbs, the Boid slows as it approaches the target and barely passes.

Simply put, these two behaviors accomplish the same goal of approaching a target in two very different ways. To say one is superior over the other is like comparing apples and oranges. If you don't care about the velocity or acceleration and just need your object to reach the target as fast as possible, Seek is optimal. If you are trying to accurately land an object inside a particular radius, Arrive is optimal. It is more dependent on the situation.

WANDER BEHAVIORS

A Wander behavior is an advanced algorithm that causes a Boid to move forward in a given range of the Boid current orientation at its max speed. That is to say it will select a random direction in range and then defer to the seek behavior on an update call. For this particular behavior, it also must face its direction motion otherwise it would simply wiggle back and forth in a small range of its initial orientation. Its general shape can be seen in Figure 5 and 6

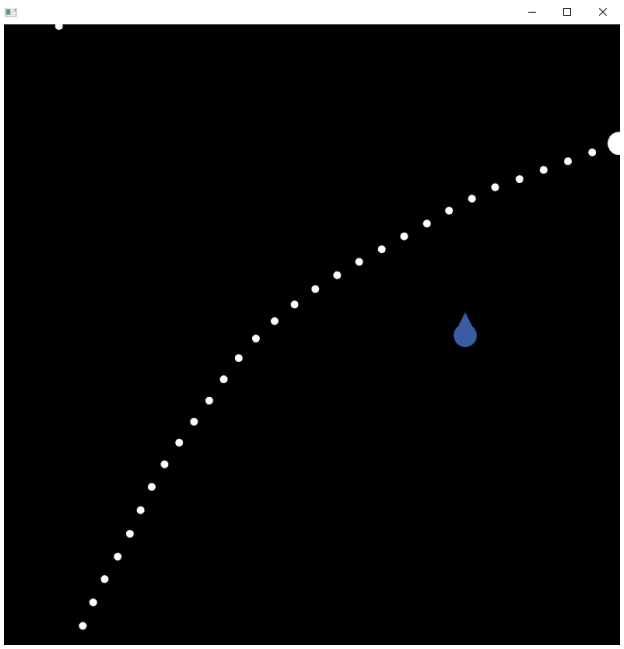

Figure 5: Wander Behavior with Dynamic Face orientation behavior

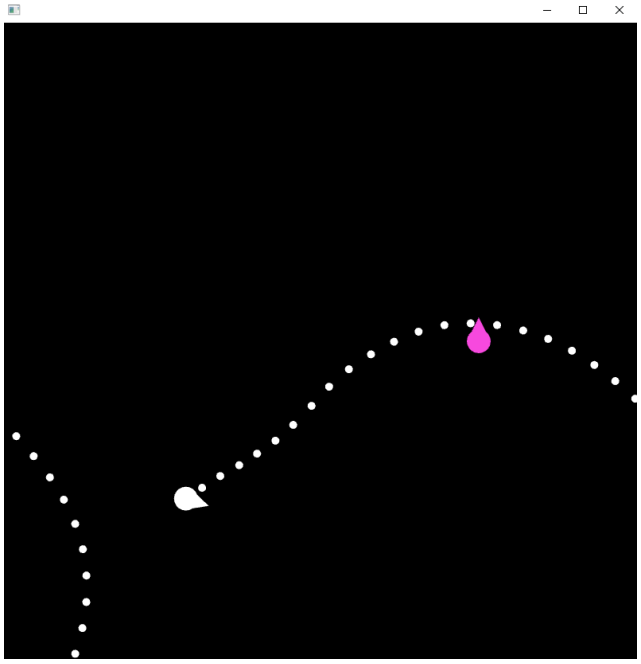

Figure 6: Wander Behavior with Dynamic LookWhereYouAreGoing orientation behavior

The difference between these two Wander behaviors is that wander with Face will tend to be more random whereas Wander with LookWhereYouAreGoing is going to move more

consistently in the direction it happens to start moving from start. This behavior is really derived from the fact that both Face and LookWhereYouAreGoing are advanced algorithms that defer to Dynamic Align.

Dynamic Align is essentially the angular motion version of Dynamic Arrive. As we established before in the previous section, Arrive tries to land as close to its target's position as possible but will almost always maintain some velocity once it reaches the target, albeit much slower than another behavior like Dynamic Seek. The same is true for Align. It will adjust the Boids angular acceleration to move towards a target orientation but never completely stop and will often overshoot its mark. Both Face and LookWhereYouAreGoing are subject to this problem by inheritance. The difference between the two orientation matching behaviors is that Face is orienting to the random direction that Wander creates and LookWhereYouAreGoing will orient itself towards the constantly updating velocity vector direction.

The effect of LookWhereYouAreGoing can again be seen in Figure 6. Once the Boid begins to drift left, the velocity vector is also pointing more left. Since the range is centered around the orientation, which is trying to follow the velocity vector, there are more chances for the Boid to seek towards a left leaning target. To make things even worse, the extra drift from Dynamic Align can cause the orientation of a Boid to swing even more left past the orientation of the already left leaning velocity vector.

Wander utilizing Face makes several Align calls on many random positions inside the given range. This will tend to normalize out the turning due to the nature of the random function we use. The binomial distribution of our random directions keeps the Boid from swerving off to one extreme over the other, as seen in the more gentle movement of Figure 5.

FLOCK BEHAVIOR

Flocking behavior is quite complace in nature amongst several fish, birds and bugs. This behavior can simply be broken down into blended weights of Separation, Velocity Matching, and Arrive. We have already discussed Arrive many times. In this particular case, each Boid in the flock is trying to Arrive to the center of mass of the flock. Velocity Match is simply matching the velocity of a target. This is independent of orientation. For flocking, all Boids are trying to match the average momentum of the flock. Finally, separation is behavior that tries to keep distance from any other boid in the flock. This makes sure that all Boids do not converge on a single point. In this study, we can create a lead Boid by creating a large weight for a single Boid. This will offset the center of mass and average momentum to generally follow that particular Boid. By applying a Wander behavior to that lead Boid, you can cause the flock to follow along with the lead Boid.

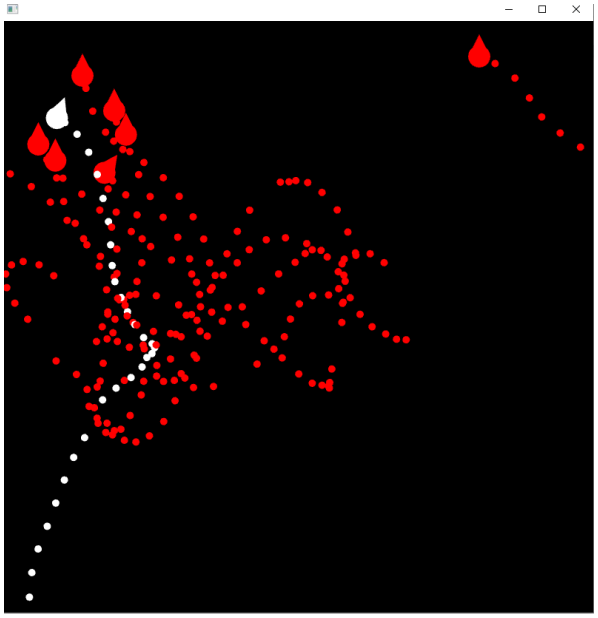

Figure 7: The white lead Boid (weight = 20) can be seen pulling most of the red Boids (weight = 1) as it wanders

The main adjustment made for this behavior was allowing the algorithm to adjust the dynamic motion of each Boid internally. Unlike other movement algorithms, Flock produces a variety of dynamic movement outputs that must be managed separately for each Boid, so ensuring the use of references in the Flocking algorithm is key.

One of the biggest factors that affects the behavior of flock is simply the Boids themselves. It is an ever evolving system so increasing the number of Boids can shift its behavior greatly. For example, as the number of Boids under a lead Boid increases, the weight of that lead Boid is lessened. This is because the flock's total mass has increased enough to offset that single huge mass of the lead. Consequencently, the less Boids means the lead Boid will have more weight in the system and move much more freely. This appears to create the phenomenon of large systems of Boids collecting closer and closer the more Boids are present, but it is hard to confirm given the nature of the toroidal environment. The swaying flock can sometimes spill over the edge of the map, shift the center of mass across the environment, interrupting cohesion in the flock. Regardless, this behavior persists even with many heavy weighted Boids.

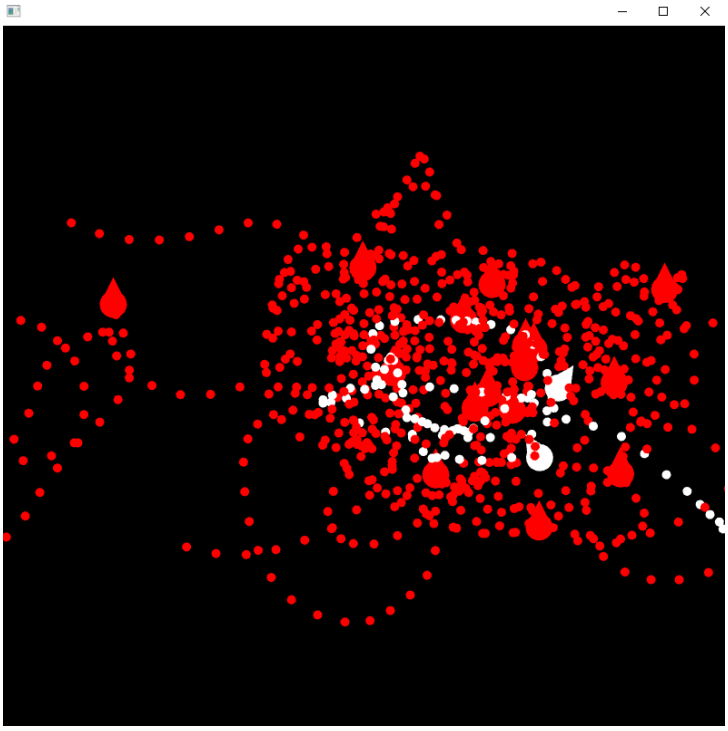

Figure 8: Both heavy Boids eventually spiral in towards each other

It seems like while two lead Boids can pull off the smaller Boids from the other, the collective weight of each lead Boids flock will eventually cause the flock to consolidate. Since the two heavy Boids are nearly on top of each other, the flock has a high cohesion. Something that is important to note is that even though a smaller Boid might have only a weight of one, it can have a massive impact on the flock once it passes over one of the boundaries. While this is highly dependent on weights and many other parameters, the distance seems to have an overall greater effect on the flock than weights which makes sense on a physical level. Consider leverage. Force is proportional to the opposite force's distance from the fulcrum, a Boid that crosses over to the other side essentially has the same effect.

Conclusion

This study posed many questions around the ideal of which behaviors looked nicer or are "better". Simply put, without a particular situation the answer is arbitrary. All of these behaviors have a variety of parameters that dictate their output. Parameters at just that, "any characteristic that can help in defining or classifying a particular system" and without a goal of a particular system you can really discern value from one over the other.

The main purpose of this study is to understand how tuning of those parameters has the possibility to adjust systems in particular ways and begin to get a grasp on how we will specifically fine tune them to our designed behaviors in future studies

Proceedings of DiGRA 2011 Conference: Think Design Play.