A Yıldız ve Akın Algoritmaları ile Otonom Sürü Sistemleri için Yol Bulma

Araştırma Makalesi/Research Article

Mehmet ALBAYRAK* , Ali Murat SÜMEN

Bilgisayar Teknolojileri, Isparta Uygulamalı Bilimler Üniversitesi, Isparta, Türkiye

mehmetalbayrak@isparta.edu.tr, alimrts07@gmail.com

(Geliş/Received:16.01.2023 ; Kabul/Accepted:20.06.2023)

DOI: 10.17671/gazibtd.1236552

Özet— Savaş ve yapay yaşam temalı bilgisayar oyunlarında, yapay zekâlı karakterlerin, ortak bir hedefe takım halinde yol bularak akın hareketi algoritması kurallarına göre sürü halinde hareket etme sistemleri üzerinde çalışılmıştır. Bu sistemlerin kullandığı yol bulma algoritmaları arasında Dijkstra, Best-First Search (en iyi ilk arama) ve A* (A yıldız) incelenmiştir. Sonuçlar, A* yol bulma algoritmasının en hızlı ve etkili olduğunu göstermiştir. Bu algoritma ve sürü hareketi modeli kullanarak, takım halinde hareket ederken kendi yolunu bulabilen yapay zekâ sistemi geliştirilmiştir. A* algoritmasını kullanan yapay zekâlı karakterlerin oluşturduğu takımların, oyun içi durumlara karşı daha başarılı sonuçlar elde ettiği görülmüştür.

Bu çalışma kapsamında, savaş temalı bir oyun için takım tabanlı yapay zekâ modülü geliştirilmiştir. Bu modül, Unity 3D oyun motoru kullanılarak oluşturulmuştur. Bayrak kapma senaryosu modu kullanılarak, lider kullanıcılı takım ile düşman takımı arasındaki karşılaşmanın sergilediği davranışlar incelenmiştir. Bu çalışma sonucunda, A* ve akın hareketi algoritmalarının kullanılması ile oyunların oynanabilirlik ve gerçekçilik gibi önemli özelliklerinin, dinamik oyun karakterleri aracılığıyla bir üst seviyeye taşınabileceği ortaya çıkmıştır.

Anahtar Kelimeler— A*, akın, sürü, takım tabanlı yapay zekâ, yol bulma algoritmaları

Pathfinding for Autonomous Swarm Systems with A Star and Flocking Algorithms

Abstract— In computer games with the theme of war and artificial life, the systems of artificial intelligence characters to move in flocks according to the rules of the raid movement algorithm by finding a way to a common goal as a team have been studied. Among the pathfinding algorithms used by these systems, Dijkstra, Best-First Search and A* have been examined. The results have shown that the A* pathfinding algorithm was the fastest and most efficient. By using this algorithm and a swarm movement model, an artificial intelligence system has been developed that can find its own way while moving as a team. It has been observed that teams formed by artificial intelligence characters using the A* algorithm achieve more successful results against in-game situations.

Within the scope of this study, a team-based artificial intelligence module was developed for a war-themed game. This module was created using the Unity 3D game engine. Using the flag grabbing scenario mode, the behavior of the leader user team and the enemy team was examined. As a result of this study, it has been revealed that important features of games such as playability and realism can be taken to the next level through dynamic game characters by using A* and raid movement algorithms.

Keywords— A*, flocking, swarm, team-based artificial intelligence, pathfinding algorithm

1. GİRİŞ (INTRODUCTION)

Karakter yapay zekâsı, bir oyunun içinde yer alan yapay zekâ tarafından yönetilen karakterlerin davranışlarını ve hareketlerini belirleyen bir sistemdir. Bu sistem, karakterlerin oyun içinde doğal ve inandırıcı bir şekilde davranmasını sağlar ve oyunun gerçekçiliğini arttırır. Örneğin, bir karakter yapay zekâsı ile programlanmış bir karakter, oyun içinde bir çatışma sırasında diğer karakterlerle etkileşime girerek savunma pozisyonuna geçebilir veya kaçabilir. Ayrıca, karakter yapay zekâsı sayesinde karakterler oyun içinde doğal bir şekilde yüz ifadeleri ve mimikler kullanarak duygularını ifade edebilirler. Bu sayede oyun oynayan kişi, karakterlerle daha etkileşimli ve inandırıcı bir şekilde oyun oynayabilir. Özetle, karakter yapay zekâsı, oyunların gerçekçiliğini ve oynanış deneyimini artıran önemli bir öğedir. Bu çalışmada takım halinde hareket eden yapay zekâlı karakterler için yol bulma algoritmaları araştırılmış ve en iyi ve performanslı yol bulma algoritmalarından biri olan A* algoritmasının 3D (3 boyutlu) oyun ortamında takım tabanlı yapay zekâ sistemleri için kullanımı üzerinde çalışılmıştır.

Bilgisayar oyunlarında kullanılan yapay zekâ sistemlerinin temel amacı, oyuncu ile etkileşimde bulunan karakterlerin ve oyunun geçtiği ortamın gerçek insan, topluluk ve dünya yaşam ortamına olabildiğince benzetilmesidir. Bu amaç doğrultusunda, oyun tasarımı ve yapay zekâ algoritmaları aracılığıyla oyunun gerçekçi ve doğal bir etkileşim ortamı oluşturulmaya çalışılır. Gelişmiş yapay zekâ sistemlerine sahip oyunlar, oyun içi deneyimin gerçek yaşam ortamlarına benzeyen bir ortamda yaşanmasını sağlar. [1]. Oyunlarda kullanılan yapay zekâ sistemlerinin başka bir önemli amacı, oyuncunun yeteneklerine karşılık vermek ve oyuncunun kurduğu planları veya tuzakları taklit etmektir. Bu amaç doğrultusunda, yapay zekâ algoritmaları oyuncunun yeteneklerini ve stratejilerini analiz ederek, oyuncuya karşı uygun bir tepki oluşturmaya çalışır. Aynı zamanda, yapay zekâ sistemi oyuncuyu zor duruma düşürerek oyunu kazanabilmek için gerektiğinde tuzaklar kurabilir. Bu, oyun programındaki yapay zekâ kalitesini belirleyen önemli bir faktördür [2].

Geleneksel yol bulma algoritmaları ile takım halinde hareket eden yapay zekâlı oyun karakterlerinin bir noktadan başka bir noktaya doğru hareket ederken, karınca sürüsü gibi tek bir yol izlemesi gerçeklikten uzak bir hareket örneği oluşturmaktadır. Her bir yapay zekâlı takım bireyinin kendi konumuna en uygun yolu bularak ve oyun içi dinamik engelleri aşarak ilerlemesi oyun içi gerçekçiliği arttıran bir yaklaşımdır.

Takım halinde hareket eden yapay zekalı karakter sistemleri, savaş temalı FPS (First Person Shooter/Birinci Şahıs Nişancı) türü oyunlar için önemli bir öğedir. Bu sistemler sayesinde, oyun içinde yer alan yapay zekâ tarafından yönetilen karakterler takım halinde hareket edebilir ve birbirlerine yardımcı olabilirler. Örneğin, takım tabanlı karakter yapay zekâsı ile programlanmış bir takım, oyun içinde bir çatışma sırasında birbirlerine yardım etmek için pozisyon değiştirebilir veya birkaç karakterin bir hedefe birlikte saldırmasını sağlayabilir. Bu sayede oyun oynayan kişi, takımını daha etkili bir şekilde yönetebilir ve oyunun gerçekçiliğini arttıran inandırıcı bir takım deneyimi yaşayabilir. Ayrıca, takım halinde hareket eden yapay zekâlı karakterler, oyun içinde dinamik ve değişen koşullara uyum sağlama konusunda da önemli bir rol oynar.

Bu çalışmada, yol bulma problemlerinde kullanılan algoritmalar incelenerek, literatürde en iyi sonuçları verdiği görülen A* algoritması seçilmiştir. A* algoritması, yol bulma problemlerinde etkili bir şekilde kullanılan bir yöntemdir. Bu algoritmanın yanı sıra, sürü halinde hareket eden gruplar için de A* algoritmasına entegre olarak çalışan bir yaklaşım üzerinde durulmuştur. Bu yaklaşım, grubun bir arada tutulmasını sağlarken, bireylerin de kendi yollarını bulabilmelerine olanak tanır. Bu sayede, grup içindeki bireylerin birbirlerini takip etmeleri yerine, her bir bireyin kendi yolunu bulabilmesi hedeflenmiştir.

Geleneksel yol bulma algoritmalarından farklı olarak, A* algoritmasının 3D harita üzerine uygulanarak takım karakterlerinin takımın varış noktası amacına uygun bir şekilde en doğru ve özgün yolları bulabilmesi sağlanmıştır. Bu sayede oyun yazılımının zaman ve işlem gücünden tasarruf edilirken daha gerçekçi hareket eden yapay zekâlı karakter sistemleri elde edilmiştir.

2. TEORİK METOD (THEORETICAL METHOD)

Bu bölümde kullanılan metot, yol bulma algoritmaları, sürü hareketi ve takım tabanlı yapay zekâ uygulaması hakkında bilgi verilmektedir.

2.1. Yol Bulma ve Engel Geçişi (Pathfinding and Obstacle Avoidance)

Savaş temalı bilgisayar oyunları, oyun oynayan kişinin savaş alanında yer alan bir karakteri yönetmesini ve/veya savaşın içinde yer almasını sağlar ve genellikle taktiksel ve stratejik hareketler gerektiren dinamik bir oynanış deneyimi sunar. Özellikle Battlefield ve Call of Duty gibi etkileşimli ve dinamik oyunlarda, yol bulma probleminin geliştirilmiş çözümleri sayesinde karakterler oyun içinde dinamik koşullara uyum sağlayabilir ve bu sayede oyunun zorluk seviyesi de arttırılabilir.

Günümüzdeki oyunlar genellikle çok geniş oyun içi dünyaları ve oynanış sırasında değişen koşulları içerir. Bu nedenle, oyun içi karakterlerin yol bulma işlemleri farklı algoritmalar kullanarak gerçekleştirilir. Bu yeni nesil oyunlar, gerçekçi ortamlar ve karakterler içerir ve bu nedenle, gerçekçi bir yapay zekâ sistemi olmadığı takdirde boş ve düşük kaliteli görünebilirler. Bu

nedenle, oyun içi karakterlerin sürekli değişen oyun ortamına uyum sağlamak için yeni yollar bulma gibi işlemleri gerçekleştirebilmesi gerekir.

2.1.1. A Algoritması (A* Algorithm)*

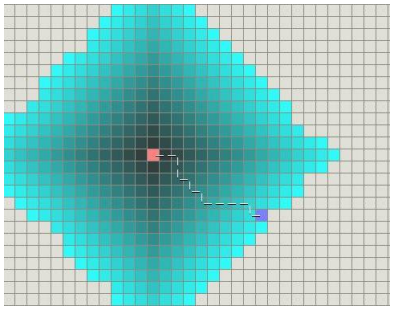
A*, bir grafik üzerinde iki nokta arasındaki en kısa ve en az maliyetli yolu bulmak için kullanılan bir yol bulma algoritmasıdır. Aramayı yönlendirmek ve en umut verici yolları seçmek için sezgisel bir işlev kullanarak hem Dijkstra hem de Breadth-First Search algoritmasının faydalarını birleştirir.

A* algoritması sezgisel fonksiyonu $f(n)$ ifadesi aşağıdaki gibidir;

$$f(n) = g(n) + h(n) \quad (1)$$

Burada, $g(n)$ başlangıç düğümünden mevcut düğüme kadar gelme maliyetini, $h(n)$ ise mevcut düğümden hedef düğüme varmak için tahmin edilen mesafeyi temsil eder.

Şekil 1'de gösterildiği gibi en kısa yol bulma algoritmalarından olan Dijkstra algoritmasında başlangıç düğümünden itibaren dışarıya doğru bütün yönlerde ilerleme tekniği kullanılır. A* algoritmasının temel fikri, her yolun maliyetini göz önünde bulundurarak ve aramayı hedefe doğru yönlendirmek için sezgisel işlevi kullanarak arama sınırını yinelemeli olarak genişletmektir. Her adımda, algoritma en düşük tahmini toplam maliyete $f(n)$ sahip yolu seçer ve arama sınırına komşu düğümleri ekleyerek onu genişletir. $f(n)$, yolun o ana kadarki maliyeti olan $g(n)$ ve hedefe giden tahmini maliyet olan $h(n)$ toplamı olarak hesaplanır. $g(n)$ Değerinde olan artış hedeften uzaklaşıldığını gösterir. Algoritma daha düşük $g(n)$ değerleri ile hedefe ulaşılması sonucunda doğru yolu tespit eder [3].

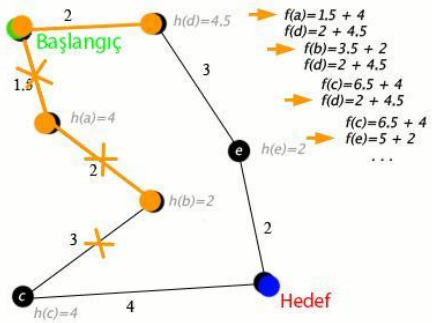


Şekil 1. Dijkstra algoritması yol bulma şeması
(Dijkstra algorithm pathfinding schema) [4]

A* algoritması, bir açık liste ve bir kapalı liste kullanarak çalışır. Açık liste, A* algoritması tarafından ziyaret edilmeyi bekleyen noktaların bir listesidir. Her adımda, açık listesinden en düşük maliyetli noktayı seçer ve bu noktayı kapalı listeye taşır. Kapalı liste ise, A* algoritması tarafından ziyaret edilen noktaların bir listesidir. Bu sayede, aynı noktayı ikinci kez ziyaret etmekten kaçınılır ve algoritma daha hızlı çalışır [5].

Şekil 2'de noktalar arası bağlantılı yolların birleştirilerek hedefe ulaşma süreci görülebilir. Şekil 2'deki yeşil nokta başlangıç noktası, mavi nokta ise hedef noktasıdır. Bağlantı yolları turuncu çizgiler ile temsil edilmektedir. Her bir arası $g(n)$ (maliyet) ve $h(n)$ (mesafe) değerleri toplanıp $f(n)$ hesaplanır ve son nokta olan hedefe ulaşılır [6].

A* algoritması her işlem adımında açık listesinden en düşük maliyetli noktayı seçer. Seçilen noktanın çevresindeki noktalara gider ve çevresindeki noktaların maliyetlerini hesaplar. Çevresindeki noktaların maliyetlerini karşılaştırır ve en düşük maliyetli noktayı seçer. Seçilen en düşük maliyetli noktayı kapalı listeye taşır ve işlemi tekrar başa döndürür. Algoritma, hedef noktaya ulaşana kadar bu adımları tekrar eder ve hedef noktaya ulaştığında en kısa yolu bulmuş olur [7].



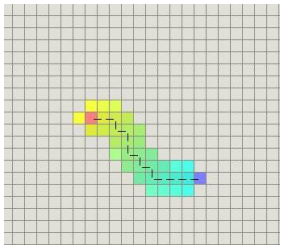
Şekil 2. A* algoritması düğüm ve yol bulma şeması
(A* algorithm node and pathfinding schema) [8]

A* algoritması, sezgisel fonksiyon tarafından hesaplanan tahmini maliyetin gerçek maliyetten daha az olduğu durumlarda, en kısa yolu garanti eder. Ancak, tahmini maliyet gerçek maliyetten daha fazla olursa, bulunan yol optimal olmayabilir. Bu nedenle, A* algoritması için kullanılan sezgisel fonksiyonun doğruluğu, bulunan yolun en kısa yol olup olmadığını belirler [9].

Algoritma 1: A* Algoritması				
	Input: *Başlangıç (start), Hedef (goal) noktaları*			
	Output: *Bulunan noktalar listesi (came_from)*			
1	***function*** *A*(start, goal)*			
2		*// Arama sınır başlangıcı noktasını belirle*		
3		*frontier = başlangıç içeren öncelik sırası listesi*		
4		*// came_from (önceki aranan) listesini belirle*		
5		*came_from = boş liste*		
6		*// cost_so_far (şimdiye kadar) listesini belirle*		
7		*cost_so_far = boş liste*		
8		*cost_so_far[start] = 0*		
9		*// Arama listesi boşalana kadar döngü*		
10		***while*** *frontier boş değil*		
11			*// Arama listesinden en düşük f-maliyetli noktayı al*	
12			*current = frontier.get_lowest_f_cost_node()*	
13			*// Mevcut nokta hedef ise, algoritma sonlanır*	
14			***if*** *current == goal*	
15				***return*** *came_from, cost_so_far*
16			*// Geçerli noktayı arama listesinden kaldır*	
17			*frontier.remove(current)*	

```
18  // Geçerli noktanın komşularını arama listesine ekle
19  for each neighbor of current
20      // Komşuya ulaşmanın maliyetini hesapla
21      new_cost = cost_so_far[current] + cost(current, neighbor)
22      // Komşu ziyaret edilmemişse veya yeni maliyet önceki maliyetten düşükse, maliyeti güncelle ve komşuyu arama sınırına ekle
23      if neighbor not in cost_so_far or new_cost < cost_so_far[neighbor]
24          cost_so_far[neighbor] = new_cost
25          priority = new_cost + heuristic(neighbor, goal)
26          frontier.add(neighbor, priority)
27          came_from[neighbor] = current
```

Algoritma 1'de A* algoritmasının çalışma prensibini açıklayan kod örneği belirtilmiştir.



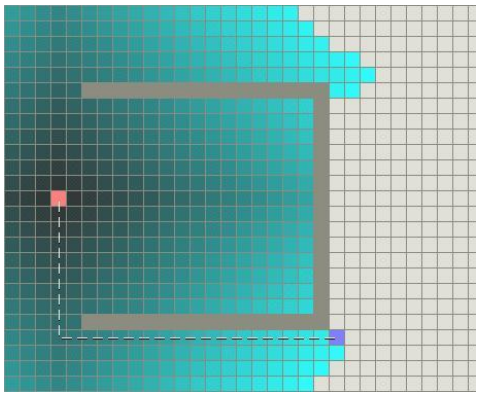
Şekil 3. A* yol bulma şeması
(A* algorithm pathfinding schema) [4]

Sezgisel çalışma prensibi sayesinde, A* algoritması ile engelsiz ortam koşullarında çok daha hızlı ve başarılı yol bulma işlemi gerçekleştirilmektedir. [5].

Karakterlerin oyun içi ortamda hedeflerine gitmesi için, engel oluşturabilecek objelerin varlığına göre yol hesaplama fonksiyonları kullanılır. Bu fonksiyonlar sürekli olarak çalışarak, karakterin hedefe gitmesi için en kısa yolu hesaplar. Ayrıca kaynak noktadan hedef noktaya ve hedef noktadan kaynak noktaya olan yollar da hesaplanır ve en kısa ve hızlı olan yol seçilir.

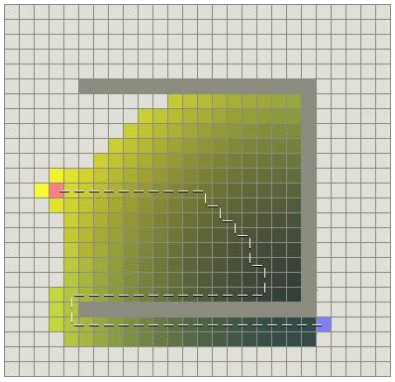
Kaynak nokta ve hedef nokta arasında engel oluştuğunda, yol bulma performansı için farklı algoritmalar kullanılabilir. Örneğin, Dijkstra algoritması yolun maliyetine bakarak en kısa yolu bulur, Best-First Search algoritması ise yolun hedefe olan yakınlığına bakarak en kısa yolu seçer ve A* algoritması ise hem maliyeti hem de hedefe olan yakınlığı dikkate alarak en kısa yolu hesaplar. Bu algoritmaların performansları arasında zaman ve maliyet açısından farklılıklar vardır.

Dijkstra algoritması ile engellerin olduğu ortamda yol bulma işlemi sırasında, algoritma tüm yönlerde arama yaparak doğru yolu bulmaya çalışır. Bu nedenle, zaman açısından büyük bir kayıp yaşanabilir. Örneğin, Şekil 4'de gösterildiği gibi, algoritma tüm yönleri taradıktan sonra doğru yol bulunmuş olsa da büyük bir alan taranmış ve vakit kaybedilmiştir.



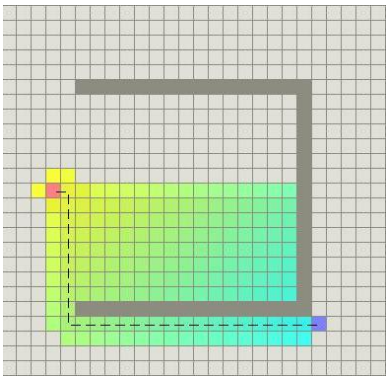
Şekil 4. Dijkstra algoritması engel ortamı
(Dijkstra algorithm environment with obstacles) [4]

Best-First Search algoritması, engel durumunda tarama yapacağı alanı azaltır, bu da Dijkstra algoritmasına göre daha hızlı bir hedefe ulaşma sağlar. Ancak, Best-First Search algoritması engeli fark edene kadar sezgisel hareket etmez, bu nedenle izlediği yol daha uzun olabilir. Bu noktada, Best-First Search algoritmasının bir avantajı hedefe hızlı ulaşmak için tarama alanını azaltmasıdır, ancak dezavantajı ise engeli fark edene kadar sezgisel bir hareket sergilememesidir [4]. Aynı konumlardaki başlangıç ve hedef noktaları arasındaki en kısa mesafe Şekil 5'de görüldüğü gibi yanlış hesaplanmıştır.



Şekil 5. Best-First Search engel ortamı
(Best-First Search environment with obstacles) [4]

A* algoritması, engel içeren ortam durumları karşısında sezgisel hareket sergiler. Bu sayede, diğer algoritmalardan farklı olarak, hedef yönündeki engeli önceden tayin edebilir. Sezgisel hareket etme özelliği sayesinde engel yapısına göre en az maliyetli yol hesaplanarak hem hızlı hem de az maliyetli bir şekilde en iyi yol belirlenebilir. Bu nedenle A* algoritması diğer algoritmalara göre daha efektif bir yol bulma yöntemi olarak kabul edilmektedir. Şekil 6'da A* algoritmasının aynı engel ortamında bulduğu yol ve aradığı noktalar görülmektedir.



Şekil 6. A* algoritması engel ortamı
(A* algorithm environment with obstacles) [4]

Şekil 4 ve Şekil 5'de görülen Dijkstra ve Best-First Search algoritmalarına göre daha az nokta taranmıştır. Buna rağmen Şekil 6'da görüldüğü gibi en kısa yolun bulunduğu anlaşılmaktadır.

Farklı ihtiyaçlara göre farklı akademik alanlarda, A* algoritmasının temel özelliklerini kullanarak geliştirilen farklı algoritmalar üzerinde çalışmalar yapılmıştır.

IDA* (Iterative Deepening A*, Yinelemeli Derinleşen A*), belirlenmiş bir başlangıç düğümü ile bir kümenin herhangi bir üyesi arasındaki en kısa yolu bulmak için yinelemeli derinleştirme öncelikli arama fikrini içeren A* algoritmasının bir çeşididir. A* 'ın optimallığını ve eksiksizliğini derinlemesine öncelikli aramanın düşük bellek gereksinimleriyle birleştirir. Bu, bir düğümün derinliği yerine ardışık iterasyonlarda hedefe ulaşmak için kalan maliyeti değerlendirmek için kullanılan buluşsal işleve daha büyük bir sınır koyarak yapılır [10].

SMA* (Simplified Memory-Bounded A*, Basitleştirilmiş Bellek Sınırlı A*), bir grafikteki en kısa yolu bulmak için kullanılan A* algoritmasına dayalı bir en kısa yol algoritmasıdır. SMA* ve A* arasındaki temel fark, SMA*'nın sınırlı bir bellek kullanması, A* algoritmasının ise daha fazla belleğe ihtiyaç duymasıdır. Sezgiye göre en umut verici dalları genişletmek için A* gibi benzer bir yöntem kullanır. Ancak SMA*, bellek kullanımını kontrol altında tutmak için genişletilecek açık düğümler listesinden en az umut vadeden dalları budar [11]

RBFS (Recursive Best-First Search, Özyinelemeli En İyi İlk Arama), A*'ya benzer bir algoritmadır, ancak düğümlerin genişletilmesini sıralamak için bir öncelik sırası kullanmak yerine, yinelemeyi kullanır. Bu, daha esnek bir f-değerinin uygulanmasına izin verir ve A* algoritmasının uygun olmadığı belirli durumlarda yararlı olabilir. RBFS (Recursive Best-First Search), A* aramasına benzer, ancak yinelemeli bir yaklaşım kullanan ve lineer uzay karmaşıklığına sahip bir arama algoritmasıdır. RBFS'nin arkasındaki fikir, düğümlerin genişletilme sırasını belirlemek için bir değerlendirme işlevi kullanan en iyi ilk arama algoritması olan A* aramasının çalışmasını taklit etmektir. A*'daki değerlendirme işlevi, yol maliyetini ve buluşsal yöntemi toplar, ancak RBFS algoritması, düğümleri en iyi birinci sıraya göre sıralamak için monoton olmayan bir maliyet işlevi kullanır [12].

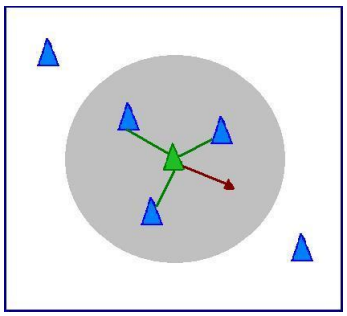
2.2 Akın Hareketi (Flocking)

Flocking, birçok nesnenin birlikte hareket ettiği bir sistemde, nesnelerin nasıl birlikte hareket ettiğini modellemenin bir yoludur. Flocking algoritması, nesnelerin izlediği üç temel kurallar üzerine inşa edilir: konsensüs, özdeşleşme ve sınırlama.

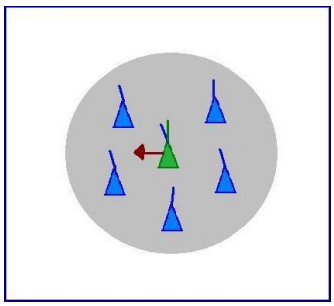
Konsensüs kuralı, nesnelerin ortalama pozisyon ve hızlarını izlemesini sağlar. Özdeşleşme kuralı, nesnelerin diğer nesnelerin yönünü izlemesini sağlar. Sınırlama kuralı, nesnelerin birbirlerine çok yaklaşmamasını sağlar [13].

Örneğin, bir kuş sürüsünün uçuş simülasyonu yapılmak istendiğinde, her bir kuşun pozisyonu ve hızı veri olarak alınır. Kuşlar, konsensüs, özdeşleşme ve sınırlama kurallarını izleyerek ortak bir düzende uçarlar. Bu yolla, sürünün bütünü hakkında bilgi elde edilebilir.

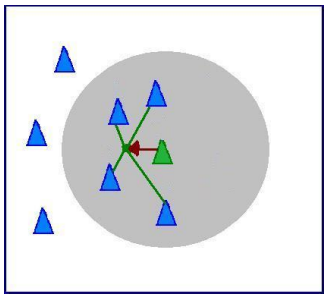
Sürüyü oluşturan nesneler (boids) komşulara ve engellere çarpışmadan kaçınmak (Şekil 7), hız ve doğrultuyu komşularla yaklaşık aynı tutmak (Şekil 8), komşulara yakın durmaya çalışmak (Şekil 9) kurallarını uygulayarak akın hareketi (sürü hareketi) işlemini gerçekleştirir.



Şekil 7. Komşulara ve engellere çarpışmadan kaçınmak
(Avoiding collisions with neighbors and obstacles) [13]



Şekil 8. Hız ve doğrultuyu komşularla aynı tutmak
(Keeping speed and direction the same as neighbors) [13]



Şekil 9. Komşulara yakın durmaya çalışmak
(Trying to stay close to neighbors) [13]

Sürü hiyerarşisi için belirlenen kurallar sonucunda, her bir nesnenin hız vektörü hesaplanmaktadır. Bu işlem, sürünün bütün nesnelerine uygulandığında, günlük hayatta gözlemlediğimiz sürü hareketlerine benzer bir hareket elde edilir. Bilgisayar oyunlarındaki karakterlerin oluşturduğu kalabalık grubun hareketi benzer bir şekilde modellenmiş olur.

Takım olarak hareket eden yapay zekâlı karakterlerin bir takım kaptanını takip ederken aynı zamanda takımdan kopmadan ve diğer takım üyelerine çarpmadan hareket etmeleri gerekir. Karakterlerin birbirlerine olan mesafelerini sürekli olarak kontrol edecek ve herhangi bir karakterin takımdan aşırı derecede uzaklaştığını tespit ettiğinde onu takıma geri çeken bir algoritma geliştirilebilir. Bir cisim hareketinin son yönü v, aşağıdaki gibi akın hareketi algoritmasının kuralları uygulanarak bu kuralların sonuçlarının ağırlıklı toplamı olarak hesaplanır:

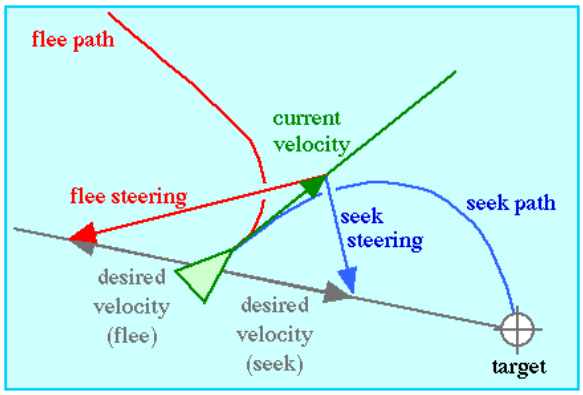
$$v = wref.vref + wcoh.vcoh + wsep.vsep \quad (2)$$

$vref$, hesaplanan sürü yolunun yönünü yansıtan referans vektörüdür. $vcoh = posavg - pozunit$, takım elemanlarının konumundan sürünün ortalama konumuna kadar olan vektördür. $vsep$, ayrılma vektörüdür. Karşılık gelen ağırlıklar $wref$, $wcoh$ ve $wsep$ olarak tanımlanmıştır. $|vsep|$, ayırma vektörünün uzunluğudur ve "Ayırma Mesafesi" olarak bilinir [14].

Şekil 10'da görüldüğü gibi, Seek (aramak veya statik bir hedefin peşinden gitmek), karakteri küresel uzayda belirli bir konuma yönlendirmek için hareket eder ve hızı hedefe doğru radyal olarak hizalanacak şekilde ayarlar.

Şekil 10' da görülen kavramların açıklamaları:

Flee path: Kaçış yolu.
Flee steering: Kaçış manevrası.
Current velocity: Mevcut hız vektörü.
Desired velocity (flee): Kaçış durumundaki amaçlanan hız vektörü.
Desired velocity (seek): Arama durumundaki amaçlanan hız vektörü.
Seek path: Arama yolu.
Seek steering: Arama manevrası.
Target: Hedef.



Şekil 10. Takım üyesine etki eden sürü hareketi kuvveti
(Flocking force acting on team member) [15]

Karakterden hedefe doğru olan vektöre istenen hız denir. Belirli uygulamaya bağlı olarak "istenen hızın" uzunluğu $\max\_speed$ veya karakterin mevcut hızı olabilir. Yönlendirme vektörü, bu istenen hız ile karakterin mevcut hızı arasındaki farktır.

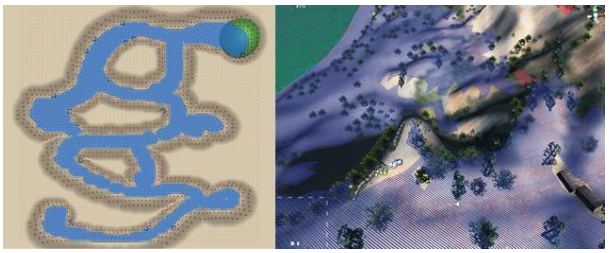
$$desired\_velocity = normalize(position - target).max\_speed \quad (3)$$

$$steering = desired\_velocity - velocity$$

Bir karakter aramaya devam ederse, sonunda hedefi geçecek ve sonra tekrar yaklaşmak için geri dönecek ve bir güvenin bir ampulün etrafında vızıldaması gibi hareket üretmeye devam edecektir. Kaçmak, basitçe aramanın tersidir ve karakteri yönlendirmek için hareket eder, böylece hızı hedeften radyal olarak hizalanır. İstenen hız zıt yönü işaret eder.

2.3. 3B Yüzey Analiz Haritası (3D Surface Analysis Map)

Yüzey analiz haritaları, Unity 3D oyun motorunda NavMesh (Navigation Mesh, Gezinme Ağı) adı verilen bir sistem ile oluşturulur. NavMesh, oyun içindeki nesnelerin ve engellerin yerleşimini analiz eder ve hareket alanlarını belirler. NavMesh, karakterlerin yol bulmasını sağlamak için bir grid (ızgara) grafik olarak görüntülenir ve oyun içinde kullanılır (Şekil 11).



Şekil 11. 3B yüzey analiz haritası
(3D surface analysis map)

Şekil 11'de görülen yüzey analiz haritası örneklerinde, mavi tonlar ile belirtilen alanlar 3B arazi üzerinde hareket edilebilecek alanları temsil etmektedir. A*

algoritmasının çalışması için gereken ızgara (grid) sistemi bu alanlar üzerindeki noktalar kullanılarak oluşturulmaktadır. A* algoritması ile ızgara üzerindeki

$$f(n) = g(n) + h(n) + m(n) \quad (5)$$

nokta (vertex) arası mesafelere göre yol bulma işlemi gerçekleştirilir.

2.4. 3B A* ve Akın Algoritmalarının Birlikte Kullanımı (Combining A* and Flocking Algorithms)

A* ve akın algoritmaları, farklı fiziksel ortamlar üzerinde konum belirleme amaçlı kullanılmaktadır. Kullanım amaçlarına göre farklılıkları aşağıdaki Tablo 1'de belirtilmiştir.

Tablo 1. A* ve Akın Algoritması Farklılıkları
(A* and Flocking Algorithm Differences)

A*	**Akın**
Grafik tabanlı (düğümler ile)	Serbest biçimli (kayan koordinatlar ile)
Optimum yörünge	Fiziksel yörünge (hız, atalet)
Bireysel	Grup
Genel bilgi kullanımı	Yerel bilgi kullanımı

A* ve sürü algoritmasını birlikte kullanarak grup halinde bir noktadan başka bir noktaya ulaşma işlemi gerçekleştirmek için, bireyin (node, boid) aynı anda A* algoritması ile yol bulurken diğer bireyler ile ortak hareket etmesi gerekmektedir. Algoritmaların kullanımında uygulanan adımlar aşağıda listelenmiştir:

- Bir makro hedef ve bu hedefe giden küresel bir yol belirlenir.
- İstenen küresel yol boyunca hareket edilecek yönlendirme davranışı için yerel bir izleme hedefi belirlenir.
- Birey fiziksel dünyadaki davranışlarına göre (nokta arama ve çarpışma) hareket ettirilir.
- Bireyin mevcut konumunu temsili grafik düğümüne eşlenir.
- Birey önceki düğümünden ayrıldıysa, yolu yeniden hesaplanır ve en yakın bireye göre arama yapılır.

Grup hareketi ağırlığının hesaplanması ve kullanılması için aşağıdaki gibi bir örnek eşitlik geliştirebilir:

$$m(n) = w * (1 - \mathrm{d}) \quad (4)$$

Burada; $m(n)$, grup hareketi ağırlığını temsil eder. Grup halindeki bireylerin birlikte hareket etme önceliğini veya etkisini gösterir.

w, bir ağırlık faktörüdür ve grup hareketine ne kadar önem verileceğini belirler. Bu faktörü isteğe bağlı olarak ayarlanabilir.d, bireyin diğer bireylerden olan uzaklığını gösteren değerdir. Bu değer, bireyin etrafındaki diğer bireylerin yoğunluğunu temsil eder.Bu durumda geliştirilen yol bulma algoritmasın matematiksel formülü aşağıdaki gibi temsil edilebilir;

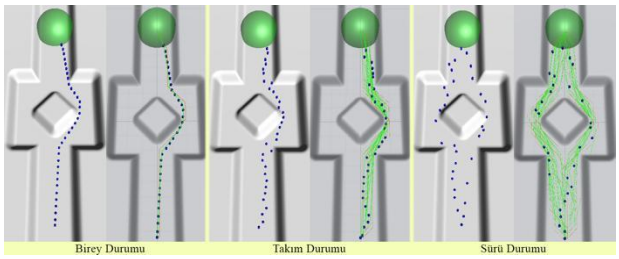
Geliştirilen yol bulma algoritmasın sözde kodu Algoritma 2'deki gibi temsil edilebilir;

Algoritma 2: Geliştirilen Yol Bulma Algoritması	
1	**function** *findPath(startNode, goalNode, otherNodes, w)*
2	*let globalPath = AStar(startNode, goalNode);*
3	*let localTarget = getNextTarget(startNode, globalPath);*
4	*moveBoid(startNode, localTarget, otherNodes, w);*
5	*mapNodeToGraph(startNode);*
6	**if** *(hasLeftPreviousNode(startNode))*
7	*globalPath = AStar(startNode, findNearestBoid(startNode, otherNodes));*
8	*localTarget = getNextTarget(startNode, globalPath);*
9	*moveBoid(startNode, localTarget, otherNodes, w);*
10	*mapNodeToGraph(startNode);*
11	**function** *AStar(startNode, goalNode)*
12	*let fScore = gScore + hScore;// A* algoritması ile startNode'dan goalNode'a giden yol bulunur // ve bu yolun bir listesi olan path döndürülür.*
13	**function** *getNextTarget(currentNode, path)*
14	*let fScore = gScore + hScore;// İstenen küresel yol boyunca hareket edilecek yönlendirme davranışı için // currentNode'in konumuna göre path'teki bir sonraki hedef düğüm döndürülür.*
15	**function** *moveBoid(currentNode, targetNode, otherNodes, w)*
16	*let d = calculateDistance(currentNode, otherNodes); // Diğer bireylere olan uzaklık hesaplanır*
17	*let m = w * (1 - d); // Grup hareketi ağırlığı hesaplanır*
18	*let fScore = gScore + hScore + m; // Toplam maliyet hesaplanır // Bireyin fiziksel dünyada hareket etmesini sağlayan işlemler gerçekleştirilir, // grup hareketi ağırlığı kullanılır.*
19	**function** *mapNodeToGraph(node)*
20	*let fScore = gScore + hScore;// Bireyin mevcut konumunu temsili grafik düğümüne eşlemek için gerekli işlemler yapılır.*
21	**function** *hasLeftPreviousNode(node)*
22	*let fScore = gScore + hScore;// Bireyin önceki düğümünden ayrılıp ayrılmadığı kontrol edilir. // Örneğin, bir çarpışma sonucu birey yön değiştirdiyse true, aksi takdirde false döndürülür.*
23	**function** *findNearestBoid(currentNode, otherNodes)*
24	*let fScore = gScore + hScore;// currentNode'e en yakın bireyi diğer bireyler listesinden bulup döndürür.*
25	**function** *calculateDistance(currentNode, otherNodes)*
26	*let fScore = gScore + hScore; // currentNode ile diğer bireyler arasındaki uzaklığı hesaplar ve döndürür.*

Bu güncellenmiş sözde kodda, w değeri grup hareketi ağırlık faktörünü temsil ederken, grup hareketi ağırlığını hesaplamak için d değeri kullanılır. m Değeri grup hareketi ağırlığını ifade eder ve $fScore$ hesaplamasına eklenir.

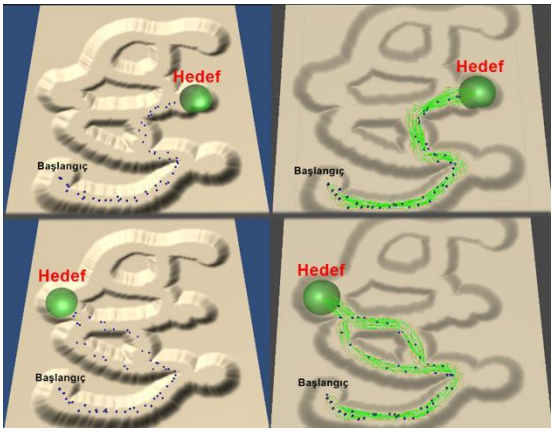
Bireylerin kalabalık gruplar halinde hareket etmesi durumunda yol üzerinde bazı sıkışma durumları gerçekleşebilmektedir. Sıkışma durumlarını engellemek için bireylerin birbirlerine olan mesafelerine kısıtlama getirilerek fazla yaklaşma durumlarında itici bir güç parametresi kullanılması gerekmektedir. Bireyler arası mesafe için kullanılan parametreye benzer olarak yürünemez bölgeler için de bir yaklaşma parametresi kullanılabilir.

Şekil 12' de bireysel olarak hareket eden birey ile takım ve sürü halinde hareket eden bireylerin yol bulma çalışması örneği gösterilmektedir.



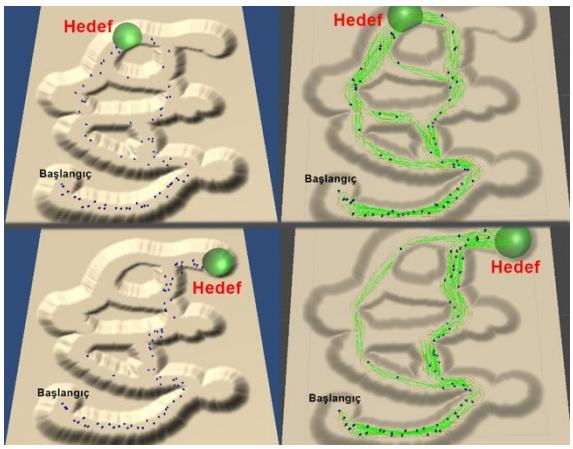
Şekil 12. A* ve sürü algoritmasının birlikte kullanımı
(Combined use of A* and flocking algorithm)

Bireylerin birbiri ile arasındaki mesafeye göre hareket ederken, engellere çarpmadan en kısa yoldan hedefe ulaşması için farklı ortamlarda hareket durumları test edilmiştir. Bireyin fiziksel özellikleri (ebat ve hareket hızı), ortamdaki pozisyonuna göre diğer bireylere göre seçtiği yolu değiştirebilmektedir. Bu şekilde her birey ortak bir hedefe giderken birebir aynı yolu kullanmak yerine farklı ama kendine göre en kısa ve hızlı yolu seçerek hareket edebilmektedir. Şekil 13'de bireylerin sürü halinde hareket ederken, yeşil çizgiler olarak belirtilen farklı yollar üzerinden istenen hedefe ulaştıkları gösterilmiştir.



Şekil 13. Sürü halinde hareket etme durumu
(The state of moving in a flock)

Bireylerin yol üzerinde karşılaştıkları engellerin etrafından dolaşarak istenen hedefe farklı yollardan ulaşabildikleri ve trafik oluşturmadan karmaşık yollardan geçmeleri durumu Şekil 14'de gösterilmiştir.



Şekil 14. Sürü halinde karmaşık yol durumu
(Complex road situation in flock)

2.5. A ve Akın Algoritması ile Takım Oyunu (Team Play with A* and Flocking Algorithm)*

Takım olarak hareket eden yapay zekalı karakterlerin bir takım kaptanını takip ederken aynı zamanda takımdan kopmadan ve diğer takım üyelerine çarpmadan hareket etmeleri için birkaç farklı yöntem bir arada kullanılmaktadır. Öncelikle, A* algoritmasını kullanarak hedef noktasına ulaşan takım kaptanının diğer takım üyeleri tarafından takip edilmesini sağlayan bir yönlendirme algoritması geliştirilmiştir. Bu algoritma, karakterlerin yönünü takım kaptanının yönüne göre ayarlayarak onları takip etmeye zorlar. Daha sonra, takımdan kopmadan hareket etmelerini sağlamak için bir takım bütünlüğü algoritması geliştirilmiştir. Akın hareketi kurallarına göre geliştirilen bu algoritma, karakterlerin birbirlerine olan mesafelerini sürekli olarak kontrol eder ve herhangi bir karakterin takımdan aşırı derecede uzaklaştığını tespit ettiğinde onu takıma geri çekmeye çalışır. Son olarak, karakterlerin diğer takım üyelerine çarpmadan hareket etmelerini sağlamak için bir çarpışma önleme algoritması geliştirilmiştir. Bu algoritma, karakterlerin birbirleriyle olan mesafelerini sürekli olarak kontrol eder ve birbirlerine çok yaklaştıklarını tespit ettiğinde takım üyelerinin yönlerini değiştirerek çarpışmalarını önler [16].

Düşman takımının hareket etme sistemi, belirli bir takım lideri tarafından belirlenir. Her birey, karşı takımın askerlerini sürekli olarak arar ve bir karşı askeri bulduğunda, ateş etme menziline girene kadar yaklaşır ve atışa başlar. Düşman takımının ateş etme önceliği, oyunu oynayan kullanıcının kontrol ettiği karakterdir. Eğer kullanıcının kontrol ettiği karakter bulunmuyorsa, diğer takım üyelerine ateş edilir.

Tablo 2'de iki takım arasındaki mücadele sırasında kullanılan yapay zekâ sisteminin dinamikleri ve uygulama aşamaları yer almaktadır.

Tablo 2. Sistem Dinamikleri ve Aşamaları
(System Dynamics and Stages)

Dinamik Türü	Açıklaması
EnemyRespawn_team	Takım arkadaşı oluşturulması
EnemyRespawn_enemy	Düşman kuvveti oluşturulması
AIFollowUSE	Takım liderinin takip edilmesi
Seeker	Yüzey analiz haritasının çıkarılması
AstarPath	Eğim ve engellere göre yol bulunması
FunnelModifier	Bulunan yolun parabolik hale getirilmesi
SimpleSmooth	Bulunan yolun yumuşatılması

İki farklı zekâ seviyesine sahip yapay zekâlı karakter takımı savaş ortamında karşılaşırlarsa, daha yüksek zekâ seviyesine sahip olan takım daha etkili ve uygun stratejiler geliştirerek daha az kayıp verirken, daha düşük zekâ seviyesine sahip olan takım daha az etkili stratejilerle daha fazla kayıp verir. Aynı zamanda daha yüksek zekâ seviyesine sahip olan takım diğer takımın hamlelerini daha kolay analiz ederek karşı hamleler geliştirebilir.

Bayrak kapma oyunun, yapay zekâlı oyun karakterleri işaretlenmiş bölgeleri ele geçirmek için, taktikler geliştirebilir ve diğer takımla mücadele etmek için çeşitli oyun mekaniklerini kullanabilirler. Örneğin:

- Karakterler, bölgenin çevresindeki bariyerleri kullanarak saldırıya karşı korunma sağlayabilir.
- Karakterler, öncü bir takım oluşturarak bölgenin önünde hareket edebilir.
- Karakterler, bölgenin çevresindeki diğer bölgelerin kontrolünü elde etmek için taktikler geliştirebilir.
- Karakterler, diğer oyuncularla birlikte bölge içindeki diğer oyunculardan ve oyun mekaniklerinden gelen bilgileri analiz edip hızlı bir şekilde kararlar alarak oyunun dinamiklerini değiştirebilir.
- Karakterler, bölgenin çevresindeki diğer oyuncularla iş birliği yaparak saldırıya karşı korunma sağlayabilir.

Bayrak kapma oyunu ortamı için belirlenen zekâ seviyelerine göre karakterlerin özellikleri Tablo 3'de gösterilmiştir [17]. Takımlar harita üzerinde belirlenen bölgeleri ele geçirmeye çalışmaktadırlar. Yapay zekâlı karakterler, bölge ele geçirme mücadelesi sırasında oyun haritası ortamı ve yaşanan durumlara göre farklı hareket ederek galip gelmeye çalışır.

Tablo 3. Karakter Yapay Zekâ Özellikleri
(Character AI Traits)

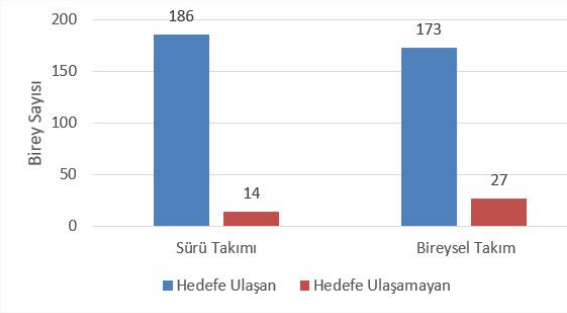
Karakter Türü	Açıklaması
OpportunisticBot (fırsatçı bot)	Karakter sürekli bölgeleri gezer. Karşı takıma ait olan bölgeye saldırır.
PossesiveBot (sahip çıkan bot)	Karakter tek bir bölgeye gider ve o bölgeyi ele geçirip savunur.
GreedyBot (aç gözlü bot)	Kaybedilen bölgeyi tekrar ele geçirmeye çalışır.

3. ARAŞTIRMA BULGULARI (RESEARCH FINDINGS)

Bu bölümde çalışma sırasında geliştirilen uygulamadan elde edilen araştırma bulguları sunulmuştur.

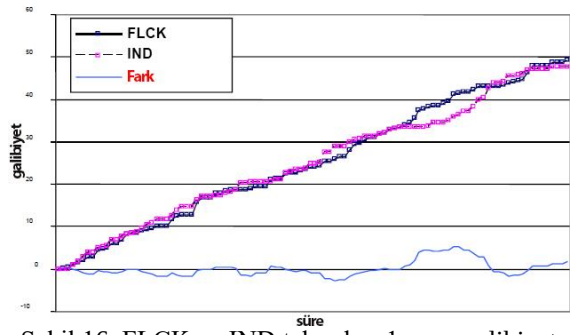
3.1. Bireysel ve Akın Takımlarının Yol Bulma Karşılaştırmaları (Pathfinding Comparisons of Individual and Flock Teams)

Bireysel ve akın halinde hareket eden takımlar olarak ikiye ayrılan takımların, 100 saniye içinde 200 adet bireyden hedefe ulaşan birey adeti karşılaştırmaları yapılmıştır. Şekil 17'de görüldüğü gibi, sürü halinde hareket eden bireyler daha yüksek oranda hedefe ulaşmıştır. Hareket edilen yol karmaşıklaştıkça sürü halinde hareket eden bireylerin başarı oranı artmaktadır.



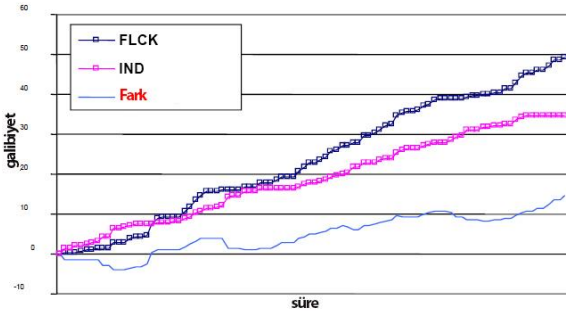
Şekil 15. Bireysel ve sürü takımlarının hedefe ulaşma durumu
(Achievement of individual and flock teams)

Şekil 18'de görülen sonuçlara göre "IND" isimli karakter takımı ile "FLCK" isimli rakip takımın her ikisi de ilk maçta bireysel olarak hedefine ulaşmaya çalışırken, kazanma oranları arasındaki fark azdır. İlk maçta her iki takım karakterleri için aynı özelliklere sahip kural tabanlı yapay zekâ sistemi kullanılmıştır. Dijksta algoritması ile yol bulan karakterler, takım halinde hareket etmek için sadece takım liderini takip etmişlerdir.



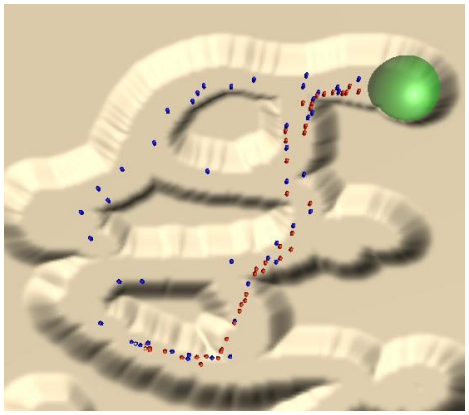
Şekil 16. FLCK ve IND takımları 1. maç galibiyet grafiği
(Game 1 win chart of FLCK and IND teams)

İlk maç sonucunda, iki takımın da dengeli bir karşılaşma sonucu benzer bir başarı oranı elde ettiği görülmüştür. İkinci maçta ise "FLCK" takımı için, A* yol bulma algoritmasını kullanan ve sürü halinde hareket eden daha gelişmiş yapay zekâ sistemi kullanılmıştır. İlk maçta görülen karakter davranışlarının ve kullanılan yolların dışında hareket eden "FLCK" takımı, daha başarılı bir sonuç elde etmiştir.



Şekil 17. FLCK ve IND takımları 2. maç galibiyet grafiği
(Game 2 win chart of FLCK and IND teams)

"FLCK" takımı, ikinci maçın başladığı andan itibaren, rakip takımın davranışlarından farklı olarak haritaya daha farklı bölgelerden yayılmış ve hedefe ulaşmada daha başarılı olmuştur. Şekil 20'de mavi renk ile simgelenen "FLCK" takımı bireyleri ile kırmızı renk ile simgelenen "IND" takımı bireylerinin harita üzerindeki yayılmaları gösterilmektedir.



Şekil 18. FLCK ve IND takımları harita yayılımı
(FLCK and IND teams map spread)

A* ve akın hareketi algoritmalarını bir arada kullanmak, takım halinde hareket eden oyun karakterleri için oldukça gerçekçi ve inandırıcı bir hareket sağlayabilmiştir. A* algoritması bir hedefe giden en uygun yolu bulmak için kullanılabilirken, akın hareketi algoritması hareket sırasında grup davranışını ve oluşumunu kontrol etmek için kullanılmıştır.

4. TARTIŞMA ve SONUÇLAR (DISCUSSION and CONCLUSIONS)

Takım halinde hareket eden oyun karakterleri için yapay zekâ sistemi geliştirilmiştir. Bu sistemde kullanıcı, takım lideri olarak kabul edilir ve rakip takım ile savaşmak için kullanılır. A* algoritması, takım içinde haberleşebilen karakterlerin yol bulma becerilerini optimize etmek için kullanılmıştır. Bu sayede, oyun karakterleri daha etkili bir şekilde hareket edebilir ve rakip takım karşısında daha etkili bir şekilde savaşabilir.

A* algoritması, Dijkstra ve Best-First Search algoritmalarına kıyasla zaman ve maliyet açısından daha etkili bir çözüm sunmaktadır. Bu sonuç, işlemci yükünü azaltarak hızlı ve sağlıklı çözümler sağlamasından kaynaklanmaktadır. Bu algoritmanın zamanlama sonuçları, oyun içi rakip takımların karşılaşması ve savaşması durumunda gerçekleşen bekleme ve saldırma gibi olaylar için karşılaştırılmıştır. Bu karşılaştırma sonucunda, A* algoritmasının kullanılmasının olayların çok daha kısa ve etkili bir şekilde gerçekleştirildiği gözlemlenmiştir.

Birçok senaryoda, takım halinde hedeflere ulaşmanın önemli bir bileşeni olan yol bulma için A* ve akın hareketi algoritmalarının kombinasyonunun kullanılması, daha gelişmiş bir yapay zekâ sistemi elde edilmesini sağlamaktadır. Bu konuda yapılan çalışmalarımızda, standart yol bulma algoritmalarını kullanan ve bireysel hareket eden karakterlere sahip olan bir takım ile A* algoritması ve akın algoritmasını bir arada kullanan bir takımın bayrak kapma oyun tipindeki mücadelesi gözlemlenmiştir. Sonuçlar, A* ve akın algoritmalarını kullanan takımın daha başarılı sonuçlar elde ettiğini göstermektedir.

Yapay zekâlı karakterlerin daha gerçekçi hareket etmesi için farklı alanlarda çeşitli çalışmalar yapılmaktadır. Özellikle, takım halinde ortak bir hedefe ulaşabilen karakterlerin yapay zekâ sistemi ne kadar iyi olursa, oyun içi kullanıcı deneyimi o kadar iyi olacaktır. Buradan hareketle gelişmiş takım tabanlı algoritma sistemleri üzerinde çalışarak, yeni nesil yol bulma ve sürü hareketi sistemleri geliştirilebilecektir.

KAYNAKLAR (REFERENCES)

[1] Kocabaş, Ş. ve Öztemel, E., "AISim: An Intelligent Agent for Distributed Interactive Simulation". *Seventh Computer Generated Forces and Behavioral Representation*, 12-15 May 1998, Orlando, Florida, USA.

[2] Kent, S., "The Ultimate History of Video Games", *Prima Publishing*, USA, 608p., 2001.

[3] Foeada, D., vd., "A Systematic Literature Review of A* Pathfinding", *Procedia Computer Sciences*, c. 179, ss. 507-514, Oca. 2021, doi: 10.1016/j.procs.2021.01.034.

[4] Patel, A., "Heuristics" "http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html", Erişim Tarihi:10.05.2021.

[5] Patel, A., "Heuristics" "http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html", Erişim Tarihi:10.05.2021.

[6] Delling, D., Sanders, P., Schultes, D., ve Wagner, D., "Engineering Route Planning Algorithms", *Algorithmics of Large and Complex Networks - Design, Analysis and Simulation*, 2009, ss. 117–139, doi: 10.1007/978-3-642-02094-0_7.

[7] Zafar, Z., ve Kant, K, "Novel Optimization using Hierarchical Path Finding A* (HPA*) Algorithm for Strategic Gaming Setup", *International Journal of Engineering & Technology*, c. 7, ss. 54-57, Mar. 2018, doi: 10.14419/ijet.v7i2.6.10067.

[8] Gunda, R., A* Search Algorithm. "https://ramahanishagunda.medium.com/a-search-algorithm-8233683c5d60", Erişim Tarihi: 28.05.2021.

[9] Rafiq, A., Kadir, T., ve Normaziah, I., "Pathfinding Algorithms in Game Development", *IOP Conf. Ser.: Mater. Sci. Eng*., ss. 769, Oca. 2021, doi: 10.1088/1757-899X/769/1/012021.

[10] Jiang, Wenrong., "Analysis of Iterative Deepening A* Algorithm", *IOP Conference Series. Earth and Environmental Science*, 2021, Vol. 693, Iss. 1, doi: 10.1088/1755-1315/693/1/012028.

[11] Lovinger, Justin and Xiaoqin Zhang. "Enhanced Simplified Memory-bounded A Star (SMA*+)." GCAI (2017).

[12] Rodler, P., "RBF-HS: Recursive Best-First Hitting Set Search", *Artificial Intelligence, Elsevier*, 2022, doi: 10.48550/arXiv.2010.04282.

[13] Reynolds, C., Boids. "http://www.red3d.com/cwr/boids/", Erişim Tarihi: 08.06.2021.

[14] H. Fathy, O. A. Raouf ve H. Abdelkader, "Flocking Behaviour of Group Movement in Real Strategy Games", **9th International Conference on Informatics and Systems**, ss. PDC-64-PDC-67, 2014, doi: 10.1109/INFOS.2014.7036679.

[15] Reynolds, C., Steering Behaviors For Autonomous Characters. "http://www.red3d.com/cwr/steer/gdc99/", Erişim Tarihi: 12.11.2022.

[16] D. Sigurdson, V. Bulitko, W. Yeoh, C. Hernández ve S. Koenig, "Multi-Agent Pathfinding with Real-Time Heuristic Search", **IEEE Conference on Computational Intelligence and Games (CIG)**, 2018, ss. 1-8, doi: 10.1109/CIG.2018.8490436.

[17] Lee-Urban S., Vasta, M., Munoz-Avila, H., "RETALIATE: Learning Winning Policies in First-Person Shooter Games", *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*, Tem. 2007.