T.C.
SÜLEYMAN DEMİREL ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

A* ALGORİTMASI İLE TAKIM TABANLI YAPAY ZEKÂ MODÜLÜ GELİŞTİRİLMESİ

Ali Murat SÜMEN

Danışman
Yrd. Doç. Dr. Mehmet ALBAYRAK

YÜKSEK LİSANS TEZİ
ELEKTRONİK-BİLGİSAYAR EĞİTİMİ ANABİLİM DALI
ISPARTA - 2014

TEZ ONAYI

Ali Murat SÜMEN tarafından hazırlanan **"A* Algoritması ile Takım Tabanlı Yapay Zekâ Modülü Geliştirilmesi"** adlı tez çalışması aşağıdaki jüri üyeleri önünde Süleyman Demirel Üniversitesi Fen Bilimleri Enstitüsü **Elektronik-Bilgisayar Eğitimi Anabilim Dalı**'nda **YÜKSEK LİSANS TEZİ** olarak başarı ile savunulmuştur.

Danışman	**Yrd. Doç. Dr. Mehmet ALBAYRAK** Süleyman Demirel Üniversitesi	.....
Jüri Üyesi	**Doç. Dr. Ecir Uğur KÜÇÜKSİLLE** Süleyman Demirel Üniversitesi	.....
Jüri Üyesi	**Doç. Dr. Okan BİNGÖL** Süleyman Demirel Üniversitesi	.....

Enstitü Müdürü **Prof. Dr. Ahmet ŞAHİNER** .....

TAAHHÜTNAME

Bu tezin akademik ve etik kurallara uygun olarak yazıldığını ve kullanılan tüm literatür bilgilerinin referans gösterilerek tezde yer aldığını beyan ederim.

Ali Murat SÜMEN

İÇİNDEKİLER

ÖZET

Yüksek Lisans Tezi

A* ALGORİTMASI İLE TAKIM TABANLI YAPAY ZEKÂ MODÜLÜ GELİŞTİRİLMESİ

Ali Murat SÜMEN

Süleyman Demirel Üniversitesi
Fen Bilimleri Enstitüsü
Elektronik-Bilgisayar Eğitimi Anabilim Dalı

Danışman: Yrd. Doç. Dr. Mehmet ALBAYRAK

Bu tez çalışmasında; bilgisayar oyunlarının tarihi incelenerek, bilgisayar oyunları karakterlerinin AI (Artificial Intelligance, Yapay Zekâ) sistemleri analiz edilmiştir. Savaş temalı ve yapay yaşam temalı oyunların sahip oldukları gelişmiş yapay zekâ sistemlerinden olan Team-Based AI (Takım Tabanlı Yapay Zekâ) sistemi araştırılmıştır. FPS (First Person Shooter, Birinci Şahıs Nişancı) türündeki oyunların takım tabanlı yapay zekâ sistemlerinin çalışmaları analiz edilmiştir. Bu sistemlerin kullandığı Dijkstra, Best-First Search (en iyi ilk arama) ve A* (A Star, A Yıldız) gibi yol bulma (pathfinding) algoritmaları karşılaştırılmış ve A* algoritmasının en hızlı ve etkili yol bulma algoritması olduğu görülmüştür.

A* algoritması ile yeni nesil oyunlardaki yapay zekâlı karakterlerin oluşturduğu takımların birlikte hareket ederek değişken oyun içi durumlara karşı adapte olup strateji değiştirebildikleri tespit edilmiştir.

Çalışma kapsamında, savaş temalı örnek bir FPS oyunun takım tabanlı yapay zekâ modülü geliştirilmiştir. Bu modül Unity 3D oyun motoru ile oluşturulmuştur. Geliştirilen modül çalıştırılarak kullanıcının lider konumda olduğu takım ile düşman takımın karşılaşma durumunda sergiledikleri davranışlar gözlemlenmiştir.

Oyun içi oluşabilecek her türlü durum karşısında yeni stratejiler belirleyerek öğrenme becerisi gösteren karakter yapay zekâ sistemleri üzerinde çalışılmıştır. Bu çalışmaların doğrultusunda, A* algoritmasını kullanarak geliştirilen takım tabanlı yapay zekâ sistemleri ile, dinamik ve öğrenebilen oyun karakterlerinin gerçekçi hareketler sergileyerek oyun içi gerçekçiliği üst düzeye taşıdığı sonucuna ulaşılmıştır.

Anahtar Kelimeler: A*, takım tabanlı yapay zekâ, bilgisayar oyunları, bilgisayar oyunu karakterleri, yol bulma algoritmaları.

2014, 80 sayfa

ABSTRACT

M.Sc. Thesis

DEVELOPING TEAM-BASED ARTIFICIAL INTELLIGENCE MODULE WITH A* ALGORITHM

Ali Murat SÜMEN

Süleyman Demirel University
Graduate School of Applied and Natural Sciences
Department of Electronics-Computer Education

Supervisor: Asst. Prof. Dr. Mehmet ALBAYRAK

In this thesis the history of the of the computer games were studied and AI systems of the computer games characters were analyzed. Team-Based AI system was investigated that is one of the advanced artificial intelligence system of the war-themed and artificial life-themed games. Operations of the First Person Shooter game's Team Based AI systems were analyzed. Advanced pathfinding algorithms of this systems like the A*, Dijkstra and Best-First Search Algorithms were compared and it was seen that A* algorithm is the best efficent and fastest patfinding algorithm.

It was identified that AI character teams of the new generation games with A* algorithm, could change their strategies to adapt to variable game situations with working together.

With this study, Team-Based AI module of an example war themed FPS game is developed. This module is created with Unity 3D game engine. User ruled team and enemy team's fighting situations behaviors were observed with running this module.

Character artificial intelligence systems were studied which are capable of learning to identify new strategies for any in-game situaitons. With these studies it was concluded that the elements such as playability and realism that are indispensable for the next generation games can move to the next level with dynamic game characters with team-based artificial intelligence using A* algorithm.

Keywords: A*, team-based artificial intelligence, computer games, computer game characters, pathfinding algorithms.

2014, 80 pages

TEŞEKKÜR

Çalışmalarım boyunca bilgi birikimi ile beni yönlendiren danışmanım Yrd. Doç. Dr. Mehmet ALBAYRAK'a sonsuz teşekkürlerimi sunarım.

Tezimin her aşamasında beni yalnız bırakmayan aileme ve arkadaşlarıma sonsuz sevgi ve saygılarımı sunarım.

Ali Murat SÜMEN
ISPARTA, 2014

ŞEKİLLER DİZİNİ

ÇİZELGELER DİZİNİ

Sayfa

SİMGELER VE KISALTMALAR DİZİNİ

AI	Yapay zekâ
A*	A Yıldız algoritması
Bot	Yapay zekâlı karakter
CD	Kompakt disk
CGA	Renk grafik uyarlayıcısı
CPU	Merkez işlem birimi
EGA	Geliştirilmiş grafik uyarlayıcısı
FPS	Birinci şahıs nişancı
f(n)	Sezgisel fonksiyon
GPU	Grafik işlem birimi
g(n)	Maliyet
h(n)	Mesafe
IDA*	Tekrarlamalı derinleşen A Yıldız algoritması
MA*	Hafıza sınırlamalı A Yıldız algoritması
PC	Kişisel bilgisayar
RBFS	Yinelemeli en iyi ilk arama algoritması
RPG	Rol yapma oyunu
SMA*	Basitleştirilmiş hafıza sınırlamalı A Yıldız algoritması
SVGA	Süper video grafik dizisi
VGA	Video grafik dizisi
3D	Üç boyutlu

1. GİRİŞ

Gelişen teknoloji ve hızla büyüyen bilgisayar oyunları sektöründe kullanılan programlama teknikleri arasında en önemli konulardan biri olan karakter yapay zekâsı konusu ele alınarak, oyun içi yapay zekâ kullanımı ve bilgisayar oyunlarında karşımıza çıkan yapay zekâ tekniklerinin araştırması yapılmıştır. Bu araştırmalar sonucunda, bilgisayar ve video oyunlarının gelişimi ile birlikte klasik ve modern oyunların programlanmasında yapay zekâ kullanımı farklılıkları incelenmiştir.

Bilgisayar ve video oyunları pazarı son yıllarda giderek ivmelenen bir gelişme göstermeye başlamıştır. Bunun temel nedenlerinden birisi gelişmiş oyunların artık sadece belirli oyun salonlarından çıkıp masaüstü sistemlerin ve kullanılan özelleşmiş grafik hızlandırıcıları ile hemen her alanda karşımıza çıkmaya başlamasıdır (Russel ve Norvig, 1995). Hemen her yıl ikiye katlanan işlemci ve grafik işleyici yongaların hızı oyun konusundaki rekabeti körüklemiştir. Özellikle son yıllarda yeni pazarlar arayışına giren dev şirketlerin oyun programlama ve özelleşmiş oyun konsolu tasarımına ağırlık vermeleri de pazardaki rekabetin ve potansiyelin açık bir göstergesidir (Kent , 2001).

Yıllar geçtikçe işlem gücünün artması sadece oyunların daha hızlı çalışmasını değil oyunların biçimlerini de değiştirmeye başlamıştır (Pratt, 1996). Bilgisayar oyunlarının ilk zamanlarında sadece hamle tabanlı satranç ve dama gibi basit zekâ oyunları ile düşük kaliteli iki boyutlu karakterlerin tekdüze bir mantığa göre hareketlerine dayanan oyunlar mevcuttu. Günümüz bilgisayar oyunlarında gerçek zamanlı ve neredeyse gerçeğe yakın görünümdeki ortamlarda onlarca karakterin birbiri ile insan davranışına yakın hareketlerle etkileşimi sergilenmektedir (Kocabaş ve Öztemel, 1998). Oyunların ve oyun şirketlerinin her geçen gün artması da kullanıcıların daha gerçekçi grafik ve daha akıllı oyun karakterlerine olan isteğini arttırmıştır. Bu noktada ise oyun programlamanın en zor ve en etkileyici yönlerinden yapay zekâ devreye girer.

Oyun programlamada kullanılan yapay zekânın amacı oyuncu ile etkileşimde bulunan karakterlerin ya da oyunun geçtiği ortamın mümkün olduğunca gerçek insan, topluluk ve dünya (kimyasal-fiziksel-biyolojik olarak anlamlı ortam) yaşam ortamına benzetilmeye çalışılmasıdır (Kocabaş ve Öztemel, 1998). Başka bir anlamda da bilgisayarın yetenekli bir oyuncuya aynı derecede yetenekli karşılık vermesi, oyuncunun kurduğu planlara benzeyen planlar ya da tuzaklar kurması, gerektiğinde oyuncuyu zor duruma düşürüp onu yenebilmesi oyun programındaki yapay zekâ kalitesini belirler (Kent, 2001).

Ancak burada gözden kaçırılmaması gereken bir nokta oyunlardaki yapay zekâ dozunun iyi ayarlanmasıdır. Eğer bilgisayarı oyuncuya karşı çok hızlı reaksiyon verecek şekilde ayarlarsanız ya da oyuncuyu bunaltacak şekilde iyi taktikler gerçekleştirirse oyunun "oynanabilirlik" seviyesi düşer ve kaçınılmaz bir pazar başarısızlığı ortaya çıkabilir. Bunun önüne geçmek için oyunlarda çeşitli yöntemler izlenir. Bunlardan bir tanesi oyuna "seviye" özelliği eklenir ve oyuncu bilgisayarın ne kadar "zeki" davranacağını kendisi belirler. Bütün bu gerekçelerden dolayı modern oyunlarda günümüz akademik yapay zekâ araştırmalarının popüler konularından olan yapay sinir ağları ve genetik algoritmalar gibi konular birkaç oyun türü hariç halen yeni oyunlarda tercih edilmemektedir (Russel ve Norvig, 1995).

Savaş temalı ve yapay yaşam temalı bilgisayar oyunlarında karşımıza çıkan oyun karakterlerinin daha çok takımlar halinde haraket ettiği görülür. Temel olarak bireysel yapay zekâ sistemlerine göre hareket eden karakterlerin takım halinde hareket etmeleri istendiği zaman daha gelişmiş algoritmalar ile takım tabanlı yapay zekâ sistemleri geliştirilmelidir. FPS tarzında olan savaş oyunlarında sürekli olarak aynı oyun deneyimini yaşamak kullanıcıların oyundan sıkılmasına neden olacağından, oyun içi değişen şartlar ve durumlar karşısında sürekli olarak strateji değiştirerek hareket eden yapay zekâlı karakterleri görmek istemektedirler.

Oyun geliştiren firmalar, geliştirdikleri oyunlarda dinamik olarak değişen oyun koşulları karşısında dinamik olarak davranış değiştirebilme özelliğine sahip

olan yapay zekâ sistemleri geliştirebilmek için yoğun bir şekilde çalışmaktadırlar. Yeni nesil oyunlarda görülen aynı oyun bölümlerini oynarken farklı oyun deneyimleri yaşamak bu tür oyunları daha cazip hale getirerek bilgisayar oyunu tutkunları arasında popüler hale getirmektedir.

Bu tez çalışmasında, 3D bilgisayar oyunlarında kullanılabilecek, A* algoritması ile takım tabanlı çalışan entegre bir yapay zekâ sistemi modülü üretilmiştir. Üretilen bu modül, bilgisayar oyunu geliştiren firmaların kullandıkları oyun motoru sistemlerine otomatik olarak yüklenerek çalıştırılabilecektir. Bu sayede zaman ve işlem gücünden tasarruf sağlanmaktadır. Modül için tasarlanmış hazır fonksiyonlar kullanılarak yapay zekâ ayarlamaları geliştirilebilir ve yapay zekâ yazılımcısı kolay ve hızlı bir şekilde takım halinde hareket eden karakterlerin yönetimini sağlayabilir.

2. KAYNAK ÖZETLERİ

Young (1988) çalışmalarında bilgisayar oyunlarının gelişiminde çok önemli etkileri olan Nolan Bushnell' ın hayatı ve geliştirdiği sistemler üzerinde durmuştur. Nolan Bushnell, Atari gibi bilgisayar oyunu sektörünün en büyük firmalarından biri olan ve birçok yeni teknolojinin mimarı olan şirketin kurucusudur. Nolan Bushnell, bilgisayar oyunlarında yapay zekâ kullanımını jeton ile çalışan makineler üzerinde geliştirmeye başlamış ve yapay zekâlı oyun karakterlerinin daha akıllı ve gerçeğe uygun hareketler halinde olabilmesi için çalışmıştır. Yaptığı çalışmalar ve geliştirdiği teknolojiler sayesinde birçok ödül almıştır.

Pratt (1996) çalışmalarında altıncı nesil bilgisayarların sahip olduğu yeni performans artışı ile bilgisayar ile gerçekleştirilen yapay zekâ sistemlerinin kullanım alanları üzerinde araştırmalar yapmıştır. Bu kullanım alanlarında askeri simülasyon yazılımları üzerinde durmuş ve bu yazılımların kullandığı sistemlerin yapay zekâ tekniklerini incelemiştir. Askeri bir simülasyon yazılımında olması gereken düşman ve dost unsurların hareketlerinin yapay zekâ sistemleri ile gerçeğe uygun bir şekilde canlandırılması üzerinde çalışmalar yapmıştır.

Kocabaş ve Öztemel (1998) simülasyon ortamlarında kullanılan yapay zekâ teknikleri hakkında çalışmalar yapmışlardır. Yapılan çalışmalarda, yedinci nesil bilgisayarların sahip olduğu işlemci gücünün getirdiği performans ile, günlük hayatta yapılan deneylerin bilgisayar ortamında gerçekleştirilmesi için geliştirilen simülasyon yazılımlarında kullanılan yapay zekâ sistemlerinin, ortaya çıkabilecek durumları gerçeğe en yakın şekilde uygulayabilmesi için kullanılan yapay zekâ teknikleri sorgulanmıştır. Örüntü ve ses tanıma, robotik, makine öğrenmesi, bilgi tabanlı sistemler, doğal dil anlama ve işleme, teorem ispatlama ve bilgisayar oyunları alanlarında kullanılan yapay zekâ teknikleri hakkında araştırmalar yapılmış ve özellikle savaş oyunlarında kullanılan yapay zekâ teknikleri üzerinde durulmuştur.

Herman (2001) bilgisayar oyunlarında kullanılan yapay zekâ sistemlerinin gelişimini ele alarak, gelişen teknoloji ve ortaya çıkan yeni oyun türleri ile şekillenen yapay zekâ sistemleri üzerinde durmuştur.

Munoz-Avila (2006) bilgisayar oyunlarında kullanılan yapay zekâ sistemleri ile akıllı oyun karakterleri tasarımları hakkında çalışmalar yapmıştır. Yapılan çalışmalarda oyun içindeki yapay zekâlı karakterlerin koordine edilmesi ve takım halinde hareket eden yapay zekâlı karakterlerin gerçeğe yakın davranışlarını programlamak için gerekli teknikler üzerinde durulmuştur.

Lee-Urban (2008) birinci şahıs nişancı türündeki oyunların yapay zekâlı karakter davranışları hakkında çalışmalar yapmışlardır. Bu tür savaş oyunlarındaki düşman ve dost tarafların, oyun sistemi içinde gerçekleşen savaş olayları sırasında takım halinde hareket ederek galibiyet alabilmesi için kullanılması gereken yapay zekâ tekniklerinin programlanması konularında yaptıkları çalışmalarda, takım halinde gerçekleşen hareketler için algoritmalar geliştirmişlerdir.

Zang ve Church (2009) bilgisayar oyunlarında yer alan karakter yapay zekâ sistemleri için yol bulma algoritmaları üzerine yaptıkları araştırmalarda, A* algoritmasını kullanarak oyun içi yol ağı üzerindeki en kısa yolu bulma çalışmaları yapmışlardır. Delling ve arkadaşları (2009) yol bulma algoritmaları üzerinde çalışarak daha karmaşık yol bulma sistemleri geliştirmişlerdir.

Mocholi ve arkadaşları (2010) bilgisayar oyunu karakterlerinin topluluklar halinde hareketleri üzerine çalışmalar gerçekleştirmişlerdir. Bu çalışmalarda karınca kolonisi algoritmasını kullanarak bireyler arası iletişim halinde olan oyun karakterlerinin yol bulma teknikleri üzerinde durulmuştur.

Tez çalışmasında kullanılan kaynaklar bahsi geçen kaynaklarla sınırlı olmamakla birlikte, gereken yerlerde diğer kaynakların atıfları yapılmıştır.

3. BİLGİSAYAR OYUNLARI

3D bilgisayar oyunlarında yer alan yapay zekâlı karakterlerin hareketlerini ve oyun içi etkilerini anlayabilmek için öncelikle bilgisayar oyunlarının tarihi gelişimini ve yeni oyun türlerinin ortaya çıkması ile gelişen bilgisayar oyunu pazarının incelenmesi gerekir.

3.1. Bilgisayar Oyunları Tarihi

Bilgisayar oyunları tarihi, donanımı yetersiz ilk bilgisayarın kullanılmasıyla birlikte programcıların bilgisayar oyunu programlamaya ilgi duyması ile başlar.

1958: Newyork Brookhaven Laboratuvarı'nda William Higinbotham vakum tüplü analog bir bilgisayar geliştirmiştir. O zamanlarda bilgisayarlar yavaş çalışıyordu ve Willim Higinbotham ekranda zıplayan noktalar gördü. Aklına, o anda bilgisayarda bir tenis oyunu yapabileceği gelmiştir (Herman, 2001).

1960: William Higinbotham geliştirdiği bilgisayar oyununu tanıtmıştır. Bu basit bir tenis simülasyonudur.

1962: Cambridge-Massachusetts'teki Hingham Enstitüsünde araştırma görevlisi Steve Russell, iki kişilik oyun seçeneği ve torpido fırlatma özellikleri içeren bir uzay gemisi oyunu geliştirmiştir (Markoff, 2009).

1969: Steve Russell, Stanford Üniversitesine geçti ve geliştirdiği oyun burada mühendislik öğrencileri tarafından büyük ilgi görmüştür. Bu öğrencilerden biri de Nolan Bushnell olmuştur.

1971: Bushnell, Russell'ın oyununu alarak, kafasındaki "bilgisayar oyunlarını eğlence sektörüne sokma" fikriyle yeniden düzenlemiş ve böylece ilk jetonlu oyun makinası oluşmuştur. Adı "Computer Space" konmuştur (Yagoda, 2008).

1971-1981: Computer Space oyununun geliştirilmesinden sonra Nutting Associates bu oyunun haklarını 500$'a alarak 1500 makine daha üretmiştir. Fakat bu oyun beklenen ilgiyi görmemiştir. Neyseki Bushnell bunun sebebini anlamakta gecikmemiştir. İnsanlar henüz televizyon ekranındaki şeyleri kontrol etmeye alışık değillerdir. Bushnell kazandığı 500$'la kendi oyun geliştirme şirketini kurmuştur. Adı "Atari" olmuştur (Kent, 2001).

Bushnell, Alan Alcorn isimli bir mühendisi işe aldı ve ondan en kolay oynanabilir bilgisayar oyununu geliştirmesini istemiştir. Alcorn "Pong" isimli bir oyunla geri gelmiştir. Bu oyun iki oyuncunun yukarı ve aşağı hareket edebilen iki dikdörtgeni kontrol ederek, hareket eden noktayı, ortadaki çizginin üzerinden geçirerek bir o tarafa bir bu tarafa göndermesi şeklinde oynanmıştır.

Bushnell hangi üretici firmaya gittiyse geri çevrildi. Bushnell prototipini bir California'da bir barın bir köşesine koymuştur. İlk günün sonunda oyunun bozulduğuna dair bir telefon almıştır. Tamir etmeye gittiğinde sorunun ne olduğunu hemen anlamıştır. Bozuk para atılan kısmı dolduğu için sıkışmıştır. Böylece bilgisayar oyunları gelmeye başlamıştır. 1981'de Atari en hızlı büyüyen şirket haline gelmiştir. Bushnell kurduğu 500$'lık şirketi 1976'da Warner Communications'a 28 milyon $'a satmıştır.

80'lerin başında birçok yeni şirket daha kurulmaya başlamıştır. Epyx, Broderbund, Sierra Online ve SSI bu sıralar kurulmuştur.

1977: Atari çıkardığı 2600 modeliyle ortalığı kasıp kasıp kavurmuştur. "Burgertime" ve "PC Man" gibi oyunlar en sonunda evde oynanmak üzere piyasaya çıkmıştır. Ebeveynler de mutluydu çünkü çocuklarını bozuk paralarla oyun oynamaktan kurtarmıştır.

1981: Oyun makinalarının popularitesi artmıştır. Apple'lar, Atari'ler ve TRS-80'ler raflarda yerini almış, üreticiler birçok oyun üretmeye başlamıştır. Bu sıralar üreticiler o zamanlar 5'25" lik disketlere göre daha kullanışlı olan bantlı

kaset oyunlarını üretmiştir. 90'lara kadar kasetler tercih edilmiş ama 90larla birlikte hızından dolayı disketler tercih edilmeye başlanmıştır.

1982: Apple II için "Zork I", Broderbund tarafından "Choplifter" ve o zamanlar daha yeni kurulmuş olan Microsoft tarafından "Olympic Decathlon" geliştirilmiştir. O zamanlarda Microsoft'un oyun sektöründeki bu denemesi pek başarılı olmamıştır. Bu yüzden Microsoft çok uzun bir süre oyun sektörüne girmemiştir. Bu yılda ayrıca Access Software, Electronic Arts ve Lucasfilm Games (Lucas Arts) kurulmuştur (Young ve Jeffrey, 1988).

1983: Bu yılda Apple II'ler yerini yeni çıkan Atari 400 ve Atari 800'lere bırakmak zorunda kalmıştır. Electronic Arts ilk oyununu üretmiştir ve oyununu dergilerde iddialı bir başlıkla duyurmuştur.

1984: Kartuşlu oyun makineleri birden gözden düşmüştür. – Computer Gaming World isimli dergi bu durumu şöyle açıklamıştır; "1983'te bu makineler hızla zirveye tırmandı ve 1984'te o zirveden düştü." Bu sırada Commodore ve disketli makineler daha fazla ilgi görmüştür. C-64 ya da Atari 800'ün daha fazla ilgi görmesinin sebebi şu anda PClerin (Personal Computer, kişisel bilgisayar) konsollara tercih edilmesiyle aynı olmuştur. Bu yeni makinelerde oyun dışında şeyler de yapabilmiştir. Ailelerin de bahanesi "Çocuklarımız programlar yazıp yeni şeyler öğrenebiliyorlar. Sadece oyun için değil." olmuştur.

Bu sıralarda IBM küçük bir PC üretmiştir. Bu PC 5'25'lik disket sürücü 2 kartuş boşluğu ve basit bir DOS versiyonu içermektedir. Ne yazık ki o zamanlar bu diğerlerine göre daha donanımlı olan makine pahalı olduğundan çok az firma onu destekleyebilmiştir.

1985: Bu yıllarda ev bilgisayarları ilgi görmeye devam etmiş ve piyasaya yeni katılan iki rakip olan Commodore'un Amiga'sı ve IBM'in kapışması sürmüştür. Ama üretimci firmalar hangi platformda oyun üreteceğine karar verememiştir. Electronic Arts verdiği ilanlarla Amiga'yı desteklediğini duyururken, diğer birçok firma IBM'de ısrar etmiştir. Bu yıllarda Amiga özel tasarımıyla grafik ve

müzik alanında ilklere imza atmış ve günümüzün Multi Medya (çoklu ortam) kavramını oluşturarak yeni bir akım meydana getirmiştir (Young ve Jeffrey, 1988).

1986: Firmalar aynı anda birçok platformla nasıl baş edeceğini öğrenmeye başlamıştır. Oyunlarını birden fazla platform için geliştirmeye başlamışlardır: Apple, C64, IBM, Amiga. Üretim maliyeti artmasına rağmen mümkün olan en fazla sayıda kullanıcıya ulaştıklarından emin olmuşlardır. "Cross-platform" denilen bu teknik bugünlerde konsollar ve PClerde de uygulanmaktadır (Young ve Jeffrey, 1988).

1987: Adventure ve RPG (Role Playing Game, rol yapma oyunu) bu yılların tercih edilen oyunları olmuştur. Origin'in "Ultima IV"ü ve Microprose'un "Pirates"ı listelerde bir numara olmuştur.

1988: Oyunlar yavaş yavaş renklenmeye başlamıştır. 4 renk CGA (Color Graphic Adapter, Renk Grafik Uyarlayıcısı) grafiklerden 16 renk EGA (Enhanced Graphic Adepter, Geliştirilmiş Grafik Uyarlayıcısı) grafiklere geçilmiştir. Sierra Online bu yeni grafik modunu ilk kullanan firma olmuştur. Bu yeni geçiş sadece 1 yıl gibi kısa bir süre almıştır.

1989: Bu yıl ilklerin yılı olmuştur. 256-VGA (Video Grafik Dizisi) grafik modunu kullanan ilk oyun olan "Mean Streets" bu yıl üretilmiştir. İlk ses kartları, Adlib, Sounblaster, oyunların biplerden daha iyi müziklere, efektlere sahip olabileceğini göstermiştir. Modem üzerinden oynanabilen ilk oyun da bu yıl piyasalara sürülmüştür. Ayrıca ilk CD-ROM (kompakt disk) oyunu "The Manhole" Activision tarafından geliştirilmiştir. Bilgisayar oyunları sektörü bugünlerde ulaştığı noktayı 1989 yılında yapılan sıçramaya borçludur.

1990: Ses kartlarının ve grafiklerin güçlenmesiyle iyice karmaşıklaşan oyun sektöründe kullanıcı-dostluğu (user-friendliness) diye bir kavram oluşmuştur. Lucas Arts hala kullandığı "seç ve tıkla" arabirimini tanıtmıştır. IBM-286 için SimCity bu yıl geliştirilmiştir (Lobo, 2007).

1991: Birçok firma yeni yeni oluşan CD-ROM teknolojisine ayak uydurmakta zorlanmaya başlamıştır. İlk olarak birçok firma daha henüz market payı çok az olan CD-ROMlar için pahalı girişimlerde bulunmak istememiştir. Bu sırada stratejilerini belirleyemeyen Commodore firması Amiga için CD32 ürününü piyasaya sürerken bu teknolojiyi kullanan ilk firma olmuştur. Aslında tasarımcılar da bu teknolojiyi kullanmayı çok iyi bilememiştir. Bu yüzden CD-ROMun piyasadaki ilk yılı pek parlak olmamıştır. Üretimciler Cdlerde boş kalan yerlere oyunla alakası olmayan filmler koymaya başlamıştır. Cdler disket oyunlarının tek bir yerde toplanma aracı olmuştur. Cdnin potansiyelinin keşfedilmesi birkaç yıl almıştır. Birçok popüler oyun bu yılda çıkarılmıştır. "Wing Commander", "Willy Beamish'in Maceraları" ve "Monkey Island" bu yılın oyunları arasında olmuştur (Lobo, 2007).

1993: Daha kaliteli oyunlar, SVGA (Super Video Graphic Array, Süper Video Grafik Dizisi) grafik ve yeni ses teknolojisiyle piyasaya sürülmüştür. "Day of the Tentacle" Lucas Arts'ın SCUMM grafik ve oyun makinesiyle oluşturulmuştur. X-Wing büyük iş yapmıştır. Star Wars hayranları kendi X-Wing savaşçısını yönetmiştir.

1994: "Wing Commander III" belki de CD-ROM video teknolojisini kullanmayı becerebilen ilk oyun olmuştur. Oyun ara demolarında film yıldızlarının videolarını içermektedir. "Indy Car Racing" ise bir Indy arabası kullanmanın yaklaşık hissini veren ilk simülasyon oyun olmuştur. Bu yılda çıkarılan "Magic Carpet" grafik açıdan o zamanlar için en başarılı oyunlardan biri olmuştur (Lobo, 2007).

Bu yıl ayrıca "Doom" tarzı dediğimiz first person shooterların da çıkış yılı olmuştur. Doom, iD tarafından piyasaya sürülmüştür. Doom'un uzun bir hikâyesi var ama özetlemek gerekirse birkaç ayda oyun piyasasını kırıp geçirmiştir.

Oyunlarda bu yıldan itibaren "Midi" ve "Wavetable Synthesis" teknolojisi kullanılmaya başlanmıştır. Böylece sesler ve müzikler daha az yer kaplayarak daha etkili olmuştur. Midi bundan sonra oyunlar için mükemmel bir ses teknolojisi olmuştur.

1995: CD-ROM'ların disketleri fethetmesi bu yıllara rastlamıştır. Daha hızlı, daha geniş kapasiteli Cdler programcılara daha fazla grafik, ses ve video opsiyonu verirken kullanıcılara da daha iyi bir oyun zevki yaşatmıştır. Bu yıldan sonra CD-ROM sürücüsü olmayanlar oyunları takip edememeye başlayınca bir CD-ROM sürücü almaya karar vermişlerdir (Lobo, 2007).

1995 Eylül ayında Sony PlayStation Amerika'da 299$'a satışa sunulmuştur. Aynı yıl Nintendo 64 Japonya'da satışa sunulmuştur (Laird, 2005).

1996: Dövüş ve savaş oyunlarına alternatif olarak kayak, snowboard, jet ski gibi sporların oyunları popüler hale gelmiştir.

Atari'nin kurucusu Nolan Bushnell sektöre tekrar dönerek platform oyunları için internet istasyonları kurmuştur.

Aynı yıl "Tamagotchi" isimli sanal bebek oyunu Japonya'da popüler hale gelmiş ve Mayıs ayında Amerika satışa sunulması ile birlikte 3 günde 30.000 adet satış yapmıştır (Kudler, 2007).

Sektöre giren konsolların popüler hale gelmesi ile birlikte bu yıllarda geliştirilen oyunların türleri çeşitlilik göstermeye başlamıştır. Artan çeşitlilik içinde savaş, yarış, sanal yaşam gibi temalı oyunlar kendi alanlarında özelleşmeye başlamışlardır.

2000: Sony PlayStation 2 Amerika'da satışa sunulmuş ve 500.000 adet olan konsol sayısı satışa sunulduğu gün sabahın ilk saatlerinde tamamen tükenmiştir.

"The Sims" isimli sanal yaşam temalı oyun yayınlanmış ve kısa sürede popüler hale gelmiştir. Öyle ki 2002 yılında "Myst" isimli oyunu geçerek en çok satan oyun olmuştur.

2001: Microsoft ve Nintendo yeni nesil sistemlerini duyurmuştur. Microsoft ürünü olan Xbox konsolu ile "en güçlü oyun deneyimi" iddiasında bulunmuştur. Aynı yıl Sega bir daha donanım ürünleri üretmeyeceğini duyurmuştur.

Varolan konsol sistemleri ve bilgisayar donanımlarının imkanları doğrultusunda üretilen bir çok oyun, sahip oldukları oyun mekanikleri ve içerik zenginliği ile adeta bir yarış halinde olmuşlardır (Herman vd., 2002).

2006: Nintendo, devrim niteliğinde olan "Wii" sistemini satışa sunmuştur. Bu sistemde kullanıcıların tenis raketi sallama, direksiyon döndürme gibi hareketleri bir kumanda ile gerçekleştirerek oyun kontrolü yapması sağlanmıştır.

Sony ise PlayStation 3 sistemini satşa sundu ki bu sistem çok gelişmiş olmasının yanı sıra çok pahalı olan bir sistem olarak kayıtlara geçmiştir.

Aynı yıllarda Xbox 360 ile birlikte PlayStation 3 konsolları için üretilen oyunlar arasında bir pazar yarışı oluşmuştur. Kullanıcılar oynamak istedikleri oyunlara göre konsol seçmek zorunda kalmışlardır. Pc kullanıcıları için üretilen oyunlar ise çok ayrı ve zengin bir pazar içinde yarış halinde olmaya devam etmişlerdir.

2008: "Grand Theft Auto 4" isimli sanal yaşam temalı oyun satışa sunulduğu ilk hafta 6 milyondan fazla olan satış rakamı ile bir rekora imza atmıştır. Eğlence dünyasında oyun sektörünün sahip olduğu pazar payının, Hollywood film sektöründen daha büyük olmaya başladığı görülmüştür.

Gelişen oyun sektöründe gün geçtikçe yeni oyunlar üretilerek piyasaya sürülmeye devam ederken her yeni çıkan oyunda görülen yapay zekâ sistemleri giderek daha güçlü ve daha gerçekçi olmaya başlamıştır. Yapay yaşam temalı

oyunlarda görülen karakter yapay zekâları ile savaş oyunlarında görülen karakter yapay zekâları, oyunların dinamik ve oynanış çeşitine daha gerçekçi ve doğru sonuçlar verdikçe oyuncular bu tür oyunlara daha çok ilgi duyar hale gelmişlerdir.

Seri halinde üretilen savaş oyunlaru ile yapay yaşam temalı oyunlar satışa sunuldukları anda yeni satış rekorları kırmaya başlamışlardır.

2010-2011: Nintendo firmasının üretimi olan Wii sistemine karşılık olarak Sony firması "Move" sistemini satışa sunarken Microsoft firması "Kinect" sistemini satışa sunmuştur. Hareket algılayıcı bu sistemler ile kullanıcıların vücut hareketlerine göre oyun içi dinamikleri kontrol edebilmesi sağlanmıştır. Bu tür kontrollerin yapılabildiği spor ve dans oyunları daha eğlenceli bir hal aldığı için oyun firmaları bu sistemler için oyunlar üretmeye başlamışlardır.

Aynı yıllarda henüz yeni yeni gelişmeye başlayan mobil oyun sektörü için "Angry Birds" isimli oyun 100 milyondan fazla yüklenme başarısı ile bu sektörün popüler hale gelmesini sağlamıştır.

Donanım gücü gün geçtikçe artan mobil cihazlar, akıllı cep telefonları ve tabletler için oyunlar geliştirilmeye başlanmıştır. Küçük ve bilgisayar oyunlarına göre karmaşık olmayan basit oyunlar kullanıcıların kısa sürede ilgisini çekerek bu sektörün hızlı bir şekilde gelişmesini sağlamıştır.

2012-2014: Bilgisayar oyunları artık sadece pc (kişisel bilgisayar) ve konsollar için değil mobil cihazlar için de üretilmeye başlanmıştır. Geliştirilen oyunların her platform için ayrı versiyonları üretilmeye başlanmıştır. İnternet üzerinden oynanan oyunların çoğu internet erişimi olan mobil cihazlar ile de oynanabilir hale gelmiştir.

2013 sonlarında Xbox One ve PlayStation 4 satışa sunulmuş ve bu yeni nesil konsollar sahip oldukları donanım gücü ile daha gerçekçi görseller içeren oyunların üretilmesine destekleyici olmuşlardır. Mobil oyun dünyasında hızla

üretilen ve çabuk tüketilen mobil oyunların yanısıra konsollar için anlaşmalı ve özel olarak geliştirilen oyunlar arasında oluşan rekabet daha da kızışmaya başlamıştır. Bunun yanında kişisel bilgisayarlarda oynanabilen oyunların büyük bir çoğunluğu neredeyse aynı grafik kalitesi ile artık konsollarda da oynanabilir hale gelmiştir.

Bütün platformlarda görülen hızlı donanım gelişimi ile oyunlar için kullanılan CPU (merkez işlem birimi) ve GPU (grafik işlem birimi) performansları özellikle geliştirilen oyunlara göre belirlenmeye başlamıştır. Donanım üretici Intel, AMD, IBM gibi firmalar ürettikleri yeni donanımları oyun firmalarında test ederek piyasaya sürmeye başlamışlardır.

Gelişen donanımlar ile birlikte üretilen oyunlar sahip oldukları temalara göre özelleştirilmiş yapay zekâ sistemlerine sahiptirler. Yapay yaşam temalı bir oyunda gündelik hayata dair karakter davranışları sergilenirken yarış temalı bir oyunda yarış içinde kullanıcı ile mücadele halinde olan özelleştirilmiş yapay zekâ sistemleri görülmüştür.

2011 yılında satışa çıkan "Skyrim" gibi bilgisayar oyunlarında görülen yapay yaşam içinde oyun içi karakterler bulundukları ortam ve sahip oldukları özelliklere göre gündelik bir hayat yaşamaktadırlar. Oyun içi bir köy veya saray ortamında görülen karakterler çiftçi veya saray muhavızı gibi ayrı özelliklere göre kullanıcı ile farklı şekillerde iletişime geçebilir ve oyun dinamiği oyunu oynayan her bir kullanıcıya göre farklı şekillerde gelişebilmektedir. Oyunun sahip olduğu yapay zekâ sistemi, tehlike durumunda kullanıcıdan kaçan hayvanlar ya da hırsızlık gibi suç içeren bir eylem karşısında kullanıcıya saldıran muhafızlardan, kullanıcının durumuna göre yardım eden ya da basit bir gündelik ders veren yapay zekâlı karakterlere kadar farklılık gösterebilmektedir. Bu tür yapay yaşam temasına sahip oyunlarda basit düşman grupları görülebilmektedir. Bu gruplar saldırı durumunda birlikte hareket ederek farklı yönlerden kullanıcıya saldırabilir ya da karakterler yalnız kaldıkları zaman basit bir şekilde kullanıcının saldırısından kaçarak kurtulmaya çalışabilirler.

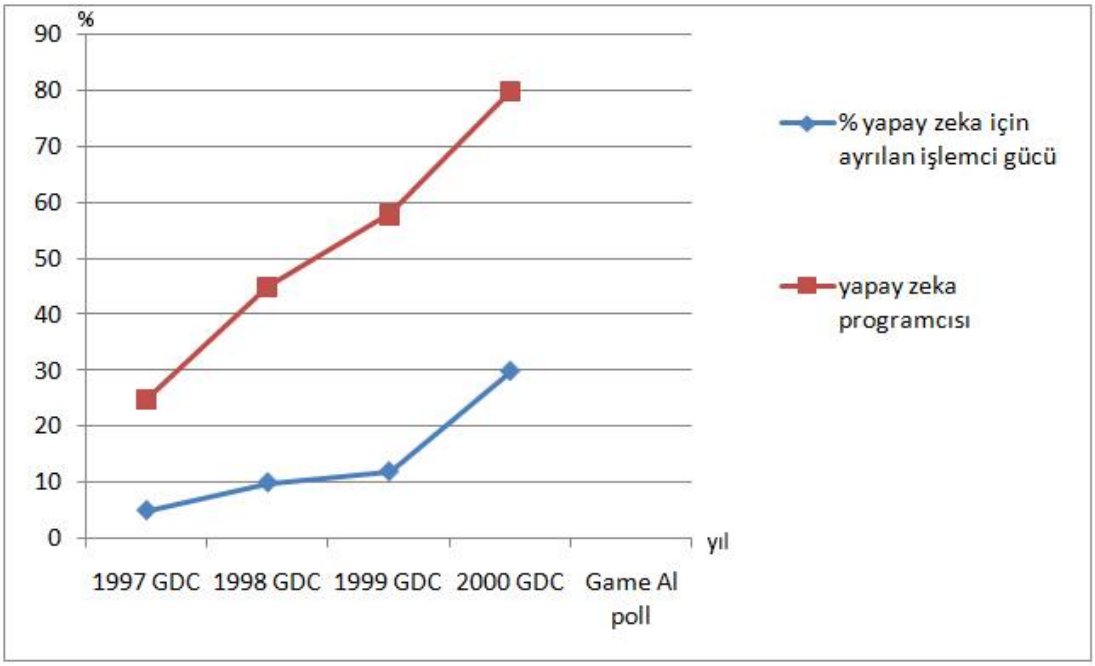
Savaş temalı oyunlar belli bir hikâyeye sahip olan oyunlar olup hikâyenin gelişimine göre oyun için düşman ve dost karakterlerin olabildiği aksiyon oyunlarıdır. Bu tür oyunlarda eğer takım halinde hareket ediliyor ise takım arkadaşları kullanıcı ile hareket eden ve gerektiği durumlarda verilen göreve göre kullanıcıyı koruyabilen yapay zekâlı karakterlerdir. Düşman karakterler oyun içi gelişen durumlara göre çarpışma sırasında saklanabilen, strateji değiştirerek farklı yönlerden saldırı düzenleyebilen, saklanma durumunda kullanıcıyı arayıp bulabilen yapay zekâlı karakterler olabilirler. Bazı oyunlarda takım halinde saldırı düzenleyen karakterlerin oyun içinde kullanıcıya tuzak kurabilen yapay zekâ sistemlerine sahip oldukları görülebilir.

Temel yapay zekâ sistemlerinin yanı sıra takım halinde hareket eden karakterlerin oyun içi gelişen ve değişken durumlara göre bireysel olarak hareket edebildiği ve sonuçları hiç bir zaman aynı olmayan davranışlar sergilediği oyunlar gün geçtikçe duyurulmaktadır. Bu tür oyunların kullanıcıları, oyunu defalarca oynasalar da her defasında farklı bir oyun deneyimi yaşayabilmektedir.

3.2. Günümüz Bilgisayar Oyunları Pazarına Bakış

Günümüzde gelişen teknoloji ile birlikte meydana gelen yüksek donanım gücüne sahip bilgisayar sistemlerinin kullanılması ile 3D oyunların programlanması artık öyle bir hal almıştır ki sektörün en ileri gelen yapay zekâ programcıları oyun şirketlerinin en aranan ve önemli çalışanları haline gelmiştir.

Oyun programlamada yapay zekâ kullanımı giderek artarak son yıllarda şirketlerin programcı sayılarının yarıdan fazlası yapay zekâ programcısıdır. Şekil 3.1’ de görüldüğü gibi yıllar geçtikçe yapay zekâ programcı sayısında artış olmuştur.



Şekil 3.1. Yapay zekâ programcısı ve işlemci gücü oranlarının grafiği

Kırmızı renk ile gösterilen grafik oyun şirketlerinin hangi oranda yapay zekâ programlama için en az bir kişi çalıştırıldığını gösteriyor. Buna göre 1997'de bu oran %25, 1998'de %45, 1999'da %59 ve inanılmaz biçimde 2000 yılı içerisinde %80 seviyesine ulaşmıştır. Bu, yapay zekânın oyun şirketleri için artık kaçınılmaz bir konu haline geldiğinin çok açık bir göstergesi. Alttaki mavi renk grafik ise yine günümüzdeki bir gerçeği göz önüne sermektedir. Bu grafik günümüz oyunlarında işlem gücünün ne kadarının yapay zekâ hesaplamalarına ayrıldığını göstermektedir. 1999 yılına kadar %10'un altında olan bu değer 2000 yılında %30 seviyesine yükseliyor. Bunun nedenini anlamak için işlemci ve grafik alanındaki gelişmeleri izlemek yeterlidir. 1998 yılına kadar bir lüks olan yüksek hızlı grafik hızlandırıcı yongaları son iki yılda artık herkes tarafından kullanılacak düzeye gelmekle kalmadı bu "grafik işlemcileri" her yıl neredeyse dörde katlanan performans artışı ile önceden işlemcinin bütün gücünü tüketen grafik çizme işlemlerini neredeyse tamamen üzerine almıştır. Son yıllarda ateşlenen masaüstü işlemci hız yarışının yüksek hızlara sahip işlemcileri 200$ seviyesine indirmesi performans konusunda büyük zorluk çeken programcılara rahat bir nefes aldırmıştır.

Günümüzde kullanılan işlemci (CPU, Central Process Unit) ve ekran kartı (GPU, Graphic Processing Unit) donanımları sayesinde oyun şirketleri geliştirdikleri sistemlerin işlem yükünü parçalayarak, oyunun grafik işlemlerini sadece GPU üzerinde çalıştırılmasını programlamaktadırlar. CPU için sadece oyun içi olaylar (game play), çevrim içi (online) data alış-veriş işlemleri ve yapay zekâ sistemleri gibi işlem ünitelerini ayırarak yüksek performanslı oyunlar geliştirilmektedir. (Lobo, 2007).

3.3. Klasik ve Modern Oyunlar

Klasik oyunlar için Satranç, Dama, Tavla, Okey, Briç ve çeşitli kâğıt oyunları gibi oyunları örnek olarak verebiliriz. Bu tür oyunlar oyunu oynayan oyuncu için tek bir amaç doğrultusunda çalışan bir yapay zekâ sistemine sahiptir. Oyun içi çalışan yapay zekâ sistemi, oyuncunun hareketlerine göre yapılacak yeni hamlenin amaca göre en uygun olanını seçmektedir. Yapay zekâ sisteminin takip etmesi gereken birçok unsur yoktur. Sadece oyuncunun hamlelerine göre hareket etmektedir.

Modern oyunlar için Call Of Duty, Gears Of War, Diablo, Sims, Grand Theft Auto gibi oyunları örnek olarak verebililmektedir. Çizelge 3.1'de görüldüğü gibi modern oyunların sahip olduğu yapay zekâ sistemleri sadece oyunu oynayan oyuncuya göre değil, aynı zamanda oyun içi sürekli olarak değişen çevresel faktörler ve oyuncunun hareketlerine göre ortaya çıkabilecek milyonlarca olasılığın sonuçlarına göre hareket etmektedir. Bu tür gelişmiş yapay zekâ sistemine sahip olan oyunların karakter davranışları ve son yıllarda oyuncuya göre değişebilen senaryo ile oyunu her oynayan oyuncu için farklı deneyimler sunmaktadır.

Modern oyunlar olarak tanımlanabilecek savaş oyunlarında görülen karakter yapay zekâları, bireysel ya da takım halinde hareket etme durumlarına göre farklılık gösterebilirken çok zeki diye tanımlanabilecek tuzak kurma beceresine sahip olabilmektedirler. Diğer yandan "Watch Dogs" gibi yapay yaşam temalı oyunlarda görülen yapay zekâ sistemlerinde araba kazası durumunda kilitlenen

bir kavşak etrafında bulunan yapay zekâlı karakterler, kontrol ettikleri arabalar ile kazaya karışmadan etraftan dolaşıp normal seyirlerene devam edebilecek kadar zeki davranışlar gösterebilmektedirler. Bu tür oyunlarda kullanıcılar, oyun içi yaşanan ve kullanıcıdan bağımsız gerçekşeleşen olaylar karşısında yapay zekâ sisteminin oyun içi dinamik ve oynanışa getrdiği gerçekçilik ile oyuna bağlanmış olur ki kullanıcı uzun süre oyun oynayarak yapay bir yaşamın içine çekilmektedir. Grand Theft Auto oyun serisinde var olan gelişmiş yapay zekâ sisteminin oyun içi olayların gidişatındaki rolü kullanıcılara hiç bir zaman aynı deneyimi sunmaz ki bu iyi bir oyun için dikkat edilmesi gereken bir unsur haline gelmiştir. Oyun geliştirici firmalar gerçekçi görsellerin yanında gerçekçi savaş deneyimler, gerçekçi oyun içi iletişim deneyimleri, gerçekçi yaşam simülasyonları gibi oyunu oynayan kullanıcıları cezbeden bu unsurları gün geçtikçe daha iyi ve gerçek yaşam uygun hale getirmektedir.

Çizelge 3.1. Klasik ve modern oyunlar arasındaki farklılıklar.

Klasik Oyunlar	**Modern Oyunlar**
Hamle tabanlıdır, süre kısıtlaması azdır.	Çoğunlukla gerçek zamanlıdır.
Sabit ve genelde iki boyutlu oyun alanı vardır.	Çok büyük ve genellikle üç boyutlu oyun alanı vardır.
Genellikle iki oyunculudur.	Oyundaki karakter sayısı yüzlerce olabilir.
Gerektiğinde işlem gücünün neredeyse tamamı AI için harcanır.	AI için kısıtlı işlem gücü vardır.
Gerçek zamanlı oyunlarda bilgisayarın yönettiği karakter tekdüze bir davranışa sahiptir.	Bilgisayarın yönettiği karakterler çok zeki davranışlar gösterebilir.
Sınırlı sayıda pozisyon olasılığı vardır	Çoğunlukla sonsuz sayıda durum söz konusudur. Her durum farklı davranış gerektirebilir.
Sadece tek bilgisayarda oynanır.	Ağ üzerinde çoklu oyuncu desteği olabilir.

3.4. Modern Oyun Programlamada Yapay Zekâ Kullanımı

Modern oyunlar temaları ve tarzlarına göre çeşitlilik göstermektedirler. Çeşitlerine göre özelleşmiş tek düze yapay zekâ sistemlerinden yaşayan ortamlara sahip olan sürekli değişkenlik gösteren durumlara göre kendi içinde karmaşık yapay zekâ sistemleri görülebilmektedir.

Günümüz modern oyunlarında karşımıza çıkan yapay zekâ konularının sürekli gelişerek değişmesine rağmen çoğu oyunda karşımıza çıkan başlıca konular şu şekilde sıralanabilir;

- Kural tabanlı yapay zekâ
- Stratejik düşünme ve öğrenme
- Yol bulma ve engel geçişi
- Yüzey analizi ve etki haritaları
- Görüş netlik grafı
- Sürü hareketi (flocking)
- Yapay yaşam

3.4.1. Kural Tabanlı Yapay Zekâ

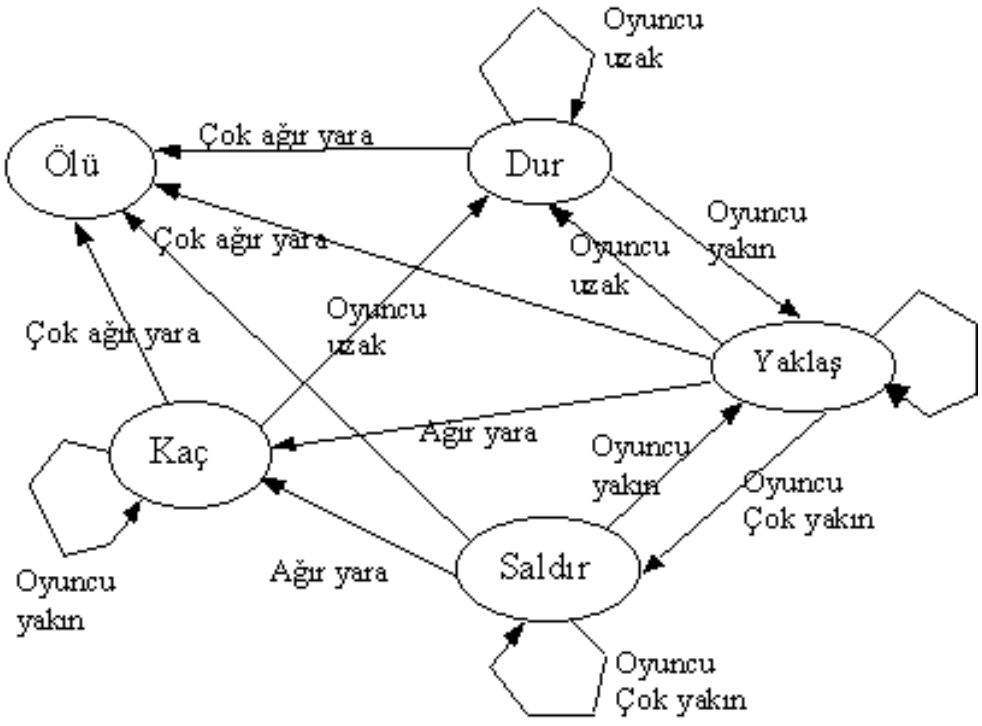
Oyunlarda bilgisayar tarafından denetlenen karakterlerin ya da ortamın davranışını tanımlamada en çok kullanılan metot kural tabanlı yaklaşımdır. Kural tabanlı yaklaşım FSM (Finite State Machine, Sonlu Durum Makineleri) ve FuSM (Fuzzy State Machine, Bulanık Durum Makineleri) olmak üzere iki alanda kendisine geliştirici bulmuştur.

Sonlu durum makineleri bir objenin sınırlı sayıdaki durum arasında meydana gelen olaylara göre geçiş yapmasını sağlayan mekanizmaya verilen isimdir (Dowsland, 1995). Bir örnek verecek olursak, bir oyundaki yaratığın davranışı aşağıdaki gibi belirlenebilir.

Durumlar: Dur, Oyuncuya yaklaş, Saldır, Kaç, Ölü
Eylemler: Oyuncu uzak, Yakın, Çok yakın, Ağır yara, Çok ağır yara
Parametreler: Mesafe, Sağlık durumu, Zaman

Elbette ki bu örnek daha da çeşitlendirilip zenginleştirilebilir. Parametre, eylem ve durumlar arttırılabilir. Bu tip durum makineleri şu andaki modern oyunların tamamında son derece yaygın biçimde kullanılmaya devam etmektedir.

Bu tip sonlu durum makinelerinin biraz daha "akıllı" örnekleri bulanık sonlu durum makineleri ile kendini göstermiştir. Burada ise durumlar arası geçişleri sağlayan eylemler oyunun ve oyuncunun davranışına göre farklılık gösterir. Yani durumlar arası geçiş parametreleri kesin mantıksal ifadelere bağlı olmayabilir. Bu da karakter davranışını biraz daha tahmin edilemez ya da insan mantığına uygun hale getirir. Buradaki handikap ise bu geçiş parametrelerinin çok iyi ayarlanması gerekliliğidir. Bulanık durum makineleri Unreal oyununda yaratık davranışını ifade etmek için yoğun biçimde kullanılmıştır (Şekil 3.2).



Şekil 3.2. Karar mekanizmalarının hiyerarşik tabakaları (Efe, 2001)

Son bir iki yılda kullanılmaya başlayan hiyerarşik kural tabanlı sistemler ise genellikle çok karmaşık olan bazı karar mekanizmalarının hiyerarşik tabakalar şeklinde ele alınması ile çözülmeye başlanmıştır. Interplay şirketinin çıkardığı Starfleet Command ve Red Storm Force 21 oyunlarında, oyunu kumanda eden oyuncu sadece üst seviyedeki askeri emirleri vermekte, alt tabakaya gelen emirler her tabakada bağımsız biçimde yorumlanarak örneğin generalden manga seviyesindeki asker grubuna kadar yayılmaktadır. Bu şekilde oyuncunun alt seviyeye olan müdahalesi minimum seviyeye inmektedir. Bu hem oyunun tasarımını hem de hata ayıklama işlemini kolaylaştırmaktadır.

3.4.2. Stratejik Düşünme ve Öğrenme

Bu konu kural tabanlı sisteme göre çok daha karmaşık bir yapıya sahiptir. Halen ufak tefek deneme alanı bulan bu tekniklerde akademik alanda yaygın biçimde kullanılan yapay sinir ağları ve genetik algoritmalar tercih edilmektedir.

Bu teknik ile geliştirilen oyunlarda bilgisayar denetimindeki karakter oyuncunun kendisine olan tepkisine göre saldırı stratejisini değiştirme yeteneğine sahiptir. Bu da oyunu her defasında daha farklı bir senaryoya sürüklediğinden oldukça favori bir konuma getirebilir. Magic & Mayhem oyununda önceki saldırılarda olanlar bir veri tabanında tutulmaktadır. Yazılım yeni saldırı önerildiğinde veri tabanındaki verilerle önerilen saldırıyı kıyaslamaktadır. Eğer saldırının fazla başarılı olamayacağı sonucun elde ederse başka bir saldırıyı denemektedir. Dosya sürekli tazelenip eski tecrübeler atılmaktadır. Ancak klasik kural tabanlı sistemlerin akademik öğrenme ve kendini geliştirme algoritmaları olarak basitçe tanımlanacak bu yapay sinir ağları ve genetik algoritmalara karşı halen seçilmelerinin bir numaralı nedeni tasarımlarının basitliği ve test etmenin çok kolay olmasıdır (De Castro vd., 2002).

Test problemi halen en büyük problemi oluşturmaktadır. Bu algoritmaların tabiatı gereği bir durumdaki sistemin davranışını önceden kestirmek çok zordur. Bu durumda oyun pazara çıktıktan sonra oyunu oynayan kişinin oyun

geliştirme sırasında yazılımın daha önce hiç test edilmediği bir hareketi yapması ve yazılımın çok kötü bir reaksiyon vermesi olasıdır. Oyun çıkarmak artık zamanla yarış haline geldiği için bu tip algoritmaların kullanımı pazar açısından son derece riskli olabilmektedir. Buna rağmen şu andaki oyun geliştiricilerinin %20'sinin özellikle yapay sinir ağları ve öğrenme konusunda deneysel araştırmalar yaptıkları bilinmektedir (Forrest vd., 1994). Özellikle Grand Theft Auto gibi yapay yaşam temalı oyunların yapay zekâ sistemleri için ekiplerin özel araştırmalar yaptıkları bilinmektedir.

3.4.3. Yol Bulma ve Engel Geçişi

Bundan dört beş yıl öncesinin ünlü oyunlarından War Craft ya da Command and Conquer ile başlayan yeni nesil oyunların en büyük problemlerinden birisi yol bulma (pathfinding) olmuştur. Bugün tasarlanan bütün oyunlarda karşımıza çıkan bu sorun, bir ya da birden fala hareketli nesnenin oyuncunun komutu ile ya da bilgisayarın kararı ile bir noktadan bir noktaya hareket ettirilmesidir.

Yol bulma algoritmaları konusunda pek çok çalışma yapılmış, sonuçta bu problem için en uygun çözümün A* isimli arama algoritması olduğuna karar verilmiştir. Bu algoritmanın üç boyuta uygulanması da söz konusu olmuş ve D* (D Star, D Yıldız) isimli arama algoritması türetilmiştir.

Algoritma oldukça karmaşık durumlarda bile kullanılsa da bazı durumlarda halen işe yaramamaktadır. Örneğin dar köprü ya da dağ geçitlerini büyük gruplar ile aşmak ya da oyun alanının hareket sırasında değişmesi ancak algoritmanın değiştirilmesi ve geliştirilmesi ile çözülebilecek konulardır. Bugün bile tam olarak mükemmel bir yol bulma algoritmasının oyunlardaki kullanımından söz etmek son derece güçtür.

Yol bulma, grafik işleme işlerinin dışında CPU kaynaklarını kullanarak gerçekleştirilen bir işlemdir. Günümüz oyunlarının geniş oyun içi dünyası ve oynanış sırasında değişen ortam koşullarına göre oyun içi karakterlerin yol bulma işlemleri de farklı algoritmalar ile gerçekleştirilebilmektedir. Bu yeni

nesil oyunlarda görülen gerçekçi ortamlar ve karakterler, gerçekçi bir yapay zekâ sistemi olmadığı takdirde boş ve kalitesiz görüneceklerdir. Bu yüzden oyun içi karakterlerin sürekli değişen oyun içi ortama göre yeni yol bulma gibi becerileri öğrenerek, kendini oyunun değişen ortamına adapte edebilmesi gerekir (Mocholi vd., 2010).

Takım halinde haraket eden oyun karakterlerinin yol bulma sistemleri için farklı yaklaşımlar olmakla birlikte günümüz oyunların karınca kolonisi algoritması da kullanılmaya başlanmıştır.

Oyun içi gelişen durumlara göre takım halinde hareket eden karakterler karşılaştıkları ve geçmeleri gereken bir oyun ortamının (orman, şehir merkezi, vb.) şartlarına göre hareket ederek herhangi bir saldırı durumunda düşman kuvvetlerden saklanma noktalarını keşfederek ilerleyişe devam eder. Yeni nesil oyunlarda görülen gelişmiş Fizik Motoru (Physics Engine) özellikleri sayesinde ortam öğeleri patlamalara tepki gösterek parçalanabilmekte ve bu durum sayesinde ortam özellikleri dinamik olarak değişebilmektedir. Oyun içi bir patlama olayı sonucu yıkılan bir bina enkazı yüzünden karakterlerin gidiş yolu kapanabilir ve bu tür durumlarda karakterler hedefe ulaşabilmek için yeni yol alternatifleri bulabilmektedir.

3.4.3.1. A* Algoritması

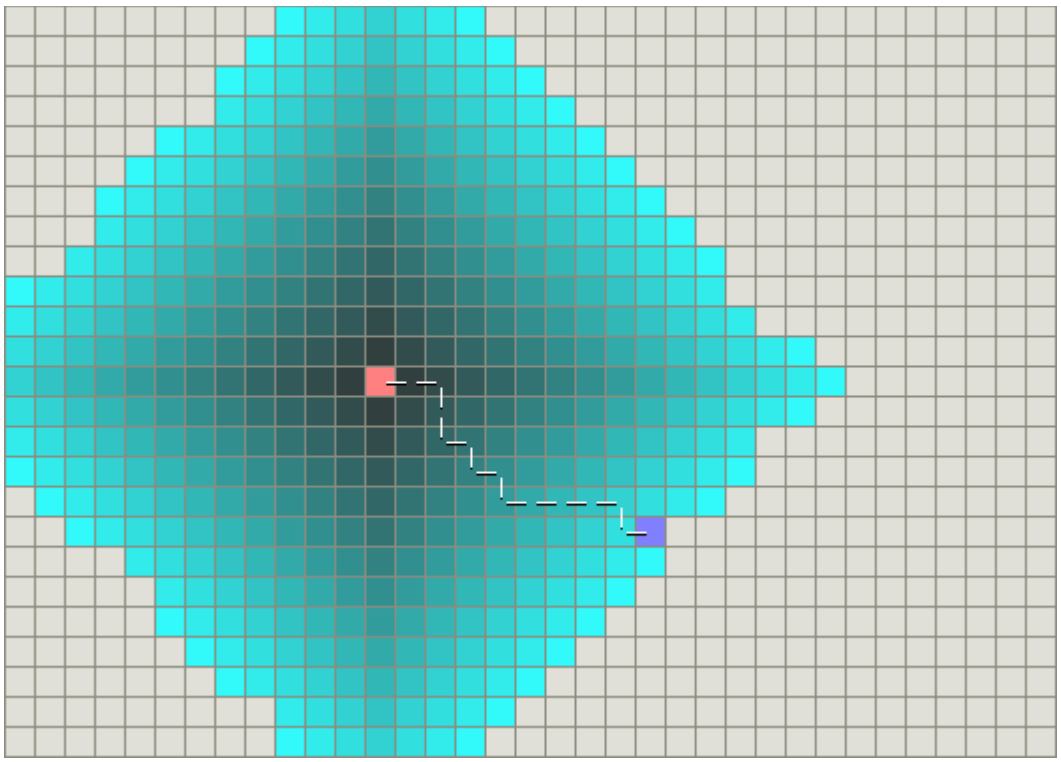
A* algoritması, iki düğüm (node) arası gidilebilecek en iyi yolu bulmak amacı ile en kısa ve ve maliyeti en az olan yolu seçerek hedefe ulaşmayı amaçlayan yol bulma algoritmasıdır.

A* algoritması buluşsal fonksiyon özelliği sebebi ile sezgisel (heuristic) algoritma olarak isimlendirilmektedir. A* algoritmasında kullanılan sezgisel fonksiyonu **f(n)** ile ifade edilecek olursa;

$$f(n) = g(n) + h(n) \quad (3.1)$$

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

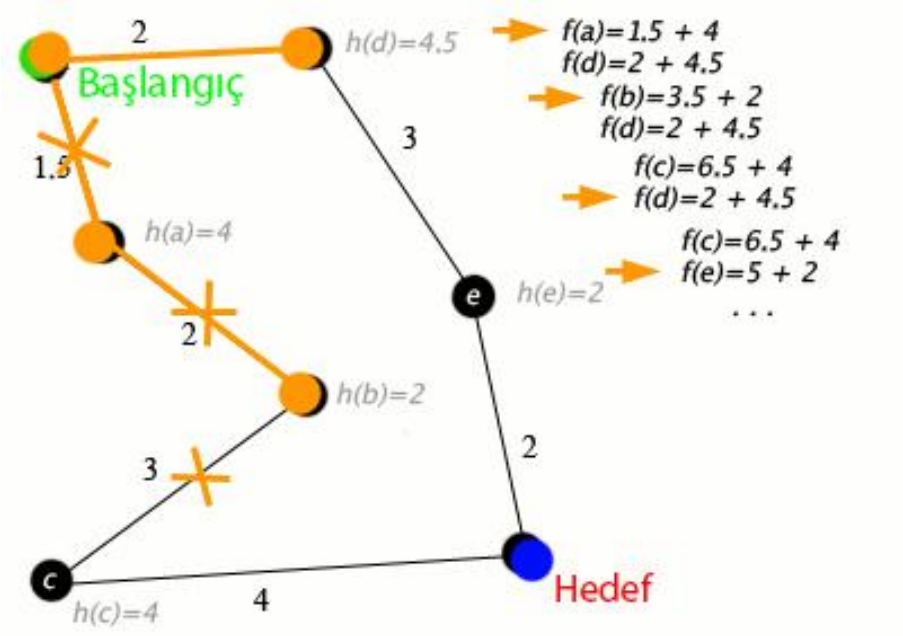
Dijkstra gibi Şekil 3.3'de gösterildiği gibi en kısa yol bulma algoritmalarında kullanılan başlangıç düğümünden itibaren bir noktadan dışarıya doğru bütün yönlerde ilerleme metodunun aksine, A* algoritmasında hedef düğüme doğru, doğrudan bir maliyet hesaplanır. Algoritma ilerleme yönünü bu maliyetin artıp azalmasına göre ayarlamaktadır. Böylece karmaşık ortamlarda dahi, algoritma **g(n)** fonksiyonunu kullanarak hedeften uzaklaştığını anlayabilir. **g(n)** parametresi aynı zamanda maliyet ile hedefe yaklaşma arasında bir denge unsurudur. Diğer bir deyişle **g(n)** fonksiyonu, algoritmanın hızı ile doğruluğu arasında belirleyici bir rol oynamaktadır.



Şekil 3.3. Dijkstra algoritması yol bulma şeması (Patel, 2014)

A* algoritmasının çalışması esnasında öncelikle kaynak düğümün komşu düğümleri ziyaret edilir. Ardından **f(n)** değeri en düşük olan düğüm öncelikli olmak üzere hedef düğüm bulunana kadar ilerlenmeye devam edilir. Her bir adımda, bir önceki düğüm, gidilen düğümün ebeveyni olarak işaretlenir. Böylece hedef düğüme ulaşıldığında düğümlerin ebeveynleri takip edilerek başlangıç ve hedef düğümleri arasındaki yol elde edilir (Zeng ve Church 2009).

A* algoritması için örnek bir yol bulma işlemi için Şekil 3.4' de düğümlerin yollarla bağlantıları sonucu hedefe ulaşım süreci görülebilir.



3.4. A* algoritması düğümler ve yolların hedef bağlantı şeması (Delling vd., 2009).

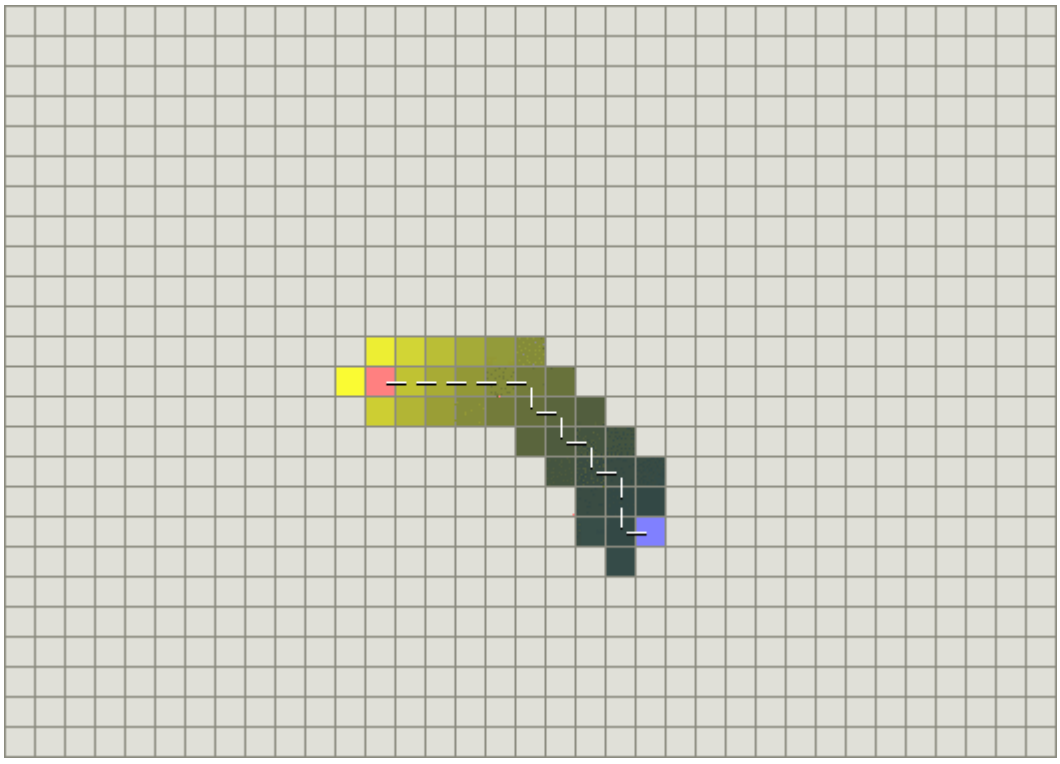
Şekil 3.4' de görüldüğü gibi yeşil nokta başlangıç noktasını, mavi nokta ise hedef noktayı temsil eder. Turuncu çizgiler ve noktalar ile ziyaret edilen düğümler ve geçilen yollar gösterilmektedir. Noktalar arası maailyet **g(n)** ve mesafe **h(n)** değerleri toplanıp **f(n)** hesaplanarak hedefe ulaşılabilmektedir.

Gidilmesi muhtemel düğümler, algoritma içerisinde "açık liste" isimli bir listede tutulmaktadır. Açık listede bulunan ve ziyaret edilen düğümler tekrar kontrol edilmemesi için "kapalı liste" isimli listeye alınmaktadır. Kapalı listede yer alan bir düğüm, daha kısa bir yol bulunması durumunda tekrar açık listeye

alınmaktadır. Açık liste içerisindeki düğümler, başlangıç düğümünden hedef düğüme kadar olan toplam maliyetlerine göre sıralıdırlar. Böylece öncelik sıralamasına sahip bir liste elde edilir. Bahse konu toplam maliyet, mevcut düğüm ile başlangıç düğümü arasındaki gerçek maliyet ile mevcut düğüm ile hedef düğüm arasındaki tahmini maliyettin toplamıdır (Buckland, 2005).

Hedef düğüme ulaşıldığında veya açık listede düğüm kalmayınca algoritma sonlandırılır. A* algoritması, sezgisel fonksiyonun hesapladığı tahmini maliyetin gerçek maliyetten fazla olmadığı durumlarda en kısa yolu verir. Ancak tahmini maliyet gerçek maliyetten fazla olursa bulunan yol, gerektiği kadar kısa değildir (Russell ve Norvig, 2003).

A* algoritması içerisinde sezgisel maliyet sıfır olarak alınırsa, diğer bir ifadeyle **h(n)** fonksiyonunun değeri sıfıra eşitlenirse, algoritma Dijkstra algoritması gibi davranış sergileyecektir. Diğer taraftan **g(n)** fonksiyonunun değeri sıfıra eşitlenirse algoritma, şekil 3.5'de gösterildiği gibi Best-First Search algoritması gibi davranış gösterecektir.

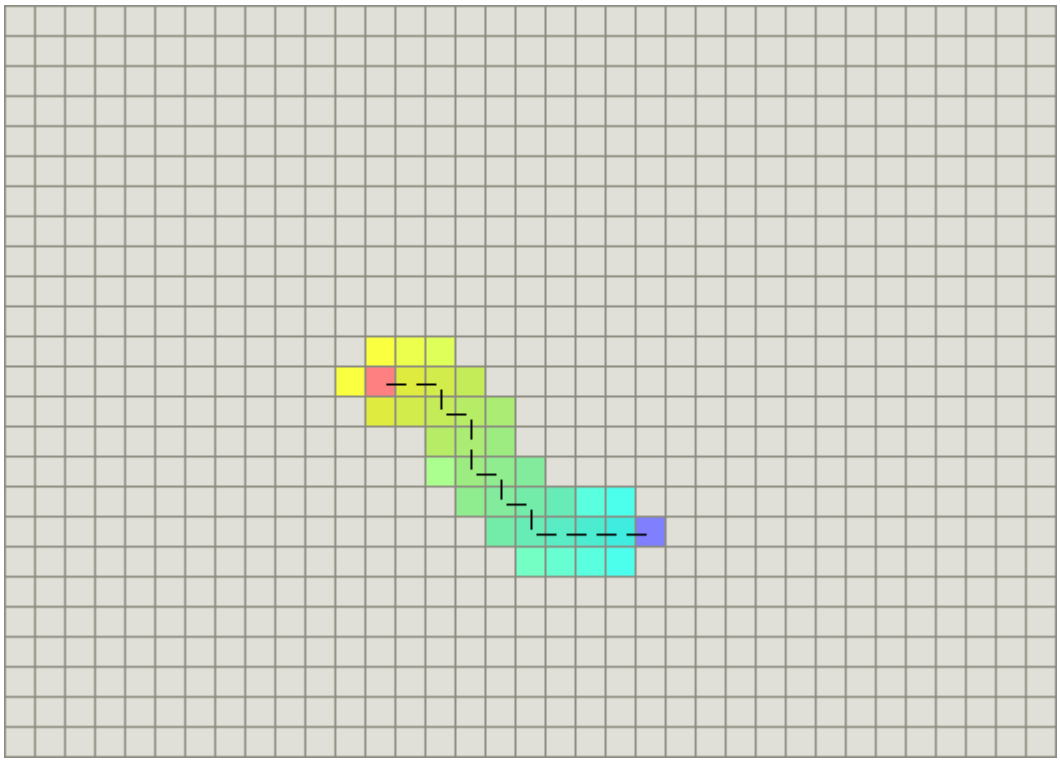


Şekil 3.5. Best-First Search algoritması yol bulma şeması (Patel, 2014)

İlk durumda taranacak düğüm sayısı ve işlem zamanı artacak, ikinci durumda ise hedefe yaklaşma-uzaklaşma kontrolü yapılamayacaktır. A* algoritmasının daha hızlı çalışabilmesi için geçilmesi istenmeyen düğümlere çok yüksek değerler verilebilir. Ayrıca yoğun bellek kullanımı, A* algoritmasının en önemli dezavantajlarından birisi olduğundan, kullanılan bellek miktarının artırılması algoritmanın daha hızlı çalışmasını sağlayacaktır (Patel, 2011).

A* algoritması içerisinde kullanılan sezgisel yöntemler sebebiyle, Şekil 3.6'da olduğu gibi engelsiz bir ortamda yol bulma süreci daha kolay ve hızlıdır.

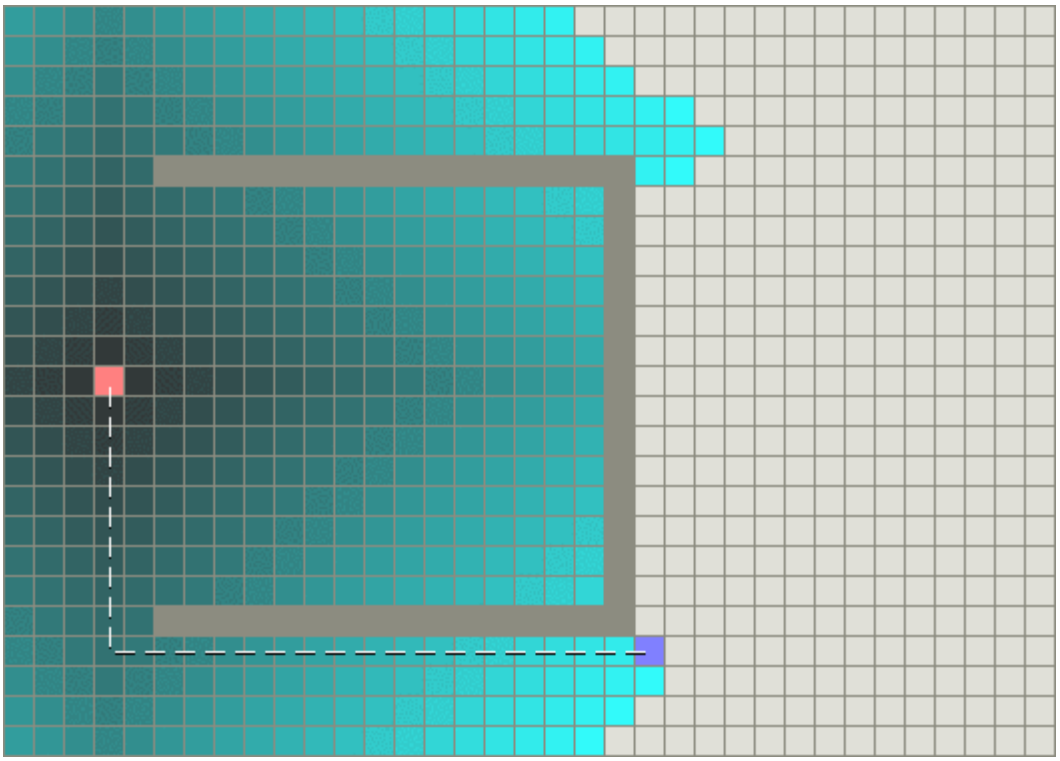
Oyun içi ortam dinamik bir şekilde değişkenlik gösterdiği için karakterlerin yolları üzerinde engel oluşturabilecek objelere göre hedefe giden yol hesaplama fonksiyonları sürekli olarak çalışır ve yeni yollar hesaplayarak karakteri hedefe yönlendirir. Bu nedenle yeterince hızlı sistemlerde, kaynak düğümden hedef düğüme ve hedef düğümden kaynak düğüme giden en kısa yollar hesaplanır ve elde edilen iki sonuçtan en uygunu seçilerek kullanılabilir.



Şekil 3.6. A* algoritması yol bulma şeması (Patel, 2014)

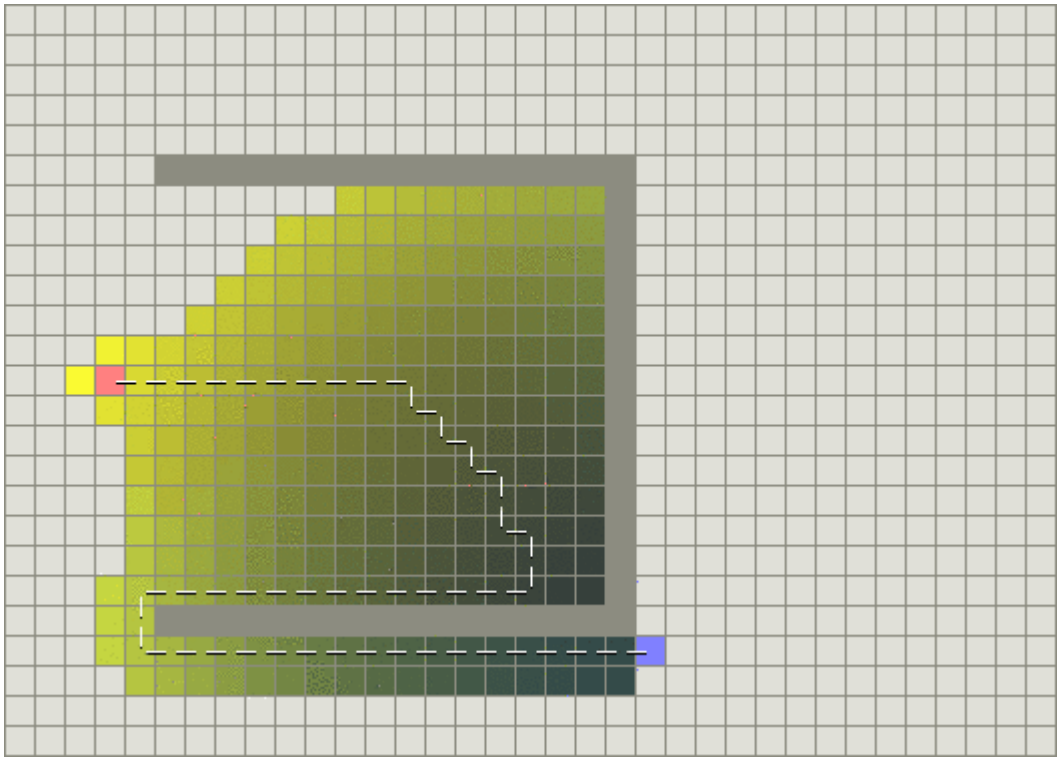
Kaynak ve hedef arasında engel olması durumunda Dijkstra, Best-First Search ve A* algoritmaları arasında zaman ve maliyet açısından farklılıklar bulunmaktadır.

Engel geçişi için Dijkstra algoritması kullanılarak hedefe ulaşma durumunda, Dijkstra algoritması bütün yönlerde tarama yaparak doğru yolu belirlemeye çalıştığı için özellikle zaman açısından büyük bir kayıp yaşanmaktadır. Şekil 3.7'de görüldüğü gibi bütün yönlerde tarama yapıldıktan sonra doğru yol kesin olarak bulunmuştur ama büyük bir alan taranarak vakit kaybedilmiştir.



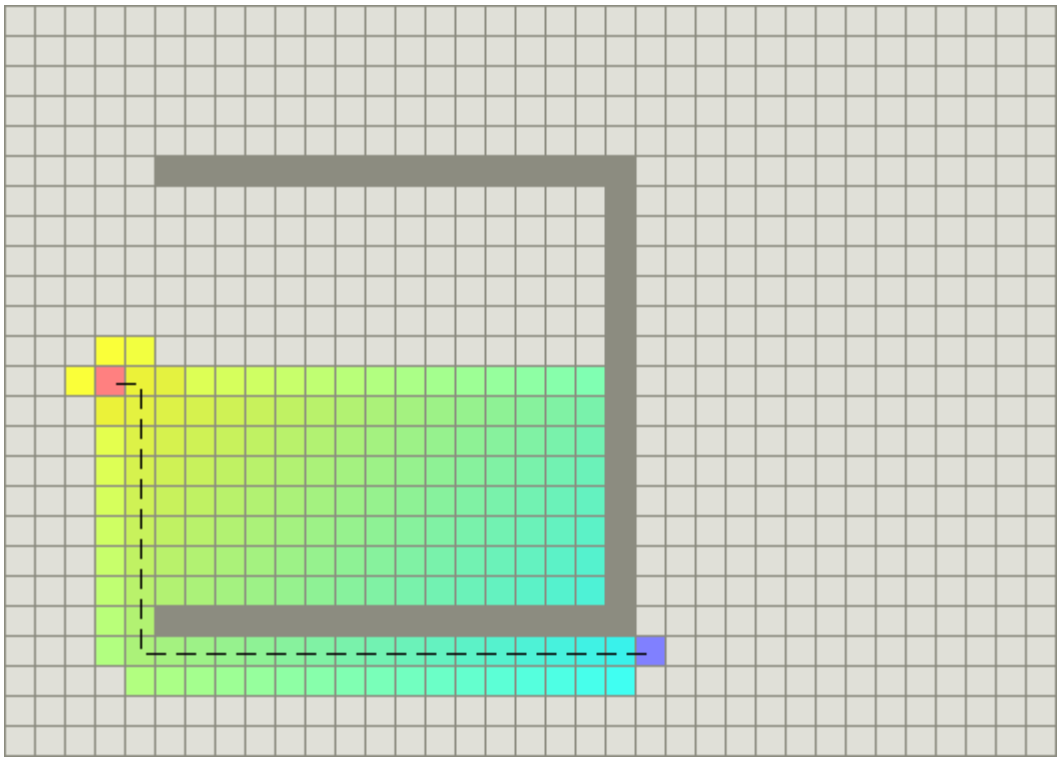
Şekil 3.7. Dijkstra algoritması engel durumu yol bulma şeması (Patel, 2014)

Best-First Search algoritmasının engel durumu karşısında taradığı alan az olacağından Dijkstra algoritmasına göre daha hızlı hedefe ulaşılır ama izlediği yol engeli farkedene kadar sezgisel hareket edilmediği için daha uzundur. Bu yüzden maliyet artmış olur ve Şekil 3.8'da görüldüğü gibi en kısa yol bulunmamıştır.



Şekil 3.8. Best-First Search algoritması engel durumu yol bulma şeması (Patel, 2014)

A* algoritması engel durumları karşısında zezgisel hareket sergilediği için diğer algoritmalara göre hedef yönündeki engeli önceden tayin edebilmektedir. Bu sayede engelin yapısına göre hedefe ulaşmak için en az maliyetli yol hesaplanarak Şekil 3.9'de görüldüğü gibi hem hızlı hem de az maliyetli bir şekilde en iyi yol belirlenmektedir.



Şekil 3.9. A* algoritması engel durumu yol bulma şeması (Patel, 2014)

Yol üzerindeki engeller karşısında hesaplanan yol, engellerin durumuna göre Şekil 3.9'da olduğu gibi dinamik bir şekilde değiştirilebilir. Ek A'da basit bir A* algoritması ile başlangıçtan hedefe yol bulma uygulamasının örnek fonksyion kodları belirtilmiştir.

Ayrıca A* algoritmasının çeşitli iyileştirmeler içeren IDA* (Iterative Deepening A*, Tekrarlamalı Derinleşen A*), MA* (Memory-Bounded A*, Hafıza Sınırlamalı A*), SMA* (Simplified Memory Bounded A*, Basitleştirilmiş Hafıza Sınırlamalı A*) ve RBFS (Recursive Best-First Search, Yinelemeli En İyi İlk Arama) şeklinde türevleri mevcuttur.

3.4.3.2. Karınca Kolonisi Algoritması

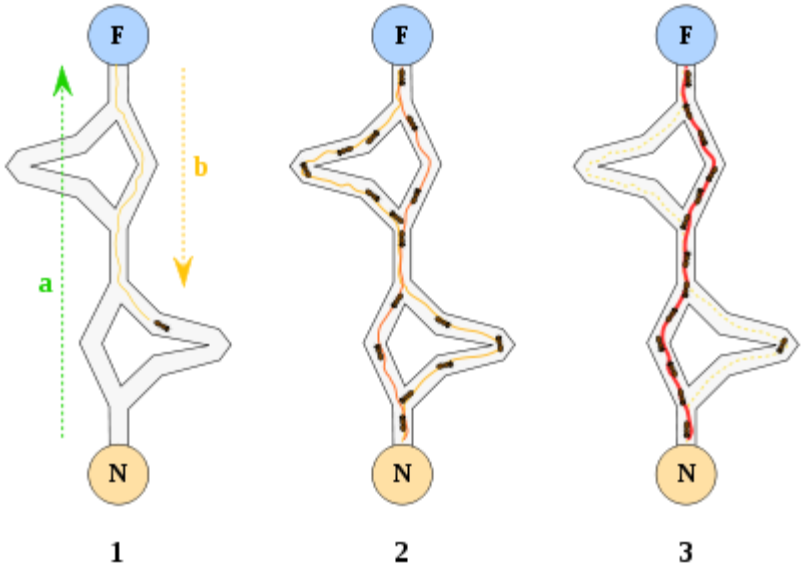
Takım halinde hareket eden yapay zekâlı karakterlerin yol bulma için izleyecekleri algoritmalardan biri de şüphesiz ki karınca kolonisi algoritmasıdır. Bu algoritma ile takım halinde hareket eden karakterlere sahip olan FPS türündeki oyunlarda, karakterlere gidilmek istenen hedefe en kısa yoldan en

hızlı şekilde gidebilmek için takımda bulunan diğer karakterler ile bilgi alışverişinde bulunurlar. Bu bilgi alışverişi sonucu hedefe ulaşmada karşılaşılan engeller ve izlenebilecek yollar analiz edilerek en kısa ve hızlı yol seçilerek karakter hareketleri sağlanır (Dorigo ve Gambardella, 1997).

Karıncaları oyun içi karakter takımlarının bir bireyi şeklinde düşünere geliştirilen algoritmalarda, karakterlerin daha hızlı bir şekilde yollarını buldukları gözlemlenmiştir. Her ne kadar işlem yükü çok fazla olan bir algoritma olsa da günümüz yeni nesil 3D FPS oyunlarında takım halinde hareket eden karakterler için bu algritma yaygın bir şekilde kullanılmaktadır.

Karıncalar, insan dâhil birçok canlının cinsel eşini bulabilmek için kullandığı koku yayma ve yayılan kokuları takip etme yönteminin benzerini kullanırlar. Karıncaların koku yayma ve koku alma yetenekleri oldukça güçlüdür ama yol bulma stratejisinde takip edilen koku, koku yayıcı karıncanın cinsiyetine bağlı değildir.

Şekil 3.10' da belirtildiği gibi, yuvadan çıkan karıncalar, bulundukları yerde diğer karıncalardan kaynaklanan baskın bir koku yoksa tamamen rastgele hareket ederler. Eğer herhangi bir yönden diğer karıncaların yaydığı bir kokuyu alabilmişse, karınca o tarafa yönelir. Birden çok yönden koku alıyorsa kokunun geldiği en baskın tarafa yönelir. Karıncanın yaydığı koku bir süre o bölgede kalabilir. Bu sayede bir yolu birden çok karınca tercih etmişse aradan belirli bir süre geçmiş olsa bile daha çok tercih edilen bölgeden daha baskın bir koku gelir.



Şekil 3.10. Karıncaların yol bulma stratejisini gösteren şema (Salzer vd., 2010)

Sürüdeki bütün karıncalar bu stratejiyi izlediğinde, başlangıçta rastgele başlayan sürü hareketi zamanla belli bir yol üstünde sabitlenir ki bu yol, yuva ile yiyecek arasındaki en kısa yoldur (Maniezzo vd., 2004).

Takım halinde hareket eden karakterler bilgi alışverişi sonucunda belirlenen en kısa ve hızlı yoldan hareket ederk hedefe ulaşmada karınca kolonisi algoritmasını kullanarak programlansalar da bazı durumlarda bu algoritmanın, oyun içi dinamik olarak değişen ortam koşulları yüzünden gereksiz bir işlem yüküne sebep olduğu görülmektedir (Rabin, 2002).

Patlama veya tuzak gibi kaos durumlarında dağılan takım karakterlerinin haberleşmede oluşan kopukluklar yüzünden davranışları anlamsız olduğu zaman, oyunun yapay zekâ sisteminin kötü olduğu düşünülebilir. Bu tür anlamsız hareketleri önlemek için karakterler takım ile tekrar bağlantı kurana kadar bireysel olarak hareket edecek şekilde farklı algortima sistemlerini kullanma moduna geçebilir. Günümüz FPS oyunlarında bu tür durumlarda karakterlerin davranışlarının değişimini gözlemlenebilmektedir.

3.4.4. Yüzey Analizi ve Etki Haritaları

Modern oyunlarda bilgisayarın sadece yol bulma algoritmaları ile rastgele gruplarla oyuncuya saldırması yeterli olmamaktadır. Gelecek oyunlarda yazılımlar oyun alanında ayrıntılı yüzey analizi yapacaklardır.

Sistem oyun alanını bir matris gibi düşünür ve her bir matris karesine komşu elemanların dostluk ve düşmanlık, arazi yapısı ve kaynaklara yakınlığı açısından değerlendirir. Yapılan tarama sonucunda ortaya çıkan iki boyutlu yüzey eğrisinin (etki haritası) analizine göre sistem oyuncuya karşı saldırılabilecek en zayıf noktaları, dost düşman bölgelerin yoğunluğunu, stratejik öneme sahip alanları tespit edebilir. Bunun sınırlı bir kullanımı Age Of Empires oyununda denenmiştir. Günümüz strateji oyunlarında daha gelişmiş türevleri olduğu görülmüştür.

3.4.5. Görüş Netlik Grafı

Görüş netlik grafı tekniğinde ise özellikle helikopter ya da stratejik askeri oyunlarda oyun alanındaki engel teşkil eden nesneleri bir kapalı poligon şeklinde temsil edip en yakın poligon kenarları arasında çizilen hatlarla bir tür graf elde etmeye dayanır. Bu sayede engellere çarpmadan gidilebilecek yere en kısa yol graf üzerinden rahatlıkla bulunabilir. Red Storm'un Force 21 oyununda bu teknikten faydalanılmıştır. Daha sonra diğer oyunlarda da bu teknik geliştirilmiştir.

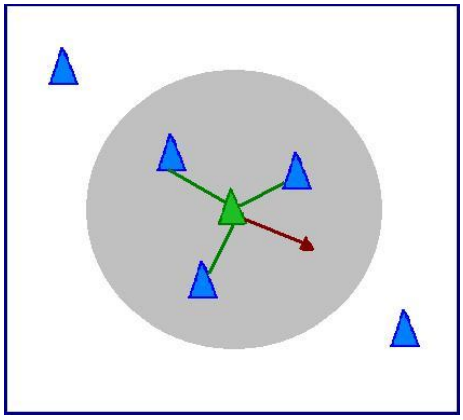
3.4.6. Sürü Hareketi

Sürü hareketi özellikle çok sayıda nesnenin bir hedefe doğru aynı anda hareketi ile oluşur. Bir hedefe doğru koşan onlarca asker, uçuşan parçalar, uçan yüzlerce kuş ya da yüzen balığın hareketini gerçeğe yakın elde etmek oldukça güç bir eylem olarak kabul edilir. Ancak 1986'da ortaya atılan pratik bir algoritma ile bu sorunun üstesinden gelinmiştir.

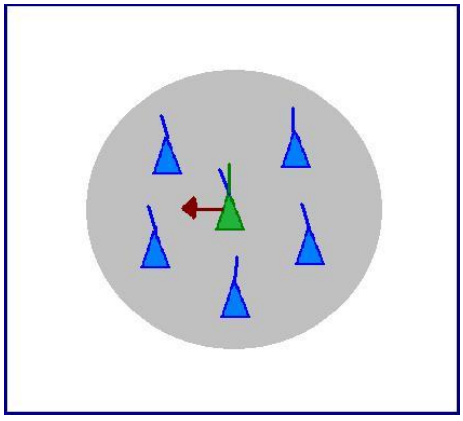
Sürü hareketi algoritmaları, günümüzde takım halinde hareket eden oyun karakterleri ya da gruplar halinde hareket eden, filmlerde gördüğümüz savaş meydanlarındaki bilgisayar üretimi karakter hareketleri için geliştirilmiş hareket algoritmalarıdır.

Algoritmalara göre sürüyü oluşturan nesneler (boids) üç farklı işlemi gerçekleştirirler.

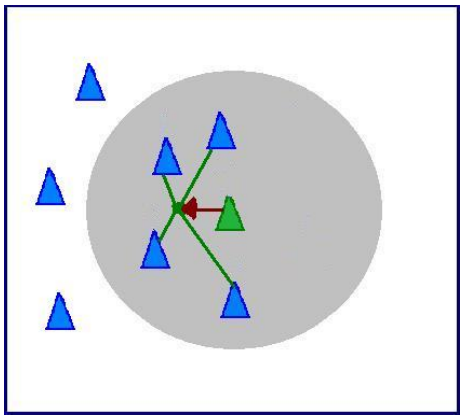
1- Komşularla ve engellerle çarpışmadan kaçınmak (Şekil 3.11).
2- Hız ve doğrultuyu komşularla yaklaşık aynı tutmak (Şekil 3.12).
3- Komşulara yakın durmaya çalışmak (Şekil 3.13).



Şekil 3.11. Komşulara ve engellere çarpışmadan kaçınmak (Reynolds, 1995)



Şekil 3.12. Hız ve doğrultuyu komşularla aynı tutmak (Reynolds, 1995)



Şekil 3.13. Komşulara yakın durmaya çalışmak (Reynolds, 1995)

Bu üç işlemin sonucunda bir hızlanma vektörü elde edilir. Bu işlem sürüyü oluşturan bütün nesnelere uygulandığında gerçekten günlük hayatta gördüğümüz sürü şeklindeki kuşların ya da kalabalık bir insan grubunun hareketine yakın bir hareket elde edilir. Bu sistem Red Allert ve Age of Empires oyunlarında olduğu gibi çoklu insan ve mekanize askeri grupların bir noktaya doğru hareketini ifade etmede yaygın olarak kullanılmıştır.

Bu algoritmalar, sinema efekti olarak 1987'de Batman Returns filmindeki sürü şeklindeki penguenlerin hareketini oluşturmak için kullanılmıştır. Daha sonra benzer şekilde birçok karakterin hareket ettiği Yüzüklerin Efendisi gibi filmlerin animasyon tasarımlarında da bu algoritma kullanılmıştır. Algoritmaların karmaşıklığı şu an için çok fazla olsa da uygun bölme teknikleri ile karmaşıklık azaltılabilir.

3.4.7. Yapay Yaşam

Özellikle son birkaç yılda kullanılan bu oyun konusu Creatures ve Sims gibi örneklerle oldukça ilgi çekmiştir. Temelde genetik algoritmalar kullanılarak gerçek hayatın bilgisayarda bir modelinin oluşturulmaya çalışmasına dayanan yapay yaşamda, canlı nesneler oyuncun müdahalesi ile ortama göre farklı karakterler elde etmekte, öğrenmekte ve tecrübe kazanmaktadır. Genellikle problemlere küresel olarak yaklaşmak yerine küçük parçalar halinde çözümler aranarak öğrenme işlemi gerçekleştirilmektedir.

Yapay yaşam ortamının oyuncuya çok gerçekçi biçimde verilebilmesi için yaşam alanları da akıllı biçimde tasarlanmıştır. Bu metot başka türlü oyunlarda da karakterlerin gerçekçi ve kestirilemez tepkiler vermesini sağlar. Buna göre oyun alanındaki nesnelerin hepsinin yayın yapma özelliği vardır.

Örneğin oyuncumuz bir evde oturuyor olsun; Mutfak büyük bir çaplı çember alanına sürekli olarak "doyma" sinyali yollar. Karakter bu çemberin içerisine yönlendiğinde mutfağa doğru gitme eğilimi gösterir. Mutfaktaki nesnelerden buzdolabı "yemek" sinyali yayınlar. Buzdolabına yönlenen karakter aldığı yemekten yayılan "pişir" sinyali ile ocağa doğru yönlenir. Senaryo uzayıp gitmektedir.

Bu yöntem kendini Baldur's Gate gibi RPG oyunlarında bazı karakterlerin örneğin "tehlike" sinyali yayan yerlerde rahatsız olmaları şeklinde kendini gösterir.

3.5. Bilgisayar Oyunlarında Yapay Zekâ Yazılımları

Oyun geliştiricilere yardımcı olan bazı yapay zekâ yazılım geliştirme araçları bulunmaktadır. Bu konuda iki farklı yaklaşım olan "Extensible AI" ve "Game AI SDK" paketleri bulunmaktadır.

Extensible AI (genişletilebilir yapay zekâ) özellikle Quake, Unreal, Half Life gibi FPS oyunlarında ortamda size karşı savaşan "Bot" isimli yapay zekâlı karakterlerin davranışını tasarlamada kullanılan bir tür script dilidir. Bu tür teknikler RPG oyunlarında da kullanılmıştır.

SDK (software development kit) yazılım geliştirme paketleri arasında ise durum makinelerini uygulamayı kolaylaştıran Direct AI, Fuzzy Durum makineleri için "Spark" yazılım aracı ve karmaşık hareket/reaksiyon tabanlı "Motivate" kütüphanesi geliştirilmiştir. Motivate kütüphanesi "Prince of Persia" isimli 3D oyunda kullanılmıştır.

3.6. Unity 3D Oyun Motoru

Bir bilgisayar oyunu yapım süreci çok uzun ve karmaşık olabilir. Oyun içi grafikler, animasyonlar, sesler, oyun kontrolleri ve ara yüz gibi birçok alan için ayrı ayrı programlama ve tasarım süreçlerine ihtiyaç duyulmaktadır. Bir oyun motoru, üzerinde bulundurduğu hazır kütüphaneler ile her bir alan için hazırlanan temel şablonlar üzerine geliştirilebilecek fonksiyonların kullanılabilmesini sağlamaktadır. Bilgisayar oyunu geliştiren kişi ya da ekipler, bu hazır kütüphaneleri kullanarak, kütüphanelerin içerdiği fonksiyonları kendilerine göre düzenleyip, tasarlamak istedikleri oyunları geliştirmektedir. Bu sayede zaman ve maliyet tasarrufu yapılmaktadır.

Unity 3D oyun motoru, diğer oyun motorlarına göre daha ucuz ve performanslı olmakla birlikte, kullanıcılara sunduğu çoklu platform desteği ve kullanışlı editörleri ile birçok oyun geliştirici tarafından kullanılan güçlü bir oyun motorudur.

Unity 3D, JavaScript, C# ve Boo programlama dillerine destek vermektedir. Tam olarak açık kaynak kodlu bir oyun motoru olmamakla birlikte, script dilleri kullanılarak, oyun içi bütün etmenlerin istenilen şekilde geliştirilmesine olanak sağlanmaktadır. Script dilleri kullanılarak geliştirilen çalışmalar performans olarak düşük kalabilir diye düşünülse de Unity 3D oyun motorunun alt yapısı ile kullanılan script dili ne olursa olsun sistem C++ kadar hızlı çalışmaktadır.

Unity 3D, programlama ortamı olarak kendi üzerinde geliştirilmiş olan MonoDevelop script editörünün yanı sıra Visual Studio gibi farklı programlama editörlerinin de kullanımına olanak sağlamaktadır. İçerdiği hazır kullanıcı kütüphanelerini, kullanışlı editörleri üzerinden kolayca sürükle-bırak yöntemi ile çalışma ortamına ekleme imkânı sunmaktadır.

Gerçek zamanlı render, ışıklandırma, seslendirme ve fizik motoru kütüphanelerinin yanı sıra, multiplayer oyun geliştirmeye olanak sağlayan altyapısı ile hızlı ve kolay bir şekilde oyun geliştirme imkânı sunmaktadır.

Geliştirilen projeler Web, iOS, Android ve Flash ortamlarına derlenebilir ve bu oyun geliştiricileri için daha çok pazar ve satış anlamına gelmektedir.

3.7. Bilgisayar Oyunu Geliştirme Aşamaları

Uluslararası Oyun Geliştiricileri Birliği (IGDA) oyun geliştirme sürecinin ana konularını şu şekilde belirlemiştir;

- Kritik oyun çalışmaları
- Oyunlar ve toplum
- Oyun tasarımı
- Oyun programlama
- Görsel tasarım
- Ses tasarımı
- Etkileşimli hikâye
- Oyun üretimi ve oyun pazarlama

Her bir ana konu kendi içinde alt konulara ayrılabilir. Oyun programlama konusu farklı alt başlıklara ayrılacak olursa;

- Matematik ve fen teknikleri (Newton kanunları)
- Stil ve tasarım prensipleri
- Bilgi tasarımı (veri yapıları)
- Prototip
- Test
- Programlama takımları
- Tasarım/teknoloji sentezi
- Gerçek zamanlı oyun ortamı ve simülasyonları için sistem mimarisi
- Veri tabanı sistemleri
- Oyun mantığı
- Multimedya programlama
- Yapay zekâ

- Network
- Tasarımcılar ve oyun analizcileri için araçlar

Bu konular oyun programlama sürecinin başlıca konularıdır (IGDA, 2003). Bu konular arasında yapay zekâ konusu, günümüz oyunlarında üzerinde durulan ve programlama takımlarında en fazla programcı bulunmasını gerektiren konudur.

Bir bilgisayar oyun geliştirilirken dört ana aşamadan geçer ve oyunun kullanıcıya sunulan son hali dördüncü aşamadan sonra ortaya çıkmaktadır.

Oyun Tasarımı: Amaçlanan son kullanıcı deneyimi ve oyun içinde yer alacak oyun unsurlarının belirlenmesi aşamasıdır.

İçerik Oluşturma: Oyundaki eğlence unsurlarının tasarlanması ve bu unsurların temsil edilme şeklinin belirlenmesidir.

Oyun Programlama: Yazılım mekanizmalarının oluşturulan içeriklere göre tasarlanması ve böylece istenilen oyun tasarımının elde edilmesidir.

Yazılım Yapısının İnşa Edilmesi: Dijital oyun bileşenlerinin birleştirilerek, çalışabilir bir oyun haline getirilmesidir (Chandler, 2006).

4. ARAŞTIRMA BULGULARI VE TARTIŞMA

Takım tabanlı yapay zekâ sistemine sahip karakterlerin hareketlerini gözlemleyebilmek için oyun motorlarına entegre olabilen bir modül geliştirilmiştir. Bu modül kendi başına bir FPS oyun olarak düşünülebilir zira sahip olduğu takım tabanlı yapay zekâ sistemi ile takımlar halinde savaşan 3D karakterler içermektedir.

4.1. FPS Oyunu Yapay Zekâ Modülü Geliştirme Süreci

FPS bir oyun içinde bulunan karakterlerin gerçekçi bir şekilde savaşmalarının çalışmaları sırasında takım halinde hareket eden karakter gruplarının, yaşanan çarpışmalara en yakın sonuçları verdiği tespit edilmiştir. Bu nedenle karakterlerin yapay zekâ sistemlerini gözlemek amacı ile A* algoritmasını kullanarak takım tabanlı çalışan bir yapay zekâ sistemi modülü geliştirilmesi uygun görülmüştür.

FPS türündeki bilgisayar oyunlarında, oyunu oynayan kullanıcının yönettiği bir karakter bulunur. Oyuncu, oyun ortamını bu karakterin gözünden görür ve bu karakteri hareket ettirmek sureti ile oyunda tamamen o karakteri kullanmakla sorumludur. Kendi karakterinden başka diğer dost karakterler ve düşman karakterler bilgisayar kontrolü ile oyunun senaryosu ve oyun içi durumlara göre hareket ederler.

FPS bilgisayar oyununun sahip olduğu senaryoya göre, takım halinde hareket eden askeri birliklerin takım kontrolünü sağlayan karakter, kullanıcının kontrol ettiği karakterdir. Takım halinde hareket eden tüm dost karakterler, kullanıcının karakterinin hareketlerine uyar ve o karakteri takip eder. Düşman takımın karakterleri ise takım kaptanı olarak belirlenen karaktere göre hareket ederler.

Modül geliştirme sürecinde ortam öğeleri, karakterler ve takım tabanlı yapay zekâ sisteminin hazırlanması ve altyapıya uygun olarak bütün nesnelerin düzenlenmesi gerekmektedir.

Yapay zekâ sistemi yazılım kısmında üretilen modülün çalıştırılması için Unity 3D oyun motoru kullanılmıştır. Bu oyun motoru ile demo niteliğinde FPS bir oyun bölümüne ihtiyaç duyulmaktadır. Sistemin çalıştılması ve denenmesi için gereken oyun bölümü üzerinde çevre modelleri ile karakter modellerine ihityaç duyulmaktadır. Oluşturulan 3D model kütüphanesi içinde modül için gerekli olan 3D çevre modelleri ve 3D asker karakterleri yer almaktadır. Karakterlerin davranışlarını test edebilmek ve sistemin çalışmasını kontrol edebilmek amacı ile karakterler ve çeşitli patlamalar için animasyon kütüphanesi oluşturulmuştur.

4.2. 3D Çevre ve Karakter Modelleme

Tasarlanan demo bölüm için gereken 3D model birimleri temel olarak üç gruba ayrılabilir;

Çevre Modelleri: Ortamdaki ağaç, siper, taş, kaya, ot vb. argümanlardır.

Silah Modelleri: Karakterlerin kullanabilecekleri silahlardır.

Karakter Modelleri: Takım elemanları için asker modelleridir.

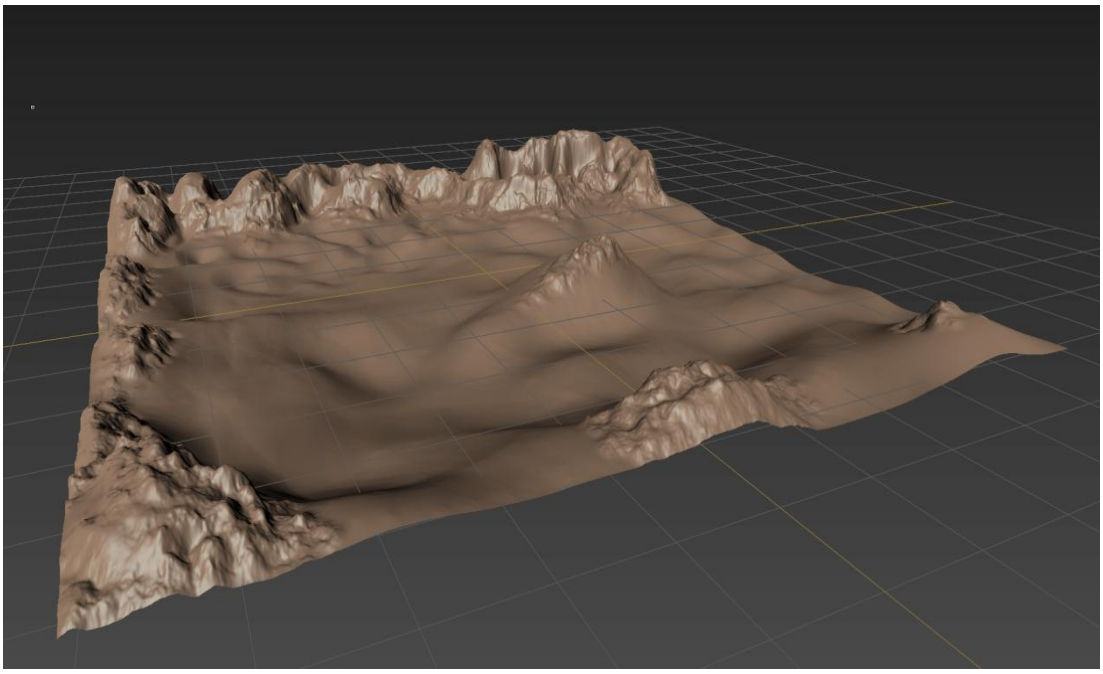
Tasarlanan demo bölüm için gereken animasyon kütühanesi için yapay zekâ sistemini gözlemlemek amacı ile gerekli olan animasyon argümanları iki gruba ayrılabilir;

Karakter Animasyonları: Karakterlerin yürüme, koşma, zıplama, ölme, ateş etme, eğilme, sürünme, konuşma vb. animasyonlarıdır.

Patlama Animasyonları: Bomba ve ateş gibi parçacık animasyonlarıdır.

Bilgisayar oyunu geliştirme sürecinde oyun tasarımı ve görsel tasarım aşamalarında oyun içi çevre ve karakterlerin oluşturulması işlemleri 3D bir oyun projesi için beceri gerektiren çalışmalardır. 3D oyunların temalarına göre iş yükü değişen bu çalışmaların en kapsamlı olanları, savaş temalı olan FPS tarzı oyunlarda görülmektedir. Oyunu oynayan kullanıcının gözünden bütün çevrenin gerçekçi bir şekilde 3D olarak modellenmesi ve objelerin düzenli bir şekilde yerleştirilmesi gerekir. Oyun içi karakterler, hareketli taşıt vb. objelerin de gerçeğe uygun bir şekilde modellenerek animasyon sistemlerinin hazırlanması oyun sistemine uyarlanması gerekir.

FPS tarzı bilgisayar oyunları bölümlere ayrılır ve kullanıcı oynanan bir bölüm üzerindeki görevleri tamamladıktan sonra yeni bir bölüme geçerek oyuna devam eder. Her bir bölüme harite (map) denir ve bu haritalar 3D modelleme programları ya da oyun motoru araçları yardımı ile hazırlanarak oyun içi ortama yerleştirilir. Haritanın oluşturulması aşamalarında öncelikle terrain (zemin) hazırlanır (Şekil 4.1).



Şekil 4.1. 3D Studio Max ortamında hazırlanmış terrain görseli

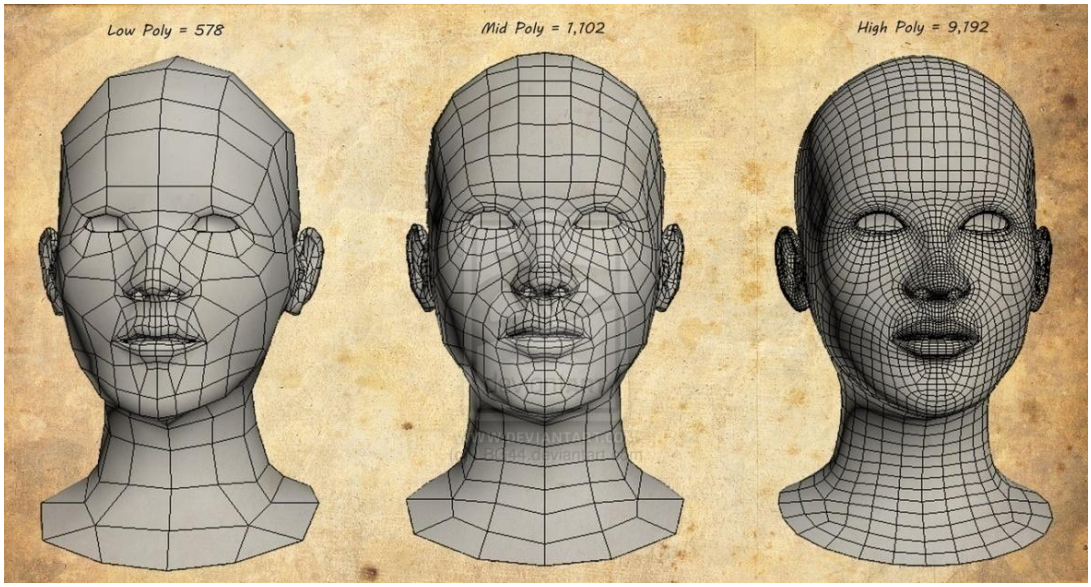
Şekil 4.2'de görüldüğü gibi terrain üzerindeki yükseltiler, çukurlar, dağlar, tepeler gibi detaylar hazırlandıktan sonra bölümün geçtiği coğrafyaya göre ağaçlar, taşlar, göller ve deniz gibi coğrafya detayları terrain üzerine işlenir. Haritanın bu tür coğrafya detayları tamamlandıktan sonra haritanın ortamına göre evler, binalar, okullar, köprüler gibi insan yapıları modellenerek harita üzerine oyun planına uygun olarak yerleştirilir (Sümen, 2013).

Şekil 4.2. Oyun motoru ortamında tamamlanmış harita görseli

3D modelleme çalışmalarının bilgisayar oyunları için farklı teknikleri vardır. Animasyon filmleri ve görsel efektler için yapılan 3D modelleme çalışmaları objelerin sahip olduğu detaylara göre yüksek poligonlu (high-poly) olarak gerçekleştirilir. Bilgisayar oyunları için ise bu durum farklıdır. Oyun içi kullanılan objelerin sahip olduğu poligon sayısına göre objelerin dosya boyutları artar. Fazla poligona sahip olan obje fazla hafıza işgal eder ki bu durum oyunun çalışmasında yavaşlamalara sebep olur. Aynı şekilde objelerin renklendirilmesi için kullanılan texture map (doku haritası) kaplama dosyalarının da fazla hafıza işgal etmemesi gerekir. Bilgisayar oyunlarının tarihsel gelişim sürecinde görülen 8bit renklendirme işlemlerinden sonra, günümüz yeni nesil bilgisayar oyunlarında kullanılan kaplama dosyaları 4096X4096 piksel (pixel, sayısal

görüntünün en küçük nokta birimi) çözünürlüklere ulaşmıştır. Bu sayede daha detaylı ve gerçekçi görseller hazırlanabilmektedir. Yine de bir bilgisayar oyunu içi hazırlanan 3D modeller düşük poligonlu (low-poly) olarak modellenir.

Çevre modellemeleri ve düzenlemeleri tamamlanan haritanın karakterleri için modelleme işlemleri için kullanılan popüler 3D modelleme programlarında olan 3D Studio Max programı ile oyun karakterleri modelleme işlemleri hızlı bir şekilde gerçekleştirilmektedir. Oyunun tarzı, teması ve hikâyesine göre oyun içi karakterlerin modelleme işlemleri yine düşük poligonlu olarak gerçekleştirilir (Şekil 4.3).



Şekil 4.3. Düşük ve yüksek poligonlu olarak hazırlanmış model görseli

Düşük poligonlu olarak modellenen oyun karakterlerinin animasyon sistemlerinin hazırlanabilmesi için insan vücudu iskelet sistemini örnek alan rig (iskelet) sistemi düzenlenmelidir (Sümen, 2012). Düzenlenen bu rig sistemi karakter modeli için hazırlanan mesh (örgü) içine uygun bir biçimde yerleştirilir ve karakter animasyonları düzenlenir. Şekil 4.4'de görüldüğü gibi karakterlerin sahip olması istenen poz ve animasyon sistemleri düzenlenerek karakterler oyun içi aksiyona hazır hale getirilir.

Şekil 4.4. Rig sistemi tamamlanan karakterin animasyona hazır modeli

4.3. FPS Oyun Modülündeki Takım Tabanlı Yapay Zekâ Sistemi

Tasarlanan yapay zekâ sisteminde çalışan yazılımsal alt birimler şunlardır;

Davranış Hiyerarşisi: Her bir karakterin karşılaşacağı durumlar için tasarlanacak olan davranış prosedürleri. Bu prosedürlerin öncelik durumuna göre programlanması ve fonksiyonlarının oluşturulmasıdır.

Takım Modeli: Oluşturulan takımların üye sayılarına göre saldırı ve savunma mekanizmalarının programlanmasıdır.

Takım Çalışması Senaryosu: Takım üyesi karakterlerin rollerine göre hareket fonksiyonlarının programlanmasıdır.

Takım Formasyonu: Takımların hareket düzenleri ve karakterlerin diziliş sisteminin programlanmasıdır.

İletişim Protokolü: Takım içi karakterler arası tehlike ve hedef tayini gibi durumlar için haberleşme protokollerinin programlanmasıdır.

Yapay zekâ sistemine sahip olan ilk bilgisayar oyunlarından günümüz yeni nesil bilgisayar oyunlarına gelinene kadar, oyun içi karakterlerin zekâ seviyelerinde belirli bir artış gözlemlenmiştir. Genel olarak bilgisayar oyunlarındaki yapay zekâlı rakip karakterler uyum sağlayamayan yapay zekâya sahiptirler (Mitchell, 1997). Bu bir dezavantajdır çünkü oyunu oynayan kullanıcı, rakip oyun karakterinin herhangi bir açığını keşfettiği takdirde sürekli olarak galip gelecektir. Bu dezavantaj, yapay zekâlı karakterin hatalarından ders çıkarmasını sağlayarak, duruma uyum sağlaması ile yok edilebilir. Uyum sağlama davranışı machine-learning (makine öğrenme) teknikleri ile geliştirilebilir. Örnek olarak artificial neural networks (yapay sinir ağları) ve evolutionary algorithms (evrimsel algoritmalar) sistemleri kullanılabilmektedir. Yeni nesil oyunlarda bu algoritmaları kullanarak geliştirilmiş yapay zekâ sistemlerini görülebilmektedir.

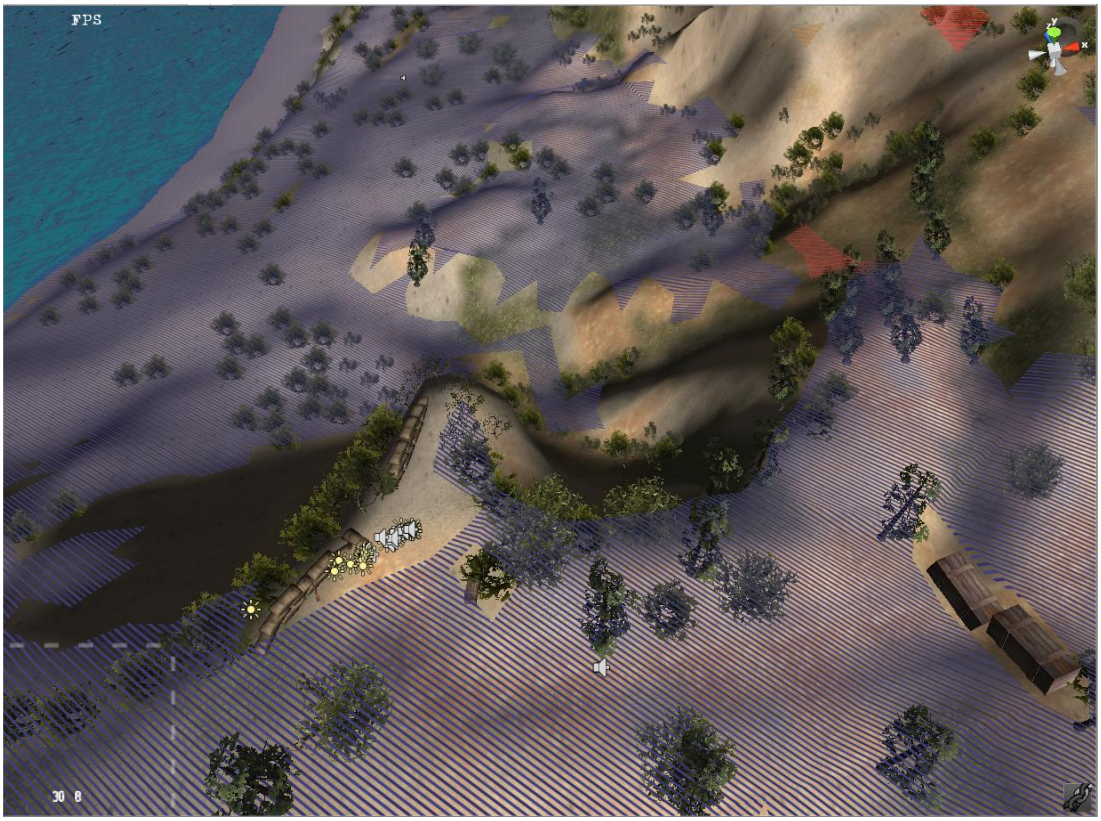
FPS oyununda bulunan karakterlerin, düşman ya da dost karakterlere göre belirlenen yapay zekâ sistemleri, oyun içi ortam ve durumlara göre dinamik bir yapıya sahiptir. Takım tabanlı yapay zekâ sistemine sahip olan karakterlerin birbirleri ile işbirliği yaparak belirlenen görevleri tamamlamaları amaçlanmıştır (Buro, 2003). Dinamik oyun içi şartlara göre yeni stratejiler belirleyerek hareket eden takım tabanlı yapay zekâ sistemleri, Çizelge 4.1'de beliritildiği gibi dört bileşen ile temsil edilebilmektedir.

Çizelge 4.1. Takım tabanlı yapay zekâ sistemi bileşenleri

Bileşen Türü	**Açıklaması**
Bireysel karakter yapay zekâsı	Her bir karakter belli bir çevrede tuzaklar ve saldırıdan korunabilecek şekilde, bireysel olarak hareket edebilecek temel yapay zekâ sistemine sahip olmalıdır.
İletişim araçları	Takım üyeleri saldırı ve düşman görünmesi gibi tehlike durumlarında birbirlerine mesaj göndererek iletişim kurabilir.
Takım organizasyonu	Takım üyelerinin lider, rütbe, sağlıkçı gibi özelliklerine göre diğer üyeler ile olan hiyerarşik komuta sistemi vardır.
Adapte olabilen mekanizma	Takımın oyun içi durumlara göre yeni strateji ve görevlendirmeler yaparak değişen koşullara uyum sağlama sistemi bulunmaktadır.

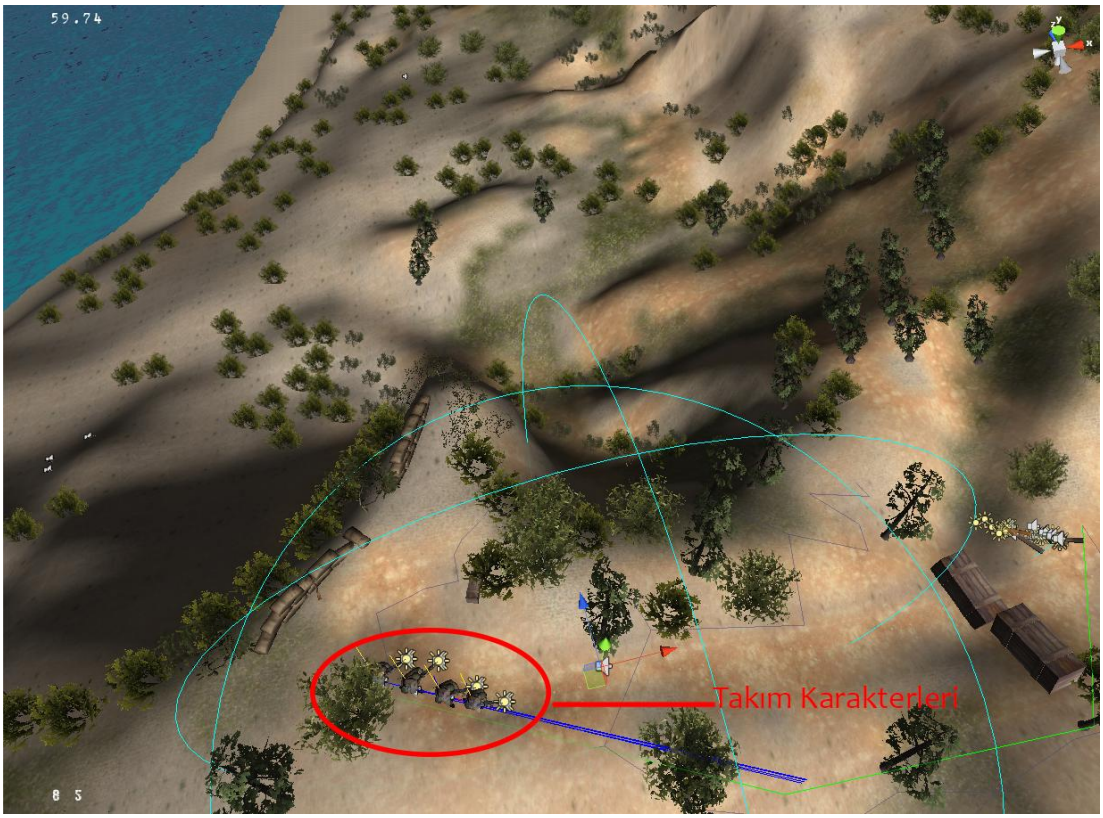
4.4. FPS Oyun Karakterlerinin Yapay Zekâ Sistemi

FPS oyunlarda bulunan karakterlerin yapay zekâ sistemleri temel olarak, düşman ve dost kuvvetlerin davranışları üzerine geliştirilmiştir. Oyun ortamında bulunan karakterler ilk olarak yüzey analizi ve takip edilecek yol ile engellerin hesaplanmasını gerçekleştirir (Şekil 4.5).



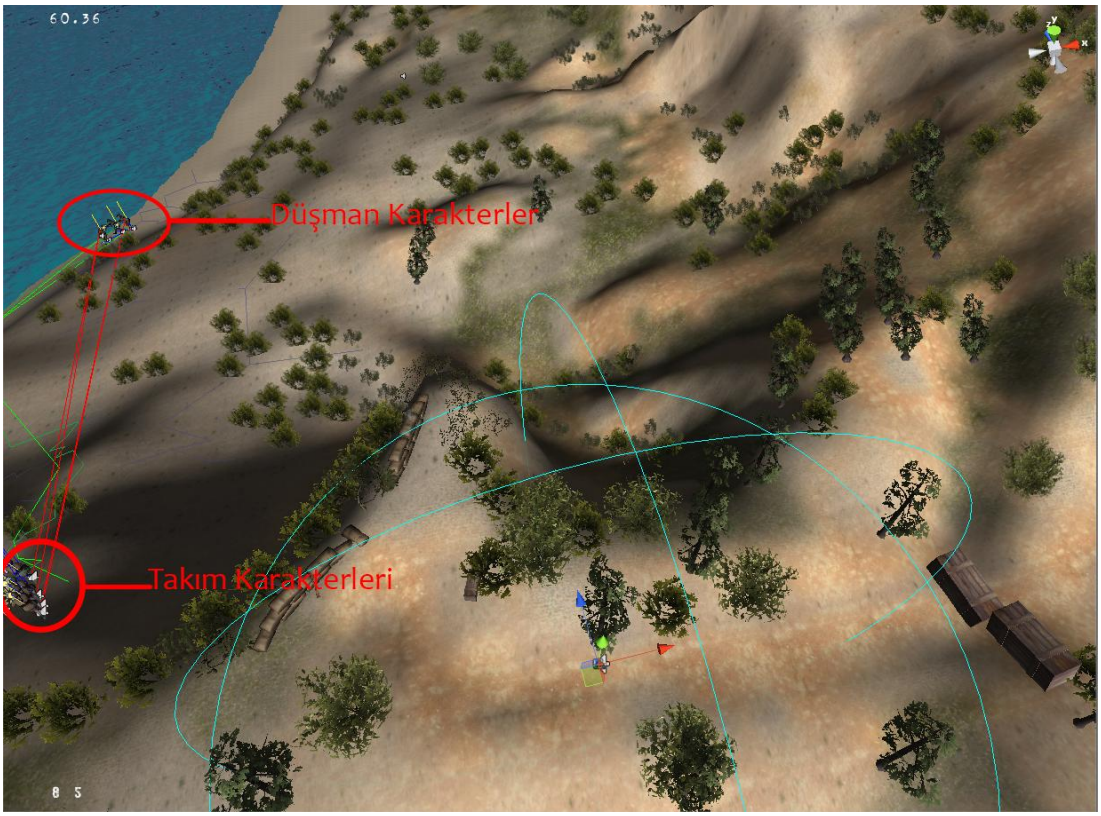
Şekil 4.5. Ortam yüzey analizi haritasını gösteren oyun içi görsel

Kullanıcı kontrollü karakterin takım bireyleri, etrafta düşman yokken kullanıcının kullandığı karakteri takip ederler. Düşman kuvvetleri görüş alanına girdiği anda takımdaki karakterler takım liderini takip etmeyi bırakıp düşman kuvvetlerinin yerini tespit ederek düşman kuvvetlerini etkisiz hale getirmek için düşman kuvvetlerinin bulunduğu yere doğru hareket ederler. Düşman kuvvetleri atış menzile girdiğinde her bir karakter karışık bir şekilde düşman kuvvetlerin askerlerine ateş etmeye başlar (Şekil 4.6 ve Şekil 4.7).



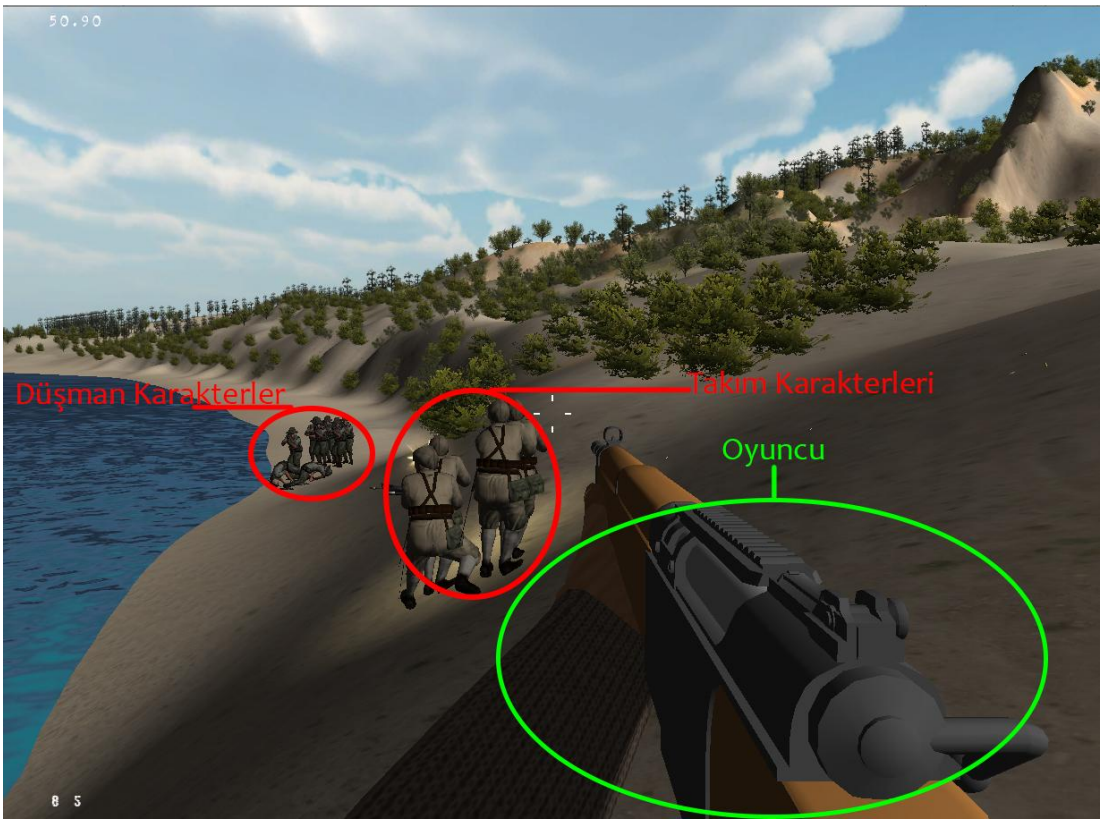
Şekil 4.6. Takım karakterleri boş ortam görüş durumu

Şekil 3.8.3.2'de görüldüğü gibi, kullanıcının takım karakterleri, görüş alanına düşman karakterler girene kadar lider olarak kabul edilen kullanıcıyı takip eder. Görüş alanına düşman karakterler girene kadar bekleme durumundadırlar.



Şekil 4.7. Dost ve düşman karakterlerin karşılaşma durumu

Eğer düşman kuvvetlerin bütün askerleri etkisiz hale getirilirse, takım bireyleri tekrar takım liderini arar takım liderini takip eder (Ponsen vd., 2006). Şekil 4.8'de görüldüğü gibi karşılaşan rakip takımların karakterleri savaşmaya başlayacaktır.



Şekil 4.8. Rakip takım karakterlerinin çatışma durumu

Düşman kuvvetlerinin hareket etme sistemi ise kendi aralarında belirlenen bir takım liderine göre belirlenmiştir. Kullanıcının dost kuvvetlerinden farklı olarak düşman kuvvetlerdeki her birey sürekli olarak karşı kuvvetleri arar ve bir karşı kuvvet askeri bulduğu anda atış menziline girene kadar yaklaşır ve ateş etmeye başlar. Düşman kuvvetlerin ateş etme önceliği oyunu oynayan kişinin kullandığı karakterdir. Eğer kullanıcı karakteri bulunmamışsa diğer takım bireylerine ateş edilir (Munoz-Avila ve Hoang, 2006). Şekil 4.9'da görülen, karşı takım malup edilerek çatışmanın bitmesi durumunda, dost karakterler lider konumunda olan kullanıcıyı takip ederek ve yeni rakipler için görüş alanını taramaya devam edip bekleme durumuna geçmektedir.

Şekil 4.9. Çatışma sonrası bekleme durumu

Bütün bu çatışma anında gerçekleşen olayların gerçeğe yakın olabilmesi için geliştirilen yapay zekâ dinamikleri Çizelge 4.2'de yer almaktadır.

Çizelge 4.2. Yapay zekâ dinamikleri listesi

Dinamik Türü	Açıklaması
EnemyRespawn_asker	Belirlenen sayıya göre takım arkadaşı oluşturulmasını sağlar.
EnemyRespawn_anzak	Belirlenen sayıya göre düşman kuvveti oluşturulmasını sağlar.
AIFollow USE	Takım üyelerinin takım liderini takip etmesini sağlar.
Seeker	Yol oluşturulmasına ve A* algoritmasına göre arazinin yol haritasının çıkarılmasına yardımcı olur.
Funnel Modifier	Funnel algoritmasına gidilecek Path için en düzgün yolu çizer.
Simple Smooth	Gidilecek yol için yumuşatılmış bir yol çizmeyi sağlar.
AstarPath	Yolbulma sisteminin özüdür. Haritanın eğimi ve üzerindeki objelere göre taranmasını sağlayarak, karakterlerin gidebileceği bölgeleri hesaplar.

Oyun içi yapay zekâlı karakterlerin özelliklerine göre karşılaşılan saldırı durumları farklılık gösterebilmektedir. "Bot" ismi verilen yapay zekâlı oyun karakterleri, oyun zorluk düzeyine göre belirlenen zekâ seviyelerine göre farklı davranışlar sergileyebilirler. Zekâ seviyelerine göre karakter takımlarının davranış özellikleri Çizelge 4.3'de belirlenen bir oyun modu türüne göre listelenmiştir (Lee-Urban vd., 2008). Bu oyun modunda karakterler belirli bir çevrede özel olarak işaretlenmiş bölgeleri ele geçirmeye çalışıp o bölge ele geçirildiği takdirde puan kazanarak bölgeyi elde tutmaya çalışmaktadır.

Çizelge 4.3. Uygulamada kullanılan yapay zekâlı karakter takımlarının listesi

Takım Adı	Açıklaması
OpportunisticBot (fırsatçı bot)	Botlar bir egemenlik bölgesinden diğerine hareket eder. Eğer bölge karşı takımın kontrolü altında ise bölgeyi ele geçirir.
PossesiveBot (sahip çıkan bot)	Her bir bot tek bir egemenlik bölgesine yönlenir ve o bölgeyi oyun boyunca ele geçirip elde tutmaya çalışır.
GreedyBot (aç gözlü bot)	Rakip tarafından ele geçirilen bölgeyi tekrar ele geçirmeye çalışır.

Bilgisayar oyunlarının gelişim süreci ve bu süreçte meydana gelen donanımsal teknolojilerin ortaya çıkması ile bilgisayar oyunlarının zamanla daha zeki oyun içi karakterlere yer verdiği gözlemlenmiştir. Donanımsal imkânlar doğrultusunda geliştirilen, klasik oyunlar kategorisine dâhil olabilecek tek

kullanıcılı ve tek bir yapay zekâ sistemine sahip olan oyunlarda görülen ilkel yapay zekâlı karakterler, oyun içi durumlara göre belirlenen doğrultuda hareket ederek belli bir amacı gerçekleştirmeye yönelik karakterlerdir. Satranç gibi tek bir amacı olan bilgisayar oyunu için, oyun içi oyunu oynayan bir karakter olmasa da bu oyunun kendisine yapay zekâlı oyun karakteri denebilir. Satranç oyununu oynayan kullanıcının hamlelerine göre hareket edecek şekilde programlanmış olan bu karakter tek amacı oyunu kazanmak olarak programlanan ilkel yapay zekâlı karakter olarak nitelendirilir (Barnes ve Hutchens, 2002).

Bilgisayar oyunlarında yapay zekânın geliştirilmesinde öncülük eden ve bu konudaki çalışmalar için ilham vermiş olan "Deep Thought" isimli bilgisayar, 1989 yılında zamanın Uluslararası Satranç Şampiyonu David Levy'i malup ederek, yapay zekâlı bir bilgisayar oyunu sisteminin insan rakip karşısında başarılı olabileceğini kanıtlamıştır (Douglas, 2013).

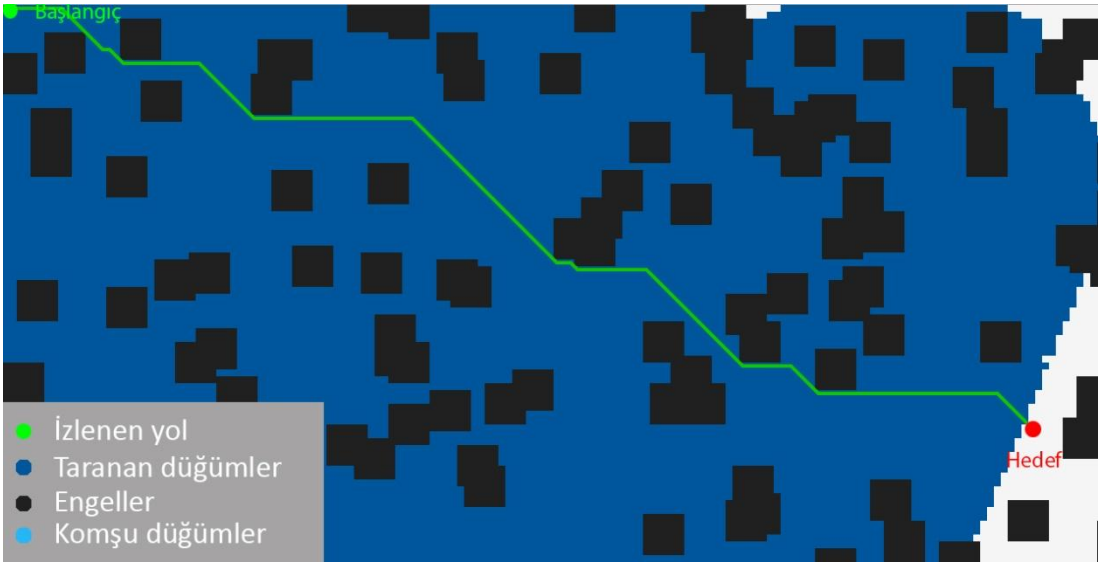
Günümüz yeni nesil 3D bilgisayar oyunlarında görülen sistemlerin gelişmesi sürecinde ortaya çıkan yapay zekâ sistemleri göstermiştir ki oyun içi yapay zekâlı karakterlerin öğrenebilme özelliğine sahip oldukları zaman, kullanıcılara daha gerçekçi ve eğlenceli oyun deneyimi yaşatabilmektedir. Bir oyunun ilgi görmesi ve üreticisinin ürününden gelir elde edebilmesi için oyun içi gerçekçilik ve eğlence deneyemi çok önemlidir. Bu yüzden oyun geliştiren firmalar, oyun içi oynanabilirlik gibi yazılımsal çözümlerin yanı sıra oyunda kullanıcı ile birlikte ya da kullanıcya rakip olarak hareket eden oyun karakterlerinin yapay zekâ sistemlerini geliştirmeye yönelik ayrı yazılım laboratuvarları kurmuşlardır. Kurulan bu laboratuvarlarda geliştirilen yapay zekâ sistemleri o kadar gelişmiştir ki özellikle savaş temalı FPS tarzı bilgisayar oyunlarında görülen yapay zekâlı karakterlerin gerçekçi davranışları gün geçtikçe kullanıcıları şaşırtmaktadır.

A* ve karınca kolonisi algoritmalarını kullanarak takım halinde ve duruma göre bireysel olarak hareket edebilen yapay zekâlı karakterler, gelişmiş fizik motorlarına sahip, ortam objeleri patlamalara karşı duyarlı yıkılabilen

bilgisayar oyunlarında, dinamik olarak değişen ortam şartlarına göre gelişmiş yol bulma becerilerine sahiptirler. "Battlefield", "Call of Duty", "Medal of Honor" gibi savaş temalı bilgisayar oyunlarında görülen dinamik ortamlarda takım halinde hareket eden oyun karakterleri yön bulma, saklanma, saldırma, görev tamamlama gibi eylemleri gerçekleştirmek için diğer karakterler ile iletişim halinde olarak paylaşılan ortam ve durum bilgilerine göre hareket edebilmektedirler. Takım halinde hareket eden karakterler takım içinde öncü, koruyucu, lider gibi özelliklere sahip olarak takım üyelerini koruyup tehlike durumlarında uyarabilmektedirler.

4.4.1. A* ve Dijkstra Algoritmaları Yol Bulma Süreleri

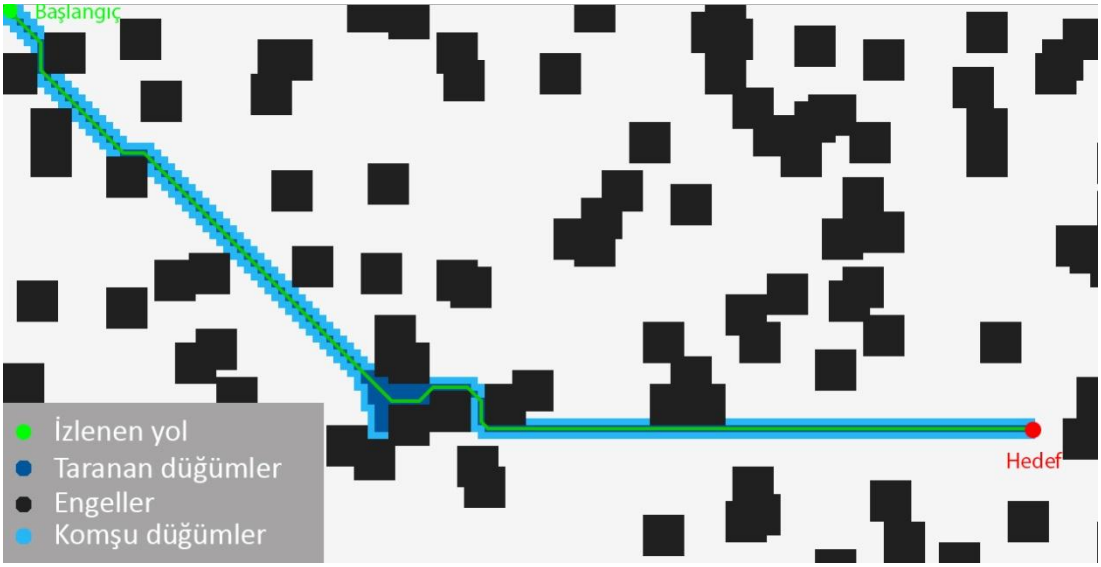
Oyun içi değişken ortam öğeleri, karakterlerin yol bulma işlemlerini sürekli olarak değiştirebilmektedir. Takım tabanlı yapay zekâ sistemi modülünü hazırlarken kullandığımız A* algoritması ile değişen engellere göre hızlı bir şekilde yol bulma işlemleri gerçekleştirilmiştir. Şekil 4.10'da görülen Dijkstra algoritması ile yol bulma işlemi sonucunda çok büyük bir alan taranmış ve toplam 200 saniyede hedefe ulaşılmıştır. Bu durum zaman kaybı ve fazla hafıza yüküne neden olmaktadır.



Şekil 4.10. Dijkstra algoritması ile hedefe ulaşma diyagramı

Dijkstra algoritmasını kullanan oyun karakterleri harita üzerindeki engelleri aşabilmek için bütün haritayı tarayarak hedefe giden yolu bulmak zorunda kalmaktadır. Bu algoritmayı kullanmak oyun içi engel öğeleri sabit olsa bile çok yavaş ve hafıza yükünü arttıran sonuçlara neden olmaktadır.

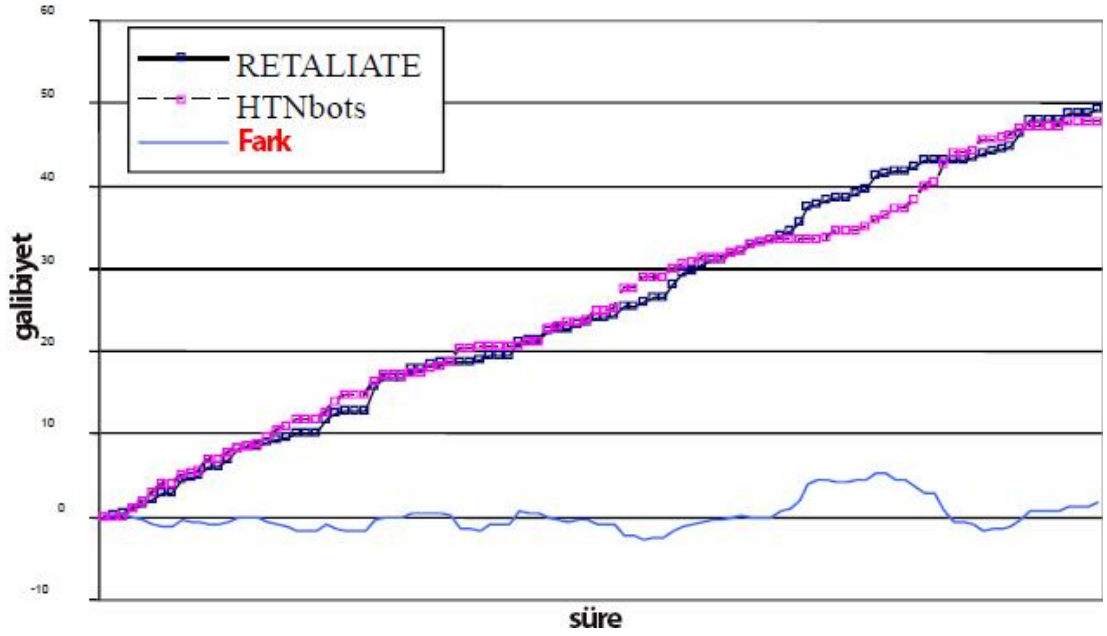
A* algoritmasını kullanarak engellerle dolu bir haritada hedefe ulaşmak 5 saniye kadar bir zaman almaktadır. Algoritmanın sezgisel hareket ederek önce hedefe en yakın komuşları tayin ederek en az maliyetli yolu seçmesi sayesinde hızlı ve az hafıza kullanımı gerektiren sonuçlar elde edilmektedir. Şekil 4.11'de görüldüğü gibi hedefe giden yol için az bir alan taranmış ve daha kısa bir yol üzerinden hedefe ulaşılmıştır.



Şekil 4.11. A* algoritması ile hedefe ulaşma diyagramı

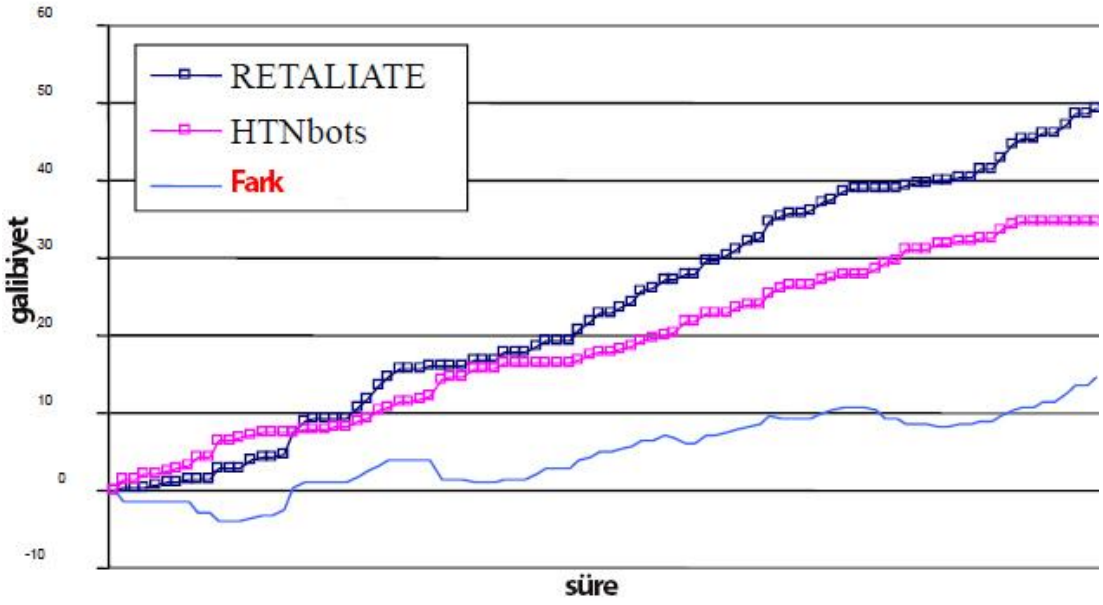
A* algoritmasını kullanarak hareket eden yapay zekâlı oyun karakterleri sürekli aynı oyun stratejisi ile haraket etmeyerek, oyun içi rakip takım davranışları ortam koşullarını analiz ederek, farklı stratejiler belirleyip rakip takımı malup edebilmektedir. Şekil 4.12 ve Şekil 4.13'de görülen grafiklerde, "HTNbots" isimli karakter takımı ile bu takımın rakibi olan "RETALIATE" isimli karakter takımının 2 müsabaka düzenleyerek savaşmaları sırasında ortam şartlarına ve rakibe göre farklı stratejiler belirleyerek galibiyet durumlarını öğrenerek değiştirebildikleri gözlemlenmiştir.

2 farklı müsabaka sonucu savaşan takımlardan daha gelişmiş ve öğrenebilir yapay zekâ sistemine sahip olan "RETALIATE" takımı, ilk müsabakada görülen karakter davranışları ve ortam durumuna göre yeni savaşma stratejileri geliştirerek 2. müsabakada daha başarılı bir sonuç elde edebilmiştir.



Şekil 4.12. RETALIATE ve HTNbots takımları 1. müsabaka galibiyet grafiği (Smith vd., 2007).

İlk müsabaka sonucunda iki takımın da dengeli bir karşılaşma ile yakın galibiyet sayısı elde ettiği görülmüştür.



Şekil 4.13. RETALIATE ve HTNbots takımları 2. müsabaka galibiyet grafiği (Smith vd., 2007).

İkinci müsabakanın başlaması ile birlikte "RETALIATE" takımının rakip takımın davranışlarına verdiği tepkiler sonucu galibiyet sayısında artış görülmüştür.

Oyun içi oynanabilirlik bir yeni nesil oyunların önemli unsurlarından olmakla birlikte oyunda kullanıcının karşılaştığı yapay zekâ sistemi ile hareket eden karakterlerin davranışları o oyunun kaliteli bir yapım olmasındaki en önemli etken haline gelmiştir. Takım tabanlı yapay zekâ sisteminin değişen oyun içi ortamda A* algoritması ile kolay bir şekilde yön bulma kabiliyeti ile gerçekçi karakter hareketleri elde edilebilmektedir.

5. SONUÇ VE ÖNERİLER

Bilgisayar oyunlarının tarihsel gelişimini araştırarak, donanımsal gelişmeler ve kullanıcı talepleri doğrultusunda, bilgisayar oyunlarında görülen yapay zekâ sistemlerinin gün geçtikçe daha gerçekçi hale geldiğini görülmüştür.

Çalışmalarımız ile elde edilen bulgular doğrultusunda takım tabanlı yapay zekâ sistemini kullanabileceğimiz maliyet ve kullanılabilirlik açısından en uygun oyun motoru olan Unity 3D oyun motorunu kullanarak, FPS oyunu için takım tabanlı yapay zekâ modülü geliştirilmiştir. Böyle bir yapay zekâ modülünün geliştirilebilmesi için gerekli olan ortam ve karakter modelleri için 3D Studio Max, ZBrush, UVLayout, Topogun, CrazyBump, BodyPaint 3D ve Photoshop programları kullnılarak gerekli modeller ve animasyonlar tamamlanmıştır. Daha sonra zemin, ağaç, siper vb. ortam materyallerini Unity 3D oyun motoru sahnesi üzerinde birleştirerek, animasyon sistemleri programlanmış olan oyun içi karakterleri sahneye yerleştirilmiştir. Takım halinde hareket ederek kullanıcıyı takım lideri olarak kabul edip rakip takım ile savaşacak oyun karakterlerinin yapay zekâ sistemi geliştirilmiştir. Bu sistem için takım içinde haberleşebilen karakterlerin yol bulma becerileri için A* algoritması kullanılmıştır. Dijkstra ve Best-First Search algoritmalarının zaman ve maliyet olarak zayıf kaldığı görülmüştür. Sağlıklı ve hızlı çözümleri işlemciyi yormadan gerçekleştiren A* algoritmasının zamanlama sonuçları karşılaştırılmıştır. Bütün bu işlemlerin sonucu, oyun içi rakip takımların karşılaşması ve savaşması durumunda oluşan bekleme ve saldırma gibi olayların A* algoritması ile çok daha kısa ve etkin bir şekilde gerçekleştiği gözlemlenmiştir.

Bu çalışma sonucu göstermiştir ki hali hazırda oyun geliştiriciler için çoklu ortam desteğine sahip olan Unity 3D oyun motoru, maliyet ve kullanılabilirlik açısından en uygun oyun motoru konumundadır. Takım tabanlı yapay zekâ modülü geliştirerek takım içi karakterlerin yapay zekâ sistemlerinin analizinde verilerin en hızlı ve doğru bir şekilde elde edilebilmesi için oyun içi ortam ve oyun karakterlerinin 3D modelleme işlerinin, bilgisayar oyunlarına uygun bir şekilde düşük poligon olarak modellenmesi gerektiği görülmüştür. Kullanılan

donanımın düşük poligon seviyelerinde daha hızlı sonuç verdiğini gözlemlenmiştir. Öğrenebilen ve rakip takımın durumuna göre davranışlarını değiştirebilen oyun karakterlerinin performanslı bir şekilde hareket edebilmeleri için daha güçlü işlemcilere ihtiyaç duyduğu görülmüştür.

Günümüz yeni nesil oyunlarının ihtiyaç duyduğu yüksek donanım özelliklerine, oyun içi görsel ve yapay zekâ sistemlerinin daha gelişmiş ve karmaşık yapıda olmaları sebep olmaktadır. Yüksek satış rakamlarına ulaşabilecek oyunlar için olmazsa olmaz unsurlardan biri olan karakter yapay zekâ sistemlerinin önemi göze alındığında, takım tabanlı yapay zekâ sisteminin yeni nesil oyunlar için vazgeçilmez bir unsur olarak gün geçtikçe daha gerçekçi ve etkili sistemler haline gelmesi beklenmektedir. A* algoritmasını kullanarak takım halinde hareket eden karakterlerin gerçekçi davranışlar sergiledikleri gözlemlenmiştir.

Bundan sonraki çalışmalarda, günümüz yeni nesil 3D bilgisayar oyunlarında kullanılabilmesi için araştırmaları devam eden D* algoritması gibi yol bulma algoritmaları gibi karmaşık algoritmalar üzerinde çalışarak, literatüre kazandırılabilecek yeni nesil yol bulma sistemleri geliştirme ve bilgisayar oyunlarında görülen dinamik ortam değişkenlerini kendi çıkarları doğrultusunda düzenleyebilme becerisine sahip olan karakter yapay zekâ sistemleri geliştirme amaçlanmaktadır.

KAYNAKLAR

Barnes, P., Hutchens, J., 2002. Testing Undefined Behavior as a Result of Learning. AI Game Programming Wisdom, Rabin, S. (Ed.), Charles River Media, 615-623, USA.

Buckland M., 2005. Programming Game AI By Example. Wordware Publishing Inc, 495p, USA.

Buro, M., 2003. RTS Games as Test-Bed for Real-Time AI Research. Department of Computing Science, Canada Proceedings of The 7th Joint Conference on Information Science, JCIS 2003, 2003, University of Alberta, Canada.

Chandler, H. M., 2006. The Game Production Handbook. Game Development Series. Charles River Media, 350p, USA.

De Castro, L. N., Von Zuben F.J., 2000. The Clonal Selection Algorithm With Engineering Applications. GECCO 2000, July 2000, Las vegas, Nevada, 36-37.

De Castro, L. N., Timmis J., 2002. A Novel Approach to Pattern Recognition. In L. Alonso, J. Corchado, and C. Fyfe, (Ed.), Artificial Neural Networks In Pattern Recognition, 67-84, University of Paisley, UK.

De Castro, L. N., Von Zuben F.J., 2002. Learning and optimization Using the Clonal Selection Principle, In The Special Issue on Artificial Immune Systems of the Journal IEEE Transactions on Evolutionary Computation, June 2002, 239-251.

Delling, D., Sanders, P., Schultes, D., Wagner, D., 2009. Engineering Route Planning Algorithms. Algorithmics of Large and Complex Networks. Springer, 117–139.

Dorigo M., Gambardella LM., 1997 Ant Colony System: A Cooperative Learning Approach to The Traveling Salesman Problem. IEEE Transactions on Evolutionary Computation, 1(1), 53–66.

Douglas, J. R., December 2013. Chess 4.7 versus David Levy. BYTE. Erişim Tarihi: 17.09.2013. https://archive.org/details/byte-magazine-1978-12-rescan

Dowsland, K., A., 1995. Simulated Annealing, In Reeves C. R. (Ed), Modern Heuristic Techniques for Combinatorial Optimizations (20-69), McGraw-Hill Book Company, Newyork.

Efe, E., 2001 Modern Oyun Programlamada Yapay Zekâ Konuları. Erişim Tarihi: 09.07.2014.
http://members.tripod.com/engin_efe/makaleler/yapayzeka2.htm

Forrest, S., Perelson, A., Allen, L., 1994, Self-Nonself Discrimination in a Computer, Proceedings of the IEEE Symposium on Research in Security and Privacy, 202-212.

Herman, L., 2001. Phoenix: The Fall & Rise of Videogames. Rolenta Press, 388p, USA.

Herman, L., Horwitz, J., Kent, S., Miller, S., 2002. The History of Video Games. Erişim Tarihi: 13.07.2013, http://en.wikipedia.org/wiki/GameSpot

Hoang, H., Lee-Urban, S., Munoz-Avila, H., 2005. Hierarchical Plan Representations for Encoding Strategic Game AI. Proceedings of Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE-2005, AAAI Press, 3-8.

IGDA International Game Developers Association, 2003. IGDA Curriculum Framework: The Study Of Games and Game Development. Erişim Tarihi: 08.12.2013. http://www.igda.org/academia/IGDA_Curriculum_Framework_Feb03.pdf

Kent, S., 2001. The Ultimate History Of Video Games. Prima Publishing, 608p, USA.

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

Kudler, A., 2007. Timeline: Video Games. Erişim Tarihi: 09.07.2014. http://www.infoplease.com/spot/gamestimeline4.html

Laird, J. E., 2005. History of Computer Games. Erişim Tarihi: 09.07.2014. http://www.emunix.emich.edu/~evett/GameProgramming/History.pdf

Lee-Urban S., Vasta, M., Munoz-Avila, H., 2008. Learning Winning Policies in Team-Based First-Person Shooter Games. Under Review for AI Game Programming Wisdom 4. Charles River Media, USA.

Lobo, D. G., 2007. Playing with Urban Life. In Borries, Friedrich; Walz, Steffen P., Böttger, Matthias. Space Time Play Computer Games, Architecture and Urbanism: The Next Level. 2007, Basel: Birkhauser.

Markoff, J., 2009. The Coming Superbrain. The New York Times. Erişim Tarihi: 27.04.2013. http://www.nytimes.com/2009/05/24/weekinreview/24markoff.html?_r=0

Mitchell, T. M., 1997 Machine Learning, McGraw-Hill Science/Engineering/ Math 81-127, 368-390.

Mocholi J. A., Jaen, J., Catala, A., Navarro E., 2010. An Emotionally Biased Ant Colony Algorithm For Pathfinding In Games. Expert Systems With Applications, 4921–4927.

Munoz-Avila, H., Hoang, H., 2006. Coordinating Teams of Bots with Hierarchical Task Network Planning. To appear In: AI Game Programming Wisdom 3. Charles River Media, USA.

Patel, A., 2014. Heuristics. Erişim Tarihi: 09.07.2014. ttp://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html

Ponsen, M., Munoz-Avila, H., Spronk, P., Aha, D., 2006. Automatically Generating Game Tactics with Evolutionary Learning. AI Magazine. AAAI press.

Pratt, D. R., 1996. Next Generation Computer Generated Forces. Proceedings of the Sixth Computer Generated Forces and Behavioral Representation, 3-8.

Rabin S., 2002. AI Game Programming Wisdom. Charles River Media, 579-589, USA.

Reynolds, C., 1995. Boids. Erişim Tarihi: 09.07.2014. http://www.red3d.com/cwr/boids/

Russell, S., Norvig, P. 2003. Artificial Intelligence: A Modern Approach. Upper Saddle River, Prentice Hall, 1078p, New Jersey, USA.

Salzer, A., Wiederschein, F., Thomas, Philipp., Schlufter, T., Mrosk, C., 2010. Lösung des TSP Problems Mittels Ameisensystemen. 5-7, Berlin.

Smith, M., Lee-Urban S., Munoz-Avila, H., 2007. RETALIATE: Learning Winning Policies In First-Person Shooter Games. Proceedings of the Seventeenth Innovative Applications of Artificial Intelligence Conference, AAAI Press, 1801-1806.

Sterren, W., 2002. Squad Tactics: Team AI and Emergent Maneuvers. AI Game Programming Wisdom, Rabin, S. (Ed.), Charles River Media, 233-246, USA.

Sümen, A. M., 2012. 3D Studio Max Karakter Modelleme ve Dokulandırma. Kodlab Yayınları, 282s, İstanbul.

Sümen, A. M., 2013. 3D Studio Max İç ve Dış Mekan Modelleme. Kodlab Yayınları, 262s, İstanbul.

Yagoda, M., 2008. 1972 Nutting Associates Computer Space. Erişim Tarihi: 17.09.2013. http://www.pinrepair.com/arcade/cspace.htm

Yannakakis G., Hallam, J., 2007. Towards Optimizing Entertainment In Computer Games, Applied Artificial Intelligence. 933-972.

Young, J. S., 1988. Steve Jobs: The Journey Is The Reward. Glenview, IL, USA: Scott, Foresman And Company. 90–91, 94, USA.

Zeng, W., Church, R. L., 2009. Finding Shortest Paths On Real Road Networks: The Case For A*. International Journal of Geographical Information Science 23 (4), 531–543.

EKLER

EK A. A* Script

EK B. AIFollwUSE Script

EK C. EnemyRespawn Script

EK A. A* Script Bloğu

```
function A*(start,goal)
    closedset := the empty set
    openset := {start}
    came_from := the empty map

    g_score[start] := 0

    f_score[start] := g_score[start] + heuristic_cost_estimate(start, goal)

    while openset is not empty
        current := the node in openset having the lowest f_score[] value
        if current = goal
            return reconstruct_path(came_from, goal)

        remove current from openset
        add current to closedset
        for each neighbor in neighbor_nodes(current)
            if neighbor in closedset
                continue
            tentative_g_score := g_score[current] + dist_between(current,neighbor)

            if neighbor not in openset or tentative_g_score < g_score[neighbor]
                came_from[neighbor] := current
                g_score[neighbor] := tentative_g_score
                f_score[neighbor] := g_score[neighbor] + heuristic_cost_estimate(neighbor, goal)
                if neighbor not in openset
```

```
                add neighbor to openset

    return failure


function reconstruct_path(came_from, current_node)
    if current_node in came_from
        p := reconstruct_path(came_from, came_from[current_node])
        return (p + current_node)
    else
        return current_node
```

EK B. AIFollowUSE Script Bloğu

```
using UnityEngine;
using System.Collections;
using Pathfinding;
 [RequireComponent (typeof(Seeker))]
public class AIFollowUSE : MonoBehaviour {
	public Transform target;
	public float repathRate = 0.3F;
	public bool isOnlyFollowPath = false;
		public float pickNextWaypointDistance = 1F;
	protected Seeker seeker;
	protected Transform tr;
	protected Vector3 targetPosition;
	protected float lastPathSearch = -9999;
	protected int pathIndex = 0;
		protected Vector3[] path;
	public void OnEnable () {
		targetPosition = Vector3.zero;
		Repath ();
	}
	public void Start () {
		seeker = GetComponent<Seeker>();
		tr = transform;
	}
	public void TargetPosition (Vector3 pos) {
		targetPosition = pos;
	}
```

```
	public void OnPathComplete (Path p) {
		if (!gameObject.active) {
			return;
		}
		if (p.error) {
			return;
		}
		path = p.vectorPath;
		float minDist = Mathf.Infinity;
		int notCloserHits = 0;
		for (int i=0;i<path.Length-1;i++) {
			float dist = Mathfx.DistancePointSegmentStrict
(path[i],path[i+1],tr.position);
			if (dist < minDist) {
				notCloserHits = 0;
				minDist = dist;
				pathIndex = i+1;
			} else if (notCloserHits > 6) {
				break;
			}
		}
	}
	public IEnumerator WaitToRepath () {
		float timeLeft = repathRate - (Time.time-lastPathSearch);
		yield return new WaitForSeconds (timeLeft);
		Repath ();
	}
	public void Stop () {
```

```
        pathIndex = -1;
    }
        public virtual void Repath () {
        lastPathSearch = Time.time;
        if( isOnlyFollowPath == true )
        {
            if (seeker == null || target == null || !seeker.IsDone ()) {
                StartCoroutine (WaitToRepath ());
                return;
            }
            seeker.StartPath (tr.position,target.position,OnPathComplete);
        }
        else {
            if (seeker == null || targetPosition == Vector3.zero) {
                StartCoroutine (WaitToRepath ());
                return;
            }
            seeker.StartPath (tr.position,targetPosition,OnPathComplete);
        }
    }
    public void PathToTarget (Vector3 targetPoint) {
        lastPathSearch = Time.time;
        if (seeker == null) {
            return;
        }
        seeker.StartPath (tr.position,targetPoint,OnPathComplete);
    }
```

```
	public virtual void ReachedEndOfPath () {

	}

	public void Update () {


		if (path == null || pathIndex >= path.Length || pathIndex < 0) {

			return;

		}

				Vector3 currentWaypoint = path[pathIndex];

		currentWaypoint.y = tr.position.y;

		while ((currentWaypoint - tr.position).sqrMagnitude <
pickNextWaypointDistance*pickNextWaypointDistance) {

			pathIndex++;

			if (pathIndex >= path.Length) {

				if ((currentWaypoint - tr.position).sqrMagnitude <
(pickNextWaypointDistance*0.2)*(pickNextWaypointDistance*0.2)) {

					ReachedEndOfPath ();

					return;

				} else {

					pathIndex--;

					break;

				}

			}

			currentWaypoint = path[pathIndex];

			currentWaypoint.y = tr.position.y;

		}

		SendMessage("CurrentWaypoint", currentWaypoint,
SendMessageOptions.DontRequireReceiver);

	}

}
```

EK C. EnemyRespawn Script Bloğu

```
#pragma strict

#pragma downcast

import System.Collections.Generic;

@script AddComponentMenu("Shooter Engine/Level/Enemy Respawn")

/*

	EnemyRespawn.js

	Bu script oyuncunun görüş alanı içinde olup olmadığını kontrol eder.

	*/

enum RespawnIf {

	AllCases = 0,

	PlayerHasFirstKey = 1,

	PlayerUsedFirstKey = 2,

	PlayerHasSecondKey = 3,

	PlayerUsedSecondKey = 4,

	PlayerHasThirdKey = 5,

	PlayerUsedThirdKey = 6,

	ItemFound = 7

}

var respawnType = RespawnIf.AllCases;

var foundItemNumber : int = 1;

var oneTimeRespawn : boolean = true;

var spawnRange : float = 50.0;

var deleteRange : float = 200.0;

var gizmoName : String;

var enemyPrefab : GameObject;

var respawnEffect : GameObject;
```

```
var respawnSound : AudioClip;

var respawnPoint : GameObject;

var respawnDelayMin : float = 0.0;

var respawnDelayMax : float = 0.0;

var numberOfEnemies : int = 1;

var radius : float = 5;

var spawnInGrid = false;

var gridX = 3;

var gridY = 2;

var spacing : float = 5.0;

var selectEnemyWeapon : boolean = true;

var weaponIndex : int = 0;

var setSightRange : boolean = false;

var sightRange = 40.0;

var setAttackRange : boolean = false;

var attackRange = 50.0;

var setRetreat : boolean = false;

var retreatProbability = 0.2;

var setShootRange : boolean = false;

var shootRange = 30.0;

var setBurstIntervalMin : boolean = false;

var burstIntervalMin : float = 0.5;

var setBurstIntervalMax : boolean = false;

var burstIntervalMax : float = 1.0;

var setBurstTimeMin : boolean = false;

var burstTimeMin : float = 0.05;

var setBurstTimeMax : boolean = false;
```

```
var burstTimeMax : float = 0.3;
var setShoot : boolean = false;
var shootEnemies : boolean = true;
var setMove : boolean = false;
var moveTowardsEnemies : boolean = true;
var setSearch : boolean = false;
var searchForEnemies : boolean = true;
var waypointsGroupName : String[] = ["Shared"];
private var target : Transform;
private var wasOutside : boolean = true;
private var updateInterval : float = 1.0;
private var arr : List.<GameObject> = new List.<GameObject> ();
private var randomNumber : int;
function Start () {
	if( GameObject.FindWithTag("Friend") != null )
		target = GameObject.FindWithTag("Friend").transform;
	else
		target = GameObject.FindWithTag("Player").transform;
	randomNumber = Random.Range(0, waypointsGroupName.length);
	var startIn : float = Random.Range(0.0, updateInterval);
	InvokeRepeating("HideEnemies", startIn, updateInterval);
}
		} else if (!oneTimeRespawn) {
			IsAnyoneAlive();
		}
	}
}
```

```
function IsAnyoneAlive () {
	var ar : List.<GameObject> = new List.<GameObject> ();
	for (var go : GameObject in arr) {
		if (go != null) {
			var anyoneDamage : CharacterDamage =
go.GetComponent.<CharacterDamage>();
			var anyoneReceiver : DamageReceiver =
go.GetComponent.<DamageReceiver>();

			if ((anyoneDamage != null && anyoneDamage.hitPoints > 0)
			|| (anyoneReceiver != null && anyoneReceiver.hitPoints > 0)
			|| (anyoneDamage == null && anyoneReceiver == null))
				ar.Add(go);
		}
	}
	if (ar.Count == 0 && arr.Count != 0) {
		arr.Clear();
		wasOutside = true;
	}
}

function HideEnemies () {
	if (arr.Count > 0) {
		for (var i : int = 0; i < arr.Count; i++) {
			if (arr[i] != null) {
				var characterDamage : CharacterDamage =
arr[i].GetComponent.<CharacterDamage>();
				var damageReceiver : DamageReceiver =
arr[i].GetComponent.<DamageReceiver>();
```

```
                    if (Vector3.Distance(arr[i].transform.position,
Camera.main.transform.position) > deleteRange) {
                        if (arr[i].active)
                            arr[i].SetActiveRecursively(false);
                    } else if (!arr[i].active) {
                        if (characterDamage != null) {
                            if (characterDamage.hitPoints > 0) {
                                arr[i].SetActiveRecursively(true);

        arr[i].BroadcastMessage("SelectWeapon", weaponIndex,
SendMessageOptions.DontRequireReceiver);
                            } else
                                arr.RemoveAt(i);
                        } else if (damageReceiver != null) {
                            if (damageReceiver.hitPoints > 0) {
                                arr[i].SetActiveRecursively(true);

        arr[i].BroadcastMessage("SelectWeapon", weaponIndex,
SendMessageOptions.DontRequireReceiver);
                            } else
                                arr.RemoveAt(i);
                        } else
                            arr[i].SetActiveRecursively(true);
                    }
                }
            }
        }
}
```

```
function Respawn () {
	yield WaitForSeconds(Random.Range(respawnDelayMin, respawnDelayMax));
		if (!wasOutside)
		return;
	if (!spawnInGrid) {
		for (var i : int = 0; i < numberOfEnemies; i++) {
			var angle : float = i * Mathf.PI * 2 / numberOfEnemies;
			var pos : Vector3 = Vector3(Mathf.Cos(angle), 0, Mathf.Sin(angle)) *
radius;
			var newpos : Vector3 = pos + respawnPoint.transform.position;
			var radiusEnemy : GameObject = Spawner.Spawn(enemyPrefab,
newpos, respawnPoint.transform.rotation);
			arr.Add(radiusEnemy);
			SetParams(radiusEnemy);
			if (respawnEffect != null)
				Spawner.Spawn(respawnEffect, newpos,
respawnPoint.transform.rotation);
			if (respawnSound != null)
				AudioSource.PlayClipAtPoint(respawnSound, newpos);
		}
	} else {
		for (var y : int = 0; y < gridY; y++) {
			for (var x : int = 0; x < gridX; x++) {
				var gridpos : Vector3 = Vector3(x, 0, y) * spacing;
				var newgridpos : Vector3 = gridpos +
respawnPoint.transform.position;
				var gridEnemy : GameObject =
Spawner.Spawn(enemyPrefab, newgridpos, Quaternion.AngleAxis(Random.Range(0, 360),
Vector3.up));
				arr.Add(gridEnemy);
```

```
                        SetParams(gridEnemy);
                        if (respawnEffect != null)
                            Spawner.Spawn(respawnEffect, newgridpos,
respawnPoint.transform.rotation);
                        if (respawnSound != null)
                            AudioSource.PlayClipAtPoint(respawnSound,
newgridpos);
                    }
                }
            }
            wasOutside = false;
}
function SetParams (enemy : GameObject) {
            if (selectEnemyWeapon)
                enemy.BroadcastMessage("SelectWeapon", weaponIndex,
SendMessageOptions.DontRequireReceiver);
            var friendlyAI : FriendlyAI = enemy.GetComponent.<FriendlyAI>();
            var enemyAI : EnemyAI = enemy.GetComponent.<EnemyAI>();
            var animalAI : AnimalAI = enemy.GetComponent.<AnimalAI>();
            var sentryGun : SentryGun = enemy.GetComponent.<SentryGun>();
            if (friendlyAI != null) {
                if (setAttackRange)
                    friendlyAI.attackRange = attackRange;
                if (setShootRange)
                    friendlyAI.shootRange = shootRange;
                if (setBurstIntervalMin)
                    friendlyAI.burstIntervalMin = burstIntervalMin;
                if (setBurstIntervalMax)
                    friendlyAI.burstIntervalMax = burstIntervalMax;
```

```
                if (setBurstTimeMin)
                        friendlyAI.burstTimeMin = burstTimeMin;
                if (setBurstTimeMax)
                        friendlyAI.burstTimeMax = burstTimeMax;
                if (setShoot)
                        friendlyAI.shootEnemies = shootEnemies;
                if (setMove)
                        friendlyAI.moveTowardsEnemies = moveTowardsEnemies;
                if (setSearch)
                        friendlyAI.searchForEnemies = searchForEnemies;
                        friendlyAI.waypointsGroupName =
waypointsGroupName[randomNumber];
        } else if (enemyAI != null) {
                if (setAttackRange)
                        enemyAI.attackRange = attackRange;
                if (setShootRange)
                        enemyAI.shootRange = shootRange;
                if (setBurstIntervalMin)
                        enemyAI.burstIntervalMin = burstIntervalMin;
                if (setBurstIntervalMax)
                        enemyAI.burstIntervalMax = burstIntervalMax;
                if (setBurstTimeMin)
                        enemyAI.burstTimeMin = burstTimeMin;
                if (setBurstTimeMax)
                        enemyAI.burstTimeMax = burstTimeMax;
                if (setShoot)
                        enemyAI.shootEnemies = shootEnemies;
                if (setMove)
```

```
                enemyAI.moveTowardsEnemies = moveTowardsEnemies;
            if (setSearch)
                enemyAI.searchForEnemies = searchForEnemies;
            enemyAI.waypointsGroupName = waypointsGroupName[randomNumber];
        } else if (animalAI != null) {
            if (setSightRange)
                animalAI.sightRange = sightRange;
            if (setAttackRange)
                animalAI.attackRange = attackRange;
            if (setRetreat)
                animalAI.retreatProbability = retreatProbability;
            animalAI.waypointsGroupName = waypointsGroupName[randomNumber];

        } else if (sentryGun != null) {
            if (setAttackRange)
                sentryGun.attackRange = attackRange;
        }
}
function OnDrawGizmos () {
        Gizmos.color = Color(1, 1, 1, 1);
            Gizmos.DrawIcon(transform.position, gizmoName + ".psd");
}
function OnDrawGizmosSelected () {
        Gizmos.color = Color(0, 1, 1);
        Gizmos.DrawWireSphere(transform.position, spawnRange);
}
```

ÖZGEÇMİŞ

Adı Soyadı : Ali Murat SÜMEN

Doğum Yeri ve Yılı : Sivas, 1987

Medeni Hali : Bekâr

Yabancı Dili : İngilizce

E-posta : alimurat@kodgraf.com

Eğitim Durumu

Lise : Bucak Y.D.A. Lisesi, 2005

Lisans : SDÜ, Teknik Eğitim Fakültesi, Bilgisayar Sistemleri Öğretmenliği, 2009

Mesleki Deneyim

SDÜ Bilgi İşlem Dairesi Başkanlığı	2006-2007
SDÜ Uzaktan Eğitim Koordinatörlüğü	2007-2008
SDÜ Uzaktan Eğitim Meslek Yüksek Okulu	2008-2009
Microsoft Inovation Center Bilkent Cyber Park	2009-2010
Kodgraf Oyun Stüdyosu	2010-2012
Animax Animasyon Stüdyoları	2012-…….. (halen)

Yayınları

Sümen Ali Murat, 2013. 3D Studio Max İç ve Dış Mekan Modelleme. Kodlab Yayınları, 262, İstanbul.

Sümen Ali Murat, 2012. 3D Studio Max Karakter Modelleme ve Dokulandırma. Kodlab Yayınları, 282, İstanbul.