

**ANKARA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ**

YÜKSEK LİSANS TEZİ

**MAKİNE ÖĞRENMESİ İLE SÜRÜ ROBOTLARI İÇİN
SAVAŞ STRATEJİSİ MODELİ**

Ümit ULUSOY

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

**ANKARA
2019**

Her hakkı saklıdır

TEZ ONAYI

ÜMIT ULUSOY tarafından hazırlanan “**Makine Öğrenmesi İle Sürü Robotları İçin Savaş Stratejisi Modeli**” adlı tez çalışması 16/10/2019 tarihinde aşağıdaki jüri tarafından oy birliği ile Ankara Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı’nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Danışman : Doç. Dr. Mehmet Serdar GÜZEL
Ankara Üniversitesi Bilgisayar Mühendisliği Anabilim Dalı

Jüri Üyeleri :

Başkan : Dr. Öğr. Üyesi Hilal Kaya
Yıldırım Beyazıt Üniversitesi Bilgisayar Mühendisliği Anabilim Dalı

Üye : Dr. Öğr. Üyesi Özgür TANRIOVER
Ankara Üniversitesi Bilgisayar Mühendisliği Anabilim Dalı

Üye : Doç. Dr. Mehmet Serdar GÜZEL
Ankara Üniversitesi Bilgisayar Mühendisliği Anabilim Dalı

Yukarıdaki sonucu onaylarım.

Prof. Dr. Özlem YILDIRIM
Enstitü Müdürü

ETİK

Ankara Üniversitesi Fen Bilimleri Enstitüsü tez yazım kurallarına uygun olarak hazırladığım bu tezin içindeki bütün bilgilerin doğru ve tam olduğunu, bilgilerin oluşturulması aşamasında bilimsel etiğe uygun davrandığımı, yararlandığım ve kullandığım bütün kaynakları atıf yaparak belirttiğimi beyan ederim.

07/07/2019



Ümit ULUSOY

ÖZET

Yüksek Lisans Tezi

MAKİNE ÖĞRENMESİ İLE SÜRÜ ROBOTLARI İÇİN SAVAŞ STRATEJİSİ MODELİ

Ümit ULUSOY

Ankara Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Doç. Dr. Mehmet Serdar GÜZEL

Bu çalışma, iki boyutlu robot simülasyon platformunda (Robocode) gerçekleşen hem tekli hem de çok ajanlı sistemlere farklı makine öğrenmeye dayalı çözümler önermektedir. Bu dinamik ve programlanabilir platform, çeşitli savaş stratejileri kullanarak robotların çevreleriyle ve birbirleriyle etkileşime girmelerini sağlar. Makine öğrenmesi temelli modellerden biri olan Pekiştirmeli Öğrenme ise bu tarz bir problemin uygulanmasında popüler çözümlerden biridir. Bununla birlikte, özellikle sürekli devam problemler için alanlarda, alan sorunu derinleşmektedir. Temel olarak, Pekiştirmeli Öğrenmenin (PÖ) ana dezavantajlarından biri, basit görevler için sadece birkaç parametre kullanılarak işlevin tanımlanabileceği uygun bir ödül işlevi tasarlamaktır, oysa ödül işlevinin amacını daha karmaşık bir şekilde tahmin etmek zor olabilir. Son araştırmalar, Yapay Sinir Ağı(YSA) tabanlı yaklaşımların bu zorluklarla başa çıkabileceğini, iki boyutlu veya bir boyutlu verilerinden kontrol stratejileri öğrenmeyi de başardığını kanıtlamıştır. Tekli robotlar için PÖ algoritmalarının bu sorunlara ek olarak, robot sayısı arttıkça ve sistemlerin çok robotlu sistemler gibi davranması gerektiğinde, genel algoritma tasarım gereksinimleri daha da karmaşık hale gelmektedir. Bu çalışmada, bu kapsam dahilinde önerilen sistem farklı savaş senaryoları dikkate alınarak doğrulanmıştır.

İlk olarak, Pekiştirmeli Öğrenme algoritması tarafından eğitilmiş tek robot, tek bir robota karşı savaşmıştır. Ayrıca, Pekiştirmeli Öğrenme algoritması tarafından eğitilmiş robotlardan çok robotlu ve merkezi bir noktadan yönetilmeyen bir sistem daha tasarlanmıştır. Tasarlanan bu sistem ile oyunun kurallarına göre diğer sürü robotları ile savaşlar yapılmıştır. Üçüncü durumda ise robot, geleneksel bir Yapay Sinir Ağı tabanlı bir sistem kullanılarak eğitilmiştir. Daha sonra, Yığılmış Otomatik Kodlayıcılar(YOK) temel alan derinlemesine öğrenmeye dayalı bir mimari, eğitilmiş robotların rakip robotlara karşı savaşmasına izin verecek şekilde tasarlanmış ve kullanılmıştır. Pekiştirmeli Öğrenme tabanlı sistemin performansı ve denetlenen öğrenme teknikleri, bu problem için farklı senaryolar kullanılarak karşılaştırılmıştır. Dördüncü olarak ise Genetik Algoritma ile bireysel robotlar geliştirilmiştir. Tüm durumlar için alınan sonuçlar neticesinde Yapay Sinir Ağları tabanlı yaklaşımın diğer yöntemlere göre üstünlüğü ortaya net bir şekilde çıkmaktadır.

Ekim 2019, 56 Sayfa

Anahtar Kelimeler: Pekiştirmeli Öğrenme, Yığılmış Otomatik Kodlayıcılar, Yapay Sinir Ağları, Savaş Stratejileri

ABSTRACT

Master Thesis

WAR STRATEGY FOR SWARM ROBOTS BASED ON MACHINE LEARNING

Ümit ULUSOY

Ankara University
Graduate School of Natural and Applied Sciences
Department of Computer Engineering

Supervisor: Asst. Prof. Dr. Mehmet Serdar GÜZEL

This study proposes different machine learning based solutions to both single and multi-agent systems, took place on a 2-D simulation platform, namely, Robocode. This dynamic and programmable platform allows agents to interact with the environment and each other by employing a variety of battling strategy. Q-Learning is one of the leading and popular machine learning based solution to be applied such a problem. However, especially for continues spaces the control problem gets deeper. Essentially, one of the main drawbacks of Reinforcement Learning (RL) is to design an appropriate reward function that the function can be described by only employing few parameters for simple tasks, whereas for more complex estimating the goal of the reward function may be a challenging problem. Recent studies prove that Neural Network based approaches can handle these challenges and achieve to learn control strategies from 2-D or 1-D data. Besides those problems of RL algorithms for single robots, once the number of robots increases and the systems need to behave as a multi agent systems, the overall design requirements become more complex. Accordingly, the proposed system is validated by considering different battle scenarios.

First, the single robot, trained by Q-Learning algorithm battles against a single robot. Besides, a multi agent system is designed from the robots trained by the Q-Learning algorithm and they battle against multiple robots based on the rules of the game. For the third case, on the other hand, the robot is trained by employing a conventional Artificial Neural Network (ANN) based system. Next, a deep learning based architecture based on Stacked Autoencoders (SAE) is designed and employed to the system so as to allow trained robots battle against competing robots. Fourthly, Genetic Algorithm(GA) is used to evolve single agent robot. The performance of the Q-Learning based system and the supervised learning techniques are compared by employing different scenarios for this problem. Results reveal the superiority of the ANN based approach over other methods.

October 2019, 56 Pages

Keywords: Multi Agent Systems, Q learning, Robocode, Autoencoder, Neural Network , Battling Strategy.

TEŞEKKÜR

Bu çalışmanın yürütülmesi ve yüksek lisans dersleri aşamasında; değerli bilgi ve önerileriyle bana destek olan danışman hocam Doç. Dr. Mehmet Serdar GÜZEL başta olmak üzere bölümümüzdeki tüm hocalarıma, desteğini hiçbir zaman esirgemeyen eşim Tuğçe KARGI ULUSOY'a, eğitim hayatım boyunca hep yanımda olan aileme, anneanneme ve arkadaşlarıma sonsuz teşekkürlerimi ve şükranlarımı sunarım.

Ümit Ulusoy
Ankara, Ekim 2019



İÇİNDEKİLER

TEZ ONAY SAYFASI	
ETİK	i
ÖZET	ii
ABSTRACT.....	iii
KISALTMALAR DİZİNİ.....	vii
ŞEKİLLER DİZİNİ	viii
ÇİZELGELER DİZİNİ	ix
1. GİRİŞ	1
2. ROBOCODE	3
2.1 Robocode Platformu	3
2.2 Oyun Kuralları.....	6
2.3 Oyun Fiziği	7
2.4 Tanımlı Robotlar	9
3. KURAMSAL TEMELLER ve KAYNAK ÖZETLERİ.....	10
3.1 Statik Yöntemler	10
3.2 Makine Öğrenmesi.....	11
3.2.1 Gözetimli öğrenme	12
3.2.2 Gözetimsiz öğrenme.....	13
3.3 Markov Karar Süreci	13
3.4 Pekiştirmeli Öğrenme	14
3.5 Yapay Sinir Ağları.....	16
3.6 Genetik Algoritma.....	21
3.7 Derin Öğrenme.....	24
3.7.1 Konvolüsyonel sinir ağları	25
3.7.2 Otomatik kodlayıcılar.....	28
4. ÖNERİLEN YÖNTEM	30
4.1 Robocode İçin Pekiştirmeli Öğrenme	30
4.2 Yapay Sinir Ağı ile Robocode İçin Q Değerlerinin Öğrenilmesi.....	32
4.3 Yığılmış Otomatik Kodlayıcılar ile Robocode için Öğrenme.....	33
4.4 Genetik Algoritma ile Robocode için Öğrenme.....	35
4.4.1 Fitness değerinin hesaplanması	38
5. ARAŞTIRMA BULGULARI VE DENEYSEL SONUÇLAR.....	40

5.1	Uygulama Platformu	40
5.2	Senaryo-1	40
5.3	Senaryo-2	43
5.4	Senaryo-3	46
5.5	Senaryo-4	47
5.6	Senaryo-5	49
6.	SONUÇ	51
6.1	Sonuçların Karşılaştırma Tablosu.....	52
6.2	Gelecekteki Çalışmalar.....	52
7.	KAYNAKLAR.....	53
	ÖZGEÇMİŞ	56

KISALTMALAR DİZİNİ

2B	İki boyutlu
3B	Üç boyutlu
AY	Ayrık Yapı
PÖ	Pekiştirmeli Öğrenme
DSA	Derin Sinir Ağları
YSA	Yapay Sinir Ağları
YOK	Yığılmış Otomatik Kodlayıcılar
OKE	Ortalama Kare Hata
FC	Fully Connected
AR	Ağırlık Regülasyonu
KSA	Konvolüsyonel Sinir Ağı
MDP	Markov Decision Process
MKS	Markov Karar Süreci
DQN	Deep Q Network
DQA	Derin Q Ağı
GP	Genetik Programlama
GA	Genetik Algoritma

ŞEKİLLER DİZİNİ

Şekil 2.1 Robot Anatomisi [20]	3
Şekil 2.2 Robocode Ortamından bir örnek ekran görüntüsü [7]	4
Şekil 2.3 Robocode platformunun logosu ve sloganı [19]	5
Şekil 3.1 Makine Öğrenmesi Yaklaşımları	13
Şekil 3.2 3 Durum ve 2 Aksiyon için Markov Karar Verme Sürecinin İşleyişi[22]	14
Şekil 3.3 Basit bir sinir ağı yapısı	16
Şekil 3.4 Öğrenme yöntemlerime göre YSA modelleri	17
Şekil 3.5 Geri beslemeli sinir ağı modeli	18
Şekil 3.6 Sıklıkla kullanılan aktivasyon fonksiyonları	19
Şekil 3.7 Perseptron Modeli	20
Şekil 3.8 Çok Katmanlı Perseptron Modeli	21
Şekil 3.9 Genetik Algoritma Akış Şeması	22
Şekil 3.10 Konvolüsyonel Sinir Ağı Örneği [27]	26
Şekil 3.11 Max Pooling Örneği [27]	27
Şekil 3.12 Basit Otomatik Kodlayıcı Yapısı [29]	28
Şekil 4.1 Q-Öğrenme ile robot eğitiminin akış şeması	31
Şekil 4.2 YSA Mimarisi	33
Şekil 4.3 YOK Mimarisi	34
Şekil 4.4 Jenerasyonlara Göre Kazanma Oranı Değişimi	37
Şekil 5.1 1. Senaryodan Alınmış Ekran Görüntüsü	40
Şekil 5.2 AUQRobot'un İterasyonlara Göre Başarı Değişimi Grafiği	41
Şekil 5.3 1. Senaryodan Alınmış Ekran Görüntüsü (TrackerBot)	42
Şekil 5.4 1. Senaryodan için Robocode Sonuçları(TrackerBot)	42
Şekil 5.5 AUNNRobot robotunun iterasyonlara göre başarı değişim grafiği	43
Şekil 5.6 AUNNRobot ile AUQRobot'un 50 turluk savaşının sonuçları	44
Şekil 5.7 Senaryo-2'den savaş sırasında alınmış ekran görüntüsü	44
Şekil 5.8 YOK robotun iterasyonlara göre başarı değişim grafiği	45
Şekil 5.9 Senaryo-2'den savaş sırasında alınmış ekran görüntüsü (AUAERobot)	46
Şekil 5.10 Senaryo-3'den çok robotlu savaş sırasında alınmış ekran görüntüsü	47
Şekil 5.11 Senaryo-3'den savaş bitişinde alınmış sonuçları gösteren ekran görüntüsü	47
Şekil 5.12 Senaryo-4'den çok robotlu savaş sırasında alınmış ekran görüntüsü	48
Şekil 5.13 Senaryo-4'den savaş bitişinde alınmış sonuçları gösteren ekran görüntüsü	48
Şekil 5.14 Senaryo-5'den GARobot ile Tracker'ın savaşından alınmış ekran görüntüsü	49
Şekil 5.15 Senaryo-5'den savaş bitişinde alınmış sonuçları gösteren ekran görüntüsü	50

ÇİZELGELER DİZİNİ

Çizelge 2.1 Robocode sonuç hesaplama parametreleri.....	6
Çizelge 2.2 Robocode simülatörünün sağladığı metotlar ve bilgiler.....	7
Çizelge 2.3 Robocode Ön Tanımlı Robotlar.....	9
Çizelge 3.1 Hareket Stratejileri.....	10
Çizelge 3.2 Radar Stratejileri.....	11
Çizelge 3.3 Atış Stratejileri.....	11
Çizelge 3.4 Pekiştirmeli Öğrenme Unsurları.....	15
Çizelge 3.5 Genetik Algoritma Unsurları.....	23
Çizelge 3.6 KSA Katmanları.....	27
Çizelge 4.1 Robocode Q-Öğrenme algoritmasında kullanılan durum ve eylem listeleri.....	32
Çizelge 4.2 Robocode GA'da çağrılan metot listeleri.....	36
Çizelge 4.3 İfade ağaçları için kullanılan terminaller.....	38
Çizelge 5.1 Yapay Sinir Ağı Konfigürasyonu.....	43
Çizelge 6.1 Senaryo sonuçlarının karşılaştırması.....	52

1. GİRİŞ

Sürü zekası çok bileşenli hareket zekasını ve kooperatif robotik alanlarını makul bir süre içinde zorlu görevleri başarmak için robotları oluşturmak, eğitmek ve koordine etmek için birleştiren bilimsel bir alandır [1,2]. Öte yandan, çok bileşenli sistemler, bir veri ağı üzerinden bilgi alışverişi yaparak ya da paylaşarak görevleri tamamlamak için bireysel robotların koordinasyonu olarak kabul edilir. Bu ise özünde sürü zekâsı disipliniyle paralel bir şekilde benzerlik göstermektedir [3,4]. Daha önce de belirtildiği gibi birden çok bileşenli ortak amaca sahip sistemler esas olarak farklı görevleri tamamlamak adına bireysel olarak gelen çoklu etkileşimleri koordine etmek ile ilgilenmektedir. Bu sistemin temel amacı, karmaşık bir robot kullanmak yerine oldukça basitçe tasarlanmış robotları koordine etmektir[5]. Robocode platformu sağladığı altyapı, makine öğrenmesine elverişli oluşu ve sürü robotlarının yönetimini desteklemesi sebebiyle bu çalışmada kullanılmıştır. Platformun tanımı ve kuralları ilerleyen bölümlerde detaylıca anlatılmıştır.

Robotların koordinasyon kabiliyeti bu sürü zekâsı sistemleri için bireysel robotların ulaşmasının mümkün olmayacağı becerileri takım halinde kazanabilme imkânı sağlamıştır [5, 6]. Bu çalışmalar, makine öğrenmesinin bu tarz bir ortam içinde stratejiler tasarlamak ve bunları uygulamak etkili bir yol olduğunu göstermektedir.

Bu bağlamda bu çalışma önceki çalışmalardan esinlenmiştir ve savaşlara robot yetiştirmek ve hazırlamak için dört farklı makine öğrenmesi temelli yaklaşım sunmuştur. İlk yaklaşımda temel olarak bir tank savaşında hayatta kalmalarını sağlamak için tek ve takım robotlarını ayrı ayrı eğitmek adına Pekiştirmeli Öğrenme(PÖ) kullanılmıştır. Ayrık yapısına rağmen PÖ'nün bu kadar karmaşık bir oyuna adapte edilebileceği kanıtlanmıştır alınan sonuçlar ile kanıtlanmıştır. Ek olarak ise daha önce benzer bir çalışmada kullanılan sonuçları karşılaştırmak adına bir Yapay Sinir Ağı(YSA) tabanlı tasarım da uygulanmıştır [13]. Yığılmış Otomatik Kodlayıcı(YOK) tabanlı bir model, bir dizi robotu eğitmek ve önceki yöntemlerden alınan sonuçlarla karşılaştırmak için tasarlanmıştır. Son olarak, Genetik Algoritma(GA) tabanlı bireysel robot eğitilerek sonuçlar alınmış ve karşılaştırılmıştır.

Benzer çalışmalar [10,11,12,13,14,15,16,7,32] 'da görülebilir. Robocode'da evrimsel metodolojiler ve GA 2000lerin başından beri daha önce birden çok kez kullanılmıştır [7][32][10]. Yakın zaman içinde GA'nın Robocode için Harper'ın çalışmasında[10] en iyi hareket yöntemini elde etmek için PÖ ve YSA ile birlikte kullanıldığını ve seçili rakip karşısında ortalama %74.8 görmekteyiz. Bu çalışmada ateşleme için PÖ kullanılmıştır[10].

Degryse'nin 2017'deki çalışmasında ise PÖ ve YSA'nın kullanıldığını, elde edilen bireysel robotlar ile deneylerin yapıldığını görmekteyiz[11]. Ayrıca yine bu çalışmada bireysel robotlara derin çok katmanlı YSA'nın Robocode platformuna nasıl uygulanabileceği konusunda da örnekler verilmiştir.

2003 yılında Gade ve Knudsen tarafından çıkarılan yayında[12] ise yine PÖ, YSA ve GA'nın Robocode problemine uygulanarak, eğitilen bireysel robotlar ile Robocode tarafından sunulan bireysel robotlara karşı deneyler uygulanmıştır. Literatürdeki Robocode alanında makine öğrenmesi ile yapılan çalışmalar incelendiğinde makine öğrenmesinin sürü robotlarına uygulanmadığını görmekteyiz.

Bu noktada kendi çalışmamızı mevcuttaki diğer çalışmalar ile kıyasladığımızda, bu çalışmanın literatüre katkısının, merkezi olarak yönetilmeyen sürü robotlarına makine öğrenmesi yöntemlerinin nasıl uygulanabildiğini elde etmek olacaktır. Sürü robotları ve bireysel robotlar için elde edilen deney sonuçları paylaşarak, literatüre referans verisi bağlamında katkı sağlanacaktır.

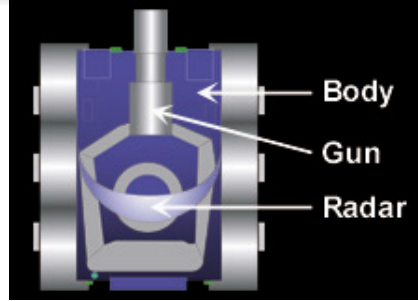
Bir sonraki bölüm Robot platformunu tanıtacaktır. Bölüm 4 ise esas olarak önerilen yöntemleri ayrı ayrı tanıtmaktadır. Tanımlanan deneyler ve bu deneylerden alınan sonuçlar deney bölümünde ayrıntılı bir şekilde değerlendirilmiştir. Son olarak, çalışma sonuç bölümünde tamamlanmıştır.

2. ROBOCODE

Bu bölümde Robocode platformu özetlenerek tanıtılmıştır ve oyunun kuralları kısaca anlatılmıştır. Verilen bilgiler Robocode resmi dokümantasyonu ve programlama dokümantasyonundan edinilmiştir[19][20].

2.1 Robocode Platformu

Robocode platformu bir oyun geliştirme platformu olup, geliştiricilerin diğer tanklara karşı savaşmak için robot savaş tankları tasarımlarına izin veriyor [7]. Savaşlar gerçek zamanlı olarak sürmekte iken oyun Şekil 2.2'de gösterildiği gibi, tek veya çoklu robotlar kullanılarak iki boyutlu bir simülasyon ortamında oynanıyor. Bu robotlar, tek robot olarak tanımlabildiği gibi bazıları takım olarak bir araya gelebilmektedir. Robotların her biri enerjilerini yüksek tutmak ve rakiplerini yok etmek için harekete, ateş etmeye ve hedeflemeye karar vermelerini sağlayacak şekilde programlanabilmektedir. Robocode'da robotlar şekil 2.1'de gösterildiği gibi üç ana kısımdan oluşmaktadır. Bunlar Gövde, Silah ve Radar'dır.



Şekil 2.1 Robot Anatomisi [20]

Robocode her bir tur içinde zaman birimleri tikler ile ölçülür ve her robot her bir tik içinde sadece bir kere hareket edebilir. Her turun (oyunun) sonunda, her katılımcının toplam puanı “ateş hasarı”, “çarpma hasarı” ve “hayatta kalma durumu” ile hesaplanır. Eğer ateş hasarı ve çarpma hasarından alınmış olan puanlar yüksek ise bir ekibin robotları hayatta kalmasa bile en yüksek puanı alabilmektedir.



Şekil 2.2 Robocode Ortamından bir örnek ekran görüntüsü [7]

Robocode platformunun esnekliği, ölçeklenebilirliği ve geliştirmeye açıklığı, geliştiricileri çok robotlu sistem problemleri için makine öğrenmeye dayalı yaklaşımlar kullanmaya teşvik etmektedir. Avantajlarına rağmen, platform getirdiği kurallar ile birlikte uygun şekilde ele alınması gereken bazı zorluklar da sunuyor. Örneğin kritik konular şu şekilde detaylandırılmıştır:

- Rakip mermiler görünmez ve çevre tam olarak gözlenemez,
- Robotlar tarafından kullanılan sensörler sınırlı sayıdadır,
- Eylem sayısı oldukça fazladır ve bu da öğrenmeyi zorlaştırır,
- Robotların ateş etme ve dönme davranışları sırasında hızı yavaşlar,
- Robotun sıcaklığı yükseldiğinde, ateş etmek işe yaramaz ve bu da kullanıcıları tüm parametreleri dikkate almaya zorlar.

Robocode, tasarım kavramları ve kaynak kodlarının paylaşıldığı araştırmacı, öğrenciler ve mühendisler de dahil olmak üzere büyük bir topluluktan büyük ilgi ve ilgi toplamaktadır. Turnuvalar ve ligler web siteleri aracılığıyla düzenlenir. Dolayısıyla, özelleştirilmiş robotların sıralaması sürekli güncellenmektedir [7]. Bu nedenle oyun stratejileri çok kritik ve farklı yaklaşımlar kullanarak sürekli gelişiyor. Robocode oyun stratejileri, bir savaşta

eylemleri ve gözlemleri kabul eden atomik elementlerin birleşimi olarak da tanımlanmaktadır. Makine öğrenmesi, tek [8,9] ve çok bileşenli sistemlerde [2,3] yaygın olarak kullanılmaktadır. Bu aynı zamanda araştırmacıları Robocode ortamındaki bu savaş sürecinde robotik ekipleri eğitmek ve hazırlamak amacıyla makine öğrenmesi veya keşifsel yöntemleri kullanmaya teşvik etmektedir. Örneğin GA'yı kullanarak çeşitli şekilde kendinden evrim eden çalışmalar yürütülmektedir [8,10]. Ayrıca liglerde daha yüksek bir sıra elde etmek adına bir strateji tahmin etmek için karar ağacı ve YSA tabanlı çözümler kullanılmıştır [11,12].

Robocode'da robotların takım halinde programlanabildiğini belirtmiştik eğer Robocode'da bir robot TeamRobot(Takım Robotu) sınıfından türetilirse takım robotlarımızı 2'li veya 5'li takımlar halinde paketleyebiliyoruz[20]. Bu ekipleri yine tekil robotlarla maça çıkarabildiğiniz gibi yine aynı şekilde farklı ekiplerle de maçlara çıkartabiliyoruz. Takım robotu oluştururken 2 farklı opsiyonumuz buluyor; Lider robot ve Droid robotlar. Droid robotların sensörleri bulunmuyor ve komutları lider robotlardan bekliyorlar. Lider robotların ise atış ünitesi(silah) bulunmuyor bu durumun oluşturduğu en büyük sorun ise eğer rakip bir şekilde lideri öldürürse droidleri mesaj ile yönetecek bir lider kalmadığı için maçı direk kaybetmiş oluyorsunuz.



Şekil 2.3 Robocode platformunun logosu ve sloganı [19]

2.2 Oyun Kuralları

Robocode’da belirlenen tur sayısının sonunda en yüksek puana sahip olan robot oyunu kazanır. Her robot belirlenen enerji miktarı ile rastgele bir noktadan rastgele bir yöne bakarak oyuna başlar. Ateşleme ünitesinin kullanımı enerji miktarını azaltmaktadır fakat bir rakibe mermi isabet ettirildiğinde bu atışı yapan robotun enerji miktarı artmaktadır. Robotlar hareket ederken diğer robotlara ve duvarlara çarptıklarında yine enerji seviyeleri azalacaktır. Turların sonucunda puan hesaplaması yapılırken ele alınan parametreler Çizge 2.1’de açıklanmıştır.

Çizelge 2.1 Robocode sonuç hesaplama parametreleri

<i>Parametre Adı</i>	<i>Açıklaması</i>
Hayatta Kalma Skoru (Survivor Score)	Bir robot öldüğünde hayatta kalan diğer robotlar 50 puan kazanır.
Son Hayatta Kalan Bonusu (Last Survivor Bonus)	Diğer tüm robotlar öldükten sonra hayatta kalan son robot ekstra 10 puan kazanır.
Mermi Hasarı (Bullet Damage)	Robotlar rakiplerine isabet ettirdikleri her mermi için 1 puan alır.
Mermi Hasarı Bonusu (Bullet Damage Bonus)	Bir robot, düşmanını mermi ile öldürdüğünde o robota verdiği toplam zararın %20’si kadar ek puan alır.
Çarpma Hasarı (Ram Damage)	Robot, rakibine yaptığı her çarpma darbesi(Ram Damage) için 2 ek puan alır.
Çarpma Hasarı Bonusu (Ram Damage Bonus)	Bir robot, düşmanını çarpma darbesi ile öldürdüğünde o robota verdiği toplam zararın %30’si kadar ek puan alır.

Robocode’da koordinat düzlemi sol alt pikselden (0, 0) olarak başlar ve Kartezyen koordinat yardımıyla konum belirlemesi yapılır. Varsayılan savaş alanı boyutu 800 x 600 pikseldir. Savaş alanında robotlar dışında herhangi bir engel bulunmamaktadır.

Robocode’da oyun sırasında simülatör yaşanan olaylar için etkinlikler(event) ve metotlar(method) tetiklemektedir. Bu etkinliklerin içinde bazı bilgiler yer almaktadır ve programcılar robotlarını sağlanan bu bilgiler ile programlamaktadır. Simülatörün tetiklediği etkinlikler Çizge 2.2’de açıklamaları ile birlikte verilmiştir.

Çizelge 2.2 Robocode simülörünün sağladığı metotlar ve bilgiler

<i>Metot Adı</i>	<i>Etkinlik</i>	<i>Alınan Bilgi</i>
BulletHitEvent	Aktif robotun rakibine mermi isabet ettirdiğini bildirir.	Rakip adı ve kalan enerjisi
BulletHitBulletEvent	Merminin başka mermiye isabet ettiğini bildirir.	Mermi Objesi
BulletMissedEvent	Atılan merminin isabetsiz olduğunu bildirir.	Mermi Objesi
HitByBulletEvent	Mermi darbesi alındığını bildirir.	Mermi Objesi
HitRobotEvent	Çarpışma durumunu bildirir.	Robot adı ve enerjisi
HitWallEvent	Duvar ile çarpışmayı bildirir.	Duvara göre olan açı, duruş
RobotDeathEvent	Rakibin ölümünü bildirir.	Robot adı
ScannedRobotEvent	Radarin bir düşmanı tespit ettiğini bildirir.	Mesafe, enerji, hız, yön ve duruş açısı
StatusEvent	Tur başlangıcını bildirir.	RobotStatus objesi
DeathEvent	Robotun ölümünü bildirir.	-
WinEvent	Robotun kazandığını bildirir.	-

2.3 Oyun Fiziği

Robotlar bölüm 2.1’de belirttiği gibi üç parçadan oluşmaktadırlar. Her bir parçanın rotası ayrı ayrı belirlenebiliyor olsa da hızları birbirinden farklı olabilmektedir. Oyundaki oynanış fiziğine dair açıklamalar üçe ayrılarak anlatılmıştır [20].

- **Robot Hareketleri**

- **Hızlanma(a):** Robotlar sıra kendilerinde olduğuna maksimum 1 birim hızlanabilirken, 2 birim yavaşlayabilmektedir.
- **Hız(v):** Robotlar artı veya eksi yönde maksimum 8 piksel/sıra hızına çıkabilmektedir.
- **Yön:** Bir tick içinde robotun gövdesi 360 derece dönebilirken diğer parçalarda maksimum dönme limiti uygulanmıştır. Radar bir tick içinde 45 derece, silah ise maksimum 20 derece artı veya eksi yönlü dönüş yapabilmektedir. Robotun maksimum dönüş miktarını ise robotun hızı aşağıdaki gibi kısıtlamaktadır.

$$(10 - 0.75 * |hız|) \text{ derece/sıra}$$

- **Mermiler**

- **Hasar:** Robocode'da atış gücü 0,1 ile 3,0 arasında değişmektedir ve atış gücü merminin hızını ve vereceği hasarı doğrudan etkilemektedir. Rakibin alacağı hasar ve merminin hız için hesaplama formülleri aşağıda verilmiştir.

$$\text{Hasar} = 4 * \text{atış gücü} + \text{argmax}(0,2 * (\text{atış gücü} - 1))$$

$$\text{Mermi Hızı} = 20 - 3 * \text{atış gücü}$$

- **Silah Sıcaklığı:** Silah sıcaklığı yüksek olduğunda atış yapmamız engellenmektedir. Varsayılan olarak her tick'de sıcaklık 0,1 olarak azalmaktadır.

$$\text{Silah Sıcaklığı} = \frac{(1 + \text{atış gücü})}{5}$$

- **Enerji:** Başlangıçta verilen enerji miktarı bir robot için her yapılan atışın gücünün 3 katı kadar azalmaktadır.
- **Çarpışmalar**
 - **Robot ile:** Çarpışmaya katılan her robot 0.6 miktarında enerji kaybeder.

- **Duvar ile:** Duvar ile yapılan çarpışmalarda alınan hasarı hız aşağıdaki gibi belirler.

$$\text{Alınan Hasar} = \text{argmax}(0 * (|\text{hız}| * 0,5 - 1))$$

2.4 Tanımlı Robotlar

Robocode platformu ile birlikte ön tanımlı bazı robotlar kullanıcı için hazır olarak gelmektedir. Bu robotlarda makine öğrenmesi metotları kullanılmamakta olup statik algoritmalar ile hareket etmektedirler. Çizge 2.2’de bu robotlardan bazıları ve onların çalışma mantıkları özet olarak listelenmiştir.

Çizelge 2.3 Robocode Ön Tanımlı Robotlar

Robot Adı	Açıklaması
CornersBot	Bu robot bir köşeye hareket ediyor, ardından ileri-geri yönde sallıyor. Eğer ölürse, bir sonraki turda ise yeni bir köşe deniyor.
CrazyBot	Bu robot mantıksız, çılgınca bir düzende hareket etmektedir.
FireBot	Bu konumunda bekler ve silahını etrafında döndürür. Vurulduğunda ise hareket eder.
InteractiveBot	Bilgisayarın faresi ve klavyesi sayesinde kullanıcıdan veri alarak hareket etmektedir.
RamFireBot	Bu robot rakiplerini takip ederek onlara çarpmaya çalışmaktadır. Çarptığında ateş eder.
SittingDuckBot	Hiçbir şey yapmadan bekleyen bu robot kalıcılık durumu göstermektedir.
SpinBot	Bu robot dairesel bir biçimde hareket eder, bir düşman tespit edildiğinde ise güçlü bir şekilde ateş eder.
TargetBot	TargetBot olduğu yerde bekler. Sadece enerji 20 düştüğünde hareket eder. Bu Robot özel olayları göstermek için eklenmiştir.
TrackerBot	Bu robot bir rakibine odaklanır , onu takip eder, yeterince yakın olduğunda ise ateş eder.

3. KURAMSAL TEMELLER ve KAYNAK ÖZETLERİ

Bu bölümde, makine öğrenmesi ile bir oyuncunun kontrol edilebildiği platform ve oyunlar üzerinde uygulanabilen güncel modeller kategorize edilmiş ve bu konular ile ilgili daha önceden yapılan çalışmalardan özetle bahsedilmiştir. Makine öğrenmesi ve derin öğrenme konseptlerine yakın olan bu bölümü atlayıp bir sonraki kısma geçebilirler.

3.1 Statik Yöntemler

Robocode’da bir robotu programlarken diğer robotların yapacakları hareketleri tahmin etmeye çalışmayan ya da stratejisinin dinamik olması yerine statik olmasını tercih eden yaklaşımların geleneksel olduğunu kabul etmekteyiz. Statik algoritmali robotlar programlanırken Hareket, Radar ve Atış stratejilerinin oyun başlamadan belirlenmesi gerekir. Robocode platformu içindeki örnek robotlarda bu stratejilerin kullanıldığını görmekteyiz. Ayrıca bu platform için en çok kullanılan statik stratejiler önceki çalışmalarda ele alınmıştır [12][10] ve bunların bir kısmı Çizge 3.1, Çizge 3.2 ve Çizge 3.3’de verilmiştir.

Çizelge 3.1 Hareket Stratejileri

<i>Strateji Adı</i>	<i>Açıklaması</i>
Hareketsiz	Rakipler için vurulması kolaydır, sabit bir noktadan ateş eden robotlar kullanmaktadır.
Çizgisel Hareket	Rakipler için tahmin edilmesi kolay olması sebebiyle çok tercih edilmez.
Dairesel Hareket	Rakipler için tahmin edilmesi kolay olmasına rağmen rastgele dairesel harekete dönüştürülerek sıkça kullanılır.
Rastgele Hareketler	Tahmin edilmesi zor olmasına karşın rakiplerle ve duvarla çarpışma ihtimalini arttırmaktadır.

Çizelge 3.2 Radar Stratejileri

<i>Strateji Adı</i>	<i>Açıklaması</i>
Diğer robotları tarama	Radarın taraması vakit aldığı için her tick’de 360 derece etraf taranarak rakiplerin koordinatları kaydedilir.
Bir hedefe kitlenme	Radar bir düşmanı tespit ettiğinde ona kitlenir ve takip edecek şekilde hareket eder. Çok katılımcılı savaşlar için mantıklı değildir çünkü öldürmesi kolay olan robotları göz ardı edebilirsiniz.

Çizelge 3.3 Atış Stratejileri

<i>Strateji Adı</i>	<i>Açıklaması</i>
Sabit Atış	Robot hareket etmiyorken rakip robotun şuan ki konumuna yapılan atıştır. Rakibin hareket etmesi durumunda atış isabetsiz olacaktır.
Lineer Atış	Rakip robotun lineer bir şekilde hareket ettiği düşünülerek yapılan atıştır. Robot hareketleri kısa bir süre içinde tahmin edilebilir hale geldiğinden kısa vadede mantıklı olsa da uzun vadede efektif olmayabilir.
Dairesel Atış	Rakibin dairesel olarak sabit bir hız ile hareket ettiği düşünülerek yapılan atıştır. Bu atışlardan rakipler hızlı manevralar ve durma/hareket etme hamleleri ile kurtulabilmektedir.
Salınımlı Atış	Rakiplerin bir sağa bir sola gittiği düşünülerek yapılan atıştır. Çok kullanılmamaktadır.
Hareket modelli atış	Rakibin hareket etme modelini öğrendikten sonra bir sonraki hareketini tahmin etmeye dayalı atıştır. Öğrenmek zaman aldığından ötürü maliyetlidir.

3.2 Makine Öğrenmesi

Yapay Zekâ, bir bilgisayarın yada bilgisayar kontrolündeki bir robotun çeşitli faaliyetleri zeki canlılara benzer şekilde yerine getirme kabiliyetidir [21]. Tarihi bilgisayar kadar eskiye uzanan yapay zekanın çıkış noktasını “Makineler düşünebilir mi?” yaklaşımı 1940larda Alan Turing (1950) ortaya çıkarmıştır. Alan Turing’in II. Dünya Savaşı sırasında Nazilerin Enigma makinesinin şifreleme algoritmasını kırmayı başarması ve bunu yaparken elektromekanik

makine kullanması bilgisayar ve yapay zekâ bilimi kavramlarının da doğurmuştur. Daha sonra ise Turing makineler insanlar gibi davranabilir mi sorusuna cevap aradığı için makinelerin ve insanların birbirini görmeden ekran ve butonlar yardımı ile iletişime geçebildiği bir test geliştirir. Bu test sırasında yaşanan diyaloglar sonucunda hangi katılımcının insan hangisinin makine olduğunu insanlardan saptamaları istenir. Bu teste Turing Testi adı verilmiştir.

Makine öğrenmesi ise bilgisayar biliminin 1959 yılında tanımlanmış bir görevi yerine getirmek ya da belirli bir veri kümesi ile alakalı bilgi edinilebilmesini sağlamak için sayısal tahminleri ve dinamik algoritmaları oluşturmaya çalışan bir bilim dalıdır. Makine Öğrenmesi, Yapay Zeka'nın bir alt dalı olarak kabul edilmektedir. Makine öğrenmesi modellerinin eğitilmesi için çok büyük miktarlarda veriye ihtiyaç duyulmaktadır. Bu noktada ise deney verilerinin sınırlı sayıda olması ve geleceğe dair kesinlik veremiyor olması sebebiyle makine öğrenmesi modellerinden alınan sonuçlar gelecekteki durumlar için mutlak doğruluk güvencesi verememektedir.

Günümüzde fotoğraf ve videolardaki objeleri tanımlamak, konuşmayı metne çevirebilmek, siber saldırıları algılamak, dolandırıcılık tespiti yapmak, hastalık tespiti yapmak, son kullanıcıya ticaret sitelerinde ürün önermek gibi örneklerini uzatabileceğimiz birçok alanda makine öğrenmesi yöntemleri etkin olarak kullanılmaya devam edilmektedir. Makine öğrenmesi algoritmaları Gözetimli Öğrenme (Supervised Learning) ve Gözetimsiz Öğrenme(Unsupervised Learning) olarak iki ana gruba ayrılmaktadır.

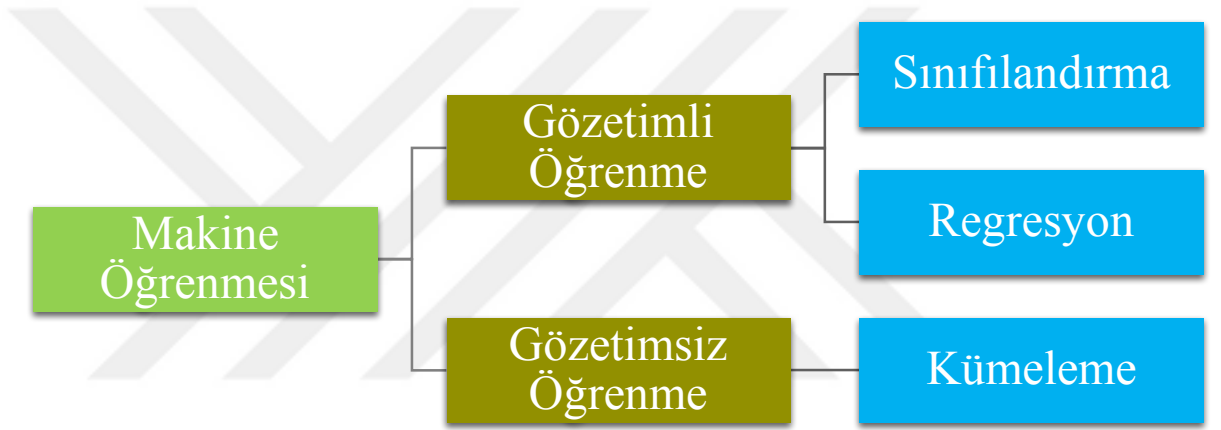
3.2.1 Gözetimli öğrenme

Girdi verileri ile çıktı verileri arasında bir korelasyon formülü ile eşleştirme yapan makine öğrenmesi metodolojisidir. Gözetimli Öğrenmede asıl amaç yukarıda bahsedilen eşleştirme formülünün bulunmasıdır. Bu yöntemde Gözetimli denmesinin sebebi eğitim verileri içinde gelen girdilere karşılık gelen çıktıların etiketli bir şekilde olmasıdır. Yani öğrenme işlemi yardımcı bir şekilde devam eder ve bu yardımcı öğrenme işlemi, sistem beklenen hata payının

altında tahminler veya sonuçlar üretmeye başladığında tamamlanır. Gözetimli Öğrenme ise kendi altında Sınıflandırma ve Regresyon olarak ikiye ayrılmaktadır.

3.2.2 Gözetimsiz öğrenme

Gözetimli Öğrenme'nin aksine, Gözetimsiz Öğrenme'de girdi verileri için eşleştirilmiş bir çıktı verileri bulunmamaktadır. Gözetimsiz öğrenme, örnek veriler arasındaki benzerliklerden hareket ederek sınıflandırma ve gruplama yapmaya çalışmaktadır. Ayrıca Şekil 3.2'de Gözetimli ve Gözetimsiz Öğrenme için ana ast dallar verilmiştir.



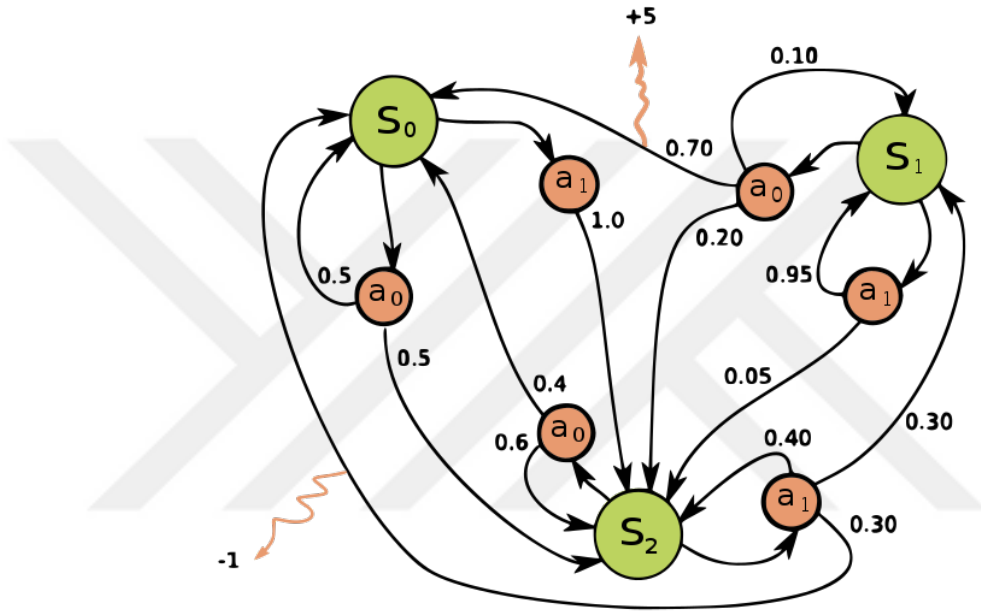
Şekil 3.1 Makine Öğrenmesi Yaklaşımları

3.3 Markov Karar Süreci

Geçmiş 1950'ye dayanan Markov Karar Süreci(MKC) ayrık zamanlı bir stokastik kontrol sürecidir[22]. Karar verme sürecini modellemek için sonuçların kısmen ve rastgele bir karar vericinin kontrolü altında olduğunu sağlayan matematiksel bir yapı sağlar. MKC'ler, dinamik programlama ve PÖ yoluyla çözülen optimizasyon problemlerini incelemek için oldukça kullanışlıdır. Basitçe tanımlayacak olursak düğümler(nodes) arasındaki geçiş ve değişimleri(edge) gösteren graf(graph) teorisinin bir modelidir.

Markov Karar Sürecinde:

- S durum kümesini,
- A aksiyon kümesini,
- $P_a(s, s^1)$ s durumundaki a aksiyonu için s^1 durumuna gidebilme ihtimalini,
- $R_a(s, s^1)$ o andaki ödülü,
- $\gamma, 0 \leq \gamma \leq 1$ indirim(discount) faktörünü göstermektedir.
- π oyunda kullanılan poliçeyi göstermektedir.



Şekil 3.2 3 Durum ve 2 Aksiyon için Markov Karar Verme Sürecinin İşleyişi[22]

3.4 Pekiştirmeli Öğrenme

PÖ, Markov Karar Süreci'ni kullanan bir Makine Öğrenmesi modelidir. Bu modelde yapay zekâ için neden sonuç ilişkisi ile önceden bilgilendirmeyi temel almaktadır. Bu neden sonuç ilişkisi yapay zekaya direk verilmek yerine modelin gözlem yaparak maksimum ödülü sağlayacak ilişkileri öğrenmesi beklenmektedir. Q öğrenme en popüler PÖ algoritması olup

q-değerlerini durum ve eylem çiftlerinden öğrenmeye çalışmaktadır. Q Öğrenme'ye ek olarak PÖ'de kullanılan diğer popüler algoritmalar Monte Carlo, SARSA ve Derin Q Ağlarıdır. PÖ'de bulunan dört unsur Çizge 3.4'de açıklanmıştır.

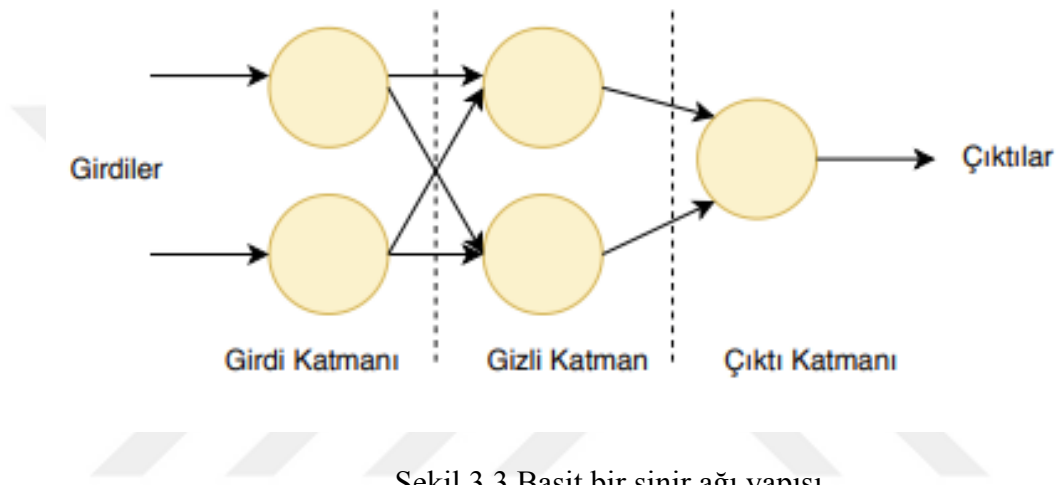
Çizelge 3.4 Pekiştirmeli Öğrenme Unsurları

Parametre Adı	Açıklaması
Politika (Policy)	Bir ajanın içinde bulunduğu durumda alabileceği aksiyonu belirlemektedir. Tüm durum-aksiyon çiftlerini içerebileceği gibi basit bir aksiyon olarak da tanımlanabilir. Çok sayıda politika olmasına karşın sıkça kullanılanları Politika tabanlı(Policy based), Değer tabanlı(Value based) ve Model tabanlı(Model based) olanlarıdır. Robocode için en iyi yaklaşım, modeli öğrenmek zorunda kalmadan ajanın gelişmesine izin verdiği için Değer tabanlı PÖ'dür.
Ödül (Reward)	Gerçekleştirilen aksiyon için çevreden alınan ödüldür. PÖ'nün amacı uzun vadeli ödülleri maksimize etmeye çalışmaktır.
Değer (Reward Function)	Ajanın içinde bulunduğu s ve onu takip eden $s+1$ durumlarında elde edebileceği ödüllerin toplamıdır. Ödül kısa vadeli getiriyi değer ise uzun vadeli getiriyi ifade etmektedir.
Çevre Modeli (Model)	Ajan için çevrenin simülasyonunu oluşturarak tahmin yeteneğini arttırmasını sağlamaktadır. Bu unsur PÖ için opsiyoneldir.

Günümüzde robot kontrolü, asansör yönetimi, telekomünikasyon, ve oyun alanlarında sıkça kullanılan PÖ ile DeepMind'ın bilgisayarının Satranç Şampiyonu Kasparov'u 1997'de yenmesi ve sonraki jenerasyon olan AlphaGo'nun dünya Go şampiyonu Ke Jie'yi yenmesi literatürdeki kilometre taşlarından olmuştur. Ayrıca Atari oyunlarında PÖ sıkça kullanılmaktadır. Robocode için ise PÖ'nün Q-öğrenme modeli geçmişteki çalışmalarda görülmüştür [11][12]. Q-öğrenmeye ek olarak bu probleme Q-öğrenmeden türetilmiş DQA'nın da uygulanması literatürdeki çalışmalarda hedeflenmiştir. Bu sistemin en büyük handikabı ise ajanın içinde bulunabileceği tüm durumları ele alıp incelemesi için gerektirdiği bellek ve işlem gücüdür.

3.5 Yapay Sinir Ağları

Biyolojik bir sinir hücresi 4 çekirdek, gövde, akson ve dentritlerden oluşmaktadır. Dentritler sinyalleri çekirdeğe, çekirdek ise genel sinyalleri aksonlara iletmektedir. YSA biyolojideki sinir hücresini modelleyeme çalışarak geliştirilmiştir. Biyolojik sinir hücrelerinde olduğu gibi YSA'da da girdiler, bağlantılar ve çıktılar bulunmaktadır. YSA modelinin geliştirilmesindeki hedef insan beyninin öğrenebilme yeteneğinin bilgisayarlara kazandırılmak istenmesidir.



Şekil 3.3 Basit bir sinir ağı yapısı

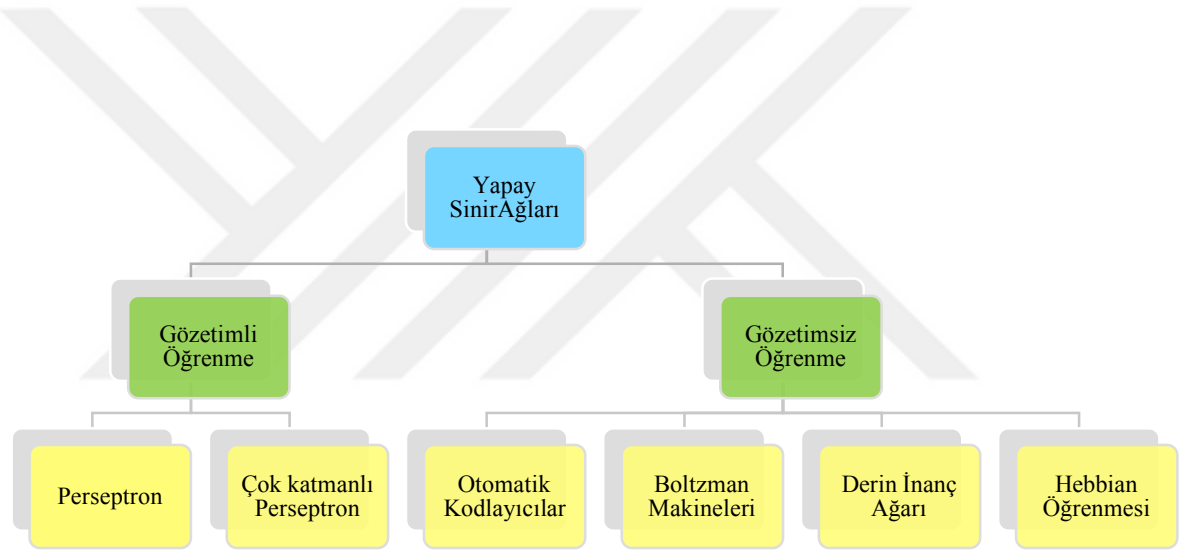
YSA modeli üzerine yapılan ilk incelemeler 1943 yılında W.S. McCulloch ve W.A. Pitts tarafından yapılmıştır [24]. Beynimizin hesaplama yeteneğinden yola çıkan W.S. McCulloch ve W.A. Pitts elektrik devreleri ile YSA'yı modellemişlerdir. Modelledikleri sinir ağında sinirsel olaylar ve aralarındaki ilişkiler ele alınırken önerme mantığı kullanılmıştır. Bu yaklaşımın kullanılma sebebi ise “Ya hep ya da hiç” karakteridir. McCulloch ve Pitts'den sonra ise YSA'lar John Hopfield, James McClelland ve David Rumelhart gibi bilim insanları tarafından geliştirilmiştir. Diğer hesaplama yöntemlerinin aksine YSA'lar probleme ve veri setine uyum sağlayabilmektedirler. Hata toleranslarının olması sebebiyle de esneklik de sunmaktadırlar. YSA kümelenme ve sınıflandırma yapmamıza yardımcı olmaktadır. Etiketlenmemiş veriler için örnek girdiler arasındaki benzerliklere göre gruplandırmayı sağladıkları gibi etiketli verilerin olduğu bir veri seti için ise sınıflandırma yaparlar. YSA'lar

katman yapılarına göre, öğrenme yöntemlerine göre ve bağlantı yapılarına göre sınıflandırılmaktadırlar.

Öğrenme yöntemlerine göre YSA'lar ikiye ayrılmaktadır;

- Gözetimli öğrenme: (Bkz: Bölüm 3.2.1) girdi ve çıktılar eğitim sürecinde verilmektedir yani bir korelasyon bilgisi sisteme verilmektedir.
- Gözetimsiz öğrenme: (Bkz: Bölüm 3.2.2) ise verilen verinin özelliklerinin ağ tarafından çıkarılması ve bu verilerin gruplandırılması beklenmektedir.

Şekil 3.5'te öğrenme modellerine göre YSA yaklaşımları hiyerarşik bir biçimde gösterilmiştir.



Şekil 3.4 Öğrenme yöntemlerine göre YSA modelleri

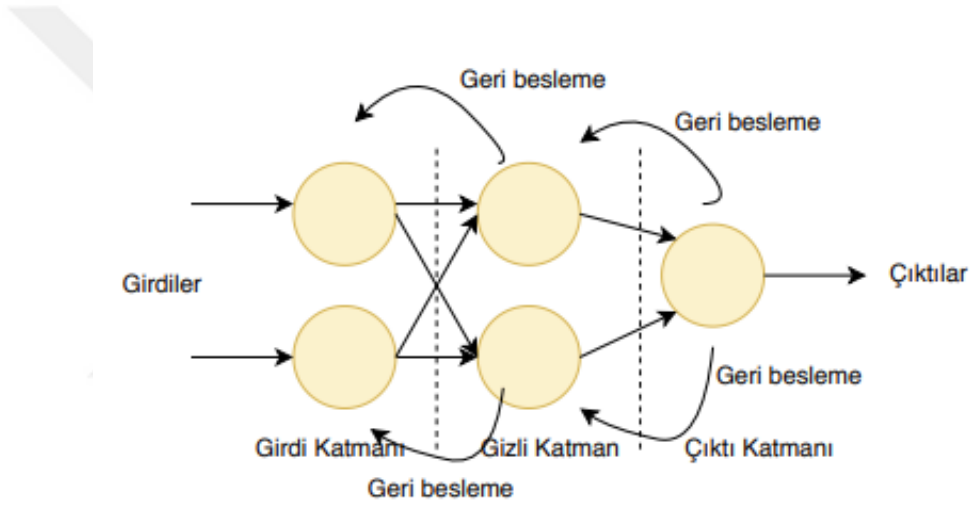
YSA'lar bağlantı yapılarına göre ise ileri beslemeli(feed-forward) ve geri beslemeli(back-propagate) olmak üzere ikiye ayrılmaktadırlar. İleri beslemeli sinir ağlarında akış girdi katmanından başlayarak varsa gizli katmandaki nöronlardan geçerek çıktı katmanına doğru ilerlemektedir. Herhangi bir geriye dönük güncelleme mekanizması bulunmamaktadır. Bu

yüzden nöron ağırlıkları ortaya çıkan hata miktarından sonra geriye doğru güncellenememektedir. Bu eksiği ise geri beslemeli ağılar çözmektedir.

İleri beslemeli(feed-forward) sinir ağıları;

- Perseptron
- Çok katmanlı Perseptron
- DİA
- Otomatik Kodlayıcılar

İleri beslemeli DSA'da çoğunlukla Kaybolan Gradyan Sorunu(Vanishing Gradient Problem) için Düzeltilmiş Doğrusal Birim(ReLU) aktivasyon fonksiyonu kullanılmaktadır. Aktivasyon fonksiyonlarının detaylarına bu tez çalışmasında kullanıldığı için değineceğiz.



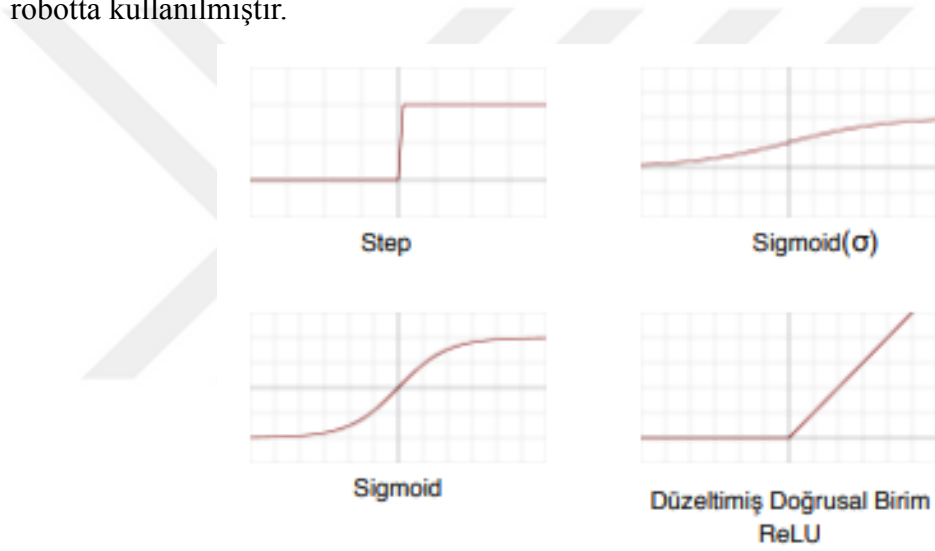
Şekil 3.5 Geri beslemeli sinir ağı modeli

Geri beslemeli sinir ağlarında ise ileri besleme yapıp girdiler çıktı katmanına kadar geldikten sonra çıktı hesaplanır. Ortaya çıkan hata miktarından hareketle geriye doğru nöron ağırlıklarını günceller ve bu güncellemeyi ise delta algoritması ve gradyan azalması algoritmasıyla yapmaktadır.

Geri beslemeli(back-propagate) sinir ağıları;

- Perseptron
- Çok katmanlı Perseptron

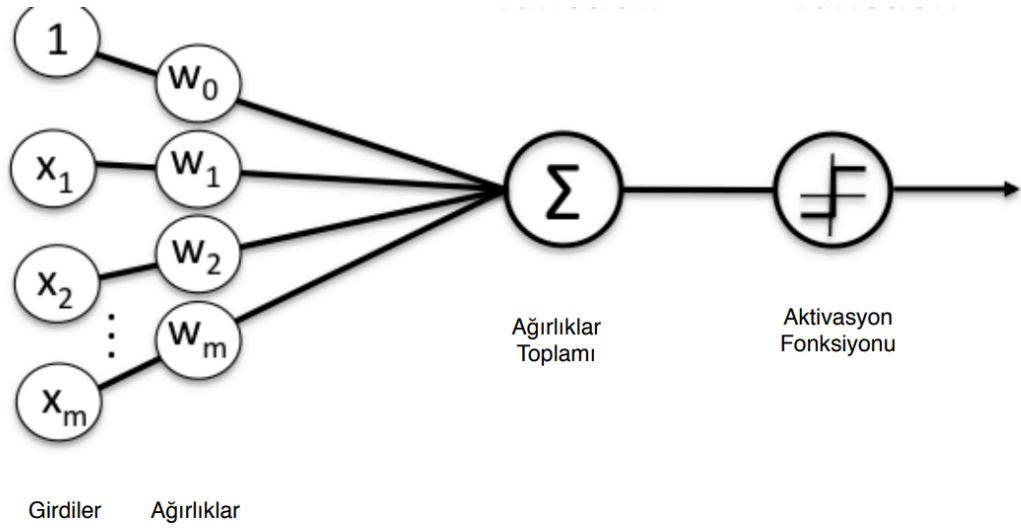
Aktivasyon Fonksiyonları: Aktivasyon fonksiyonları çıktı fonksiyonuna YSA yakınlaştırma amacıyla nöronlar tarafından kullanılmaktadır. Aktivasyon fonksiyonunun amacı nöronların çıkış verilerindeki lineerliği engellemektedir. Ortaya çıkan sonucu oluşturan ana birim olması sebebiyle aktivasyon fonksiyonu seçimi sinir ağları için çok önemlidir. 4 adet ana aktivasyon fonksiyonu bulunmaktadır. Bunlar Sigmoid(σ), Düzeltilmiş Doğrusal Birim(ReLU), Tanh ve Step fonksiyonlarıdır. Şekil 3.7’de aktivasyon fonksiyonlarının görülmektedir. Bu fonksiyonlardan en basiti Step fonksiyonudur ve genel olarak bir eşik(threshold) değeri geçen değerler için 1 diğerleri 0 olarak çıktı sonucu verir. Sigmoid fonksiyonu ise Step fonksiyonun biraz daha gelişmiş modelidir ve YSA tabanlı robotta kullanılmıştır.



Şekil 3.6 Sıklıkla kullanılan aktivasyon fonksiyonları

Perseptron: Perseptron kavramını ilk kez 1958 yılında Frank Rosenbatt çıkarmıştır.

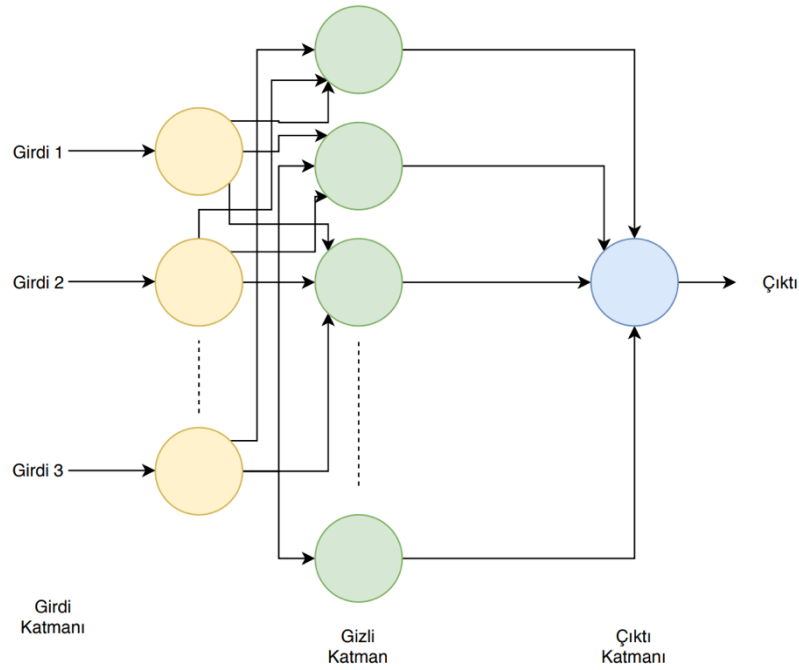
YSA’da tek katmanlı en basit işlem birimidir. Gizli katman bulunmaz sadece girdi ve çıktı katmanlarından oluşmaktadır. Çıktı üniteleri bütün girdi nöronlarına tamamen bağlıdır(FC) ve her bağlantı için bir ağırlık bilgisi tutulmaktadır.



Şekil 3.7 Perseptron Modeli

Perseptron lineer bir fonksiyonla bölünebilen ve çözülebilen problemlerde kullanılmaktadır. AND, OR ve NOT bu problemlerden birisidir. Lineer olarak bölemediğiniz problemleri perseptronlar çözemez. Örneğin XOR problemini çözmek için perseptronları geliştirmemiz gerekmektedir. Bu geliştirme ise Çok katmanlı Perseptronlar ile yapılabilmektedir.

Çok Katmanlı Perseptron: YSA için 1950 yıllarında atılan adımların bir süre devam etmesine karşın 1969 Yılında Minsky ve Seymour Papert'in An Introduction to Computational Geometry [25] isimli çalışmasında XOR probleminin tek katmanlı perseptron ile çözülemeyeceği kanıtlanmıştır. Fakat 1980lere gelindiğinde yapılan çalışmalar sayesinde XOR probleminin çözüleceği kanıtlanmış [26] ve YSA'lar yine popüler olmuştur. YSA için en yaygın olarak kullanılan modeli çok katmanlı(multilayered) perseptronlardır.



Şekil 3.8 Çok Katmanlı Perseptron Modeli

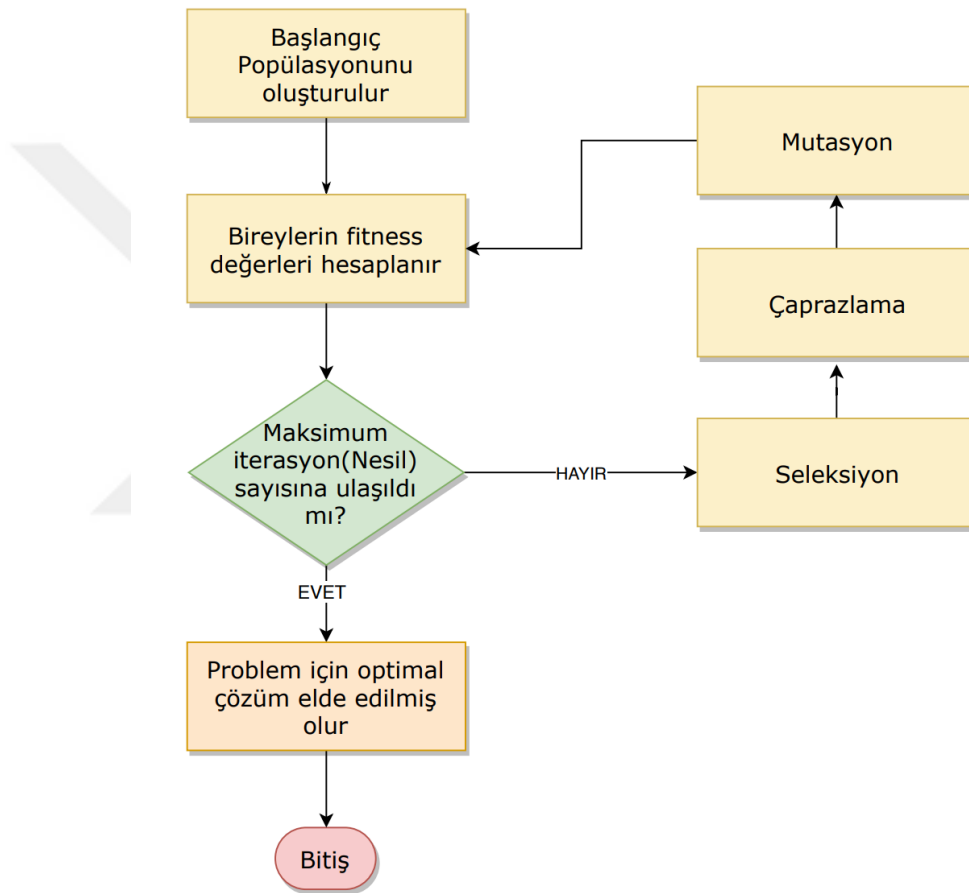
Çok katmanlı ağlar temelde şekil 3.9'deki gibi 3 bölümden meydana gelmektedir:

- Girdi(Input Layer) katmanı: Bilgi ile ilgili işlem yapılmaz, gelen veriler sadece gizli katmana iletilir.
- Gizli(Hidden Layer) Kısım: Girdi katmanındaki her bir elemanı, gizli katmandaki nöronlara tamamen bağlı durumdadır(FC). Bir tane gizli katman ile birçok problemin çözülebildiği gibi birden fazla da gizli katman kullanılabilir. Bu kısımda bir önceki katmandan gelen veriler işlenmektedir.
- Çıktı (Output Layer) Kısım: Gizli katmandan gelen verileri alıp işledikten sonra çıktıyı üretir.

3.6 Genetik Algoritma

GA doğada gözlemlenen evrimsel sürece benzer bir şekilde çalışan arama ve eniyileme yöntemidir. Karmaşık çok boyutlu arama uzayında en iyinin hayatta kalması ilkesine göre bütünsel en iyi çözümü arar. GA'ların temel ilkeleri ilk kez Michigan Üniversitesi'nde John

Holland tarafından ortaya atılmıştır. Holland 1975 yılında yaptığı çalışmaları “Adaptation in Natural and Artificial Systems” adlı kitabında bir araya getirmiştir. İlk olarak Holland evrim yasalarını GA içinde eniyileme problemleri için kullanmıştır. GA problemlere tek bir çözüm üretmek yerine farklı çözümlerden oluşan bir çözüm kümesi üretir. Böylelikle, arama uzayında aynı anda birçok nokta değerlendirilmekte ve sonuçta bütünsel çözüme ulaşma olasılığı yükselmektedir. Çözüm kümesindeki çözümler birbirinden tamamen bağımsızdır. Her biri çok boyutlu uzay üzerinde bir vektördür[30].



Şekil 3.9 Genetik Algoritma Akış Şeması

GA problemlerin çözümüne en iyi olan hayatta kalır modeliyle yaklaşmaktadır ve evrimsel süreçleri makine ortamında simüle etmeyi denemektedir. Problemlerin çözümlerinin yer

aldığı kümeye nüfus denmektedir. Nüfus bireylerden oluşmaktadır bu bireyler vektörler ya da kromozomlar olabilmektedir. GA'daki unsurlar Çizge 3.5'te listelenmiştir. Ele alınan problem için nüfusun içindeki unsurların çözüm olup olmadığını hesaplayan bir fonksiyon bulunmaktadır. Bu fonksiyona uygunluk fonksiyonu adı verilmektedir. Bu fonksiyondan çıkan değere göre yüksek puan alan bireyler nüfus içindeki diğer bir bireyle birlikte çaprazlama ile yeni bir bireyi meydana getirir. Bu yeni bireye çocuk adı verilmektedir. Çocuk gen çaprazlaması sebebiyle hem babanın hem de annenin genini taşır. Bu eş seçilimi ve çaprazlama süreci içinde uygunluk fonksiyonuna göre düşük puanlı bireyler yavaş yavaş popülasyonun dışında kalmaya başlar. Yani iyi olan genlerin seçilimi söz konusudur. Bu sayede ilerleyen nesiller için iyi genlerin aktarılması sağlanır.

Çizelge 3.5 Genetik Algoritma Unsurları

Unsur Adı	Açıklaması
<i>Gen</i>	Bireylerin özelliklerini belirleyen ve nesilden nesile aktarılan birimlerdir.
Birey (Kromozom)	Gen dizisidir, birden fazla sayıda kromozomun bir araya gelmesi ile oluşur.
Nüfus (Popülasyon)	Kromozomların bir araya gelmesiyle nüfus oluşur. GA için problemin olası çözümlerini içermektedir.
Uygunluk Değeri (Fitness Value)	Bireylerin hayatta kalmaya ne kadar elverişli olduklarını gösteren, uygunluk fonksiyonu ile hesaplanan değerdir.
Uygunluk Fonksiyonu (Fitness Value)	Bireylerin gen dizilimlerinin ortam şartlarına göre seçilebilme puanını hesaplamaktadır.

GA'da uygunluk fonksiyonuna ek olarak çeşitliliği sağlamak için mutasyon ve çaprazlama işlemlerine ihtiyaç duyulmaktadır. Bu işlemlere genetik operatörler denmektedir. Bu operatörler Çaprazlama(Crossover) ve Mutasyondur(Mutation).

- **Çaprazlama Operatörü:** Yeni bireyleri uygunluk değeri yüksek bireylerden oluşturmak için kullanılmaktadır. Çaprazlama operatörü nüfus içinde seçilen iki bireyin kromozomlarının rastgele bir noktası seçilerek yeni bireyde birleştirilmesini ifade etmektedir. Çaprazlamanın nasıl ilerleyeceğine dair birden çok yaklaşım bulunmaktadır. Bunlardan en çok bilinen ve kullanılanı rulet tekerleği seçimi kabul edilmektedir.

- Mutasyon Operatör: Jenerasyonlar arasında çeşitliliğin sağlanması ve probleme çözüm olabilecek ideal değerleri bulabilme ihtimalini yükseltmek için bu operatör kullanılmaktadır. Genelde mutasyon oranına göre kromozom içinden rastgele seçilen bir gen değerinin değişmesiyle olabildiği gibi iki genin dizilişinin kendi arasında değişmesiyle de meydana gelebilir.

3.7 Derin Öğrenme

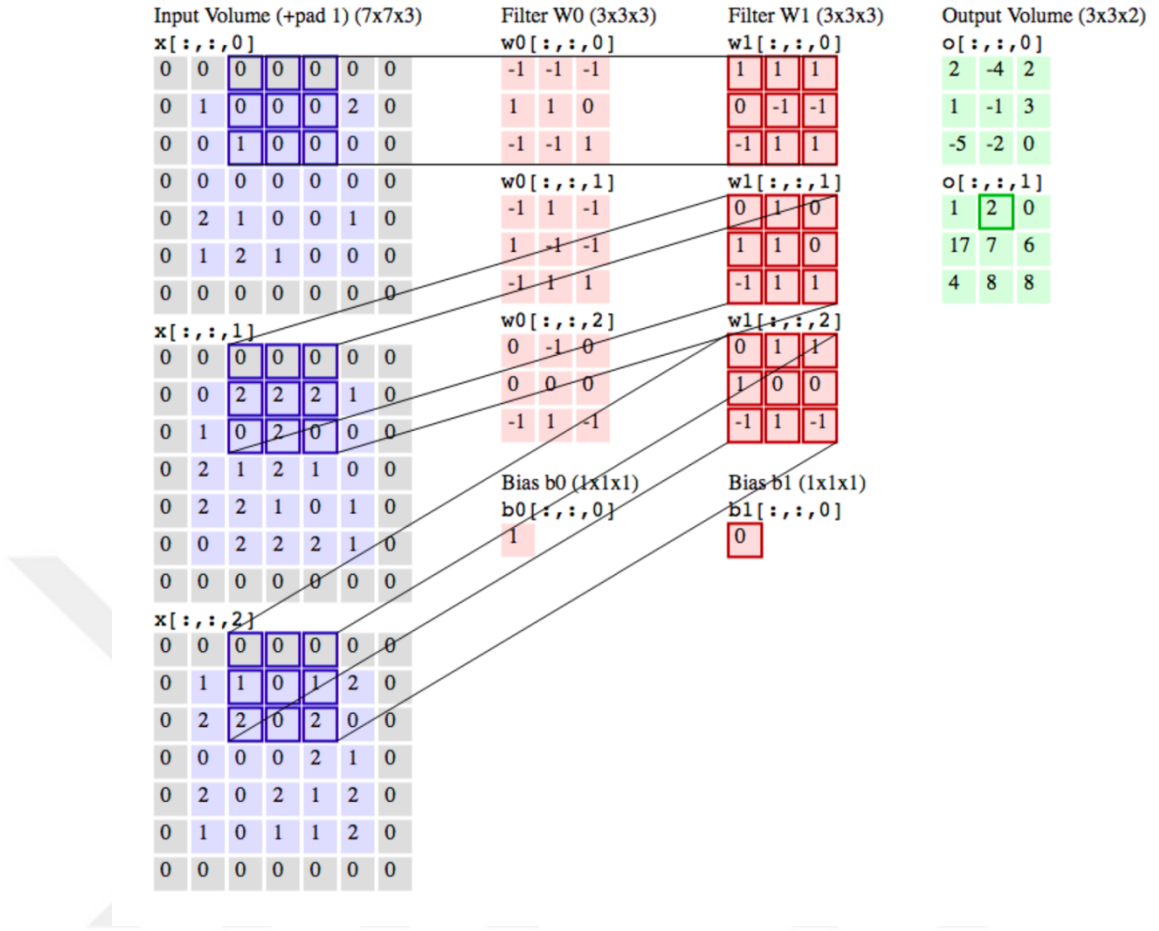
YSA'dan esinlenilen Derin Öğrenme günümüzde çok yaygın bir şekilde kullanılan ve problemlere efektif çözümler sunan yapay zekâ biliminin alt dalı olarak kabul edilmektedir. Derin öğrenme, yakın geçmiş içinde oldukça popüler olmuş ve derin öğrenme metodolojileri ile sayısız uygulama yapılmıştır. Yüksek boyutlu verileri işlemekte paralel işlem gücü olan ekran kartları sayesinde çok başarılı olan derin öğrenme literatürdeki işlem gücü darboğazının aşılmasını sağlamıştır. Bu sebeplerden ötürü bilim ve sanayide birçok uygulamada ile karşımıza çıkmıştır. Derin Öğrenme bilgisayarlı görmede, büyük veri uygulamalarında, nesne bulma, nesne takip etme ve doğal dil işleme gibi birçok uygulamada popüler olarak kullanılmaktadır.

Derin öğrenme, çoklu soyutlama(abstraction) seviyelerine sahip verilerin gösterimini öğrenmek için çoklu işlem katmanlarından oluşan hesaplama modellerine izin vermektedir. Bu yöntemler konuşma tanıma, görsel nesne tanıma, nesne algılama ve ilaç keşfi ve genomik gibi diğer birçok alanda son teknolojiyi önemli ölçüde geliştirmiştir. Derin öğrenme, bir makinen için her yeni gelen katmandaki veri gösterimini önceki katmandaki gösterimden yola çıkarak hesaplamaktadır. Bu hesaplama için kullanılan ve katmanlardaki iç parametrelerini nasıl değiştirmesi gerektiğini bulmak için geri yayılma(back-propagate) algoritmasını kullanmaktadır. Derin evrimsel ağlar, görüntü, video, konuşma ve ses işlemede cıgır açarken, tekrarlayan ağlar metin ve konuşma gibi ardışık veriler üzerinde ışı tutmaktadır [28].

Derin Öğrenme metodolojilerinden olan ve sürü robotları yönetme problemi için literatürde ve bu çalışmada kullanılmış olan Konvolüsyonel Sinir Ağları'ndan ve Otomatik Kodlayıcılar'dan bu kısımda bahsedilmiştir.

3.7.1 Konvolüsyonel sinir ağları

İleri beslemeli sinir ağları içinde eğitim maliyetlerini düşürmek için boyut azaltma uygulanmaktadır. KSA'larda bu boyut azaltma işlemi evrişimli katmanlarda uygulanmaktadır. Evrişimli katmanlarının amacı, ağ tarafından inceleme yapılmadan önce desenleri(pattern) çıkartmaktır. Bu desenler kenarlar, köşeler, dokular gibi dikkat çeken ve bir nesneyi tanımlamak için kullanılabilen ufak görüntü parçaları olarak tanımlanabilmektedir. Veriler bu katmanlara matrisler aracılığı ile kodlayarak gönderilmektedir. Evrişimli katmanlarda padding, stride ve pooling işlemleri yapılmaktadır. Çizge 3.6'te bu katmanların açıklaması yapılmıştır.



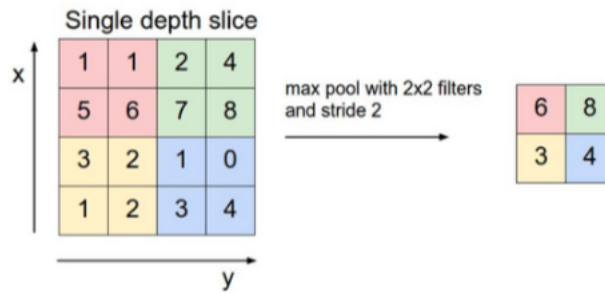
Şekil 3.10 Konvolüsyonel Sinir Ağı Örneği [27]

Şekil 3.11’da gösterilen Evrişimli bir katman örneğinde girdi üç kanallı (kırmızı, mavi ve yeşil) 5 x 5 piksellik bir görüntüdür. 2 birimlik piksel ekleme(padding) eklenmesi ile bu büyüklük 7x7x3'lük bir matrise dönüşmektedir ve sonra bu matris evrişimi kullanılarak özellik çıktısı (feature output) olan yeşile filtrelenmiş olur. Şekilde filtreyi $w0[:, :, i]$, Bias'i $b0$ ve çıktıyı $o[:, :, 0]$ göstermektedir. Adım(Stride) 2,2'dir, bu mavi matrisin her işlemde yatay ve dikey olarak 2 birim ilerlediği anlamına gelmektedir [27].

Çizelge 3.6 KSA Katmanları

Katman Adı	Açıklaması
Evrişim (Convolution)	Input bilgisinin yüksek frekanslı bölgeleri için bir gruplama yapmayı amaçlamaktadır. Öz nitelikleri bulmak için dikey ve yatay olmak üzere iki ana filtre bulunmaktadır. Genellikle KSA’larda ilk katman olarak konumlandırılmaktadır.
Piksel Ekleme (Padding)	İşlem matrisine ek hücreler eklenerek ana matris üzerinde işleme alınmayan piksel kalmamasını sağlamaktadır. Bu piksel ekleme işlemine padding adı verilmektedir. Giriş matrisi $n \times n$, filtre matrisi $f \times f$ olduğunda çıktı ile girdinin aynı boyutta olması talep edilirse $(n+2p-f+1) \times (n+2p-f+1)$ formülü kullanılır. Buradaki piksel ekleme parametresi p , $p=(f-1)/2$ formülü ile hesaplanmaktadır.
Adım (Stride)	Adım miktarı(Stride) her adımda girdi matrisi üzerinde ne kadar kaydırması gerektiği belirtmektedir. Örneğin, 5×5 girdi matrisi için 3×3 ’lük bir filtre uygulanıyorken adım 1 ve padding 2 ise çıktı matrisi 3×3 boyutlarında olacaktır.
Ortaklama (Pooling)	Pooling katmanında genellikle maksimum ortaklama(max-pooling) yöntemi kullanılmaktadır. Bu katmanda ağ için parametre öğrenilmemektedir. Gelen matris için kanal sayısını aynı tutularak yükseklik ve genişlik bilgisi azaltılmaktadır.

Şekil 3.12’de maksimum ortaklama işlemi gösterilmektedir.



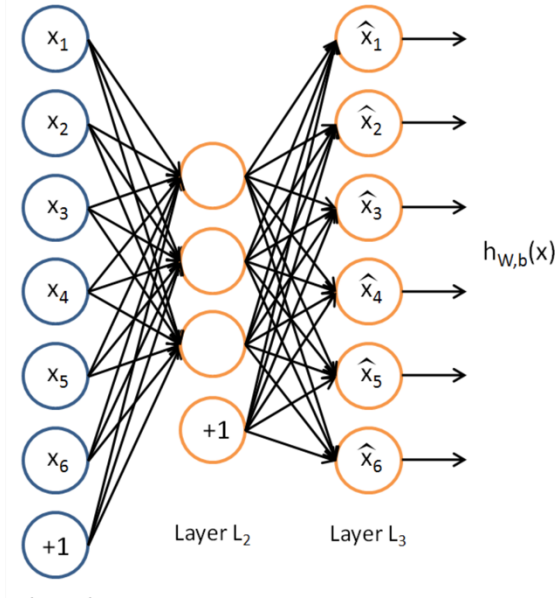
Şekil 3.11 Max Pooling Örneği [27]

Lokal bağlantılı katmanlardan meydana gelen Konvolüsyonel Sinir Ağları ağırlıklarını eşit bir şekilde paylaşmaya çalışmaktadır. Tamamen bağlı olmaması sayesinde, KSA daha çok görüntü tanıma problemlerinde tercih edilmektedir. Biz Robocode’da matematiksel konum, enerji, açı verilerinden yola çıkarak hareket ettiğimiz için oyun sırasındaki anlık görüntü verilerini kullanmamaktayız. Bunun yerine çok katmanlı perseptron yapısının önüne otomatik kodlayıcı yerleştirerek DSA yapısı tecrübe edilmiştir.

3.7.2 Otomatik kodlayıcılar

Otomatik Kodlayıcı (AutoEncoder), çok boyutlu veriyi önce gizli bir boyutta sıkıştıran sonra da sıkıştırılmış gizli boyuttaki veriyi tekrar oluşturmak için(reconstruct) kullanılan gözetimsiz(unsupervised) bir sinir ağı çeşididir.

Vanilla otomatik kodlayıcılar en temel otomatik kodlayıcı olarak kabul edilmektedir ve bu kodlayıcılarda 3 katman bulunmaktadır. Şekil 3.13’de mavi düğümler girdi katmanını, mavi düğümlerden sonra gelen Layer L2 yazan turuncu düğümler gizli boyutu, Layer L3 yazan düğümler ise çıktı katmanını göstermektedir.



Şekil 3.12 Basit Otomatik Kodlayıcı Yapısı [29]

Otomatik kodlayıcının amacı datayı yeniden inşa etmek olduğundan girdi katmanı ile çıktı katmanındaki düğüm sayısı birbirine eşittir. Girdi katmanı ile gizli boyut(katman) arasına kodlayıcı(encoder) denmektedir. Kodlayıcı çok boyutlu verinin boyutunu azaltmamızı sağlar gizli boyut ile çıktı katmanı arasına ise çözücü(decoder) denmektedir. Çözücünün amacı azalmış olan boyutu arttırarak tekrar eski boyuta döndürmektir. Otomatik kodlayıcıların başka bir amacı da verinin üzerindeki gürültüyü(noise) azaltıp nitelikleri ortaya çıkarmaktır. Bu sebeple çok katmanlı Perseptron ile yapılan deneyleri geliştirmek için Vanilla Otomatik Kodlayıcı yapısı uygulanmıştır.



4. ÖNERİLEN YÖNTEM

Bu çalışma çok oyunculu savaş oyunu Robocode'a dört farklı makine öğrenmesi tabanlı çözüm önermektedir. Bunlardan ilki, öğrenme platformundan maksimum ödülü almayı amaçlayan PÖ yaklaşımını kullanır. İkinci yaklaşım ise YSA mimarisine dayanan denetimli bir öğrenme algoritmasına dayanır. Üçüncü yaklaşım GA'yı, son olarak ise robotlarından maksimum alınabilecek başarı oranlarını zorlamak için bir YOK tabanlı model tasarlanmış ve uygulanmıştır.

Bu çözümlerin her biri sırasıyla ayrıntılı olarak açıklanacaktır.

4.1 Robocode İçin Pekiştirmeli Öğrenme

PÖ, belirli bir durumda ödülü en üst düzeye çıkarmak için uygun önlemleri almayı amaçlar. Belirli bir durumda mümkün olan en iyi davranışı bulmak için çeşitli yazılım ve makineler tarafından kullanılır. PÖ, denetlenen öğrenmeden, robotun ya da oyuncunun verilen görevle ne elde etmek için ne yapacağını kabul ettiğini gösterir.

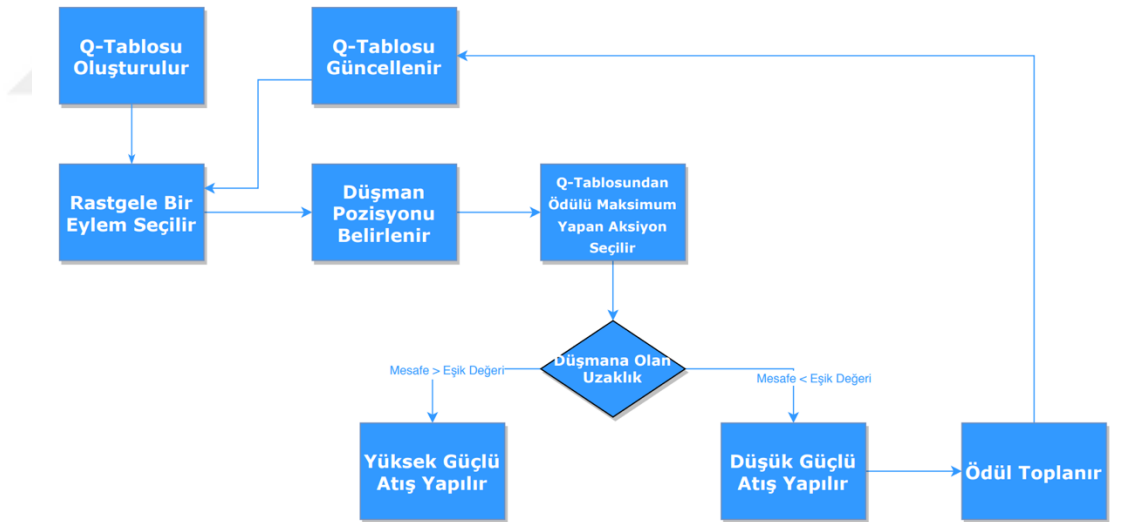
Bir eğitim veri setini kullanmak yerine, robotun deneyimlerini baz alarak öğrenmeye devam eder. Q öğrenme, en popüler PÖ algoritmasıdır ve etkinliği ve popülerliği nedeniyle bu çalışma için tercih edilmektedir [17]. Robot esas olarak çevreyi gözlemler ve önceden tanımlanmış aksiyonu kullanarak eylemi gerçekleştirir. Robot bu eylemin devamında sonucu oyundan veya platformdan alır. Her bir durum ve aksiyon çifti ödül hakkında ipucu verdiği için ileride kullanılmak üzere tutulmalıdır. Algoritma temel olarak, her bir durumda eylem için beklenen maksimum kazancı gösteren bir Q-Tablosu oluşturmayı amaçlamaktadır. Q öğrenmede güncelleme kuralı en uygun Q-Değerini tahmin etmek üzere Bellman Denklemine göre tasarlanmıştır ve Q-Öğrenme güncelleme kuralı (1) olarak verilmiştir.

$$Q(s_t, a_t)^{new} = (1 - \alpha)xQ(s_t, a_t)^{old} + \alpha x(r_t + \gamma \max Q(s_{t+1}, a_t)) \quad (1)$$

Her bir ‘t’ anında robot ya da oyuncu “a_t” eylemini seçtiğinde “r_t” ödülünü elde eder ve bir sonraki “s_{t+1}” durumuna geçer. Ayrıca, öğrenme oranını ifade ederken, indirim faktörünü gösterir.

Verilen algoritma ve yaklaşım içinde, Robocode'un kural ve ortamına göre bir Q öğrenme robotu tasarlanmıştır. Buna göre herhangi bir robot, radarını kullanarak düşmanın konumunu, duruş açısını ve düşmanın mesafesini bilir, daha sonra her adımda robot mevcut durum için yaklaşmakta olan ödülü en üst seviyeye çıkaracak olan eylemi seçer.

Bu durum ve eylem çiftleri, oyun sırasında güncellenen tabloda saklanır ve robot, uygulanan eylemler sonucunda tabloyu (Q-Tablosu) güncellemek için her tick’teki ödülleri toplar. Arama tablosu (Lookup Table), Robocode problemi için önerilen Q-Öğrenme robotu adına aşağıdaki durumları ve eylemleri içerir (Bkz. Çizge 4.1). Q-Learning robotunun akış şeması, Şekil 4.1’de verilmiştir.



Şekil 4.1 Q-Öğrenme ile robot eğitiminin akış şeması

Çizelge 4.1 Robocode Q-Öğrenme algoritmasında kullanılan durum ve eylem listeleri

<i>Durum</i>	<i>Eylem</i>
Robot konumu	Düşmandan kaç
Düşman konumu	Düşmana doğru git
Rakip ile aradaki duruş açısı	Pozisyonunu koru
Enerji seviyesi	Saat yönünde ya da tersi yönünde hareket et

Uygulanan Q-Öğrenme algoritmasının presudo kodu aşağıdaki şekilde verilmiştir:

Ensures Updated Q-Table for action state coordination

Procdure Q-Learning

Initiliaze state-actions table $Q(s,a)$

Current state “s” should be selected

While(A final state or threshold value is obtained)

Basing on the action selection policies select and action **a**

Obtain reward **r** for selected action alongside with the next state.

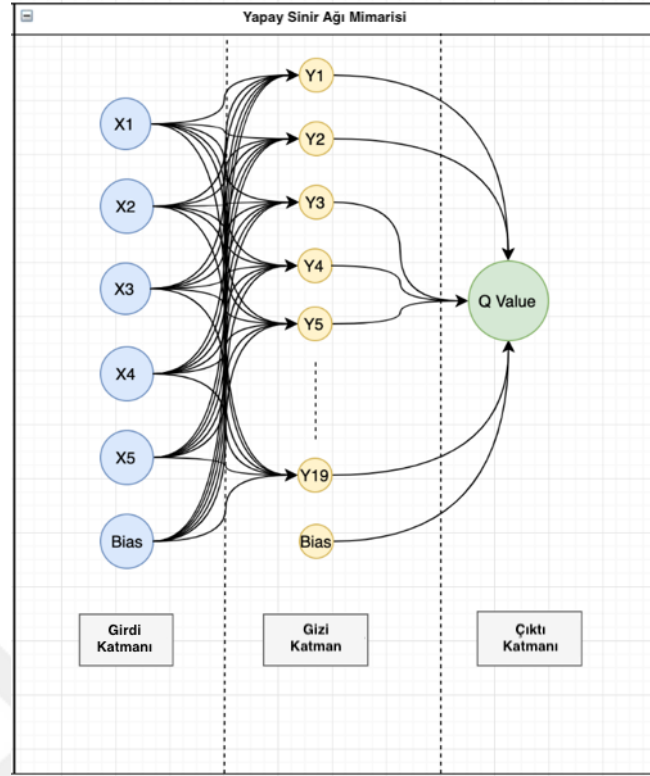
Update Q value for current state **s** and for following state according to (1) and parameters

EndWhile

EndProcedure

4.2 Yapay Sinir Ağı ile Robocode İçin Q Değerlerinin Öğrenilmesi

YSA, birçok farklı ve zorlu soruna adapte edilebilecek standart tahmin etme mekanizmaları olarak kabul edilir. Robocode ortamına mevcutta birçok çalışma uygulanmıştır, bu referansların bazıları Bölüm 1'de görülebilmektedir. Buna göre, bu çalışmalardan ilham alan çok katmanlı bir perseptron uyarlanmış ve bu çalışma için tasarlanmıştır. Esas olarak, maksimum ödülü getiren Q değerlerini yaklaşık olarak belirlememize izin veren eylem ve durumlara dayanarak Robocode savaşlarında her bir tik için en iyi çıktıyı aramayı amaçlar.



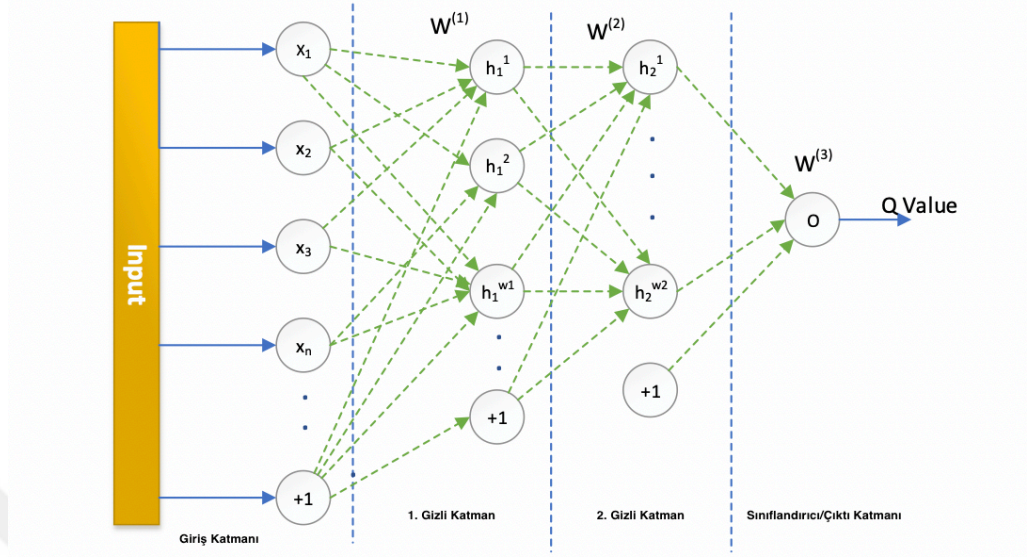
Şekil 4.2 YSA Mimarisi

Önerilen sinir ağı çok basit bir yapıya sahip olan iki katmandan oluşmaktadır. İlk katman (Input Layer) girişleri, yani X pozisyonunu, Y Konumunu, robot ile rakip arasındaki mesafeyi, rakip ile olan atış açısını, hareket ve bias değerlerini temsil eder. Diğer yandan ikinci katman (Hidden Layer) tamamen bağlı(FC) bir katmandır. Son katman (Output Layer), Şekil 4.2'de görüldüğü gibi Q değerlerini temsil eder. Sigmoid fonksiyonu, aktivasyon fonksiyonu olarak kullanılmıştır ve ayrıca gizli katmandaki düğüm sayıları, daha yüksek öğrenme başarısı için yapılan denemeler sonucunda belirlenmiştir.

4.3 Yığılmış Otomatik Kodlayıcılar ile Robocode için Öğrenme

Yığılmış seyrek otomatik kodlayıcı kendi içinde seyrek otomatik kodlayıcıları içeren bir tür DSA'dır ve bu ağlar çoğunlukla sınıflandırma veya regresyon sorunları için son katman olarak literatürde düzenli kullanılmaktadır [18]. Biz de bu modeli problemimize sonuçlarını ele almak için uygulayarak, modelin iyileştirme getirip getirmediğini elde etmeyi

amaçladır. Şekil 4.3'te gösterilen bu problem için örnek bir model tasarlanmış ve verilmiştir.



Şekil 4.3 YOK Mimarisi

Buna göre, ilk otomatik kodlayıcılar denetimsiz(unsupervised) bir eğitim yöntemi kullanılarak eğitilmiştir [18]. Temel olarak, birinci seyrek otomatik kodlayıcının çıktısı, ikincisi için bir girdi olarak kabul edilir ve ikinci otomatik kodlayıcıların çıktısı, ilgili şekilde gösterildiği gibi sınıflandırıcıya bir girdi olur. Otomatik kodlayıcılar ve "SoftMax" sınıflandırıcısı, optimum Q değerini tahmin etmek için geri yayılım algoritması kullanılarak istiflenmiş ve kalifiye olmuşlardır. Her Otomatik Kodlayıcı, Denklem 6'da gösterilen maliyet işlevi kullanılarak eğitilir. E_r değeri, ortalama kare hata (OKH) yaklaşımı kullanılarak düzenlenir.

$$E_r = \frac{1}{M} \sum_{m=1}^M \sum_{t=1}^T (x_{tm} - \hat{x}_{tm})^2 + \lambda * \Omega weights + \beta * \Omega sparsity \quad (2)$$

E_r Hata oranını, x girdiyi, $\hat{x}$ ve geri yüklenen veriyi tanımlamaktadır, " λ " katsayısı L2 "Ağırlık Regülasyonu(Weight Regularization)" tarafından, " β " katsayısı "Kısıtlılık Düzenlemesi(Sparsity Regularization)" için kullanılır. m gözlem sayısıdır, t eğitim ise veri etiketi numarasını gösterir.

$\Omega weights$ Ağırlık Regülasyonu'nu göstermektedir ve aşağıdaki gibi tanımlanmaktadır:

$$\Omega weights = \frac{1}{2} \sum_x \sum_j^J \sum_k^K w_{jk}^{(x)^2} \quad (3)$$

Burada X, gizli katmanların sayısını, n, gözlem sayılarını ve k, gizli katmanları gösterir [18].

Kısıtlılık Düzenlemesi(Sparsity Regularization) ise şöyle tanımlanabilir:

$$\Omega sparsity = \sum_{i=1}^D KL(\rho || \hat{\rho}_i) \quad (4)$$

Burada beklenen değer ρ , $\hat{\rho}_i$ olarak, "i" her bir nöronun ortalama çıkış aktivasyonunu belirtirken, "KL" ise aynı veriler üzerinde iki olasılık dağılımı arasındaki farkı değerlendiren fonksiyondur. Bu denklemlerin detayları [18] 'de görülebilir.

4.4 Genetik Algoritma ile Robocode için Öğrenme

Robocode'daki bir robot sınıfı soyut(abstract) Robot metotlarından herhangi bir alt kümesini içerebilmektedir. Bu metotların tamamına Robocode dokümantasyonundan ulaşılabilmektedir. Geliştirilen genetik robot için aşağıdaki metodların kullanımına odaklanılmıştır:

- onScannedRobot
- onHitByBullet
- onHitWall

Bu metotlar içinde parametre olarak gelen "e"den yararlanılarak parametreler ayarlanmaktadır. Program evrilmeye ve gelişmeye devam ettiği süre içinde jenerasyonlar içinde onScannedRobot metodu içinde yapılan hareketlerin ve ateşlemenin problemin çözümüne yaklaştırdığı görülmüştür. Bu sebepten ötürü bu metodu doğru şekilde programlayarak içinde kullanılan parametrelerin evrilmesini sağlamak amaçlanmıştır. Çizge

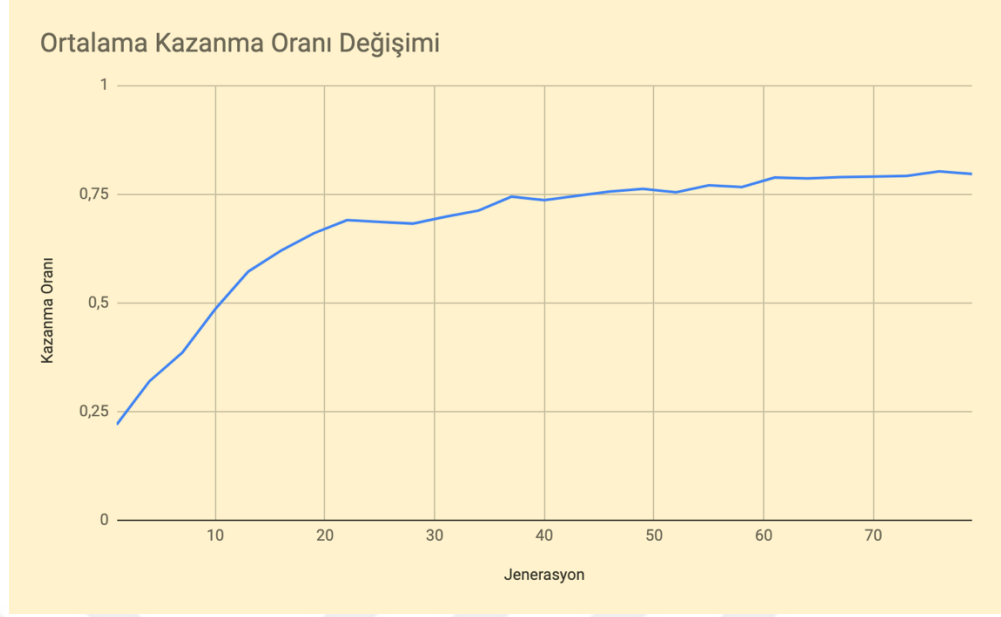
4.2 onScannedRobot içinde çağrılan metotları göstermektedir. Herhangi bir t anında bu metotların içine doğru parametrelerin veriliyor olması GA ile eğitilmiş robotun başarılı olmasını sağlayacaktır.

Çizelge 4.2 Robocode GA’da çağrılan metot listeleri

<i>Metot</i>	<i>Aldığı Parametre</i>
setAhead	İlerleme mesafesi
setTurnRight	Robotun dönme miktarı
setGunTurnRight	Silahın dönme miktarı
setTurnRadarRight	Radarın dönme miktarı
setFire	Rakibe yapılacak olan atışın gücü

Yukarıdaki metotların içine aldığı parametreler alt ifade ağaçları (expression tree) ile oluşmaktadır. Bu 5 adet alt ağacın birleşimi bir bireyi yani genomu meydana getirmektedir. İfade ağaçlarını oluşturmak için ise terminallerin ve operasyonların tamamının bulunduğu bir küme gerekmektedir.

Çizge 4.3’te ifade ağaçlarını oluşturmak için kullanılan terminaller listelenmiştir. Bu kümede statik terminaller, genel terminaller, operasyonlar ve matematiksel terminaller bulunmaktadır. Bu sebeple genetik robot yaratan programın en zorlu yanı, ifade ağaçlarının büyümesinin ya da küçülmesinin nasıl düzenleneceğini bulmaktır. Bu problem çaprazlama(Crossover) sırasında hangi alt ağaç boyutlarının kullanılabileceğini belirleyen bir algoritma ile, ikinci ebeveynde ihtiyacı olunan bir alt ağacı bularak ve birinci ebeveynde bunun için geçerli bir ekleme noktası belirlenerek çözülebilir. Bu yöntem sayesinde gelişme regüle edilmiş bir hale gelmiştir. Şekil 4.4’de jenerasyonlara göre kazanma oranının(fitness) değişimi gösterilmiştir.



Şekil 4.4 Jenerasyonlara Göre Kazanma Oranı Değişimi

Bu grafiğe bakıldığında başlangıçta %15 civarında başlayan kazanma oranının ilk jenerasyonlarda hızlı bir evirilme göstererek %70 barajına yakınsadığını görmekteyiz. 60. Jenerasyon sonrasında ise başarı oranı çok değişmemeye başlayarak %73-%75 aralığında kalmaktadır. Buna karşın ortalama yerine maksimum kazanma oranını incelediğimizde en iyi değer olarak TrackerBot'a karşı %87 kazanma oranı alınmıştır.

Çizelge 4.3 İfade ağaçları için kullanılan terminaller

<i>Terminal Tipi</i>	<i>Terminal Adı</i>
Genel Terminaller	getEnergy()
	getHeading()
	getVelocity()
	getWidth()
	getX()
	getY()
	getGunHeadingRadians()
	getGunTurnRemainingRadians()
	getHeadingRadians()
	getRadarHeadingRadians()
	getRadarTurnRemainingRadians()
Statik Terminaller	0,0001
	Math.random()
	Math.random() * 2 - 1
	Math.floor((Math.random()*10))
	Math.PI
	fitnessVar
Tarama Event'i Terminalleri	e.getBearingRadians()
	e.getDistance()
	e.getEnergy()
	e.getHeadingRadians()
	e.getVelocity()
Matematiksel Operasyon Terminalleri	Math.abs()
	Math.acos()
	Math.asin()
	Math.cos()
	Math.sin()
	Math.toDegrees()
	Math.toRadians()
	*-1

4.4.1 Fitness değerinin hesaplanması

Nüfus içindeki her birey için, genetik programlama kendi uygunluğunu savaş sonuçlarını alarak denemelidir. Bu da öğrenme sırasında Robocode Kontrol API'si aracılığıyla gerçekleştirilmiştir. Her nesilde, popülasyonun her üyesi belirli bir tur için her rakibe karşı

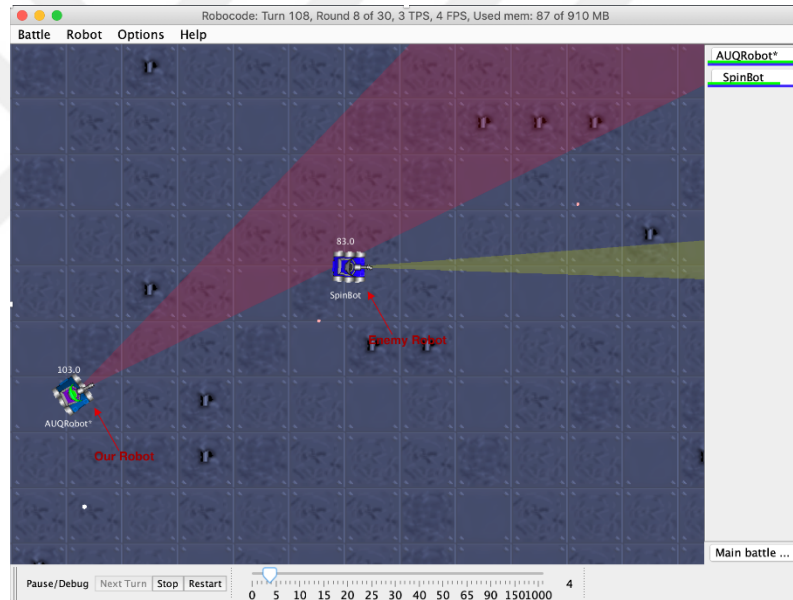
savařır. Her turun sonrasında hem GA robotu hem de rakip iin puan toplanır. Relatif puan GA robotunun puanının toplamının rakibin puanına blerek hesaplanmasıyla elde edilir. Robotun genel uygunluęunu yani fitness deęerini bulmak iin her bir turun sonu puanı toplanarak toplam tur sayısına blnr. Yani turlardaki puanların ortalaması alınır. 0.50 veya daha yksek bir fitness deęeri, robotun rakibine eřit veya ondan daha iyi bir aday olduęunu gsterir. Robocode deterministik olmayan bir oyun olması sebebiyle fitness deęerlerinde řans unsurunun da hesaba katılması gerekmektedir. řansı en aza indirmenin tek yolu ise her bir test iin tur sayısını arttırmaktır.

Gvercin delięi ilkesi(pigeon hole principle) reme iřlemi iin kullanılmıřtır. Mutasyon ihtimali parametresi 0.1, aprazlama ihtimali parametresi 0,85 ve replikasyon ihtimali parametresi 0,5 olarak belirlenmiřtir. Bu konfigrasyonlarla 100 jenerasyon sonrasında elde edilen robot iin sonular Blm 5.6’da listelenmiřtir.

5. ARAŞTIRMA BULGULARI VE DENEYSEL SONUÇLAR

5.1 Uygulama Platformu

Yukarıda belirtilen Robocode, IBM Alphaworks [7] tarafından geliştirilen bir tank savaş emülatörüdür. Temel olarak tank veya ekipler rakipleri tarafından vurulmaması için çevreye doğru yönelmek ve hareket etmek zorundadır. Rakiplerine karşı savaşı bireysel bir şekilde kazanmaları için tek ve çok robotlu sistemleri eğitmek adına üç farklı makine öğrenmeye dayalı yaklaşım bu çalışma için kullanılmıştır. Deney yapmak için Intel Core i7-6700 CPU'da @ 2.60 GHz ve 16 GB RAM'e sahip bir masaüstü bilgisayar kullanılmıştır. Her yöntem ve sonuçlar senaryolar tanımlanarak ayrı ayrı gösterilmektedir.

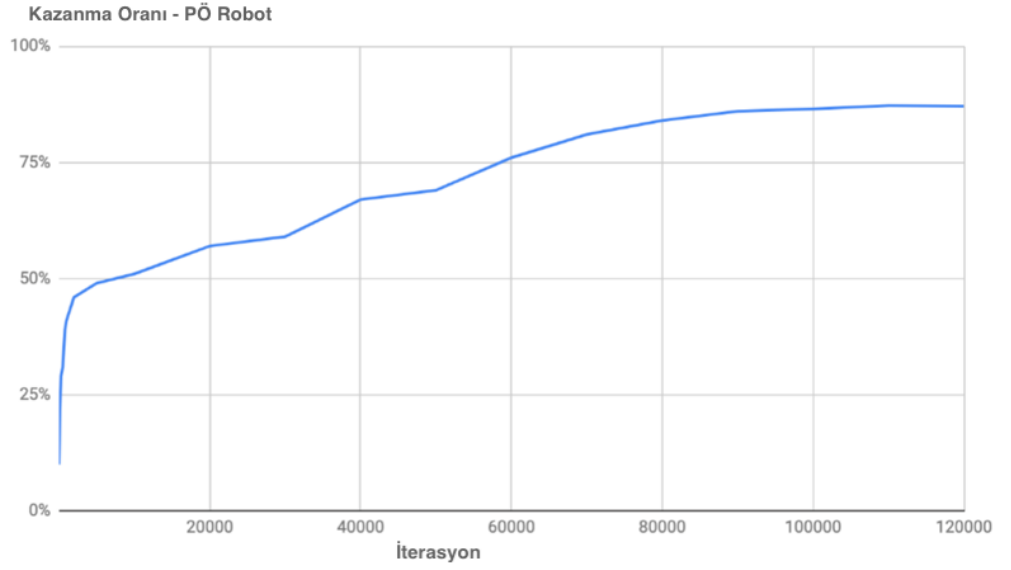


Şekil 5.1 1. Senaryodan Alınmış Ekran Görüntüsü

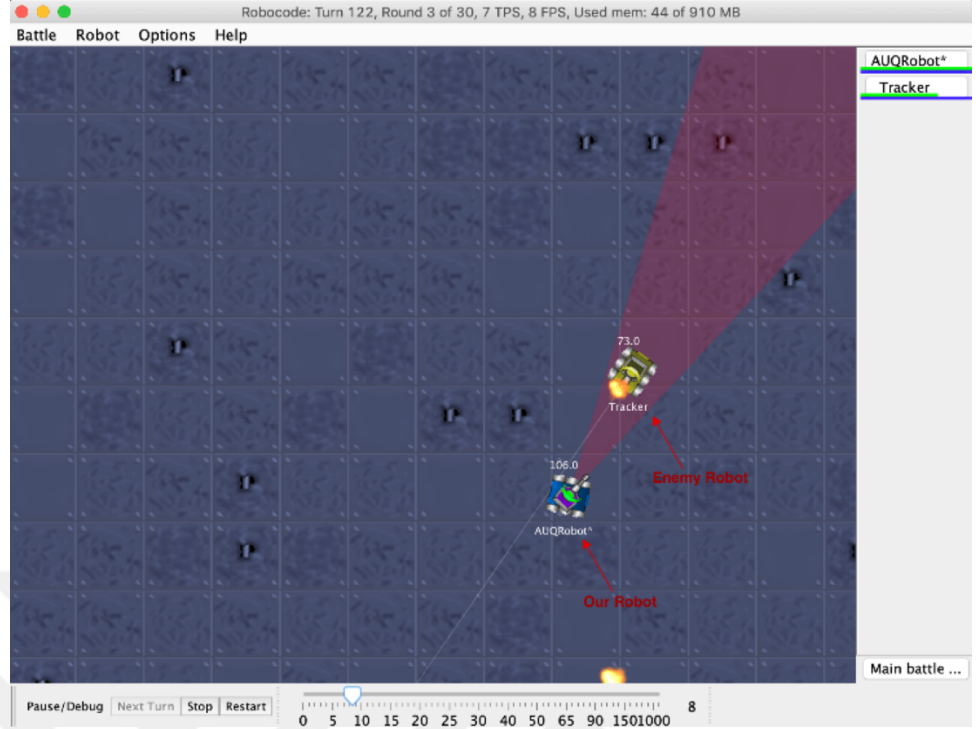
5.2 Senaryo-1

Bu senaryo, Q öğrenme ile öğretilerek özelleştirilmiş robotun (AUQRobot) Spin Robot'a karşı savaştığı tek bir robot savaşını göstermektedir. Örnek bir ekran görüntüsü Şekil 5.1'de gösterilmektedir. Senaryoda robotu eğitmek için 12.000 turluk bir eğitim savaşı yaptırıldı.

Aşağıda gösterilen Şekil 5.2, kazanan yüzdelerin tur boyunca değişimini göstermektedir. Grafiğe geri gelecek olursa, kazanma yüzdesinin hırslı yöntemle (Greedy Approach) sağlanan PÖ ile %87'ye yakınlığı kabul edebiliriz. Sağlaması yaparak kontrol etmek istersek tur boyunca toplanan kümülatif ödül değişikliği aynı eğriyi verir. Robocode maksimum veri depolama kısıtlamaları altında elde edilen bu sonuç. Sonuçlar 9216 ölçülerinde bir Q masasıyla deneyimlenmiştir ve ayrıca masa boyutunun artırılmasının muhtemelen toplam kazanma performansını artıracığı belirtilmektedir.



Şekil 5.2 AUQRobot'un İterasyonlara Göre Başarı Değişimi Grafiği



Şekil 5.3 1. Senaryodan Alınmış Ekran Görüntüsü (TrackerBot)

Bu senaryoda, PÖ ile özelleştirilmiş robot (AUQRobot), Robocode'da kullanılan popüler bir robot olan TrackerBot ile karşı karşıya gelmiştir (bkz. Şekil 5.3). Bu senaryoda aynı eğitim yapılandırması da uygulanmış ve %92 kazanma oranı elde edilmiştir.

Şekil 5.4'da gösterilen sonuçlar Robocode platformundan bu savaş için elde edilen örnek bir ekran görüntüsüdür. Sonuçlara ilişkin olarak, Tracker robot, 20 turluk bir savaşın hiçbir turunda hayatta kalmadı ve AUQRobot robot rakibine karşı hep hayatta kalmasına karşın yediği mermiler ve çarpışma puanlarından dolayı %8 miktarında puanı rakibine vermiş oldu.

Results for 20 rounds											
Rank	Robot Name	Total Score	Survival	Surv Bonus	Bullet Dmg	Bullet Bonus	Ram Dmg * 2	Ram Bonus	1sts	2nds	3rds
1st	uu.AUQRobot*	3407 (92%)	1000	200	1835	367	5	0	20	0	0
2nd	sample.Trac...	293 (8%)	0	0	292	0	1	0	0	20	0

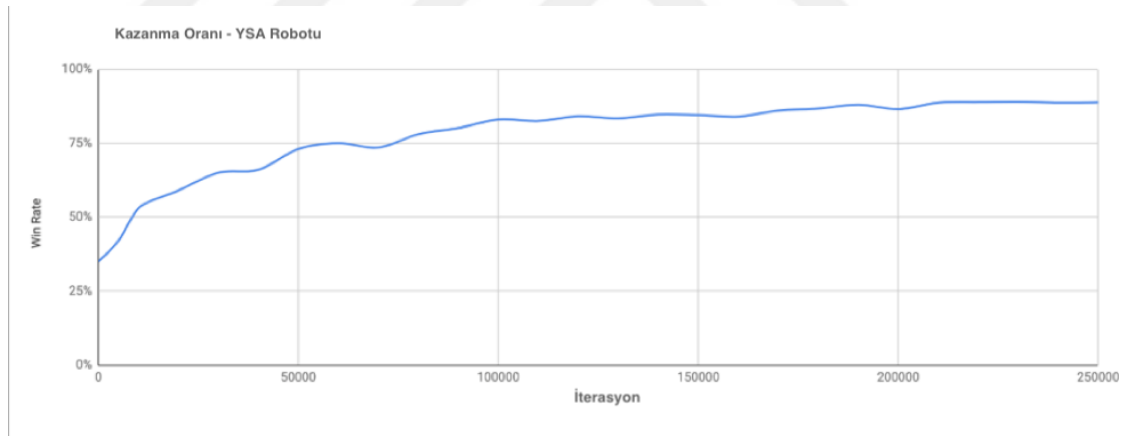
Şekil 5.4 1. Senaryodan için Robocode Sonuçları(TrackerBot)

5.3 Senaryo-2

Bu senaryo, YSA ile eğitilerek özelleştirilmiş olan bir robotun (AUNNRobot) Spin Robot'a karşı savaştığı tek robotlu bir robot savaşını ele almaktadır. Yukarıda tarif edilen YSA mimarisine göre geliştirilen robot, iki yüz elli bin iterasyon sonucunda SpinBot'a karşı eğitildi. Tablo 2, YSA tabanlı sistemin yapılandırmasını göstermektedir.

Çizelge 5.1 Yapay Sinir Ağı Konfigürasyonu

<i>Sinir Ağı Girdileri</i>	<i>Parametre Aralığı</i>	<i>Sinir Ağı Girdi Aralığı</i>
Robot Konumu X	0px - 800px	{0.0, 0.1... 7.9, 8.0}
Robot Konumu Y	0px - 600px	{0.0, 0.1... 7.9, 8.0}
Rakip ile aradaki mesafe	0px - 1000px	{0.0, 0.1... 5.9, 6.0}
Rakip ile aradaki atış açısı	0° - 360°	{0.0, 0.1... 5.9, 6.0}
Enerji Seviyesi	0 - 200	{0, 1, 2 ... 199, 200}



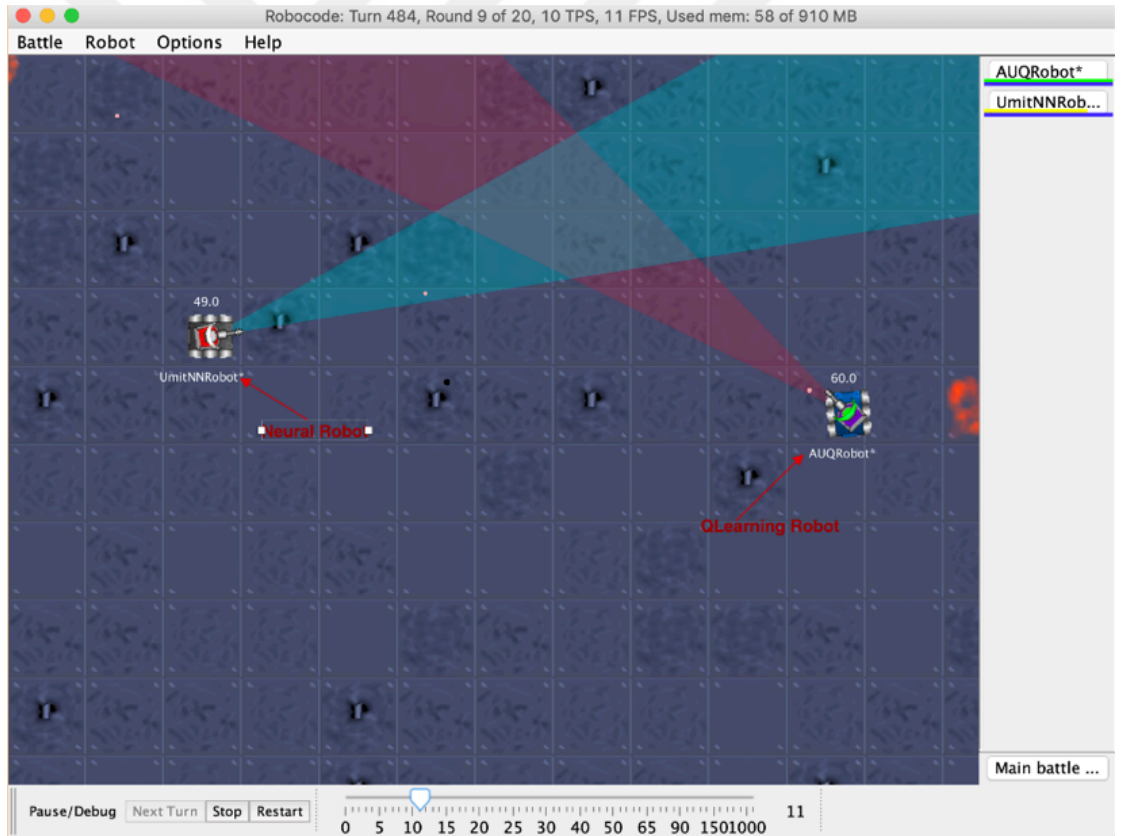
Şekil 5.5 AUNNRobot robotunun iterasyonlara göre başarı değişim grafiği

PÖ ile elde edilen lineer regresyon ve Q-değerini gösteren sinir ağı, bu sistemi eğitmek için algoritma olarak kullanılmıştır. Şekil 5.5, YSA tabanlı robotun eğitim sürecindeki kazanma yüzdesini göstermektedir. Sonuçlar, YSA tabanlı yöntemin hızla öğrenmeye başladığını ancak PÖ ile karşılaştırıldığında daha geç ideal değerlere yaklaştığını göstermektedir. Buna

göre, bu senaryoda, AUQRobot ve AUNNRobot arasındaki bir savaşın, her ikisi de aynı robot sınıfı için eğitilmiş olduğu, her iki sistemin performansını da uygun şekilde karşılaştırılabileceği düşünülmektedir. Genel olarak ele alacak olursak bu iki robot arasındaki savaş için katılımcıların hiçbiri diğerine mutlak bir üstünlük gösterememiştir ancak 50 turluk savaş sonrasında AUNNRobot, AUQRobot'a göre %54'e %46 bir sonuç elde ederek rakibini az bir fark ile yenmiştir. Görsel-12 bu 50 turluk savaşın sonuçlarını göstermektedir.

Results for 50 rounds											
Rank	Robot Name	Total Score	Survival	Surv Bonus	Bullet Dmg	Bullet Bonus	Ram Dmg * 2	Ram Bonus	1sts	2nds	3rds
1st	uu.UmitNNRobot*	3771 (54%)	900	180	2474	215	1	0	20	30	0
2nd	uu.AUQRobot*	3149 (46%)	1500	300	1186	158	6	0	32	18	0

Şekil 5.6 AUNNRobot ile AUQRobot'un 50 turluk savaşının sonuçları

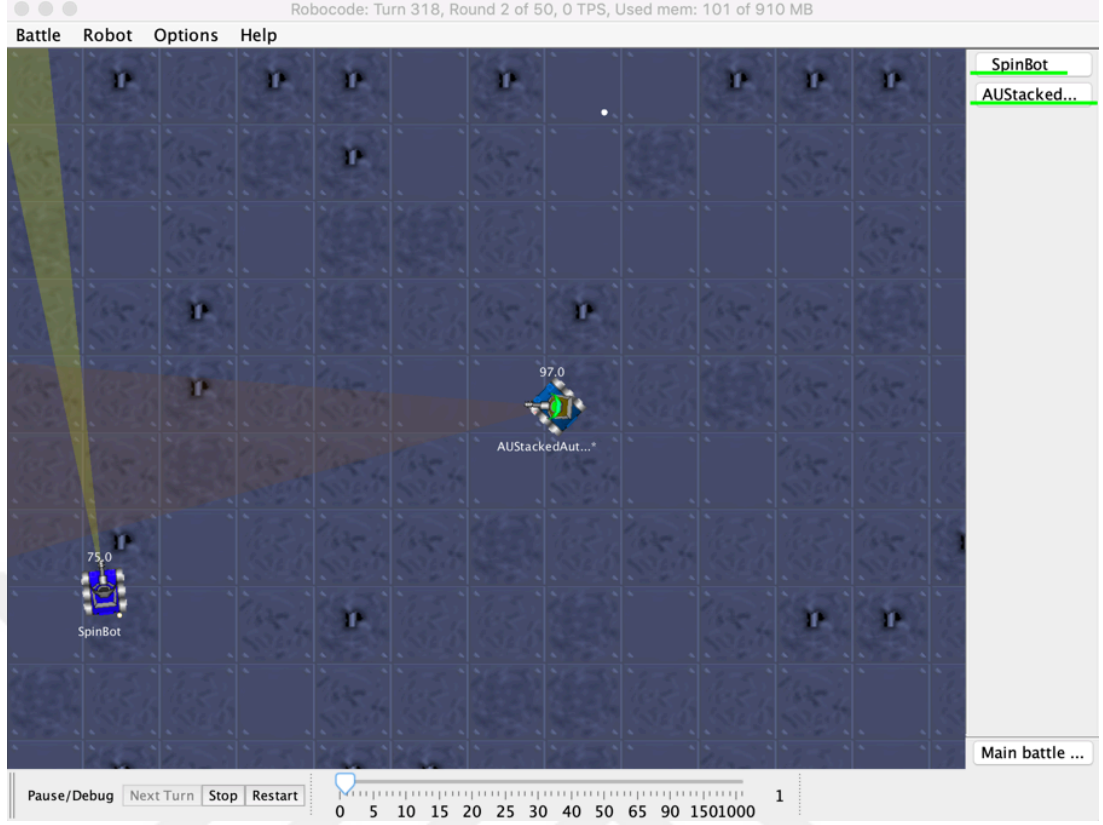


Şekil 5.7 Senaryo-2'den savaş sırasında alınmış ekran görüntüsü



Şekil 5.8 YOK robotun iterasyonlara göre başarı değişim grafiği

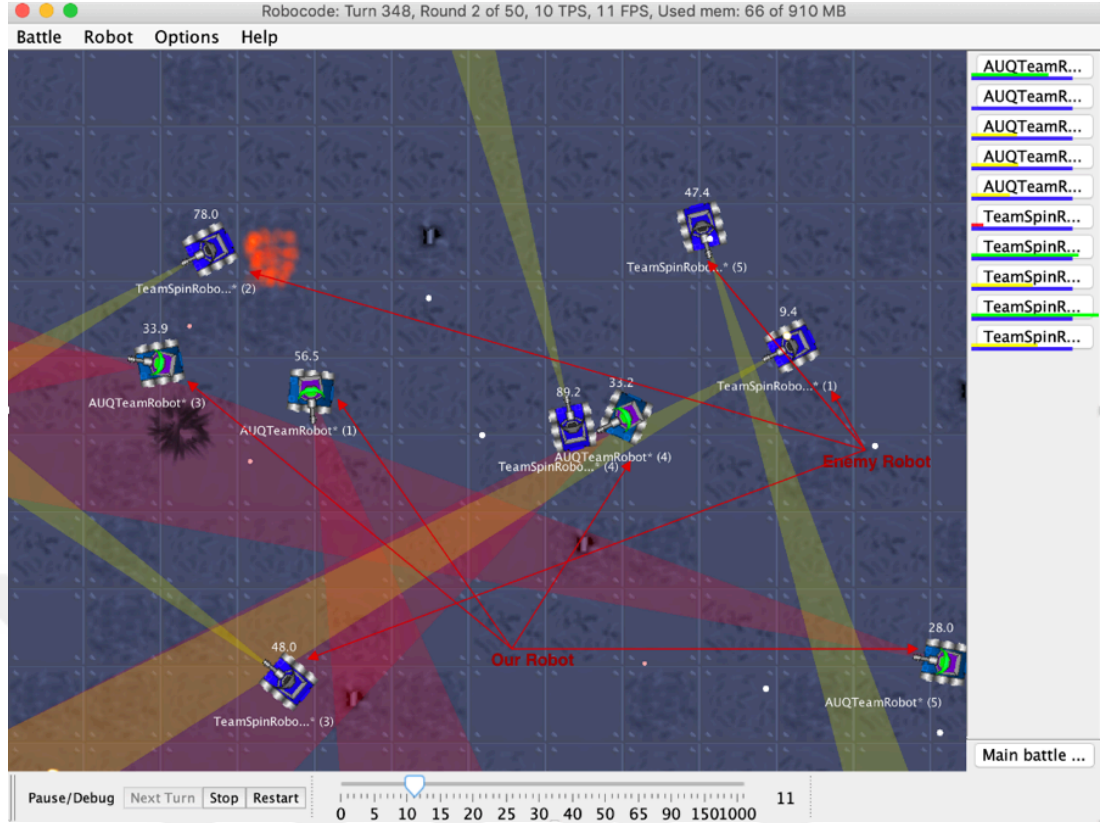
YOK tabanlı yöntemin sonuçları YSA eğitim verisi ile alınmıştır. Buna göre YOK ağıın kazanan yüzdesi Şekil 5.8'de gösterildiği gibi, AUNNRobot ile karşılaştırıldığında yaklaşık olarak %3-4 oranında daha düşük kazanma oranı sağlamıştır. YOK tabanlı AUAERobot (bkz. Şekil 5.9) içinde gösterildiği gibi online olarak eğitilmiştir. Ham görüntü verilerinin, ağa konum değerleri vermek yerine girdi olarak kullanılması durumunda, YOK (Deep Auto Encoder) ve ayrıca KSA(Convolutional Neural Network) tabanlı mimarilerin, daha yüksek başarı elde etme açısından eğitimi maliyetli olan AUNNRobot mimarisinden daha iyi performans gösterebileceğini belirtmeliyiz. Eğitim verilerinin yanı sıra eğitim sürecinin gerçekleştirilmesi yukarıda belirtilen yöntemlerin Robocode ortamında kullanılmasını zorlaştırmaktadır fakat verilen zorluklara rağmen, bu mimarileri verilen problemlere uygulamayı gelecekteki yapılacaklar içinde planlanmaktayız. Sonuç olarak, AUAERobot yerine, AUNNRobot'un AUQRobot ile rekabet etmesi daha mantıklı olacağı için tercih edilmektedir.



Şekil 5.9 Senaryo-2’den savaş sırasında alınmış ekran görüntüsü (AUAERobot)

5.4 Senaryo-3

Bu senaryoda, daha önce özelleştirdiğimiz robotun (AUQRobot) SpinBot takımı ile savaştığı çok robotlu bir robot savaşı gösterilmektedir. Bu çalışmanın başlangıç noktalarından biri de çok robotlu ekip savaşlarının yapılması olduğundan, önce eğitilmiş robot ile tek bir robot sınıfına karşı bir ekip oluşturulmuştur. Daha sonra, bu ekibin üyeleri birbirlerine saldırmamaları için programlanmıştır. Son olarak ise eğitimin verildiği robot sınıfı, AUQRobot'un bir robot ekibi olarak tanımlanmıştır. Şekil 5.10, savaş alanından bir ekran görüntüsünü göstermektedir. Sonuçlar, özelleştirilmiş fakat merkezi bir yönetimi olmayan AUQRobot takımlarının, Şekil 5.11'de gösterildiği gibi, ortalama %65 başarı sağlayarak SpinBot Takımı’ndan daha iyi performans gösterdiğini ortaya koymaktadır.



Şekil 5.10 Senaryo-3'den çok robotlu savaş sırasında alınmış ekran görüntüsü

Rank	Robot Name	Total Score	Survival	Surv Bonus	Bullet Dmg	Bullet Bonus	Ram Dmg * 2	Ram Bonus	1sts	2nds	3rds
1st	uu.AUQTeamRobot* ...	5897 (17%)	3450	450	1833	96	67	0	5	0	1
2nd	uu.AUQTeamRobot* ...	5251 (15%)	3250	0	1806	104	90	0	1	3	3
3rd	uu.AUQTeamRobot* ...	4836 (14%)	2750	180	1748	80	67	11	3	0	0
4th	uu.AUQTeamRobot* ...	4401 (13%)	2550	0	1687	77	88	0	0	2	2
5th	uu.TeamSpinRobot* (1)	3171 (9%)	2500	90	383	9	178	12	1	1	2
6th	uu.TeamSpinRobot* (4)	2978 (9%)	2250	90	415	10	210	3	1	0	0
7th	uu.TeamSpinRobot* (2)	2487 (7%)	2000	0	350	3	128	5	0	2	0
8th	uu.AUQTeamRobot* ...	2052 (6%)	950	0	1014	33	55	0	0	0	0
9th	uu.TeamSpinRobot* (5)	1939 (6%)	1450	0	348	1	140	0	0	1	2
10th	uu.TeamSpinRobot* (3)	1655 (5%)	1300	0	249	3	103	0	0	0	0

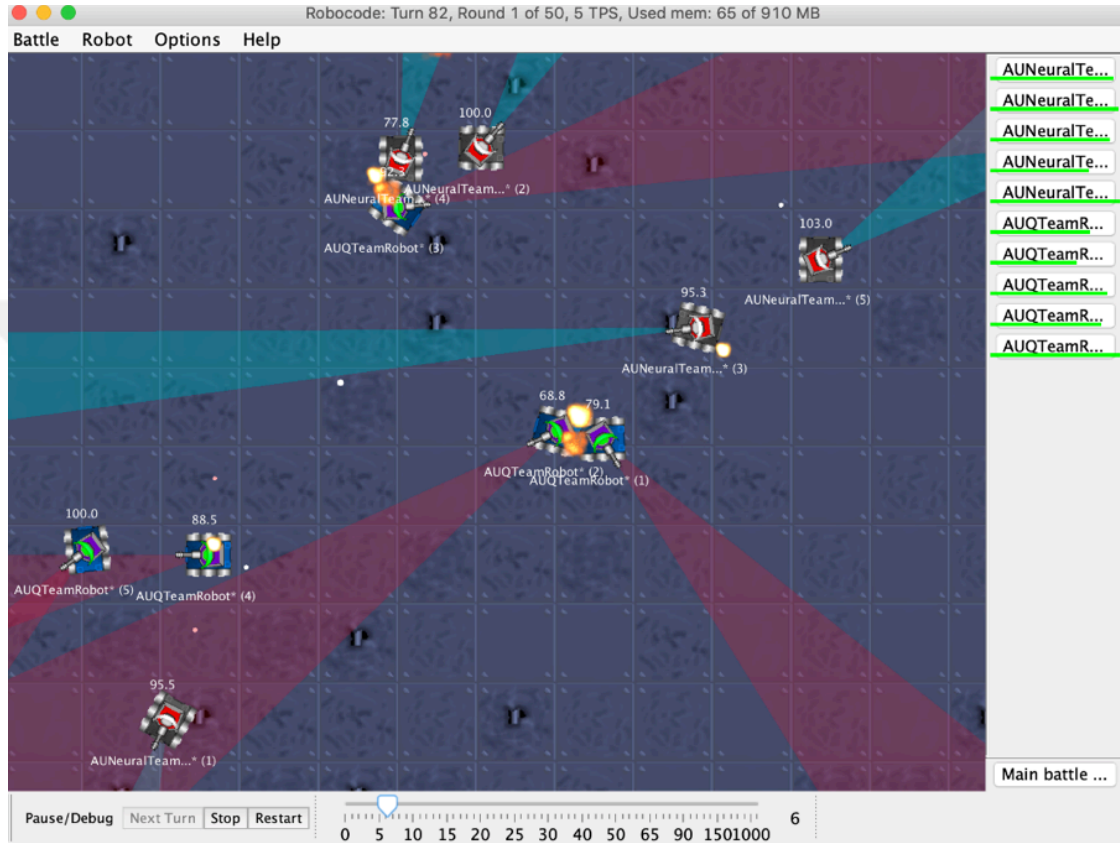
Save OK

Şekil 5.11 Senaryo-3'den savaş bitişinde alınmış sonuçları gösteren ekran görüntüsü

5.5 Senaryo-4

Bu senaryo, beş robot tankından oluşan çoklu robot sistemlerinin savaşını ele almaktadır. Buna göre, ilk takım AUQRobot'tan, ikincisi AUNNRobot modelinden türetilmiştir. Bu senaryodan elde edilen örnek bir ekran görüntüsü Şekil 5.12'de gösterilmiştir. Bu robot takımları arasında birkaç farklı yarışma (deney) yapıldı ve sonuçlar AUNNRobot takımının

rakibinden %67 (minimum) ile %74 (maksimum) başarı oranı arasında daha iyi performans gösterdiğini ortaya koydu. Deney sonuçları sırasında Robocode platformundan üretilen Şekil 5.13'deki sonuçlar 50 turluk savaşı içerir ve AUNNRobot'un rakibine karşı takım performansını gösterir.



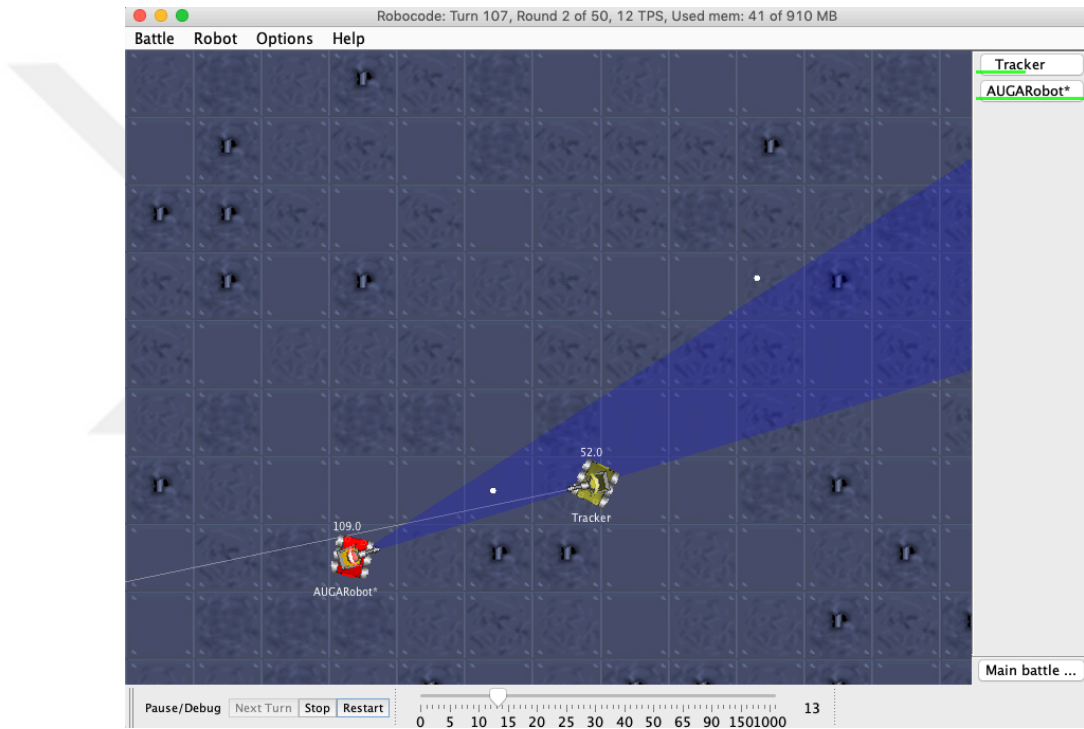
Şekil 5.12 Senaryo-4'den çok robotlu savaş sırasında alınmış ekran görüntüsü

Results for 50 rounds												
Rank	Robot Name	Total Score	Survival	Surv Bonus	Bullet Dmg	Bullet Bonus	Ram Dmg * 2	Ram Bonus	1sts	2nds	3rds	
1st	uu.AUNeuralNetworkTeamRobot* ...	25653 (16%)	16500	990	6945	401	784	33	11	13	8	
2nd	uu.AUNeuralNetworkTeamRobot* ...	24132 (15%)	15100	810	7045	390	779	8	9	7	7	
3rd	uu.AUNeuralNetworkTeamRobot* ...	23626 (15%)	14850	990	6773	336	672	5	11	7	7	
4th	uu.AUNeuralNetworkTeamRobot* ...	23419 (15%)	14950	810	6657	296	688	18	9	9	8	
5th	uu.AUNeuralNetworkTeamRobot* ...	23025 (14%)	14700	630	6687	304	680	24	7	8	11	
6th	uu.AUQLearningTeamRobot* (1)	9019 (6%)	8000	180	763	13	64	0	2	3	2	
7th	uu.AUQLearningTeamRobot* (4)	8275 (5%)	7350	0	831	3	90	0	0	0	3	
8th	uu.AUQLearningTeamRobot* (5)	8174 (5%)	7250	0	876	4	44	0	0	1	4	
9th	uu.AUQLearningTeamRobot* (2)	8075 (5%)	7100	90	831	1	53	0	1	3	0	
10th	uu.AUQLearningTeamRobot* (3)	7274 (5%)	6450	0	756	3	65	0	0	0	2	

Şekil 5.13 Senaryo-4'den savaş bitişinde alınmış sonuçları gösteren ekran görüntüsü

5.6 Senaryo-5

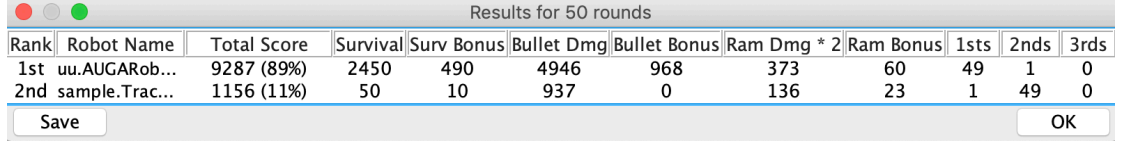
Bu senaryo, GA öğrenme ile öğretilerek özelleştirilmiş robotun (AUGARobot) SuperTracker Robot'a karşı savaştığı tek bir robot savaşını göstermektedir. Örnek bir ekran görüntüsü Şekil 5.14'de gösterilmektedir. Bu senaryodaki GP ile evirilen robot 100 jenerasyonluk 3 gün süren bir eğitimi online olarak almıştır. Bu jenerasyonlar sonucunda uğradığı değişim ile yaptığı savaşlarda ortalama %77 başarı elde etmektedir. Aşağıdaki sonuçlar ise 50 turluk bir savaşın sonucunda elde edilmiştir ve Şekil 5.15, bu savaşın sonuçlarını göstermektedir.



Şekil 5.14 Senaryo-5'den GARobot ile Tracker'ın savaşından alınmış ekran görüntüsü

Robocode platformunda diğer geliştiriciler tarafından tasarlanan robotlarda dinamik ve mantıklı stratejiler keşfedilmektedir. En yaygın yaklaşım olarak, Robotların genellikle arenada hareket ederken rakiplerini takip etmelerini sağlayan bir hedefleme sisteminin geliştirmesidir. Düşman uzaklaştığında kendi enerjisini korumak için daha çok düşük güçlü

mermilere geçilirken düşman yakında olduğu zamanlarda ise güçlü mermiler tercih edilmiştir. Bu stratejilerdeki aksiyonları GP karar ağaçları sayesinde iyi bir şekilde yönetebilmektedir.



Rank	Robot Name	Total Score	Survival	Surv Bonus	Bullet Dmg	Bullet Bonus	Ram Dmg * 2	Ram Bonus	1sts	2nds	3rds
1st	uu.AUGARob...	9287 (89%)	2450	490	4946	968	373	60	49	1	0
2nd	sample.Trac...	1156 (11%)	50	10	937	0	136	23	1	49	0

Save OK

Şekil 5.15 Senaryo-5’den savaş bitişinde alınmış sonuçları gösteren ekran görüntüsü

Bu sonuçları incelediğimizde GARobot’un ortaya ortalama başarısının üzerinde bir sonuç çıkardığını görmekteyiz. Uzun bir süre sonunda evirildiğini ele alacak olursak bu sonucu oldukça başarılı olduğunu kabul ediyoruz. Bu noktada tek bir sınıfa karşı geçirilen jenerasyonlar sonrasında GP’nin her bir tick’teki hareket ve aksiyonlarının en optimum çözümlere yakın olduğu görülmektedir. Fakat GP’nin, PÖ ve YSA’ya göre çözüme daha geç ulaştığını da unutmamak gerekmektedir.

6. SONUÇ

Bu makale, tek ve çok ajanlı sistemler için popüler bir 2-B oyun simülatörü, yani Robocode kullanarak bazı popüler makine öğrenme temelli yaklaşımları tanıtmakta ve karşılaştırmaktadır. Bu platform, temel olarak araştırmacıların rekabete katılmak ve tüm dünyadaki tank savaş oyuncularını ve tasarımcılarını fikirlerini gerçekleştirmek üzere özelleştirilmiş robot ekipleri tasarlamalarına olanak tanıyor.

Alan probleminin (Space Problem) oyunların özelliklerine göre devam etmesine rağmen, problem için PÖ tabanlı bir model tanıtılmıştır. Ayrıca, bir YSA tabanlı modeli özünde gerçekçi bir yaklaşım olmayan çok büyük bir Q-Tablosu oluşturmak yerine Q-değerlerini yaklaşık olarak belirlemek için tasarlanmıştır. Ek olarak, önceki deneyimler YOK'ların etiketli veriler (labeled data) elde edildikten sonra alternatif bir denetimli(supervised) öğrenme yaklaşımı sunabileceğini kanıtladı. Bununla birlikte, ham görüntü yerine konum verileri, mesafe, enerji gibi reel matematiksel değerler kullanıldığından, YOK'lar görüntülerde hata ayıklama(denoising) veya giriş boyutunu küçültme (reducing size) gibi bir gelişmişlik sağlamaz. Bu sonuçlarda, oyun durumlarını gösteren ham görüntülerin, bir YSA mimarisinden daha güçlü bir mimari tasarlamaya girdi olarak derin mimariler tarafından daha iyi kullanılması gerektiği belirtilmelidir. Bununla birlikte, Robocode'da verilen girdileri ele aldığımızda YSA modeli hem tekli hem de çok ajanlı sistemlerde makine öğrenmesi yaklaşımlarından daha iyi performans gösterir.

Deneyisel sonuçlar ve bu sonuçların değerlendirilmesi, yazarları gelecekteki eserler olarak ham görüntüleri kullanarak YOK'ler veya KSA tabanlı bir model tasarlamaya teşvik eder. Bu model, muhtemelen hem PÖ hem de YSA tabanlı modellerden daha iyi performans gösterecek, ancak daha büyük miktarda eğitim verisine ihtiyaç duyabilecek ve aynı zamanda uygun bir model oluşturmak için aşırı eğitim süresi gerektirebilecek modeller yetiştirmek için ham görüntü verilerine ihtiyaç duyacaktır.

6.1 Sonuçların Karşılaştırma Tablosu

Çizge 6.1’de bölüm 5’te ele alınan senaryolar karşılaştırılmıştır.

Çizelge 6.1 Senaryo sonuçlarının karşılaştırması

Senaryo Adı	Robot Adı	Rakip Adı	Başarı Oranı	Rakip Başarı Oranı
Senaryo-1	AUQRobot	Tracker	%92	%8
Senaryo-2	AUNNRobot	AUQRobot	%54	%46
Senaryo-3	AUQRobot(Takım)	SpinBot(Takım)	%65	%35
Senaryo-4	AUNNRobot(Takım)	AUQRobot(Takım)	%67	%33
Senaryo-5	AUGARobot	Tracker	%77	%23

Bu sonuçları dikkate aldığımızda YSA ile eğitilen robotun PÖ’ye göre daha iyi sonuç verdiğini görmekteyiz. PÖ robotlarının ise daha büyük q-tabloları ile daha iyi sonuçlar aldığı da ayrıca kanıtlamış oluyoruz. GP’deki robotların ise bu sıralamada PÖ robotlarına göre tek robotlu savaşlarda daha iyi sonuç verdiğini fakat eğitiminin yani evrilmesinin daha uzun bir süre aldığını da görmekteyiz.

6.2 Gelecekteki Çalışmalar

Görüntü sınıflandırma için büyük veri setlerinin erişebilirliği ve kullanılabilirliği son yıllarda oldukça artmıştır. Buna ek olarak GPU'lar ile paralel veri işleme yeteneği derin öğrenmeyi oldukça popüler hale getirmiştir. Robocode’da etiketlenilmiş görsel veri kümelerinin olmayışı sebebiyle sonraki çalışmalara kaynak sağlayacak bilgisayarlı görmede kullanılabilecek bir veri seti oluşturulabilir. Ve daha sonra oluşturulan bu veri seti ile konvolüsyonel sinir ağları modelleri uygulanarak PÖ göre daha başarılı sonuçlar elde etmek denenebilir. Bu konvolüsyonel sinir ağları GP ile harmanlanarak hibrit bir sistemin tasarlanması bu tez çalışmasından sonra yapılması planlanan çalışmalardan olacaktır.

7. KAYNAKLAR

- Artificial Intelligence Britannica, <http://britannica.com/>, Erişim tarihi: Mayıs 2019 [21]
- Andrej K. Cs231n convolutional neural networks for visual recognition.
<http://cs231n.github.io/>, Erişim tarihi Mayıs 2019 [27]
- Aykut B. Safak, E. Bostanci, A. E. Soylucicek, "Automated Maze Generation for Ms. Pac-Man Using Genetic Algorithms", International Journal of Machine Learning and Computing, vol. 6, no. 4, pp. 226-240, 2016[16]
- A. J. Abdellatif, B. McCollum and P. McMullan, "Serious Games Quality Characteristics Evaluation: The Case Study of Optimizing Robocode," 2018 International Symposium on Computers in Education (SIIE), Jerez, 2018, pp. 1-4[14]
- A. Sharma, K. Gupta, A. Kumar, A. Sharma and R. Kumar, "Model based path planning using Q-Learning," 2017 IEEE International Conference on Industrial Technology (ICIT), Toronto, ON, 2017, pp. 837-842. doi: 10.1109/ICIT.2017.7915468 [17]
- Baptiste DEGRYSE, Deep Q-Learning for Robocode, MSc thesis, Ecole polytechnique de Louvain, 2017 [11]
- Brambilla, M., Ferrante, E., Birattari, M. and Dorigo, M., "Swarm robotics: A review from the swarm engineering," Swarm Intell.7 (1), 1–41 (2013)[2]
- Dorri, A, Kanhere, S. and Jurdak, R, "Multi-Agent Systems: A survey", IEEE Access. 1-1.10.1109/ACCESS.2018.2831228, 2018[6]
- Eisenstein, Jacob. 2003. Evolving Robocode Tank Fighters. AI Memo 2003-023[32]
- E. Şahin, "Swarm robotics: from sources of inspiration to domains of application," in Swarm Robotics Workshop: State-of-the-Art Survey, E Şahin and W. Spears, Eds., Lecture Notes in Computer Science, no.3342, pp. 10–20, Berlin, Germany, 2005[1]
- Gade, M., Knudsen, M, et al., "Applying machine learning to robocode", Technical Report, 2003[12]
- Guzel, M. S. et al., "A Novel Framework for Multi-Agent Systems Using a Decentralized Strategy", Robotica, 37(4), pp. 691-707, (2019)[3]
- Harper, Robin. (2014). Evolving Robocode tanks for Evo Robocode. Genetic Programming and Evolvable Machines. 15. 10.1007/s10710-014-9224-2[10]
- Jiménez, A. C., García-Díaz, V. and Bolaños, S., "A decentralized framework for multi-agent robotic systems," Sensors 18(2),417, (2018)[5]

- Jin-Hyuk, H. and Sung-Bae, C, "Evolution of emergent behaviors for shooting game characters in Robocode," Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753), Portland, OR, USA, 2004, pp. 634-638 Vol.1[7]
- Karim, A. M., Güzel, M. S., Tolun, M. R., Kaya, H., and Çelebi, F. V., A New Generalized Deep Learning Framework Combining Sparse Autoencoder and Taguchi Method for Novel Data Classification and Processing, Mathematical Problems in Engineering, pp. 1–13, 2018[18]
- LeCun, Y., Bengio, Y. ve Hinton, G. 2015. Deep Learning. Nature, 521(7553), 436-444. [28]
- McCulloch, W. S. ve Pitts W. 1943. A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5(4), 115–133[24]
- Minsky, M. ve Papert, S. 1969. Perceptrons. M.I.T. Press Perceptrons [25]
- Nattharith, P., & Güzel, M. S. (2016). Machine vision and fuzzy logic-based navigation control of a goal-oriented mobile robot. Adaptive Behavior, 24(3), 168–180[9]
- Official Robocode Project Website, <https://robocode.sourceforge.io/>, Erişim tarihi: Mayıs 2019[19]
- Official Robocode Documentation Website, <http://robowiki.net>, Erişim tarihi: Mayıs 2019[20]
- Rodriguez-Angeles, A. and Vazquez, C. L., “Bio-inspired decentralized autonomous robot mobile navigation control for multi agent systems,” Kybernetika 54(1), 135–154 (2018)[4]
- Rumelhart, D. E., Hinton, G. E. ve Williams R. J. 1986. In Parallel Distributed Processing: Explorations in the Microstructure of Cognition. MIT, Foundations, vol. 1, 318-362[26]
- Sanjay, M S., Chirag, S. T. and Dharm, S, "Multimedia based fitness function optimization through evolutionary game learning", Emerging Trends in Networks and Computer Communications (ETNCC) 2011 International Conference on, pp. 164-168, 2011[8]
- Stanford University Unsupervised Feature Learning and Deep Learning Tutorial, <http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/>. Erişim tarihi Haziran 2019 [29]

V. Alaiba and A. Rotaru, "Agent architecture for building Robocode players with SWI-Prolog," 2008 International Multi conference on Computer Science and Information Technology, Wisia, 2008, pp. 3-7[15]

Wikipedia. Genetik Algoritmalar – Wikipedia.
https://en.wikipedia.org/wiki/Genetic_algorithm. Erişim tarihi Haziran 2019 [30]

Wikipedia. Markov decision process –
https://en.wikipedia.org/wiki/Markov_decision_process. Erişim tarihi Mayıs 2019[22]

Wikipedia. Turing Test - wikipedia. https://en.wikipedia.org/wiki/Turing_test. Erişim tarihi Mayıs 2019[23]

Y. Shichel, E. Ziserman, M. Sipper, "GP-Robocode: Using genetic programming to evolve robocode players" in Proc. 8th Eur. Conf. Genetic Program. Lect. Notes Compute. Sci., Germany, Heidelberg: Springer-Verlag, vol. 3447, pp. 143-154, 2005[13]

ÖZGEÇMİŞ

Adı Soyadı: Ümit Ulusoy

E-mail: umituluso@yandex.com

Doğum Tarihi: 06.11.1992

Medeni Hali: Evli

Eğitim Durumu

Lise : Ayyıldız Anadolu Lisesi (2010)

Lisans : Ankara Üniversitesi Bilgisayar Mühendisliği-İngilizce (2015)

Yüksek Lisans : Ankara Üniversitesi Fen Bilimleri Enstitüsü

Bilgisayar Mühendisliği Bölümü (Eylül 2016–Ekim 2019)

Çalışılan Kurumlar

Enocta Avez A.Ş., 2015 Ağustos – 2016 Şubat

Havelsan A.Ş., 2016 Şubat – 2018 Haziran

Comodo, 2018 Haziran – 2019 Şubat

Atlassian(OpsGenie), 2019 Şubat – Devam ediyor