Pac-Man Yapay Zeka Projesi

Arda Ege İsker
Department of Computer Engineering
TOBB University of Economics and Technology
Ankara, Turkey
Email: aisker@etu.edu.tr

***Öz*—Bu yazıda izlenen genel amaç, oyun ve simülasyonlar hakkında daha fazla bilgi edinmek olup yapay zekanın oyunlara nasıl uygulandığını, bu ortamda geliştirilebilecek uygulamalar için yol haritası olmasıdır. Yapay zekayı oyunlarda uygulamanın avantajları ve dezavantajları, performans karşılaştırılmaları sonucu Pacman oyunu için uygun olan yapay zeka algoritmasının seçimi olacaktır.**

Anahtar Kelimeler—Arama ağaçları, refleks aracısı, sezgisel arama, Oyunlarda arama, olasılık, Minimax, Expectimax, a-ß budaması, değerlendirme fonksiyonları

I.Giriş

Oyunlar, uzun süredir yapay zeka ve makine öğrenmesi konuları için araştırma ve test etme ortamı olmuşardır. Pac-Man Namco tarafından 1980 yılında yapılmış bir oyundur. Pac-Man'de oyuncu, bir labirent içerisinde hareket ederek sarı diskleri bitirmeye çalışır. Hedefi hayalet ve canavarlardan kaçarak tüm küçük diskleri toplamak olan oyuncu, tüm diskleri topladığında diğer aşamaya geçer. Labirent üzerinde beliren meyveleri toplamak oyuncuya fazladan puan kazandırır. Büyük sarı diskleri aldığında, canavar ve hayaletler maviye dönüşür ve bir süreliğine yenilebilir duruma gelirler[2]. Pac-Man,gerçek zamanlı ve deterministik olmayan bir oyundur: Düşmanların hareketleri rastgele faktörler içerir, bu nedenle oyun daha önce olan durumların hafızada tutularak oynanamaz. Geçmişten günümüze Pac-Man aracıları için genetik algoritmalar, pekiştirmeli öğrenme, yapay sinir ağları ve bulanık sistemler gibi teknikler uygulanmış, fakat elle yazılmış kural tabanlı sistemlerin daha yüksek puanlar aldıkları gözlemlenmiştir. Bu nedenle Pac-Man üstüne gerçekleştirilen önceki araştırmalarda amaç, oyunu yalnızca araştırma için bir platform olarak kullanmaktır. Bu projede [5] ise amaç oyun için en iyi sonucu verecek yapay zekanın gerçekleştirilmesidir. Bu projede yapay zeka dört aksiyon bileşeninden oluşmaktadır. Bunlar: (1) oyun ekran durumunu yakalamak, (2) oyun objelerini tanımak, (3) Pac-Man'in gideceği yönün seçilmesi, (4) Bu yöne uygun yön tuşunun aktif edilmesidir. Oyunun detaylarına dair Appendix:

A1. Oyun, saniyede ekrana 60 kare basar. Örneğin kare başına 16ms gecikme vardır. Böylelikle oyundaki en ufak bir değişikik için de olsa yapay zekanın çalışma süresi 16ms den düşük olmalıdır.

A2. Oyunda yapay zekanın performansının test edilmesi için 10 adet farklı labirent vardır. Bu labirentler arasındaki temel fark labirent boyutu ve labirent içerisindeki hareket kısıtıdır.

A3. Oyunda sayısı haritalara göre değişiklik gösteren güç pilleri vardır. Pac-Man aracısı bu pili yediği zaman kısa süreliğine hayaletleri yiyebilir.

A4. Oyundaki puanlama sistemi iki unsurdan oluşmaktadır. Pac-Man aracısı yem yedikçe puanı artar, fakat oyunda geçirilen süre başına puan eksilmektedir.

II. LİTERATÜR ARAŞTIRMASI

Bu kısımda daha önceden Pac-Man ve türevleri üzerine yapılan YZ ve makine öğrenmesi araştırmaları yer alacaktır. Tablo-1'de görüldüğü üzere literatürde bu alanda yapılan bazı araştırmalar, bu araştırmalarda kullanılan metodlar ve bu metodlar sonucu alınan skorlar verilmiştir.

Ref.	Scores		Methods	Categories
	Average	Highest		
[Mehrabian 2007]		20,640	Hand-coded rules	Rule based
[Jabbary 2007]	1,181	1,830		
[Wirth 2007]	1,673	3,370		
[Wirth 2008a]	2,510	4,360		
[Wirth 2008b]				
[Fitzgerald 2008]	11,167	15,970		
[Fitzgerald 2009]	11,661	16,840		
[Chan 2009]	12,634	23,740		
[Kwong 2009]	2,641	5,250		
[Li 2009]	5,947	9,000		
[Sojoodi 2010]	5,933	9,990		
[Handa 2007]	1,751	2,270	Hand-coded rules, parameters are evolved	
[Handa 2008]	9,377	12,930		
[Gallagher 2003]				
[Szita 2007]			Rules are generated by Cross Entropy method	
[Flensbak 2008]	7,371	11,990	Simple search	Search methods
[Robles 2009]	7,664	15,640		
[Kelly 2009]	5,676	8,740		
[Bell 2010]	10,812	14,660		
[Jeong 2010]	8,545	18,690		
[Butz 2010]	4,478	5,790		
[Kashifuji 2008]	4,694	6,390	A* search, parameters are evolved	
[Matsumoto 2009a]	13,059	24,640		
[Matsumoto 2009b]	17,102	30,010		
[Ashida 2010]	15,353	20,580		
[Gan 2008]	11,981	15,400	Game tree search	
[Emilio 2010]	9,343	21,250	Ant Colonies	
[Tong 2010]	5,702	10,820	Monte-Carlo simulation	
[Lucas 2005]			Neural net, connection weights are evolved	Learning algorithms
[Gallagher 2007]			Neural net with raw input, connection weights are evolved	
[Piatkowski 2008]	4,130	7,190	Neural net	
[Bonet 1999]			Incremental reinforcement	
[Ando 2008]	2,493	5,520	Support Vector Machine	

Tablo 1: Literatürdeki yapay zeka makalelerinin içerikleri

İlk kategoride kural tabanlı araştırmalar yer almaktadır. Elle yazılmış kurallar, yapay zeka tasarımı için en temel yöntemlerden biridir. Bu metodu uygulayan uygulamalarda metod birbirinden farklılık gösterebilir. Örneğin Alan Fitzgerald, Peter Kemeraitis ve Clare Bates Congdon (2008,2009) karar mekanizmasını oyun durumu, yakındaki hayaletlerin sayısı, yem sayısı, güçlü yem sayısı, yenilebilecek hayalet sayısı, vb. niteliklere göre verir [3]. Bu projede kullanılan değerlendirme fonksiyonları bu karar mekanizmalarına örnek sayılır. Ardından elle yazılmış kurallara göre karar mekanizmaları çalışır (örneğin çok yakın hayalet sayısı 1'den büyük eşitse ve güçlü pil sayısı 1'den büyük eşitse güçlü pile git).
Kural tabanlı metodları geliştirmek adına parametreleri optimize etmek için evrimsel metodlar kullanılmıştır. Örneğin Marcus Gallagher ve Amanda Ryan (2003) Pac-Man'in gidebileceği her yöne o yöndeki düşmanların sayısına göre bir olasılık ataması yapmıştır [4].
İkinci kategoride arama algoritmaları vardır. Basit arama algoritmaları, Pac-Man'in olduğu noktadan gidilebilecek tüm yönleri genişletmeye yönelik arama ağacı kuran metodlardır.
Basit arama ağaçlarını geliştirmek adına sezgisel arama yöntemleri kullanılmıştır. Örneğin Matsumoto A* arama metodu ile hedef noktaya giden minimal maliyetli yollar hesaplamıştır.
Karınca kolonisi ve Monte-Carlo simülasyonları da arama metodu olarak kabul edilebilir.
Üçüncü kategoride yapay sinir ağları ve destek vektör makinesi öğrenme algoritmaları vardır. Örneğin Simon M. Lucas (2005) tek gizli katmanlı ileri beslemeli yapay sinir ağı geliştirmiştir.
Bu projede ise üç yapay zeka uygulaması kullanılacaktır.
Bunlar: Basit refleks aracısı, Minimax aracısı ve Expectimax aracısıdır.

III. METODOLOJİ

Proje geliştirme ortamı olarak Python 2 kullanılmıştır. Hazır kullanıcı arayüzüne sahip Pac-Man projesi üstüne yapay zeka uygulaması geliştirilmiştir. Oyun çalıştırılırken aracı tipini, oynanacak oyun sayısını, harita seçimini, hayaletlerin sayısını ve oyunun hızını komut satırı argümanı olarak alır.

Argümanlar:

-n: Oynanacak toplam oyun sayısı. (Örn: -n 50)

-l: Oyunun oynanacağı haritanın seçilmesi (Örn: -l smallClassic) (argüman verilmezse mediumClassic haritası seçilir)

-p: Oyunu oynayacak yapay zeka aracısının seçimi (Örn: -p ReflexAgent, -p MinimaxAgent , -p AlphaBetaAgent, -p ExpectimaxAgent)

-t: Bu argüman verilirse sadece oyunun sonucunun çıktısı verilir, oyunun oynandığı gözlemlenemez.

-g: Hayalet aracılarının seçimi (Örn: -g RandomGhost)

-k: Oyundaki hayalet sayısı (Argüman verilmezse 4 hayalet ile oynanır.)

--frametime: Kareler arasındaki gecikme (Argüman verilmezse 0.1 ms gecikme kullanılır.)

-a depth: Ağaç araması içeren aracılar için maksimum derinlik (Örn: -a depth=4)

- a evalFn: Kullanılacak değerlendirme fonksiyonunun seçimi (Örn -a evalFn=better) (Argüman verilmezse normal değerlendirme fonksiyonu kullanılır.)

Manhattan Mesafesi:

Projede oyunun objelerin arasındaki mesafenin ölçülmesi için

$$|x_1 - x_2| + |y_1 - y_2|$$

Şekil 1. Manhattan mesafesi formülü

IV. KULLANILAN ARACILAR VE DEĞERLENDİRME FONKSİYONU

Raporun bu kısmında projede kullanılan aracılar tanıtılacaktır.

Refleks Aracısı:

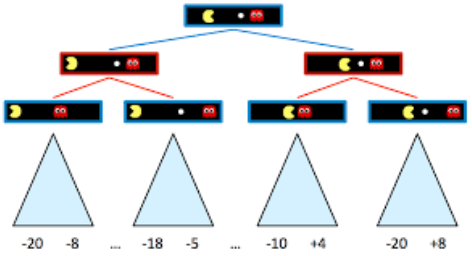
Bu aracı, bir değerlendirme fonksiyonu sonucu yapabileceği hareketler arasından en yüksek değerlendirme skoruna sahip hareketi seçer.

```
def evaluationFunction(self, currentGameState, action):
    successorGameState =
currentGameState.generatePacmanSuccessor(action)
    newPos = successorGameState.getPacmanPosition()
    newFood = successorGameState.getFood()
    newGhostStates = successorGameState.getGhostStates()
    newScaredTimes = [ghostState.scaredTimer for
ghostState in newGhostStates]
    #print newPos
    #print newGhostStates[0]
    foodPos = newFood.asList()
    foodCount = len(foodPos)
    closestDistance = 1e6
    for i in xrange(foodCount):
      distance = manhattanDistance(foodPos[i],newPos) +
foodCount*100
      if distance < closestDistance:
        closestDistance = distance
        closestFood = foodPos
    if foodCount == 0 :
      closestDistance = 0
    score = -closestDistance

    for i in xrange(len(newGhostStates)):
      ghostPos = successorGameState.getGhostPosition(i+1)
      if manhattanDistance(newPos,ghostPos)<=1 :
        score -= 1e6

    return score #successorGameState.getScore()
```

Görüldüğü üzere değerlendirme fonksiyonu yakınlardaki hayalet sayısı, yem sayısı, korkmuş hayalet sayısı parametrelerine göre hesaplanır.

Minimax Aracısı:

Bu aracı, Pac-Man ve hayalet aracıları arasında bir Minimax ağacı üstünde aramalar gerçekleştirir. Pacman = max, hayalet = min olacak şekilde gidilecek yön karar verilir.



Şekil 2. Minimax ağacı temsili.

```
def getAction(self, gameState):
    numAgent = gameState.getNumAgents()
    ActionScore = []
    def _rmStop(List):
      return [x for x in List if x != 'Stop']
    def _miniMax(s, iterCount):
      if iterCount >= self.depth*numAgent or s.isWin() or
s.isLose():
        return self.evaluationFunction(s)
      if iterCount%numAgent != 0: #Ghost min
        result = 1e10
        for a in
_rmStop(s.getLegalActions(iterCount%numAgent)):
          sdot =
s.generateSuccessor(iterCount%numAgent,a)
          result = min(result, _miniMax(sdot,
iterCount+1))
        return result
      else: # Pacman Max
        result = -1e10
        for a in
_rmStop(s.getLegalActions(iterCount%numAgent)):
          sdot =
s.generateSuccessor(iterCount%numAgent,a)
          result = max(result, _miniMax(sdot,
iterCount+1))
          if iterCount == 0:
            ActionScore.append(result)
        return result

    result = _miniMax(gameState, 0);
    return
_rmStop(gameState.getLegalActions(0))[ActionScore.index(m
ax(ActionScore))]
```

Her düğümün ağırlığı, değerlendirme fonksiyonu skoruna göre olup, Minimax algoritması uygulanarak gidilecek yön hesaplanmıştır. Ağaçta aranacak derinlik oyuna argüman olarak verilebilir.

a-ß Budaması:

Bu aracı, Minimax ağacı üstünde a-ß budaması yaparak keşfedilmesi oyunun başarımı açısından anlamsız olan düğümlerin keşfedilmesini engellemeye yarıyor.

```
def _alphaBeta(s, iterCount, alpha, beta):
  if iterCount >= self.depth*numAgent or s.isWin() or
s.isLose():
    return self.evaluationFunction(s)
  if iterCount%numAgent != 0: #Ghost min
    result = 1e10
    for a in
_rmStop(s.getLegalActions(iterCount%numAgent)):
      sdot = s.generateSuccessor(iterCount%numAgent,a)
      result = min(result, _alphaBeta(sdot, iterCount+1,
alpha, beta))
      beta = min(beta, result)
      if beta < alpha:
        break
    return result
  else: # Pacman Max
    result = -1e10
    for a in
_rmStop(s.getLegalActions(iterCount%numAgent)):
      sdot = s.generateSuccessor(iterCount%numAgent,a)
      result = max(result, _alphaBeta(sdot, iterCount+1,
alpha, beta))
      alpha = max(alpha, result)
      if iterCount == 0:
        ActionScore.append(result)
      if beta < alpha:
        break
    return result
result = _alphaBeta(gameState, 0, -1e20, 1e20)
return
_rmStop(gameState.getLegalActions(0))[ActionScore.index(m
ax(ActionScore))]
```

Görüldüğü üzere her katman için düğümlere birer alfa ve beta değerleri atanıyor. Daha sonra Minimax ağacı düğümleri üzerinde a-ß budaması algoritması çalışıyor. Eğer alfa > beta durumu sağlanırsa o düğümün çocuk düğümleri keşfedilmeden döngü kırılıyor.

Expectimax Aracısı:

Bu aracı, hayaletlerin yapabileceği hareketlerin olasılığına göre kendisi için en uygun hareketi hesaplayan aracıdır.

```
def getAction(self, gameState):
    numAgent = gameState.getNumAgents()
    ActionScore = []
    def _rmStop(List):
      return [x for x in List if x != 'Stop']
    def _expectMinimax(s, iterCount):
      if iterCount >= self.depth*numAgent or s.isWin() or
s.isLose():
        return self.evaluationFunction(s)
      if iterCount%numAgent != 0: #Ghost min
        successorScore = []
        for a in
_rmStop(s.getLegalActions(iterCount%numAgent)):
          sdot =
s.generateSuccessor(iterCount%numAgent,a)
          result = _expectMinimax(sdot, iterCount+1)
          successorScore.append(result)
        averageScore = sum([ float(x)/len(successorScore)
for x in successorScore])
        return averageScore
      else: # Pacman Max
        result = -1e10
        for a in
_rmStop(s.getLegalActions(iterCount%numAgent)):
          sdot =
s.generateSuccessor(iterCount%numAgent,a)
          result = max(result, _expectMinimax(sdot,
iterCount+1))
          if iterCount == 0:
            ActionScore.append(result)
        return result

    result = _expectMinimax(gameState, 0);
    return
_rmStop(gameState.getLegalActions(0))[ActionScore.index(m
ax(ActionScore))]
```

ExpectMinimax metodu ile hayalet düğümlerinin şans düğümü değeri hesaplanır ve buna göre Pac-Man aracısı için uygun hareket yönü seçilir.

Daha iyi değerlendirme fonksiyonu:

Bu fonksiyonda değerlendirme skoru dört duruma göre değişiklik gösterir. Bunlar oyun durumu, yemler, hayaletler ve güçlü yemlerdir.

```
def _scoreFromGhost(gameState):
  score = 0
  for ghost in gameState.getGhostStates():
    disGhost =
manhattanDistance(gameState.getPacmanPosition(),
ghost.getPosition())
    if ghost.scaredTimer > 0:
      score += pow(max(8 - disGhost, 0), 2)
    else:
      score -= pow(max(7 - disGhost, 0), 2)
  return score

def _scoreFromFood(gameState):
  disFood = []
  for food in gameState.getFood().asList():

disFood.append(1.0/manhattanDistance(gameState.getPacmanP
osition(), food))
  if len(disFood)>0:
    return max(disFood)
  else:
    return 0

def _scoreFromCapsules(gameState):
  score = []
  for Cap in gameState.getCapsules():

score.append(50.0/manhattanDistance(gameState.getPacmanPo
sition(), Cap))
  if len(score) > 0:
    return max(score)
  else:
    return 0

def _suicide(gameState):
  score = 0
  disGhost = 1e6
  for ghost in gameState.getGhostStates():
    disGhost =
min(manhattanDistance(gameState.getPacmanPosition(),
ghost.getPosition()), disGhost)
  score -= pow(disGhost, 2)
  if gameState.isLose():
    score = 1e6
  return score

score = currentGameState.getScore()
scoreGhosts = _scoreFromGhost(currentGameState)
scoreFood = _scoreFromFood(currentGameState)
scoreCapsules = _scoreFromCapsules(currentGameState)
#if score < 800 and currentGameState.getNumFood() <= 1
and len(currentGameState.getCapsules()) == 0:
#  return _suicide(currentGameState)
#else:
return score + scoreGhosts + scoreFood + scoreCapsules
```

scoreFromFood, en yakın yem sayısına göre bir skor döner.

scoreFromGhost, eğer hayalet korkmuş ise Pac-Man aracısının onu kovalamasına yönelik, eğer korkmamış bir hayalet ise ondan kaçmasına yönelik bir skor döner.

scoreFromCapsules, en az Manhattan mesafesine sahip kapsüllerin uzaklığına göre bir skor döner.

V. Test Sonuçları

Aracıların başarımını ölçmek için birbirleri ile karşılaştırmalı başarım testi uygulancaktır.
Test ortamında harita mediumClassic haritası seçilmiştir.
Testin sağlıklı sonuç verebilmesi için her aracı 50 adet oyun oynayacaktır ve sonuçlar karşılaştırılacaktır.
Yürütme konfigürasyonu:

pacman.py --frameTime 0 -p <aracıTipi> -n 50 -q

	Ortalama Skor	Ortalama Kazanma Skoru	Kazanma Oranı
Refleks Aracısı	981.98	1326.05	37/50 (%74)
Minimax Aracısı	-302.95	576.52	5/50 (%10)
a-ß Aracısı	-15.75	1436.50	5/50 (%10)
Expectimax Aracısı	-102.45	782.54	4/50(%8)

Test sonuçlarından görüldüğü üzere, en iyi sonucu Refleks aracıları veriyor. Bunun bir çok sebebi olabilir. Meseka en öncelikli sebebi, oyundaki hayaletlerinin hareketinin rastgele oluyor olması olduğunu düşünüyorum. Çünkü Minimax ve benzeri aracılarda ağaç üzerinde işlem yapılırken karşı tarafın hep optimal hareket edeceği varsayılıyor. Fakat hayaletlerin hareketleri rassal olduğu zaman güzel bir sonuç elde edemiyorlar. Refleks aracıları ise bu konfigürasyon için güzel sonuç vermiş olsa da daha büyük haritalarda ve hayaletlerin rassal değil sonuca yönelik hareket ettiği konfigürasyonlarda bu kadar iyi sonuç vereceğini düşünmüyorum. Fakat Minimax tabanlı aracıların daha iyi sonuç vermesini istiyorsak, arama derinliğini artırmamız gerekiyor.
Arama derinliğini artırıp tekrar test edince;

Yürütme konfigürasyonu:

python pacman.py -p <aracıTipi> -l -a depth=4 -q -n 50

	Ortalama Skor	Ortalama Kazanma Skoru	Kazanma Oranı
Refleks Aracısı	977.98	1336.21	34/50 (%68)
Minimax Aracısı	842.21	1129.23	27/50 (%54)
a-ß Aracısı	1052.93	1512.50	30/50 (%60)
Expectimax Aracısı	949.25	1348.45	26/50(%52)

Görüldüğü üzere Minimax ağacındaki arama derinliği arttığı zaman Minimax ve benzeri aracılarda gözle

görülür bir başarım artışı oldu. Bunun sebebi, Minimax algoritmalarının az derinlikte arama yaptığı zaman istenen hedefe uygun çözümler getiremiyor olmasıdır. Eğer oyun çok büyük bir harita ölçeğinde güçlü bir sistemde Minimax ve benzeri aracılarda çalıştırılırsa Refleks aracılarından daha iyi sonuç vereceğini tahmin ediyorum.

VI. KAZANIMLAR

Bu proje, yapay zekanın oyunlarda kullanım yöntemini anlamamı sağladı. Bu projede pratik kazandığım için non-deterministic sistemlerde puanlandırma, olasılık hesabı ve tahmin edilebilirliği kullanarak gerçek hayat problemlerine yapay zeka çözümleri geliştirmeye olan ilgim arttı.

Proje kapsamında arama algoritmasının teoride ne kadar verimli olduğu değil uygulanan probleme ne kadar uygun olduğunun önemli olduğunu fark ettim.

Ağaç yapısı üstünde traverse etmek, durum makineleri gibi kavramlar üstünde pratik yaparak kariyerimde karşılaşabileceğim bu gibi problemlerin olası çözümleri için fikir edinmiş oldum. Proje kapsamında çeşitli arama algoritmaları implement edilerek benim kullandığım algoritmalar ile performans açısından kıyaslanabilir.

Markov Decision Processes & Reinforcement Learning kullanılarak Value Iteration, Q-learning, Approximate Q-learning , Deep learning modelleri [6] de uygulanabilir.Ayrıca rakiple oynanan oyunlarda yapay zeka uygulamalarında rakibin davranış şeklinin kullanılan algoritmanın başarımına olan etkisinin ne kadar fazla olduğunu görmüş oldum.

Github proje bağlantısı:
https://github.com/ardaisker/PacmanAI

REFERANSLAR

[1] https://www.researchgate.net/publication/339680476_Role_of_AI_in_Gaming_and_Simulation

[2] https://tr.wikipedia.org/wiki/Pac-Man

[3] Ando, J., Shirakawa, S., Otsuka, J., Takahashi, R., Totsuka, Y., and Nagao, T. (2008). Active Control of Ms.PacMan using SVM learning with human playing data, WCCI 2008 Ms Pacman Competition Entry

[4] Ashida, T., Miyama, T., Matsumoto, H., and Thawonmas, R. (2010). CIG 2010

[5] https://github.com/ardaisker/PacmanAI

[6] https://towardsdatascience.com/reinforced-pac-man-8e51409f4fc