Bilgili ve Bilgisiz Arama Temelli PACMAN Oyunu

BM455 – YAPAY ZEKAYA GİRİŞ

Aralık 2020

Uygulama Ödevi - 1

161180038

Hilal Ilgaz

GAZİ ÜNİVERSİTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

Bölüm – 1

Uygulama ödevi, PyCharm editörü üzerinde Python 2.7 dili ile gerçekleştirilmiştir. Python 2.7 için Anaconda Navigator üzerinde py2 ismi verilen bir environment kurulmuştur. PyCharm üzerinde interpreter ayarları düzenlenerek conda Python 2.7 seçilmiştir. Arayüz için ekstra bir şey yapılmamıştır. Kaliforniya Üniversitesi'nin ilgili ödeve ait web sayfasında paylaşılan layoutlar projeye eklenmiş ve arayüz olarak labirenti oluşturmak için kullanılmıştır [2]. Uygulama ödevi, PyCharm üzerinde terminalden çalıştırılabilir durumda teslim edilmiştir.

Python 3.8 yerine 2.7 seçilmesinin nedeni ödev dosyasında yer alan web sitesinin baz olarak kullanılmış olmasıdır. Eğer Python 3.8 veya türevlerine ait bir environment üzerinde çalıştırılırsa kod çalışmaz.

Bölüm – 2

Uygulama ödevi, bilgili ve bilgisiz arama temelli Pacman Oyunu gerçekleştirimidir. Pac-Man ya da Türkçe piyasa adıyla Dobişko, Namco tarafından yapılmış bir oyundur. 1980 yılında çıkmış ve kısa sürede popüler bir oyun olmuştur.

Pac-Man yahut Türkçe piyasa adıyla Dobişko, Namco tarafınca yapılma bir oyundur. 1980 senesinde çıkmış ve kısa müddette popüler bir oyun olmuştur.

Pac-Man'de oyuncu, bir labirent içinde hareket ederek sarı diskleri bitirmeye çalışır. Hedefi hayalet ve canavarlardan kaçarak bütün minik diskleri toplamak olan oyuncu, bütün diskleri topladığında öteki aşamaya geçer. Labirent üstünde beliren meyveleri toplamak oyuncuya ekstra puan kazandırır. Büyük sarı diskleri aldığında, canavar ve hayaletler maviye dönüşür ve bir süreliğine yenilebilir duruma gelirler. Oyunun Atari 2600 kartuşları Türkiye'de Dobişko adıyla piyasa sürülmüştür. Oyun Japonya'da Puck-Man olarak yayımlanmış olsa da birtakım vandalların "P" harfini "F" ile değiştirerek sövgü oluşturması sebebiyle oyun Amerika'da Pac-Man olarak yayımlandı. Google 21 Mayıs 2010'da bir jest yaparak logosunda Pac-Man oynatmıştır [1].

Ödev kapsamında PacMan canavarının yol bulabilme, belirli bir konuma ulaşabilme, yiyecekleri(noktaları) toplayabilmesi gerekmektedir. Bu işlemler için arama algoritmaları kullanılacaktır. Kullanılan arama algoritmaları:

- Genişlik Öncelikli Arama / Breadth First Search
- Derinlik Öncelikli Arama / Depth First Search
- Düşük Maliyetli Arama / Uniform Cost Search

- A*

Bölüm – 3

Proje kapsamında kullanılan arama algoritmalarının uygulanabilmesi için dört çeşit veri yapısı kullanılmıştır. Bu veri yapıları; stack(yığın), queue(kuyruk), priority queue(öncelikli kuyruk) ve heap. Kod içinde bu veri yapıları İngilizce isimleriyle kullanılmıştır. Bu yapılar util dosyasında kodlanmış ve search dosyasında arama algoritmaları oluşturulurken kullanılmıştır.

Queue veri yapısı genişlik öncelikli arama algoritmasının uygulanmasında kullanılmıştır. Bir node seçilir ve sırayla komşuları gezilir. Seçilen node'a ait gezilebilecek komşu kalmadığında queue içerisindeki ilk node alınır ve onun komşuları ziyaret edilerek aynı işlemler queue boşalana kadar devam eder. Kısaca algoritma klasik bir enqueue ve dequeue mantığıyla node ekleyip çıkararak derinlemesine gezer ve birikme yapmaz.

Stack veri yapısı derinlik öncelikli arama algoritmasında kullanılmıştır. Bu algoritma için stack yapısının tercih edilmesinin nedeni stack içinden sürekli eleman alınarak bütün nodeların gezilebilmesidir. Ayrıca queue yapısının aksine stack yapısında sürekli üstten ekleme ve çıkarma yapıldığından ilerleme derinlemesine yapılır.

Priority queue ve heap yapıları düşük maliyetli arama ve A* arama algoritmalarının uygulanmasında kullanılmıştır. Priority queue veri yapısı, heap önceliklendirilmiş queue yapılarını uygulamak için kullanılır. Projede 'util'dosyasında mport edilen heapq modülü min heap mantığıyla çalışır. Yani her döndüğünde en küçük heap node u atılır. Yapı içerisindeki elementler push ve pop komutları ile manipüle edildiğinde yapı korunur. Bu iki algoritmada özel bir yapı kullanılmasının nedeni önceliklendirilme gerekmesidir. Düşük maliyetli arama algoritmasında göz ardı edilemeyecek bir maliyet faktörü söz konusudur. Bu sebeple kullanılmıştır. A* arama algoritmasında ise sezgisel bir diğer deyişle heuristic olarak adlandırılan özel bir durum söz konusudur.

Proje kapsamında bulunan sınıflar;

- layout dosyası labirent için desen okuması ve bunlara dair içeriklerin depolanması içindir.
- keyboardAgents dosyası Pacman'in klavyeden kontrolü için arayüz gibi düşünülebilir.
- ghostAgents dosyası hayaletleri kontrol eden ajanlar içindir.

- textDisplay dosyası ASCII, Pacman grafikleri içindir.
- graphicsUtils dosyası Pacman grafiklerini desteklemek içindir. Program çalıştırıldığında gözüken arayüzün rengi çerçevesi gibi faktörler bu kısımda yer alır.
- graphicsDisplay dosyası Pacman grafikleri içindir.
- pacmanAgents dosyası klasik Pacman oyununu kontrol eden ajanlar içindir.
- util dosyası arama algoritmalarının uygulanmasında kullanılan veri yapılarını içerir.
- game dosyası Pacman dünyasının nasıl çalıştığına dair mantığı içerir.
- pacman dosyası oyunu yürüten dosyadır. Main fonksiyon bu dosyanın içinde yer alır. Dolayısıyla terminalde kod çalıştırılmaya çalışılırken mutlaka import edilmesi gerekir.
- searchAgents dosyası arama bazlı ajanları içerir. Bu dosyanın içindeki heuristic ve çeşitli problem adı altındaki sınıflar algoritmaların çalışması için önemlidir.
- search dosyası arama algoritmalarını içerir. Bu dosya ile util dosyasının içeriğinden veri yapılarının tanıtımı yapılırken değinilmiştir.

searchAgents ve search dışındaki dosyalar Kaliforniya Üniversitesi'nin ilgili dersine ait web sayfasından alınmıştır [2]. search dosyasında bulunan önemli metotlar;

- derinlikOncelikliArama(problem), bu metot isminden de anlaşılacağı üzere derinlik öncelikli arama için yazılmıştır. Derinlik öncelikli arama mantığım bütün nodeları graf gibi düşünerek gezerken stack veri yapısına ait komutları basit bir şekilde kullanmak üstüne kuruludur. Kodu üstüne kurduğum algoritma Prof. Dr. Suat Özdemir'e ait Algoritmalar Ders notunda bulunan derinlik öncelikli arama konusunda bahsedilen sözde koddur [3]. Bu sözde kod birebir uygulanmamıştır. Arama algoritmasının mantığının kurulmasında kullanılmıştır. Söz konusu sözde kod aşağıda bulunmaktadır.

```
DOA(G,s)
for each vertex u
        visited[u]=false;
push.stack(s);
while(stack is not empty)do
        u=pop.stack();
        if(visited[u]=false)then
```

```
visited[u]=true;
for each unvisited neighbour v of u
        push.stack(v)
```

- genislikOncelikliArama(problem), bu metot isminden de anlaşılacağı üzere genişlik öncelikli arama için yazılmıştır. Derinlik öncelikli arama metodunu yazarken kurulan mantığın stack yerine queue yapısına uygulanmış hali kullanılmıştır. Derinleme yerine genişlemesine node gezilmesi sağlanmıştır.
- dusukMaliyetliArama(problem), bu metot isminden de anlaşılacağı üzere düşük maliyetli arama için yazılmıştır. En düşük maliyetli node ile başlanır. Başlangıçta maliyet 0 olarak atanır ve nodelar ziyaret edilirken maliyet artar. Düşük maliyetli arama için kullanılan sözde kod aşağıdaki şekildedir [4].

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier ← a priority queue ordered by PATH-COST, with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)  /* chooses the lowest-cost node in frontier */
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        frontier ← INSERT(child, frontier)
      else if child.STATE is in frontier with higher PATH-COST then
        replace that frontier node with child
```

- nullHeuristic(state,problem=None), bu metot default heuristic metodu olarak tanımlanmıştır. Herhangi bir heuristic metot çağrılmazsa kullanılacaktır.
- aStarArama(problem, heuristic=nullHeuristic), bu metot isminden anlaşılacağı üzere A* arama için yazılmıştır. Hedef node ve heuristic node'ların hesaplanması gerektiği şekilde çalışmaktadır. Her adımda en düşük değere sahip düğümü alır. Bu node'a komşu olan node'ların değerleri güncellenir ve bu çerçevede maliyet hesabı yapılıp sezgisel fonksiyon uygulanılır.

Bu metotlar birbirine benzer şekilde yazılmış fazladan parametreye ihtiyaç duyan metotlara ekleme yapılmıştır. Nodelar arasında gezme aşamasının kontrolü while(True) döngüsüyle sağlanmıştır. Tüm fonksiyonlarda ortak olmak üzere veri yapısının boş olup olmadığı kontrol edilmiştir. Tüm nodeların gezildiğinden emin olmadan arama algoritmasını sonlandırmamak için son aşamada hala hareket olup olmadığı kontrol edilmektedir. Hareket varsa path içerisine eklenir ve altındaki nodelara bakabilmek için ebeveyn olarak atanır. Hareket yoksa yolun son halini döndürür. isGoalState() metodu yardımıyla problemin başlangıç aşamasındayken Pacman'in doğduğu yerin hedef yeri olup olmadığını kontrol eder, hedef yeriyse program durur, değilse devam eder. Böylece kontrol sağlanmış olur.

searchAgents dosyasındaki önemli sınıflar ve metotlar;

- SearchAgent(Agent) sınıfı, Genel kullanıma yönelik bir ajandır. Arama yapılmak istenen problem için gerekli algoritma ile bağlantı kurar. Default olarak derinlik öncelikli arama için çalışacaktır. Default arama dışında bir metot kullanmak için fn ile belirtilmelidir. SearchAgent içinde değişiklik yapılmamalıdır. Bu kısım John DeNero ve Dan Klein tarafından yazılmıştır [2].
- PositionSearchProblem(search.SearchProblem) sınıfı, posizyon arama problemi içindir. Bir arama problemi durum uzayını, başlangıç durumunu, hedef testini, successor işlevi ve maliyet işlevini içerir. Bu arama problemi, Pacman panosundaki belirli bir noktaya giden yolları bulmak için kullanılabilir. Durum uzayı, oyunda (x,y) konumlarından oluşur [2].
- manhattanHeuristic() ve euclideanHeuristic() metotları A* arama algoritmasının sezgisel fonksiyonları için yazılmıştır. Manhattan uzaklığını ve Öklid uzaklığını hesaplar.
- CornersProblem(search.SearchProblem) sınıfı, dört köşe problemi için üstünde çalışılacak layout'da gezinme işlemi için tasarlandı. Sınıfın constructor'ında problem için köşelerin durumu belirtilmiş ve başlangıç durumu tanımlanmıştır. getSuccessors(self,state) metodunda Pacman canavarının duvarlara vurma durumunda veya vurmama durumunda sonraki adım planlanmıştır. Duvarlara vurmadığı sürece dört köşeyi gezdiğinden emin olunmaya çalışılmıştır. getCostOfActions(self,actions) metodu, hareketlerin maliyetini döndürür. Geçerli olmayan hareket yapılırsa 999999 döndürür. cornersHeuristic(state,problem) metodu, köşe problemi için sezgisel fonksiyondur. Sezgisel başlangıçta sıfır alınır. Daha sonra ziyaret edilmeyen köşeler ve diziye eklenir. Manhattan uzaklık formülasyonu kullanılarak mesafe hesaplanır. Mesafe

sezgiselden büyük olduğu sürece sezgisel mesafeye eşitlenir ve bu durum köşelerin sayısını problemdeki köşelerin uzunluğuna eşitlenene kadar devam eder.

- AStarCornersAgent(SearchAgent) sınıfı, FoodSearchProblem için A* ve foodHeuristic metodunu kullanan SearchAgent sınıfından türetilmiş ajan sınıfıdır [2].
- FoodSearchProblem sınıfı, Pacman oyunun mantığı olan tüm noktaları toplama ile ilgili problemin çözümü içindir. Noktalar oyunda yiyecek anlamına gelmektedir. Game dosyası içinde bir Grid sınıfı vardır. Bu sınıfa bağlı olarak foodGrid oluşturulur ve bu foodGrid için kalan yiyeceği belirten True ya da False değer döndürür. foodHeuristic(state,problem) metodu bu sorunun A* arama algoritması ile çalışırken kullanılacak çözümü için bir sezgisel fonksiyondur. Grid sınıfının içinden foodGrid değişkeni listeye dönüştürülerek alır. Sezgisel başlangıçta 0 olarak alınmıştır. mazeDistance() metodu kullanılarak iki nokta arasındaki mesafe hesaplanır ve bu hesaplama sezgisel değere atanır. foodGrid içindeki bütün yiyecekler bitene kadar bu işlem devam eder.
- ClosestDotSearchAgent(SearchAgent) sınıfı, aramaları kullanarak tüm yiyecekleri aramak için yazılmıştır. Sınıf dahilindeki findPathToClosestDot(self,gameState) metodu gameState'den başlayarak en yakın noktaya bir path döndürür. Döndürülen path A* arama ile bulunabilir.
- AnyFoodSearchProblem(PositionSearchProblem) sınıfı, herhangi bir yiyeceğe giden bir yol bulmak için kullanılır. PositionSearchProblem sınıfından ayıran kısmı ulaşılması gereken hedef durumun farklı olmasıdır. isGoalState(self,state) metodu hedef durumun kontrolü içindir. Verilen koordinatlarda yiyecek bulunmuşsa True, bulunmamışsa False döndürür.
- mazeDistance(point1,point2,gameState) metodu, search dosyasında oluşturulan arama metotlarını kullanarak herhangi iki nokta arasındaki mesafeyi döndürür. gameState herhangi bir oyun durumu olabilir [2].

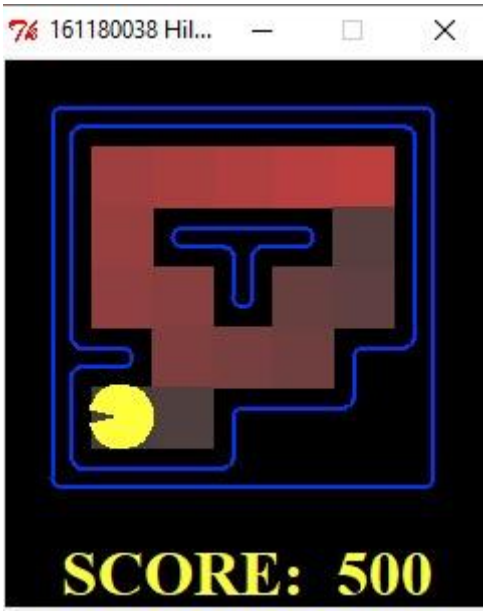
Game dosyası Pacman'in mantığı için yazılmıştır. Pacman'in mantığı dendiğinde anlatılmak istenen durum, Pacman canavarının hareketi, hayaletlerin durumları, oyunun amacı, Pacman canavarının gidebileceği yönler ve oyunun üstüne kurulu olduğu panel(Grid) gibi gereçlerdir. Aynı zamanda bu dosya oyunun sabit ajanlarını da içerir. Klavyeden basılan tuştan, otomatik olarak hareket eden Pacman'e kadar tüm çeşitli varyasyonlar bu dosyanın farklı dosyalara import edilmesiyle çalışır. Aynı zamanda panel olarak bahsi geçen yapı yiyeceklerin

toplanmasını ve bununla alakalı sezgisel fonksiyonların çalışmasını da sağlayacak çeşitli metotlar içerir [2].

Bölüm – 4


```
Path found with total cost of 8 in 0.0 seconds
Search nodes expanded: 16
Pacman emerges victorious! Score: 502
('Average Score:', 502.0)
('Scores:       ', '502.0')
Win Rate:      1/1 (1.00)
('Record:       ', 'Win')
```

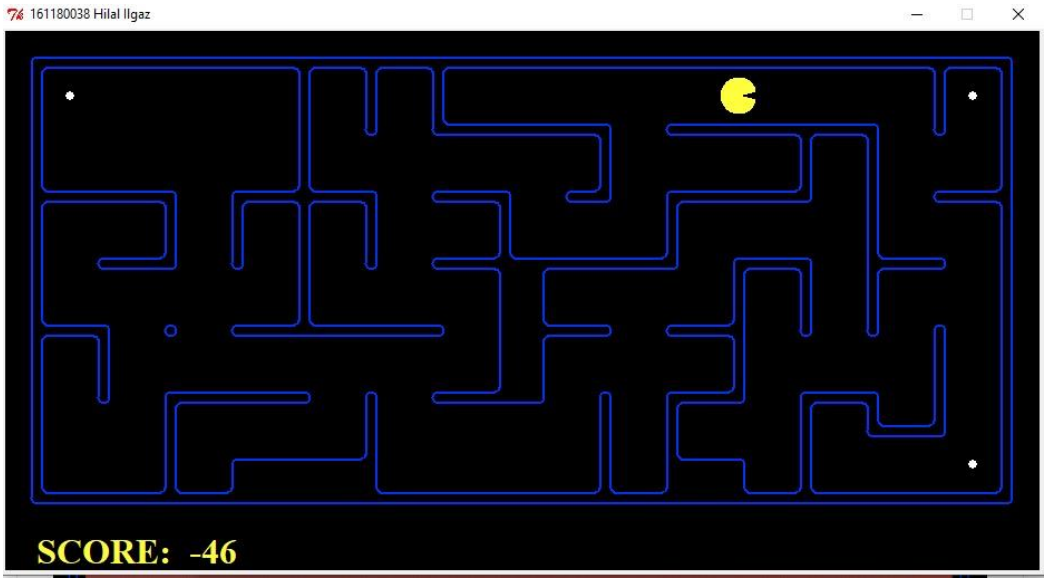
Genişlik öncelikli arama algoritmasının küçük labirentte sadece arama algoritmasının çalışma sistemi denendiğinde SearchAgent eşliğinde alınan sonuçlar; yiyeceği bulmak için Pacman canavarı 8 adım attı ve 16 hücreyi genişletti.




```
Path found with total cost of 10 in 0.0 seconds
Search nodes expanded: 16
Pacman emerges victorious! Score: 500
('Average Score:', 500.0)
('Scores:       ', '500.0')
Win Rate:      1/1 (1.00)
('Record:       ', 'Win')
```

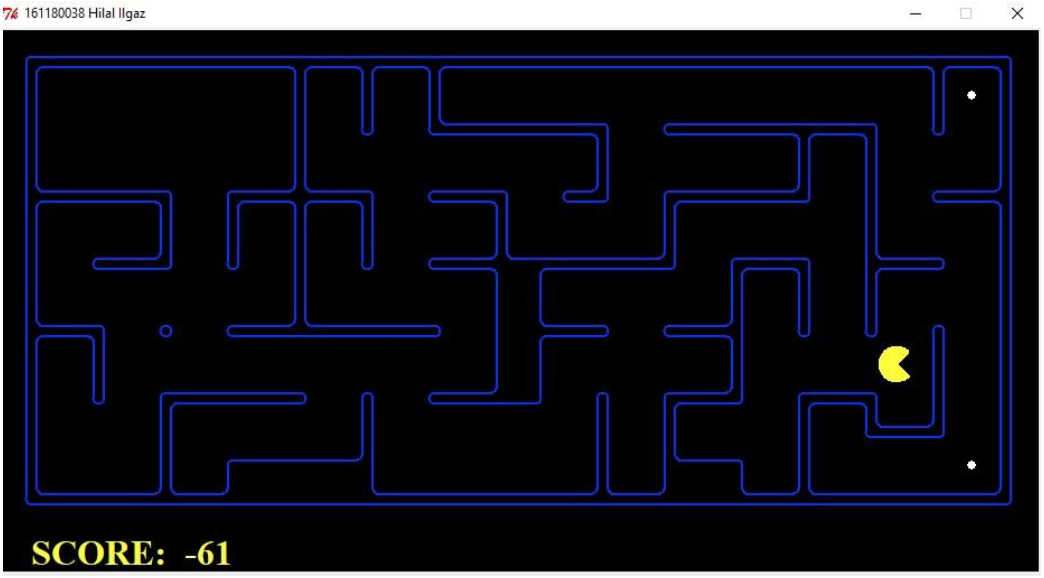
Derinlik öncelikli arama algoritmasının küçük labirentte sadece arama algoritmasının çalışma sistemi denendiğinde SearchAgent eşliğinde alınan sonuçlar; yiyeceği bulmak için Pacman canavarı 10 adım attı ve 16 hücreyi genişletti.

Küçük labirentte genişlik öncelikli ve düşük maliyetli arama algoritmaları derinlik öncelikli arama algoritmasına göre daha iyi sonuç verdi. Bir diğer deyişle genişlik öncelikli arama algoritması optimal sonucu buldu.



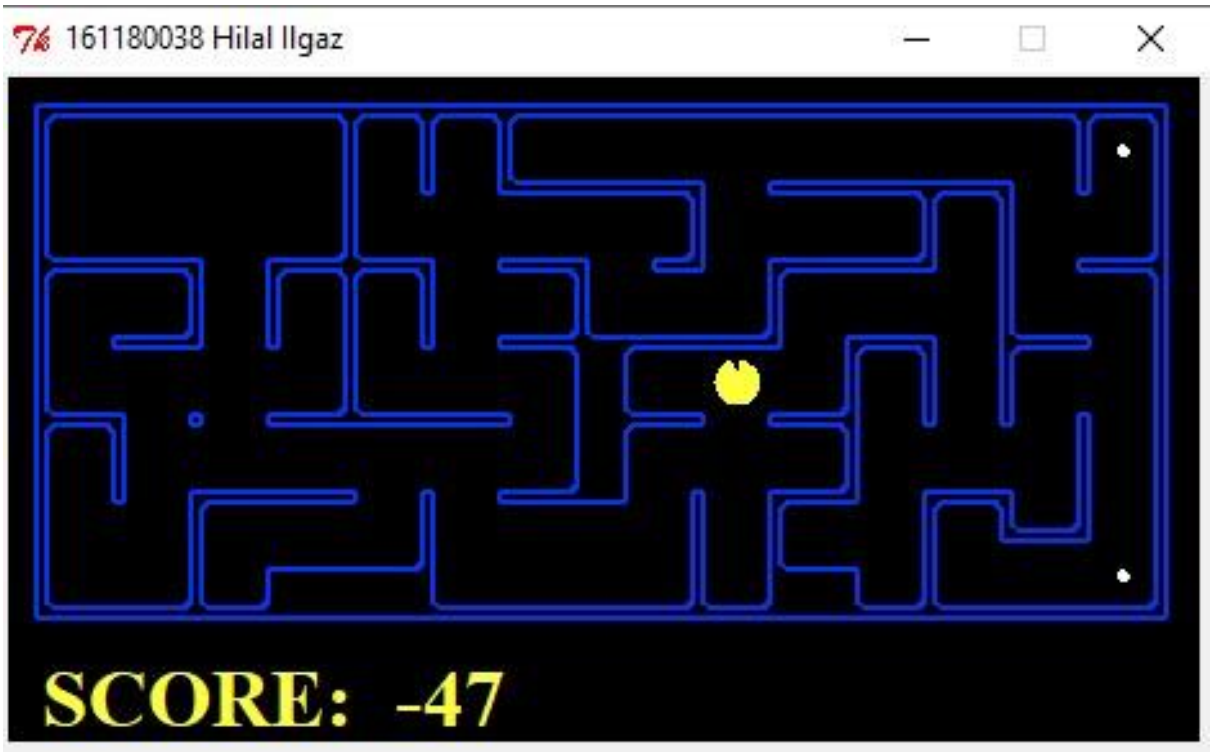

```
Path found with total cost of 221 in 0.0 seconds
Search nodes expanded: 372
Pacman emerges victorious! Score: 319
('Average Score:', 319.0)
('Scores:       ', '319.0')
Win Rate:      1/1 (1.00)
('Record:       ', 'Win')
```

Derinlik öncelikli arama algoritması orta büyüklükte labirentte köşe probleminde 221 adım attı ve 372 hücre genişletti. Bu çözümü bulması 0.0 saniye sürdü.



```
Path found with total cost of 106 in 0.3 seconds
Search nodes expanded: 1968
Pacman emerges victorious! Score: 434
('Average Score:', 434.0)
('Scores:       ', '434.0')
Win Rate:      1/1 (1.00)
('Record:       ', 'Win')
```

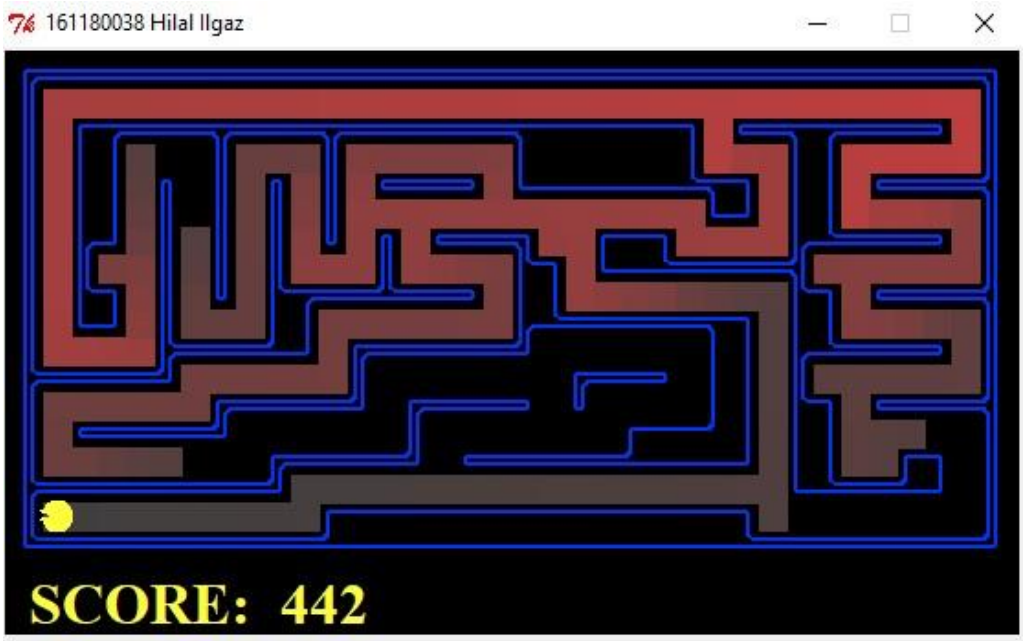
Genişlik öncelikli arama algoritması orta büyüklükte labirentte köşe probleminde 106 adım attı ve 1968 hücre genişletti. Bu çözümü bulması 0.3 saniye sürdü. Bu kısımda bulunan score yazılarına dikkat edilmemesi gerekir. Ekran görüntüleri oyun devam ederken alınmıştır. Oyun sonlandığında hepsi olması gerektiği gibi score değerlerine ulaşmıştır.



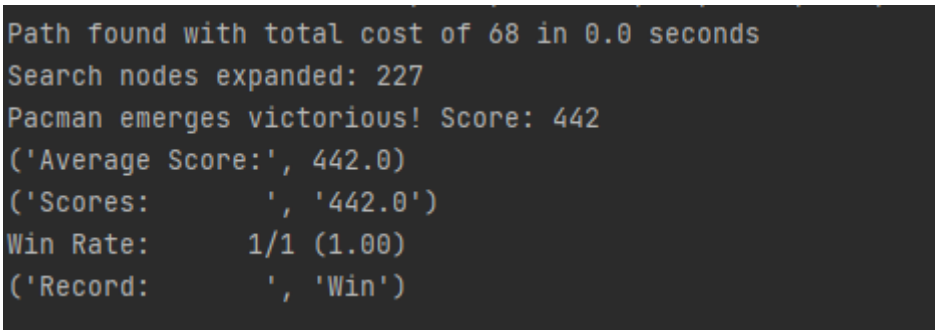

```
Path found with total cost of 106 in 0.3 seconds
Search nodes expanded: 1655
Pacman emerges victorious! Score: 434
('Average Score:', 434.0)
('Scores:       ', '434.0')
Win Rate:      1/1 (1.00)
('Record:       ', 'Win')
```

A* algoritması aynı problem için genişlik öncelikli arama algoritması ile aynı sayıda attı ve 1655 hücre kullandı. Fakat bu çözümü bulması 0.3 saniye sürdü.

Genişlik öncelikli arama daha kıza yol bularak derinlik öncelikli aramadan daha iyi olduğunu gösterdi. Fakat buradaki problem yer karmaşıklığı oldu. Genişlik öncelikli arama, derinlik öncelikli aramanın kapladığının 5 katını kapladı ve bu süre bakımından da uzun sürmesini sağladı. Derinlik öncelikli aramanın avantajı bu noktada ortaya çıktı. A* arama algoritması, genişlik öncelikli arama algoritmasından daha kısa sürmüş ve daha az yer kaplamıştır. Adım sayısı ise aynıdır. A* arama algoritmasının çalıştırılırken onun için tasarlanan köşe ajanlarını kullandığı unutulmamalıdır. Diğer üç arama algoritması ise aynı harita için cornersHeuristic() fonksiyonunu kullanmıştır. A* arama algoritması ajanlar yardımıyla aynı algoritmayı kullanmıştır. Düşük maliyetli arama algoritması sonuçları genişlik öncelikli arama algoritmasıyla aynı çıktığı için yer verilmemiştir.




```
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 222
Pacman emerges victorious! Score: 442
('Average Score:', 442.0)
('Scores:       ', '442.0')
Win Rate:      1/1 (1.00)
('Record:       ', 'Win')
```



A* arama algoritması, sezgisel fonksiyon olarak Öklid ve Manhattan uzaklıkları çerçevesinde denendiğinde her ikisi de 68 adımda tamamlamış olmasına rağmen Öklid için 227 hücre genişletildi, Manhattan için ise 222 hücre genişletildi. Bunun anlamı, Manhattan uzaklığı sezgisel fonksiyon olarak seçildiğinde daha az alan kaplar.

Bölüm – 5

1. Web sitesi: https://tr.wikipedia.org/wiki/Pac-Man
2. Web sitesi: https://inst.eecs.berkeley.edu/~cs188/sp10/projects/search/
3. Ders notu: Özdemir S. BM218 Algoritmalar. Çizge Algoritmaları.
4. Gupta A.(2019). *AI Search Algorithms Implementations.* Web sitesi: https://towardsdata science .com/ai-search-algorithms-implementations-2334bfc59bf5