Bu raporda, BM455 Yapay Zekaya Giriş dersi kapsamında verilen "Yapay Zeka ile Pac-Man" uygulaması detaylarıyla anlatılmaktadır. Rapor içeriği; Bölüm-1: Gerçekleştirilen Platform, Bölüm-2: Gerçekleştirimi Yapılan Problemin Kısa Tanımı, Bölüm-3: Veri Yapısı Kataloğu, Bölüm-4: Elde Edilen Sonuçlar, Bölüm-5: Kaynaklar şeklindedir.

"Yapay Zeka ile Pac-Man" uygulamasının amacı, Pac-Man karakterinin, labirent içerisinde yol bulabilmesi, belirli bir konuma ulaşabilmesi ve etkin bir biçimde yiyecekleri toplayabilmesidir. Verilen ek-kodlarda "search.py" içerisinde boş bırakılan arama algoritmalarının kodları da eklenerek proje gerçekleştirilmiştir. Raporun içindekiler aşağıda sıralanmıştır.

İçindekiler

BÖLÜM 1: GERÇEKLEŞTİRİLEN PLATFORM

“Yapay Zeka ile Pac-Man” uygulaması Visual Studio Code platformunda gerçekleştirilmiştir. Visual Studio Code, birden çok dil için yerleşik desteğe ve diğerleri için zengin bir destek ekosistemine sahip bir uzantı modeline sahiptir. VS Code aylık olarak güncellenir ve Microsoft Python blogunda güncel bilgilere sahip olabilmeye olanak sağlar. Visual Studio Code, iyi belgelenmiş bir uzantı modeli aracılığıyla birden çok programlama dilinde geliştirmeyi destekler. Python uzantısı sıralanan özelliklere sahip Python gelişimini sağlar: Python 3.4 ve üstü ile Python 2.7 desteği, IntelliSense ile kod tamamlama, Linting, Hata ayıklama desteği, Kod parçacıkları, Birim test desteği, Conda ve sanal ortamların otomatik kullanımı, Jupyter ortamlarında ve Jupyter Not Defterlerinde kod düzenleme[1].

Ayrıca bu projenin gelişiminde programlama dili olarak ise Python 2.7.13 kullanılmıştır. Python 2.7.13'ün ilk resmi sürümü 17 Aralık 2016'da yayınlanmıştır[2].

BÖLÜM 2: GERÇEKLEŞTİRİMİ YAPILAN PROBLEMİN KISA TANIMI

Bu projede amaç eğlenmek için yapılan bir oyundan ziyade, istenilen arama algoritmalarının çalışma prensiplerini göstermektir. Oyun, Pacman’in ödül jetonuna ulaşmak için hareketli grafikte gezinmesi gereken ızgara tabanlı bir labirentte oynanır. Bu projedeki problem, Pacman’in istenilen arama algoritmalarını kullanarak, en kısa yoldan verilen jetona ulaşmasıdır. Ek-kodlarda verilen layout içerisindeki çeşitli labirentler kullanılarak istenilen DFS, BFS, UCS, A* algoritmaları için tek tek denemeler yapılmış ve sonuçlar kıyaslanmıştır. Ayrıca, Pac-Man’in belirli bir labirent içerisinde dört köşede bulunan sabit yiyecek kaynaklarına erişmesi için A* algoritması tabanlı bir uygulama yaparak bu işlemi en kısa yoldan yapılması da incelenmiştir.

BÖLÜM 3:VERİ YAPISI KATALOĞU

3.1. Kullanılan Veri Yapıları

Bu projede kuyruk, yığın, öncelik kuyruğu ve liste gibi veri yapılarından faydalanılmıştır.

DFS Algoritması, kör arama stratejisi olarak da adlandırılan bilgisiz bir arama stratejisidir; yani bu algoritmalar, problem tanımında sağlananlar dışında durumlar hakkında hiçbir ek bilgiye sahip değillerdir. Arama ağacının mevcut sınırındaki en derin düğümü genişletir. Oluşturulan düğümleri takip etmek için bir “yığın veri yapısı” kullanır. Yığına bir öğe eklemek için “Push”, yığından bir öğe çıkarmak için ise “Pop” kullanılır. Yığın veri yapısı LIFO ilkesini takip ettiğinden, en son oluşturulan düğüm, genişletme için seçilen ilk düğümdür. Bir düğüm tamamen genişletildiğinde, yığından çıkar.

BFS Algoritması, seviyeye göre bir düzey aramadır. Bir sonraki seviyeye geçmeden önce belirli bir seviyedeki tüm düğümler genişletilir. Önce en yüzeysel düğümü genişletir. Genişletilmiş tüm düğümlerin bir listesini tutmak amacıyla "kuyruk veri yapısı" kullanır. Kuyruğa bir öğe eklemek için "Enqueue", kuyruktan bir öğe çıkarmak için ise "Dequeue" kullanılır.

UCS Algoritması, BFS gibi en sığ düğümü genişletmek yerine, en düşük maliyetli yol g(n) ile düğüm n'yi genişletir. Sınırın g ile sıralanan bir öncelik sırası olarak saklanmasının nedeni budur. UCS algoritması hedef testini, ilk oluşturulduğu zamandan ziyade genişletme için seçildikten sonra bir düğüme uygular. Bunun nedeni, üretilen ilk hedef düğümün optimalin altında bir yolda olabilmesidir. Ayrıca, daha iyi bir hedef durumuyla karşılaşılıp karşılaşılmadığını kontrol etmek için bir test içerir. "Öncelik kuyruğu(priority queue) veri yapısını" kullanır. Öncelik kuyruğu tarafından tek tip maliyetli bir arama algoritması uygulanır. En düşük kümülatif maliyete maksimum öncelik verir. UCS algoritması, tüm kenarların yol maliyeti aynı olduğu zaman BFS algoritmasına eşdeğerdir.

A* Arama Algoritması, bilgilendirilmiş arama stratejisi kategorisine girer. Bu tür aramaya en iyi ilk arama da denir. A* algoritmalarının uygulanmasında düğümlerden maliyetleri düşük olanları seçmek ve tekrarlanan seçimleri gerçekleştirmek için "Öncelik kuyruğu(priority queue) veri yapısını" kullanılır. A* araması, g(n), yani düğüme ulaşma maliyeti ve h(n) yani mevcut düğümden hedef düğüme ulaşma maliyeti kullanılarak hangi düğümün birleştirileceğini değerlendirir ve şu şekilde gösterilir: "f(n) = g(n) + h(n)"[3].

3.2.Projede Kullanılan Bazı Fonksiyonların ve Sınıfların Açıklamaları

class SearchProblem	Bir abstract class olan SearchProblem sınıfı, bir arama probleminin yapısının ana hatlarını çizer. Ancak yöntemlerin hiçbirini uygulamaz.
fonksiyon getStartState	Arama probleminin başlangıç durumunu döndürür.
fonksiyon isGoalState	Sadece durum geçerli bir hedef durumuysa True döndürür.
fonksiyon getSuccessors	Belirli bir durum için mevcut durumun halefini, oraya ulaşmak için gereken eylemi ve o düğümü genişletmenin maliyetini döndürür.
fonksiyon getCostOfActions	Bu fonksiyon, belirli bir eylem dizisinin toplam maliyetini döndürür.
fonksiyon tinyMazeSearch	TinyMaze'i çözen bir dizi hareket döndürür.

fonksiyon depthFirstSearch	İstenilen komutlara uyularak hazırlanmıştır. Derinlik öncelikli arama algoritmasının kodlarıdır.
fonksiyon breadthFirstSearch	İstenilen komutlara uyularak hazırlanmıştır. Genişlik öncelikli arama algoritmasının kodlarıdır.
fonksiyon uniformCostSearch	İstenilen komutlara uyularak hazırlanmıştır. Düşük maliyetli arama algoritmasının kodlarıdır.
fonksiyon nullHeuristic	Mevcut durumdan sağlanan arama problemindeki en yakın hedefe kadar olan maliyeti tahmin eder
fonksiyon aStarSearch	İstenilen komutlara uyularak hazırlanmıştır. A* arama algoritmasının kodlarıdır.

BÖLÜM 4: ELDE EDİLEN SONUÇLAR

Bu bölümde; derinlik öncelikli arama(DFS), genişlik öncelikli arama(BFS), düşük maliyetli arama(UCS) ve A* algoritmalarının performansları aşağıda bulunan ekran görüntüleri kullanılarak değerlendirilmiş ve kıyaslanmıştır. İncelenen algoritmalar arasında "derinlik öncelikli arama(DFS)" algoritması, "düşük maliyetli arama "UCS" algoritması ve "A*" algoritmalarının bigMaze labirentinde, genişletilen arama düğümleri ve ortalama skorları eşit gelmiştir. Ayrıca A* algoritmasının dört köşe probleminde her bir hedefe varıldığında algoritmayı tekrar işleyerek kalan hedefler için yeni bir işlem yapılır. Burada hesaplama maliyeti önemli bir kriter olmuştur ve işlem "Manhattan uzaklığı" ile çözülmüştür.

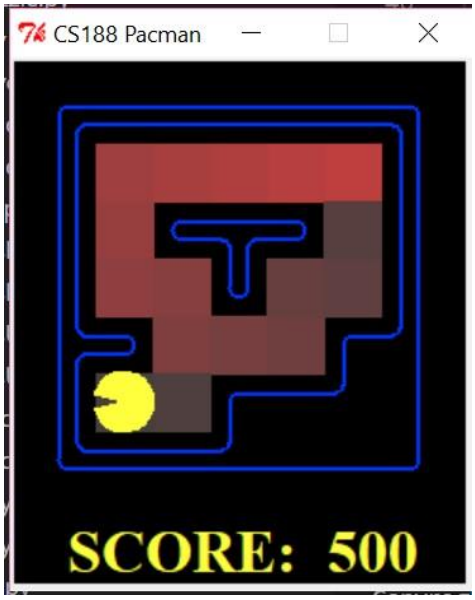
Aşağıdaki ekran görüntüleriyle daha detaylı inceleyelim;

Pac-Man'in arama algoritmalarını kullanarak nasıl performans gösterdiğini görmek için komut satırından aşağıdaki şekilde komutlar çalıştırılmalıdır:

"python pacman.py -l MAZE_TYPE -p SearchAgent -a fn=SEARCH_ALGO"

Burada; MAZE_TYPE: Labirent düzenini tanımlar, SearchAgent ve SEARCH_ALGO ise parametrede sağlanan algoritmaya göre Pac-Man'i labirent boyunca yönlendirir.

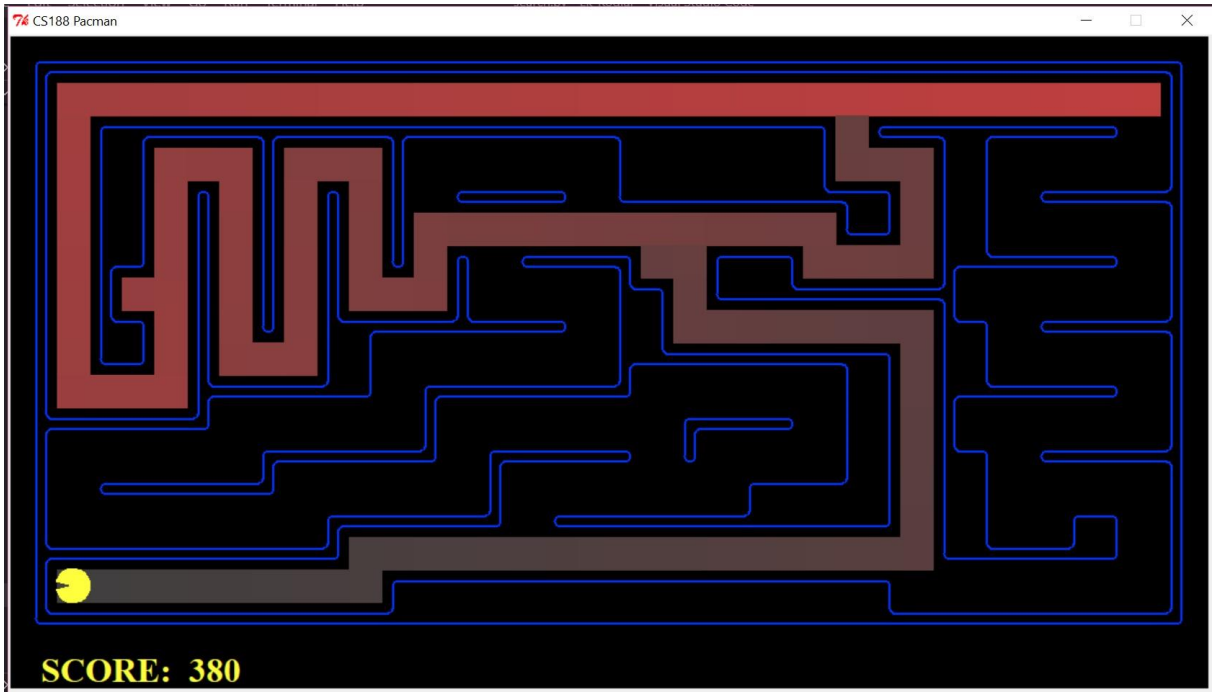
- Öncelikle, DFS algoritmasının tinyMaze, mediumMaze, bigMaze ve mediumScaryMaze labirentlerindeki performansını inceleyelim:



Şekil 1- DFS algoritmasının tinyMaze labirentinde araması

```
PS C:\Users\bulent.dirican\Desktop\Ek_Kodlar> python pacman.py -l tinyMaze -p SearchAgent -a fn=dfs
[SearchAgent] using function dfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 10 in 0.0 seconds
Search nodes expanded: 15
Pacman emerges victorious! Score: 500
Average Score: 500.0
Scores:        500.0
Win Rate:      1/1 (1.00)
Record:        Win
PS C:\Users\bulent.dirican\Desktop\Ek_Kodlar>
```

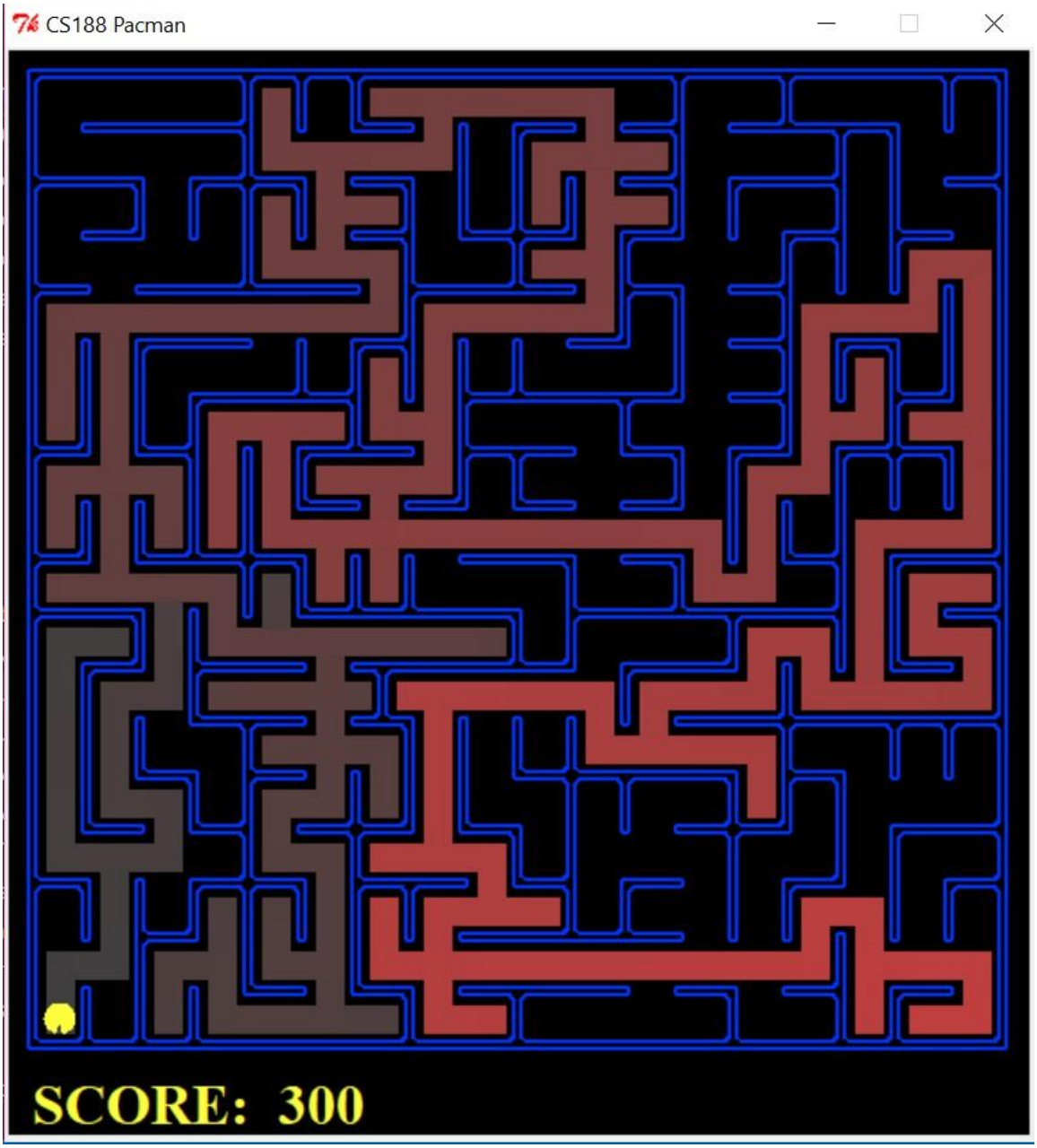
Şekil 2- DFS algotirmasının tinyMaze labirentinde aramasından elde edilen sonuçlar



Şekil 3- DFS algoritmasının mediumMaze labirentinde araması

```
PS C:\Users\bulent.dirican\Desktop\Ek_Kodlar> python pacman.py -l mediumMaze -p SearchAgent -a fn=dfs
[SearchAgent] using function dfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 130 in 0.0 seconds
Search nodes expanded: 146
Pacman emerges victorious! Score: 380
Average Score: 380.0
Scores:        380.0
Win Rate:      1/1 (1.00)
Record:        Win
PS C:\Users\bulent.dirican\Desktop\Ek_Kodlar>
```

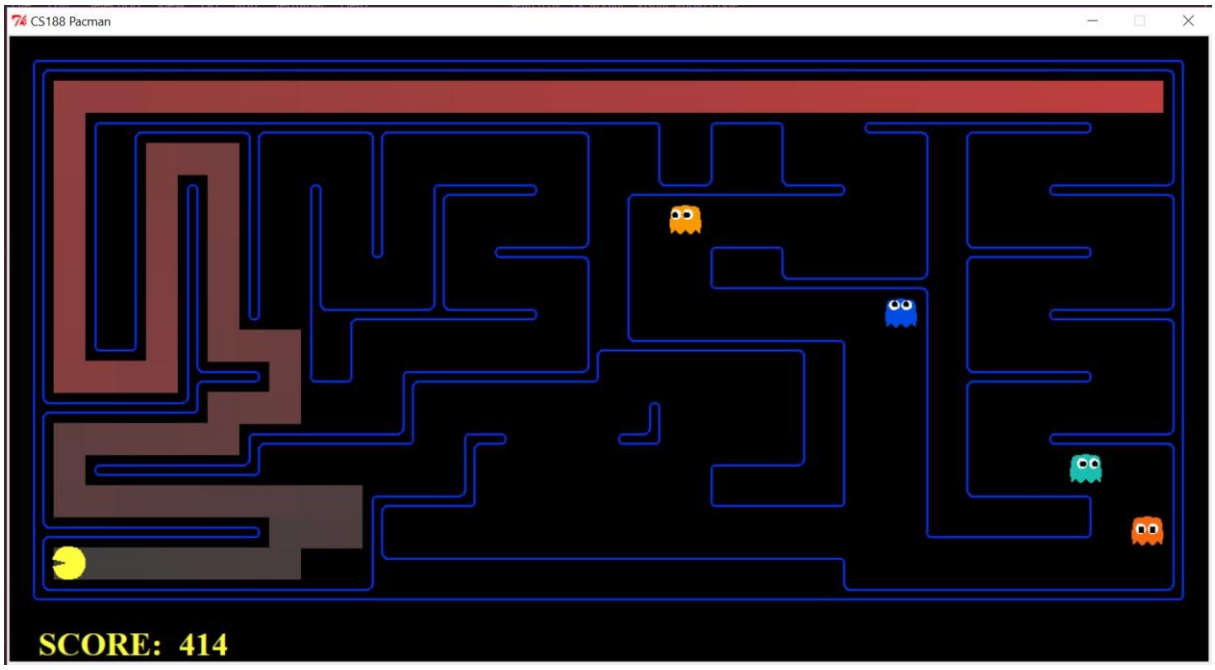
Şekil 4- DFS algoritmasının mediumMaze labirentinde aramasından elde edilen sonuçlar



Şekil 5- DFS algoritmasının bigMaze labirentinde araması

```
PS C:\Users\bulent.dirican\Desktop\Ek_Kodlar> python pacman.py -l bigMaze -p SearchAgent -a fn=dfs -z .5
[SearchAgent] using function dfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 390
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:        300.0
Win Rate:      1/1 (1.00)
Record:        Win
PS C:\Users\bulent.dirican\Desktop\Ek_Kodlar>
```

Şekil 6- DFS algoritmasının bigMaze labirentinde aramasından elde edilen sonuçlar



Şekil 7- DFS algoritması mediumScaryMaze labirentinde hayaletle karşılaşmadan jetona ulaşmıştır

```
PS C:\Users\bulent.dirican\Desktop\Ek_Kodlar> python pacman.py -l mediumScaryMaze -p SearchAgent -a fn=dfs
[SearchAgent] using function dfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 96 in 0.0 seconds
Search nodes expanded: 96
Pacman emerges victorious! Score: 414
Average Score: 414.0
Scores:        414.0
Win Rate:      1/1 (1.00)
Record:        Win
PS C:\Users\bulent.dirican\Desktop\Ek_Kodlar>
```

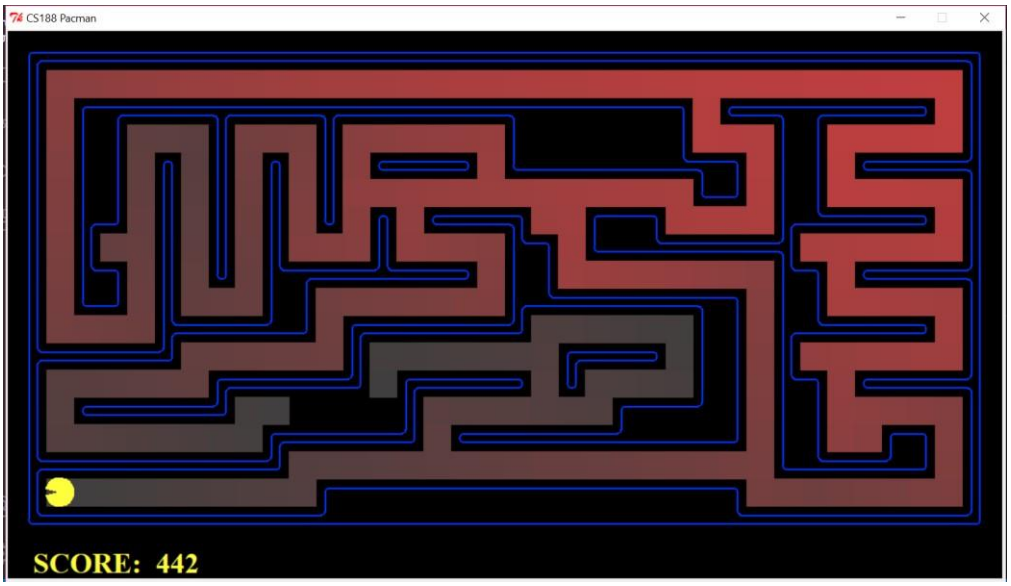
Şekil 8- DFS algoritmasının mediumScaryMaze labirentinde aramasının sonucu

- BFS algoritmasının tinyMaze, mediumMaze, bigMaze ve mediumScaryMaze labirentlerindeki performansını inceleyelim:

Şekil 9- BFS algoritmasının tinyMaze labirentinde araması

```
PS C:\Users\bulent.dirican\Desktop\Ek_Kodlar> python pacman.py -l tinyMaze -p SearchAgent -a fn=bfs
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 8 in 0.0 seconds
Search nodes expanded: 15
Pacman emerges victorious! Score: 502
Average Score: 502.0
Scores:        502.0
Win Rate:      1/1 (1.00)
Record:        Win
PS C:\Users\bulent.dirican\Desktop\Ek_Kodlar>
```

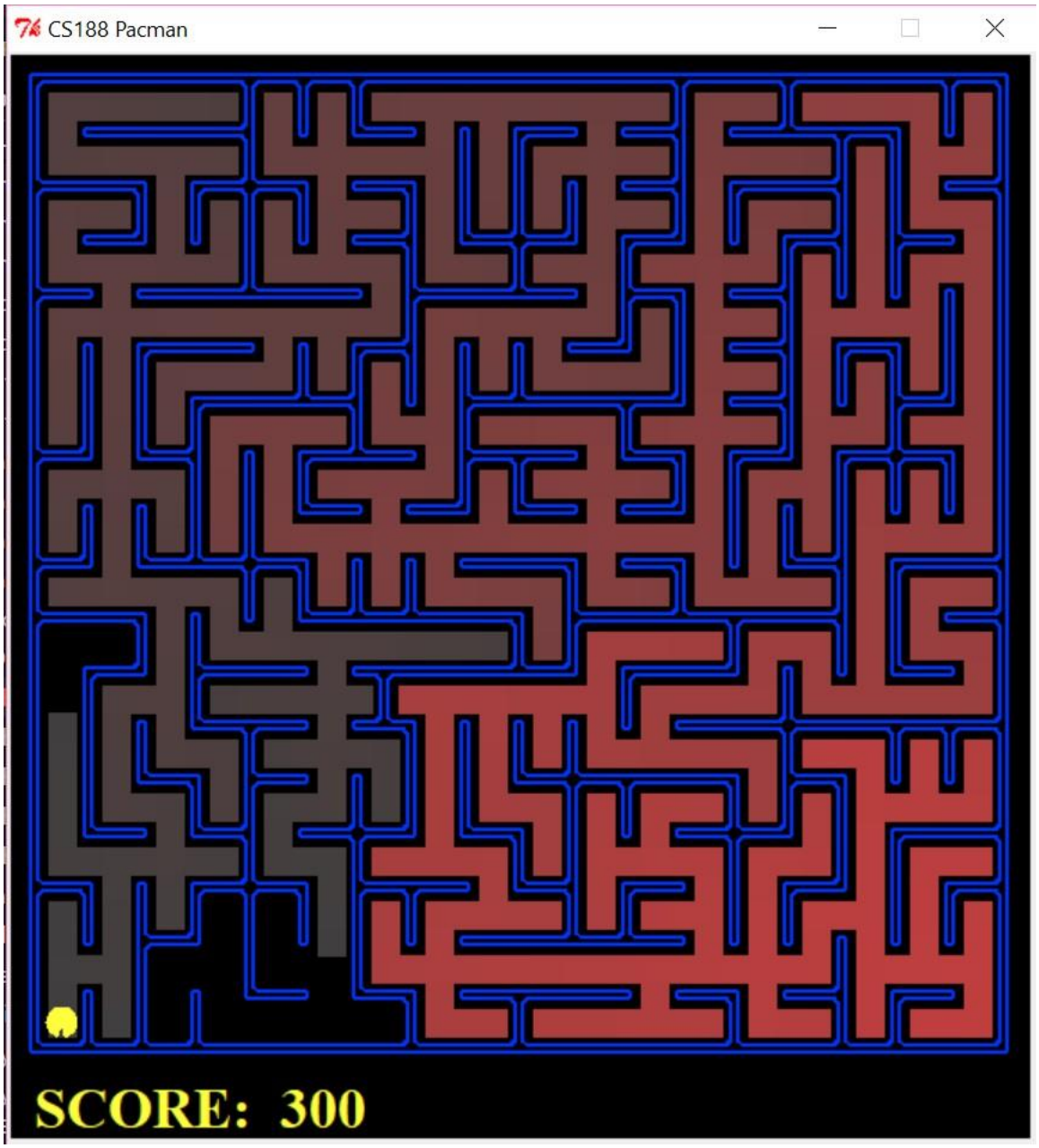
Şekil 10- BFS algoritmasında tinyMaze labirentinde aramasından elde edilen sonuçlar



Şekil 11- BFS algoritmasının mediumMaze labirentinde araması

```
PS C:\Users\bulent.dirican\Desktop\Ek_Kodlar> python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 269
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores:        442.0
Win Rate:      1/1 (1.00)
Record:        Win
PS C:\Users\bulent.dirican\Desktop\Ek_Kodlar>
```

Şekil 12- BFS algoritmasında mediumMaze labirentinde aramasından elde edilen sonuçlar



Şekil 13- BFS algoritmasının bigMaze labirentinde araması

```
PS C:\Users\bulent.dirican\Desktop\Ek_Kodlar> python pacman.py -l bigMaze -p SearchAgent -a fn=bfs -z .5
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 620
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:        300.0
Win Rate:      1/1 (1.00)
Record:        Win
PS C:\Users\bulent.dirican\Desktop\Ek_Kodlar>
```

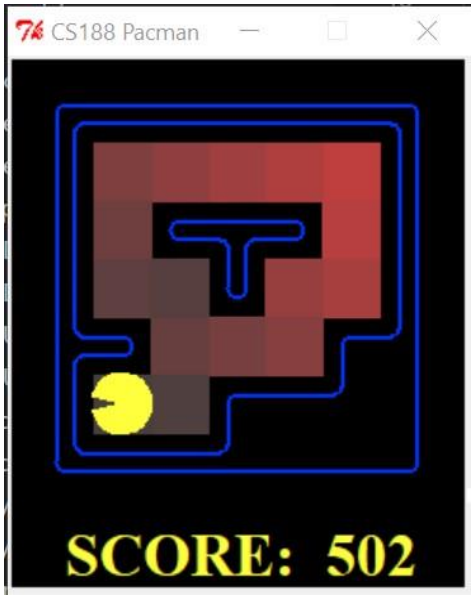
Şekil 14- BFS algoritmasında bigMaze labirentinde aramasından elde edilen sonuçlar

Şekil 15-BFS algoritması mediumScaryMaze labirentinde hayaletle karşılaşarak başarısız olmuştur

```
PS C:\Users\bulent.dirican\Desktop\Ek_Kodlar> python pacman.py -l mediumScaryMaze -p SearchAgent -a fn=bfs
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 72 in 0.0 seconds
Search nodes expanded: 279
Pacman died! Score: -523
Average Score: -523.0
Scores:        -523.0
Win Rate:      0/1 (0.00)
Record:        Loss
PS C:\Users\bulent.dirican\Desktop\Ek_Kodlar>
```

Şekil 16-BFS algoritmasının mediumScaryMaze labirentinde aramasının sonucu

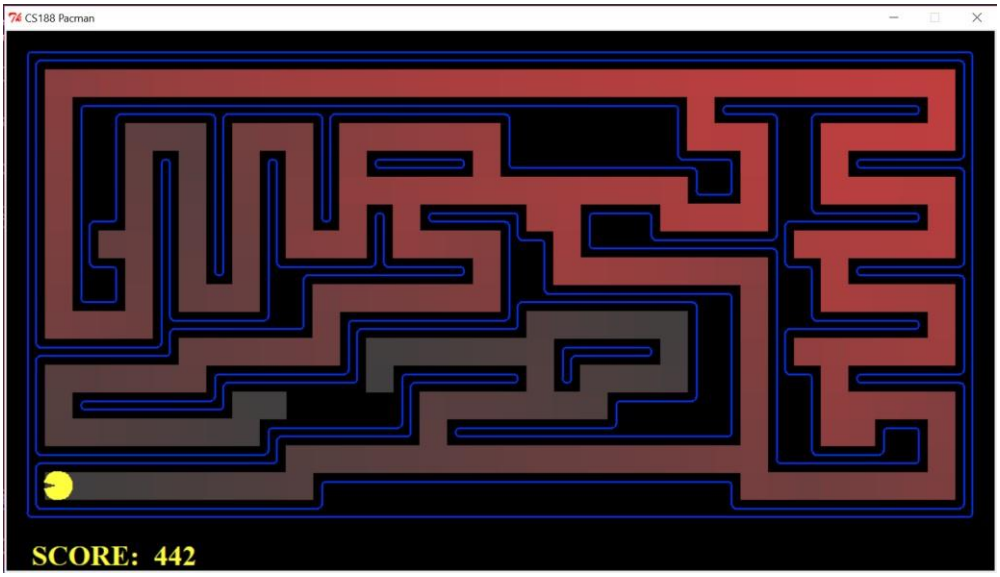
- UCS algoritmasının tinyMaze, mediumMaze, bigMaze ve mediumScaryMaze labirentlerindeki performansını inceleyelim:



Şekil 17- UCS algoritmasının tinyMaze labirentinde araması

```
PS C:\Users\bulent.dirican\Desktop\Ek_Kodlar> python pacman.py -l tinyMaze -p SearchAgent -a fn=ucs
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 8 in 0.0 seconds
Search nodes expanded: 15
Pacman emerges victorious! Score: 502
Average Score: 502.0
Scores:        502.0
Win Rate:      1/1 (1.00)
Record:        Win
PS C:\Users\bulent.dirican\Desktop\Ek_Kodlar>
```

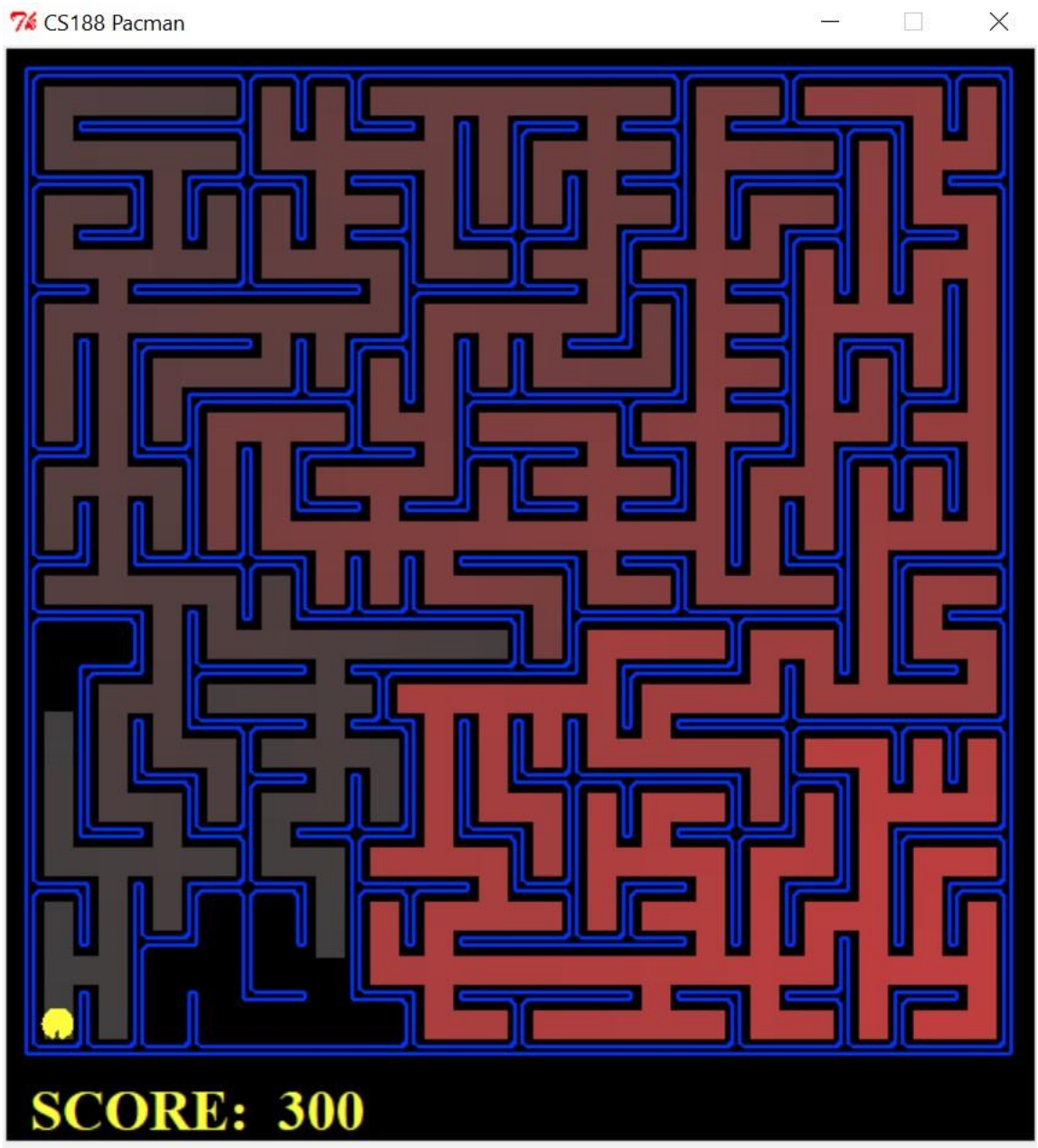
Şekil 18- UCS algoritmasında tinyMaze labirentinde aramasından elde edilen sonuçlar.



Şekil 19- UCS algoritmasının mediumMaze labirentinde araması

```
PS C:\Users\bulent.dirican\Desktop\Ek_Kodlar> python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 269
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores:        442.0
Win Rate:      1/1 (1.00)
Record:        Win
PS C:\Users\bulent.dirican\Desktop\Ek_Kodlar>
```

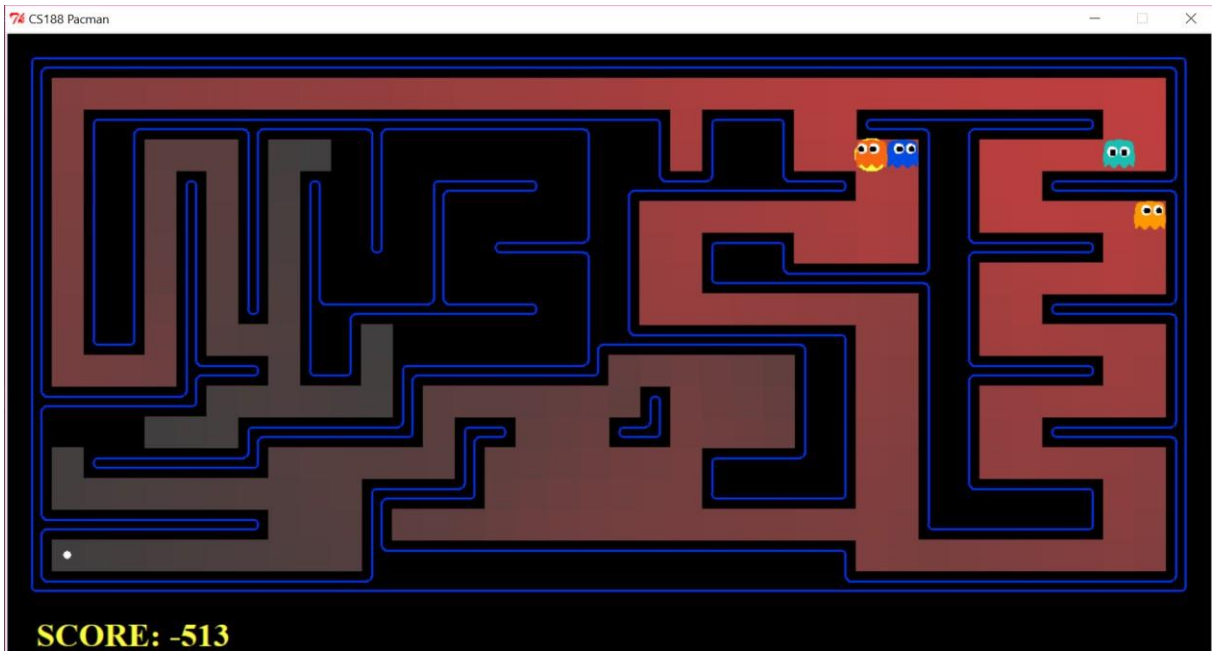
Şekil 20- UCS algoritmasında mediumMaze labirentinde aramasından elde edilen sonuçlar.



Şekil 21- UCS algoritmasının bigMaze labirentinde araması

```
PS C:\Users\bulent.dirican\Desktop\Ek_Kodlar> python pacman.py -l bigMaze -p SearchAgent -a fn=ucs -z .5
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 620
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:        300.0
Win Rate:      1/1 (1.00)
Record:        Win
PS C:\Users\bulent.dirican\Desktop\Ek_Kodlar>
```

Şekil 22- UCS algoritmasında bigMaze labirentinde aramasından elde edilen sonuçlar.

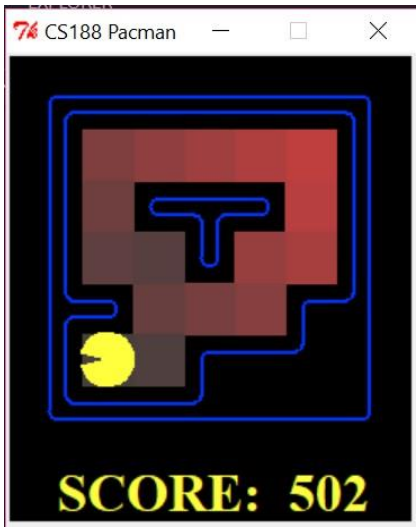


Şekil23- UCS algoritması mediumScaryMaze labirentinde hayaletle karşılaşarak başarısız olmuştur

```
PS C:\Users\bulent.dirican\Desktop\Ek_Kodlar> python pacman.py -l mediumScaryMaze -p SearchAgent -a fn=ucs
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 72 in 0.0 seconds
Search nodes expanded: 279
Pacman died! Score: -513
Average Score: -513.0
Scores:        -513.0
Win Rate:      0/1 (0.00)
Record:        Loss
PS C:\Users\bulent.dirican\Desktop\Ek_Kodlar>
```

Şekil 24- UCS algoritmasının mediumScaryMaze labirentinde aramasının sonuçları

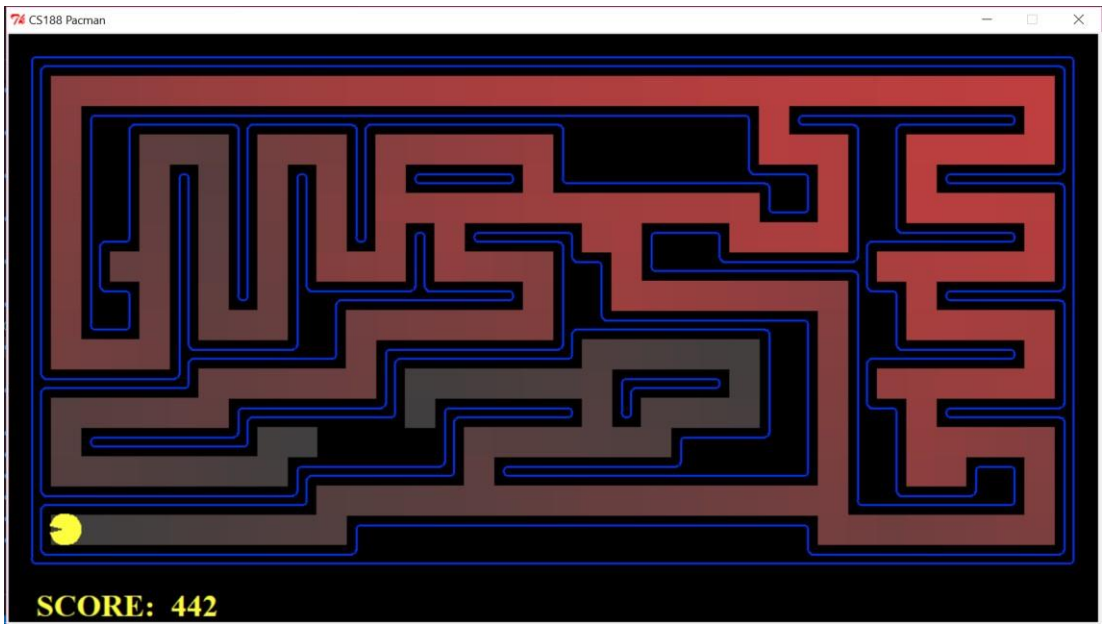
- A* algoritmasının tinyMaze, mediumMaze, bigMaze, mediumScaryMaze labirentlerindeki performansını ve dört köşede aramasını inceleyelim:



Şekil 25- A* algoritmasının tinyMaze labirentinde araması

```
PS C:\Users\bulent.dirican\Desktop\Ek_Kodlar> python pacman.py -l tinyMaze -p SearchAgent -a fn=astar
[SearchAgent] using function astar and heuristic nullHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 8 in 0.0 seconds
Search nodes expanded: 15
Pacman emerges victorious! Score: 502
Average Score: 502.0
Scores:        502.0
Win Rate:      1/1 (1.00)
Record:        Win
PS C:\Users\bulent.dirican\Desktop\Ek_Kodlar>
```

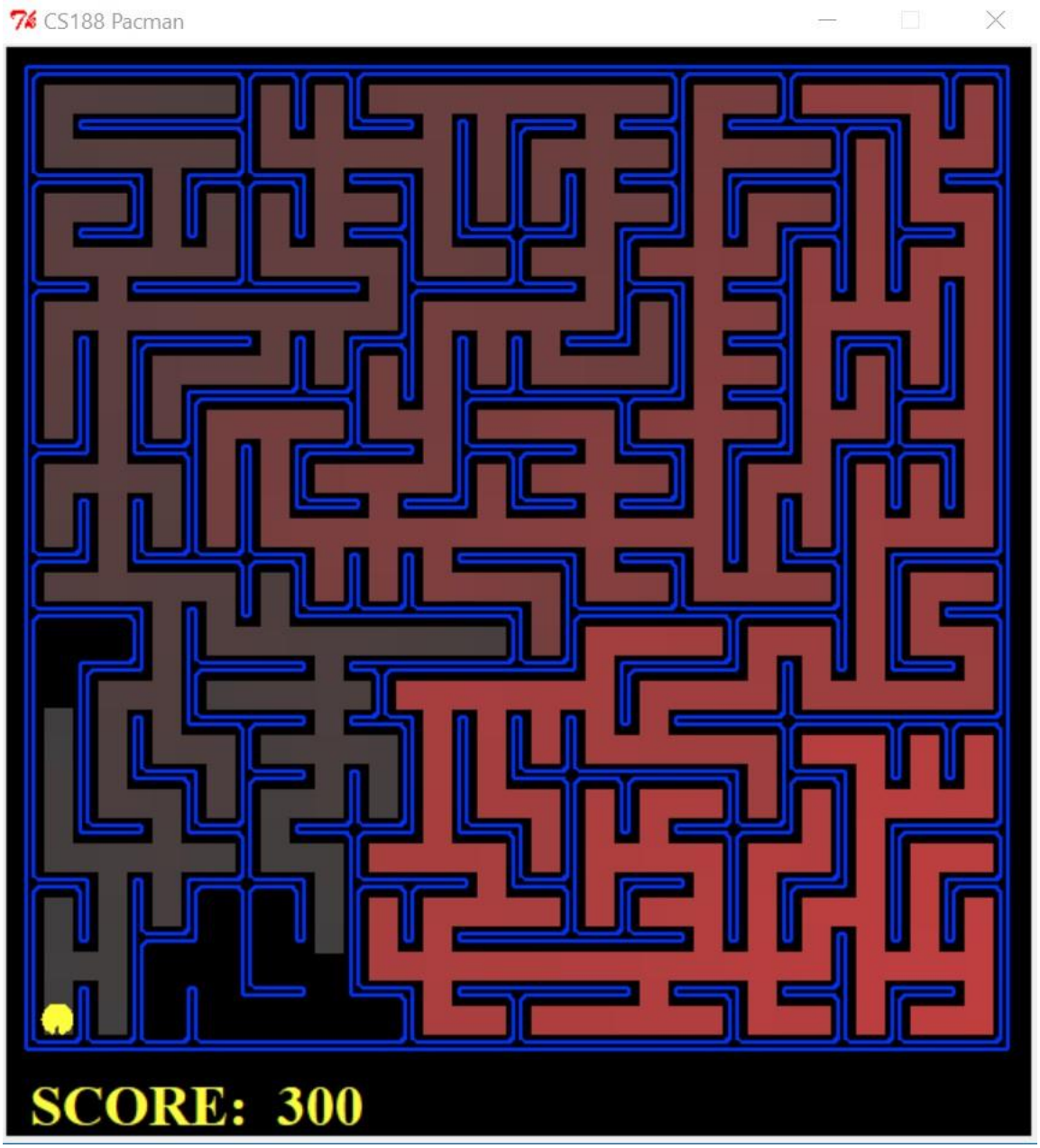
Şekil 26- A* algoritmasında tinyMaze labirentinde aramasından elde edilen sonuçlar.



Şekil 27- A* algoritmasının mediumMaze labirentinde araması

```
PS C:\Users\bulent.dirican\Desktop\Ek_Kodlar> python pacman.py -l mediumMaze -p SearchAgent -a fn=astar
[SearchAgent] using function astar and heuristic nullHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.1 seconds
Search nodes expanded: 269
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores:        442.0
Win Rate:      1/1 (1.00)
Record:        Win
PS C:\Users\bulent.dirican\Desktop\Ek_Kodlar>
```

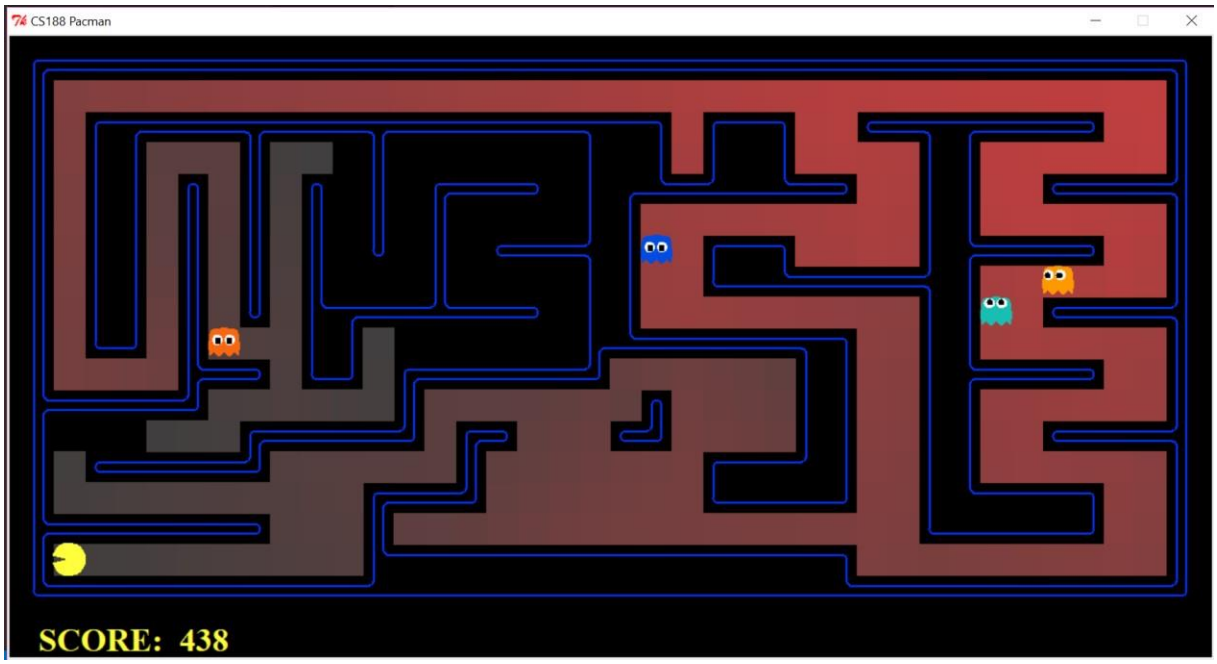
Şekil 28- A* algoritmasında mediumMaze labirentinde aramasından elde edilen sonuçlar.



Şekil 29- A* algoritmasının bigMaze labirentinde araması

```
PS C:\Users\bulent.dirican\Desktop\Ek_Kodlar> python pacman.py -l bigMaze -p SearchAgent -a fn=astar -z .5
[SearchAgent] using function astar and heuristic nullHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 620
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:        300.0
Win Rate:      1/1 (1.00)
Record:        Win
PS C:\Users\bulent.dirican\Desktop\Ek_Kodlar>
```

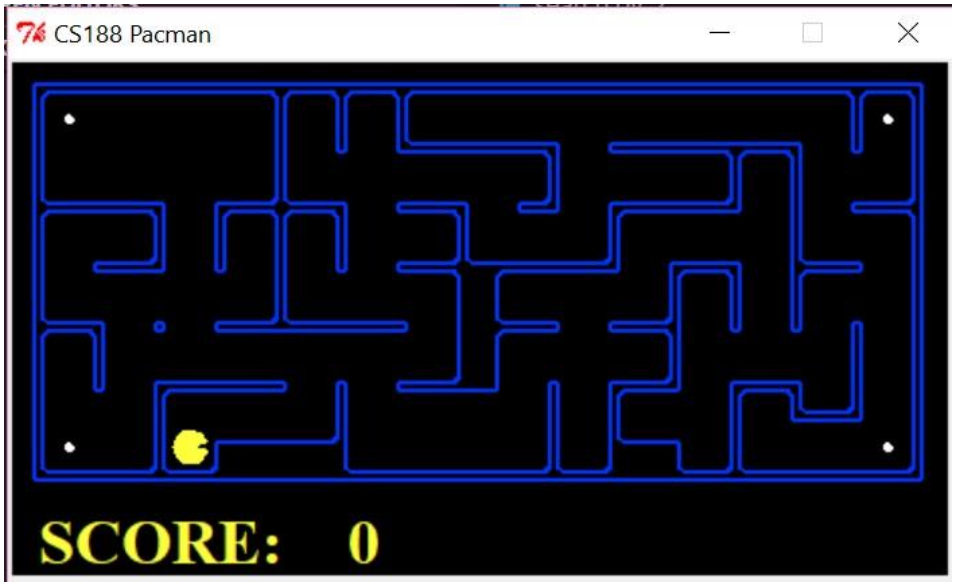
Şekil 30- A* algoritmasında bigMaze labirentinde aramasından elde edilen sonuçlar.



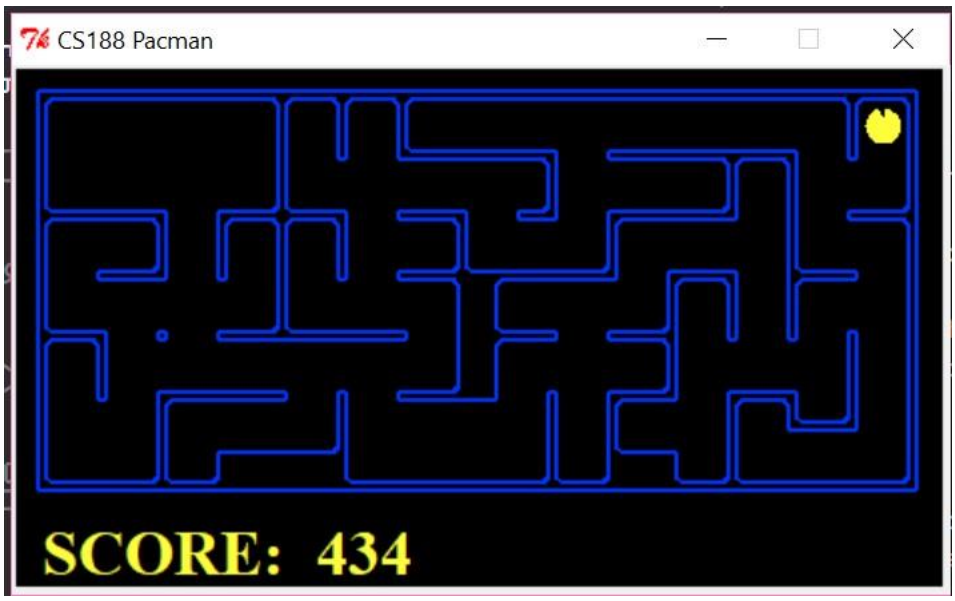
Şekil 31- A* algoritması mediumScaryMaze labirentinde hayaletle karşılaşmadan jetona ulaşmıştır

```
PS C:\Users\bulent.dirican\Desktop\Ek_Kodlar> python pacman.py -l mediumScaryMaze -p SearchAgent -a fn=astar
[SearchAgent] using function astar and heuristic nullHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 72 in 0.1 seconds
Search nodes expanded: 279
Pacman emerges victorious! Score: 438
Average Score: 438.0
Scores:        438.0
Win Rate:      1/1 (1.00)
Record:        Win
PS C:\Users\bulent.dirican\Desktop\Ek_Kodlar>
```

Şekil 32- A* algoritmasının meziumScaryMaze labirentinde aramasının sonucu



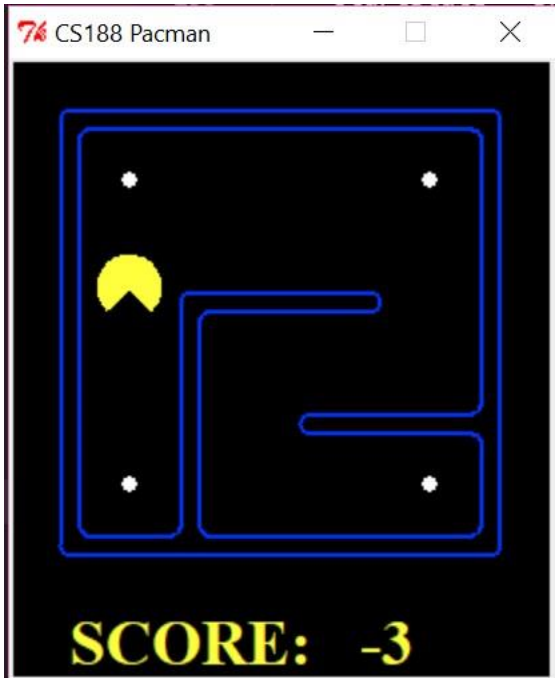
Şekil 33- A* algoritmasının mediumMaze dört köşede arama yapmaya başlama durumu



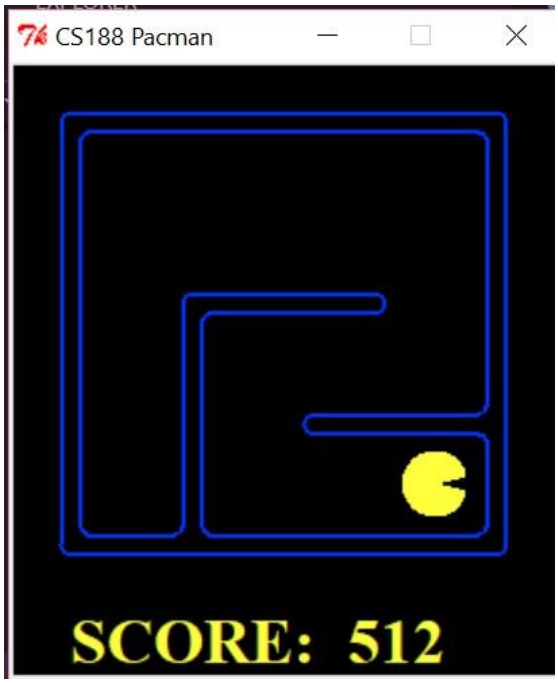
Şekil 34-A* aramasının mediumMaze dört köşede arama yapma skoru

```
PS C:\Users\bulent.dirican\Desktop\Ek_Kodlar> python pacman.py -l mediumCorners -p AStarCornersAgent -z 0.5
Path found with total cost of 106 in 1.7 seconds
Search nodes expanded: 459
Pacman emerges victorious! Score: 434
Average Score: 434.0
Scores:        434.0
Win Rate:      1/1 (1.00)
Record:        Win
PS C:\Users\bulent.dirican\Desktop\Ek_Kodlar>
```

Şekil 35-A* algoritmasının mediumMaze dört köşede arama yapmasının sonucu



Şekil 36- A* algoritmasının tinyMaze dört köşede arama yapmaya başlama durumu



Şekil 37- A* aramasının tinyMaze dört köşede arama yapma skoru

```
PS C:\Users\bulent.dirican\Desktop\Ek_Kodlar> python pacman.py -l tinyCorners -p SearchAgent -a fn=bfs,prob=CornersProblem
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 28 in 0.0 seconds
Search nodes expanded: 252
Pacman emerges victorious! Score: 512
Average Score: 512.0
Scores:        512.0
Win Rate:      1/1 (1.00)
Record:        Win
PS C:\Users\bulent.dirican\Desktop\Ek_Kodlar>
```

Şekil 38- A* algoritmasının tinyMaze dört köşede arama yapmasının sonucu

BÖLÜM 5: KAYNAKLAR

1. Python Development Visual Studio Code, Erişim Tarihi: Aralık 14,2020. https://realpython.com/python-development-visual-studio-code/
2. Python 2.7.13, Erişim Tarihi: Aralık 14,2020.

https://www.python.org/downloads/release/python-2713/

3. Pal,J., Chartterjee, D., Modak,S. (2019), Designing of Search Agents Using Pacman.
4. DFS Algoritması kodu için, Pal,J., Chartterjee, D., Modak,S. (2019), Designing of Search Agents Using Pacman, sayfa 4. ‘teki sözdekoddan yararlanışmıştır.
5. BFS Algoritması kodu için, Pal,J., Chartterjee, D., Modak,S. (2019), Designing of Search Agents Using Pacman, sayfa 6. ‘daki sözdekoddan yararlanışmıştır.
6. UCS Algoritması kodu için, Pal,J., Chartterjee, D., Modak,S. (2019), Designing of Search Agents Using Pacman, sayfa 8. ‘deki sözdekoddan yararlanışmıştır.
7. A* Algoritması kodu için, Pal,J., Chartterjee, D., Modak,S. (2019), Designing of Search Agents Using Pacman, sayfa 10. ‘daki sözdekoddan yararlanışmıştır.