YAPAY ZEKA DERSİ DÖNEM SONU PROJESİ

MAZE SOLVER

OCAK 2013

Proje Ekibi

Emre Can YILMAZ (09060286)
Nurdan Tuğçe YEŞİLYURT (09060295)
Ayşe Begüm TOPYILDIZ (09060252)

MAZE SOLVER

Tanım: Maze Solver programı kendisine verilen bir labirenti A* Algoritmasını kullanarak en kısa yoldan çözmeyi amaçlayan bir programdır.

Platform: Program Ubuntu 12.04 üzerinde ve Ruby Programlama Dili (versiyon 1.9.3) kullanılarak geliştirilmiştir.

Çalıştırma: Xterm açılır, proje dizinindeyken şu komut verilir: “$ ruby main.rb”

Eğer ki labirenti kendimiz oluşturmak istiyorsak:
1) kod içerisindeki “plan” dizisi gerektiği gibi doldurulur.
2) “map = build_map(15, 15)” satırının başına # konularak etkisiz hale getirilir.
3) “map = build_planned_map(plan)” satırı başındaki # silinerek aktif hale getirilir.
4) komut satırına yine “$ ruby main.rb” yazılarak program çalıştırılır.

Programın ne kadar sürede çalıştığını görüntülemek için:
1) path.size.times # ile etkisiz hale getirilir.
2) komut satırına “$ time ruby main.rb” yazılarak program çalıştırılır.

Not:
|: Duvar . : Kullanılmayan Yol

Uygulama Dosyaları
Maze Solver programı main.rb ve node.rb dosyalarından oluşmaktadır.

main.rb: A* algoritmasının uygulandığı ana program dosyasıdır.

Bu dosyada bulunan fonksiyonlar ve işlevleri:

`def adj(curr, goal, open, close, map) ;` adjugate matrix
`def build_map(x_dim = 10, y_dim = 10) ;` verilen x ve y boyutlarına göre rasgele bir labirentin oluşturulduğu fonksiyon.
`def build_planned_map(plan);` planlanmış bir labirentin oluşturulduğu fonksiyon.
`def print_map(map, start, goal, path = Array.new) ;` labirentin görsel şekile aktarıldığı fonksiyon. X: Gidilen Yol, S: Başlangıç, G: Hedef, |: Duvar, . : Kullanılmayan Yol
`def in_path?(path, cell) ;` doğru yolda mıyım? kontrolünü yapan fonksiyon.
`def find_path(start, goal, map) ;` yol bulma fonksiyonunun hesaplanması.

node.rb: Bağlantı düğümlerinin oluşturulması, bu düğümlerin maaliyetini ve sezgisel fonksiyonun gerçekleştiği yardımcı bir kitaplıktır.

Bu dosyada bulunan fonksiyonlar ve işlevleri:

`def initialize(x, y, c)` ; x, y koordinatları ve cost değişkenlerinin tanımlandığı fonksiyondur.
`def s_parent ;` `kendisinin ebeveynini bulan fonksiyondur.`
`def total_cost ;` toplam maliyet hesabını yapan fonsiyondur.
`def value(goal);` hedefteki değerin hesabını yapan fonksiyondur.
`def heur(goal) ;` sezgisel hedeflerin hesabının yapıldığı fonksiyondur.
`def cost_from_parent ;` ebeveynden gelen maliyet hesabının yapıldığı fonksiyondur.
`def g_x ;` x değerinin getirildiği fonksiyon.
`def g_y ;` y değerinin getirildiği fonksiyon.
`def g_parent ;` ebeveyn değerinin getirildiği fonksiyon.
`def g_cost ;` maliyet değerinin getirildiği fonskiyon.
`def to_s ;` string dönüşümünün yapıldığı fonksiyon.

Sezgisel (Heuristic) Algoritma:

Sezgisel algoritmalar geçiş süresinde daha verimli hale gelebilmek için en iyi çözümü aramaktan vaz geçerek çözüm zamanını azaltan algoritmalardır.

Sezgisel algoritmalar en iyi sonucu bulacaklarını garanti etmezler fakat makul bir süre içerisinde bir çözüm elde edeceklerini garanti ederler. Genellikle en iyiye yakın olan çözüm yoluna hızlı ve kolay bir şekilde ulaşırlar.

Sezgisel fonksiyonumuzu yol bulma algoritmalarınının en iyilerinden olan Manhattan algoritmasını kullandık.

```
Manhattan Distance
h(n) = D * (abs(n.x-goal.x) + abs(n.y-goal.y))
```

```
Alternatif
Diagonal Distance
h(n) = D * max(abs(n.x-goal.x), abs(n.y-goal.y))
```

A-Star (A*) Algoritması:

A-star algoritması, en kısa yolu bulmak için kullanılan algoritmalardan birisidir. Bir düğümden (node) hedef bir düğüme, en kısa hangi düğümler üzerinden gidileceğini bulmaya yarar.

A* algoritması, sezgisel (heuristic) bir algoritma olarak sınıflandırılabilir. Bunun sebebi algoritmasının mesafe hesaplarken kullandığı fonksiyondur:

$f(n) = g(n) + h(n)$ denklemindeki

$f(n)$ = hesaplama yapan sezgisel (heuristic) fonksiyon.
$g(n)$ = Başlangıç düğümünden mevcut düğüme kadar gelmenin maliyeti
$h(n)$ = Mevcut düğümden hedef düğüme varmak için tahmin edilen mesafe.

f(n) fonksiyonunun sezgisel olma sebebi, bu fonksiyon içerisinde bulunan ve tahmine dayalı olan h(n) sezgisel fonksiyonudur.

Algoritmanın çalışması:

Algoritma yukarıdaki toplama işlemini kullanan bir yapıya sahiptir. Veri yapısı olarak bir öncelik sırası (priority queue) kullanan algoritmada en öncelikli olan düğüm f(n) değeri en düşük olan düğümdür.

1. Algoritma her adımda çocuk düğümler için f(x) fonksiyon değerlendirmesi yapılarak minimum değerli düğüm seçilir. Aynı değere sahip olan birden fazla düğüm varsa bu düğümler içerisinden operatörlerin kullanım sırasına göre herhangi birisi seçilir.
2. Açılan minimum değerli düğümün, benzeri şekilde çocuk düğümlerinin değerlendirilmesi yapılır. O anki düğümden yeni bir çocuk düğüm elde edilemiyorsa ve henüz hedefe varılmadıysa, üst seviyeye geri dönülerek arama yönü minimum olan fonksiyon değerine göre değiştirilir.
3. Gidilen düğüme göre komşu olan bütün düğümlerin değerleri güncellenir (artık bu düğüme gelmenin bir maliyeti vardır ve f(n) fonksiyonu içerisinde bu değer yer almaktadır.)
4. İşlemler hedefe ulaşıncaya kadar tekrarlanır. Hedef bulunmuşsa başlangıç durumundan hedefe olan yol çözümü verilir.

Kaynaklar:

http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html
http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html
http://tr.wikipedia.org/wiki/Sezgisel_algoritma
http://en.wikipedia.org/wiki/A*_search_algorithm
http://en.wikipedia.org/wiki/Adjugate_matrix