**Labirent Oyunu**

[**Selda Yapal**](mailto:seldayapal@anadolu.edu.tr)

[**Nedret Gegeoğlu**](mailto:nedretgegeoglu@windowslive.com)

**Kocaeli Üniversitesi**

**Bilgisayar Mühendisliği 2017 – 2018**

1. **Problem Tanımı:**

Bizden istenen rastgele oluşturulan bir labirentin içerisinde bir giriş, çıkış kapısı belirterek iki kapı arasında yolculuk edip labirenti çözmekti. Burada kullanabileceğimiz yöntemler birden fazla olabilirdi fakat bizden “bağlı liste” ve “yığın” yapısı kullanmamız istendi.

1. **Yapılan Araştırmalar:**

İlk olarak araştırmamız gereken C dili üzerinde bağlı liste kullanımıydı. Bunun için **struct** yapısı üzerinde oluşturulan nesne yapısı ile bağlı listeye eleman ekleme yöntemini oluşturabileceğimizi öğrendik ve uyguladık.

Yığın yapısını (LIFO) inceleyip uygulamaya dökmek için oluşturulan struct düzenine bağlı liste üzerinde sona ekleme metodu oluşturup, sondan silme metodu oluşturduk. Böylelikle ilk giren eleman son çıkar mantığını yakaladık.

Labirent üzerinde dolaşma mantığı projenin en önemli kısımıydı. Bunu uygulamaya dökmek için düşündüğümüz fikir:

Kullanılan giriş kapısından itibaren dört bir yanını tarayarak yol (“1”) olup olmadığını kontrol ettik. Labirent çıkan bu uygun yollardan “alt, sağ, sol, üst” sıralamasını takip ederek ilk uygun yolu kullanıma soktu.

Labirent üzerindeki bir diğer kritik sorun çıkış kapısına ulaşıp ulaşmadığını tespit etmekti.

* Bunun için ilk olarak giriş ve çıkış kapılarının yolun üzerinde olduğuna dikkat ettik Yani giriş ve çıkış kapıları “1” olana dek yazılım tekrar sormaya devam eder.
* Bu aşamayı hallettikten sonra ikinci adım olarak eğer labirent üzerindeki giriş çıkışa ulaşamıyorsa kaldığı son noktadan geriye sararak başlangıç noktasına dönüyor ve geriye döndükçe son elemanı silip listeyi (yolu) güncelliyordu. Bu aşamada önce yolun geldiği son noktada artık ileriye gidilemeyeceği yazılım tarafından teyit edildiğinde geriye sarma başlıyordu.
* Böylelikle bu sorun çözüldü.

Labirente dair belki de karşılaştığımız en büyük sorun verilen sınırları aşmasıydı. Burada olay labirentin sınırları dışarısında oluşturulan hafızada kalmış herhangi 1 bitine rastlaması halde orayı da yola dahil edip kullanıma almasıydı. Bu sorunu da labirentin kontrol mekanizmasını sınırlar içerisinde kısıtlayarak çözdük ve taşmayı engelledik.

1. **Tasarım:**
   1. **Yazılım Mimarisi:**

Projeyi C dilinde oluşturduk. Bağlı liste yapısını kullanmak için ve her bir kutuya ait verileri (koordinatları) saklamak için struct yapısı kullandık.

Bununla beraber bağlı liste yöntemi için bir çok farklı metot kullanıldı.

* Sona ekle();
* Listeden sil();
* Son eleman();
* Listeyi yazdır();
* Listede Var mı();

1. **Genel Yapı:**

Projede ilk olarak bağlı liste işlemlerini gerçekleştirdik Bunun için sırasıyla:

* **Struct dugum();** oluşturulup içerisinde koordinat verilerini saklayan, ve struct dugum yardımıyla önceki ve sonraki elemana bağlantı yaparak listenin çift yönlü olmasını sağladık.
* **Struct dugum ekle();** metodu ile yeni eleman ekleme metodunu hazırladık
* **Sona ekle();** struct dugum ekle(); metodu ile hazırlanan yeni eleman verilerini listede sona eklemek için hazırladığımız metot. Girilen elemanın bulunduğu x ve y koordinatını ekler. Ekleme yapıldıktan sonra listede en son elemana geçilir.
* **Listeden Sil();** Eğer eklenilen elemanı silmemiz gerekirse bu fonksiyon yardımı ile x ve y koordinatı vererek silebiliyoruz. Silme işlemi tamamlandıktan sonra fonksiyon son elemana geri dönüş yapar.
* **Son eleman();** Bağlı liste üzerindeki son elemanı yazılım içerisine aktarmamız gerektiğinde x ve y koordinatı için 2 değişken verip sonuç almamızı sağlayan metot.
* **Listeyi Yazdır();** Oluşan bağlı listeyi ekrana yazdırmaya yarayan metot.
* **Listede Var mı();** Liste içerisinde eleman aramamıza yarayan metot.

Bağlı liste içeriği için kullanılan bu metotlardan sonra labirentin görsel görünümü için **labirent();** metodu oluşturulup düzenlendi.

* **Labirent();**

Bu metot içerisinde **Graphics.h** kütüphanesi yardımıyla kullanıcı için görsel bir labirent ekranı yaratıldı.

* 1000x700 boyutunda oluşturulan bu ekranda görüntü düzenlemeleri yapıldı.
* Kullanıcının belirlediği labirent boyutuna göre ekran eşit boyutlarda kutulara paylaştırıldı.
* Rastgele 1ler ve 0larla oluşturulan labirentte 1leri beyaz renk, 0ları siyah renk kutular temsil etti.
* Son olarak labirent probleminde sorun çözüme ulaşırsa ekranda giriş noktasında oluşturulan bir resim, çıkış noktasına ulaştığı bir animasyon oluşturuldu.

Görsel olarak labirentin temsili de hazırlanıldıktan sonra son olarak **matris();** metodu oluşturuldu.

* **Matris();**

Proje üzerinde labirent giriş çıkışı belirlemekten, bağlı listenin düzenlenmesinden, sınırların kontrolünün yapıldığı, çizilen yolun ulaşıp ulaşmadığını kontrol eden metottur.

İlk olarak kullanıcıdan alınan matris boyutuna göre 1 ve 0 dan oluşan rastgele bir labirent dizayn eder.

Labirentin giriş ve çıkış kapısı belirlenir ve iki kapının da 1ler üzerinde olmasına dikkat edilir. Olmadıkça tekrar kullanıcıdan adres girmesi istenir.

Oluşturulan giriş kapısının x ve y koordinatı int sonuç[2]; isimli değişkene [0] ve [1] adresine aktarır ve labirent çözümlenene kadar dönecek bir while döngüsü içerisine girer.

Oluşturulan döngünün çıkış kriteri sonuc[0] ve sonuc[1] koordinatlarının çıkış kapısına eşit olmasıdır. Ulaştığında döngüden çıkar. Eğer yol yoksa tespit edilip döngü kırılır ve sonuç olmadığı bildirilir.

Labirent içerisinde piyonumuzun nereye gideceğini belirlemek için etrafındaki 1leri kontrol ederek **“alt check,sol check, sag check, ust check”** isimli oluşturulan değişkenlere piyonun dört bir yanındaki birleri kontrol ettirilir.

**Eğer gidilecek tek bir durum var ise:**

Liste kontrolü yapıldıktan sonra o koordinat listede var ise yani oraya daha önce gidilmiş ise o kutudaki 1i 0 yaparız ve tekrar o kutuya gitme işleminden kurtuluruz.

Kontrol yapıldıktan sonra listede yok ise mevcut yöne ilerlenir.

**Eğer gidilecek birden fazla durum var ise:**

Bulunan karede piyon etrafında ki kutulara ilerler bu ilerleme sayısında toplam gidilen komşu sayısı yani etrafındaki her kutuya gidip geri dönmüşse o kutu çıkmaz olduğu için gidilen komşu sayısı ile toplam check değişkeni eşit olduğunda o kutu 1 den 0 yapılır ve listeden silinir.

Bir diğer işlem ise piyonun bulunduğu kutuda etrafındaki kutunun daha önce gidilip gidilmediğini kontrol etmektir. Buna göre daha önce gidilmemiş ve o kutu gidilebilir ise bu elemanı listeye ekleriz.

Gidilen adresleri listeleriz ve kaç hamle sarf ettiğimizi görürüz.

Tüm bu işlemler sonucunda hiçbir kutuya gidemiyorsak labirent içinde bulunduğumuz rota çıkışa ulaşmıyor demektir. Bunu kontrol eden mekanizma aynı kutuda defalarsa döngünün döndüğünü fark ettiğinde hatalı çıkış uyarısı verir ve bu sayede labirentin çıkışı olmadığını tespit ederiz. Böylelikle döngü kırılır ve “Labirentin çıkışı yok” mesajı verilir.

Eğer bulunduğumuz kutudan bir sonraki kutu sınırların dışında ise ve biz bu kutuya girersek o kutunun bit değerini 0 yapar bağlı listeden siler ve önceki kutuya dönmeyi sağlarız.

Tüm bu işlemlerden sonra labirent içerisinde uygun yol var ise yol bulunur ve piyon giriş noktasından çıkış noktasına doğru ilerler ve attığı her adımda liste güncellenip ekrana yansıtılır. Bunun sonucunda çıkışa ulaşıldığında döngü sonlanır ve ekrana “Labirentin sonuna ulaşıldı.” Mesajı verilir ve oluşturulan labirent görseli gerekli bilgilere göre oluşturulur ve ekrana yansıtılır.

Böylelikle labirentte piyonumuzun çizdiği rotayı görsel olarak da takip edebiliriz.

1. **Referanslar:**

[1.] Bağlı Listeler – Embedded Koü :

<https://docs.google.com/document/d/1qKg4_edAoyakXnn5R16JhcHw6jPypV1QBln35rqJT5E/edit>