DERS : VERİ YAPILARI VE ALGORİTMALAR

AD SOYAD : DUA HALLUF EDİB

NUMARA : 23210401243

Bir labirenti çözmek için kullanılan bir uygulama içeriyor. Programın amacı, belirli bir labirent haritası üzerinde bir başlangıç noktasından çıkış noktasına kadar bir yol bulmaktır. Aşağıda, bu kodun bileşenleri hakkında bir rapor bulabilirsiniz:

1. Maze (Labirent) Sınıfı (Maze.h ve Maze.cpp)

- **Sınıfın Amacı:**
  Bu sınıf, bir labirenti temsil eder ve labirent üzerinde hareket etmek ve çözmek için gerekli metotları sağlar. Labirent bir dosyadan okunur ve ardından üzerinde belirli bir yol aranır.

- **Üyeler:**

  - `std::vector<std::string> labirent;`
    Labirentin her satırını bir `string` olarak saklayan bir vektör. Her satır, labirentin bir satırını temsil eder.

  - `Konum baslangic;`
    Labirentteki başlangıç noktasını temsil eden bir `Konum` yapısı.

  - `Konum cikis;`
    Labirentteki çıkış noktasını temsil eden bir `Konum` yapısı.

- **Metotlar:**

  - `Maze(const std::string &dosyaAdi);`
    Labirenti verilen dosya adından yükleyen bir kurucu. Dosyayı satır satır okuyarak `labirent` vektörüne yükler.

  - `void labirentiYazdir();`
    Labirenti konsola yazdırır.

  - `void labirentiGuncelle();`
    Labirenti konsolda günceller ve ardından yeniden yazdırır. (Linux/Unix/MacOS için `clear`, Windows için `cls` komutunu kullanır.)

  - `bool hareketEt(Konum konum);`
    Labirent üzerinde bir yol bulmaya çalışır. Hareketi simüle eder ve adım adım labirenti günceller.

  - `bool labirentiCoz();`
    Labirentte bir çözüm bulmak için `hareketEt` fonksiyonunu çağırır.

  - `bool gecerliHareket(Konum konum);`
    Bir konumun labirent üzerinde geçerli bir hareket olup olmadığını kontrol eder.

2. Hareket Algoritması (Maze.cpp)

- **Hareket Algoritması:**
  - Algoritma, başlangıç noktasından başlayarak her adımda dört yönde (yukarı, aşağı, sağ, sol) hareket etmeye çalışır.
  - Geçerli bir hareket bulunduğunda, bu hareketi gerçekleştirir ve bu adımı '>' karakteri ile işaretler.
  - Eğer çıkmaz bir yola girerse, adım geri alınır ve yol eski haline döndürülür.
  - Program, belirli bir süre bekleyerek (`std::this_thread::sleep_for`) her adımın görsel olarak kullanıcıya gösterilmesini sağlar.
  - Eğer çıkış noktası bulunursa, labirent başarıyla çözülmüş olur. Aksi halde, çıkış bulunamadığı anlamına gelir.

3. Harita (Harita.txt)

- **Harita Yapısı:**
  - Harita bir ASCII labirenti olarak temsil edilir.
  - '#' karakteri duvarları temsil eder.
  - ' ' karakteri boş alanları, yani geçilebilir yolları temsil eder.
  - 'e' karakteri çıkış noktasını temsil eder.
  - Bu belirli harita, sol üst köşede bir başlangıç noktası (1, 1) ve sağ alt köşeye yakın bir çıkış noktası içerir.

4. Ana Program (main.cpp)

- **Amaç:**
  - Program, `Maze` sınıfını kullanarak bir labirenti çözmeyi dener.
  - `labirentiCoz` metodu çağrılır ve eğer bir çözüm bulunursa, başarı mesajı; bulunamazsa hata mesajı yazdırılır.

5. Makefile

- **Derleme ve Çalıştırma:**
  - `Makefile`, programın derlenmesi ve çalıştırılması için gerekli komutları içerir.
  - `derle` komutu, kaynak dosyaları (`main.cpp` ve `maze.cpp`) derleyerek ikili dosyayı oluşturur.
  - `calistir` komutu, derlenmiş programı çalıştırır.

6. Programın İşleyişi

- Program çalıştırıldığında, harita dosyasından labirenti yükler.
- Başlangıç noktasından hareket etmeye başlar ve çıkış noktasına ulaşmak için yolları dener.
- Her adımda ekran güncellenir ve yolun nasıl bulunduğu görsel olarak kullanıcıya gösterilir.
- Sonuç olarak, program ya bir çözüm bulur ya da çıkmaz yollara girdiği için çözüm bulunamadığını bildirir.

Bu program, özellikle algoritmaların görsel olarak gösterilmesi gereken durumlarda, labirent çözme ve yol bulma problemlerinin anlaşılmasına yardımcı olabilir.