PROGRAMLAMA LABORATUVARI II

PROJE 1:

ŞİRİNLER OYUNU

SENA ÖKTEM

Kocaeli Üniversitesi

Mühendislik Fakültesi

Bilgisayar Mühendisliği(İÖ)

190202054@kocaeli.edu.tr

Özet:

Bu projede .txt dosyasından oyuna girecek düşman karakterler, bu düşman karakterlerin hangi kapıdan gireceği ve oyunun haritası okunmuştur. Okunan bu verilerle oyunun haritası oluşturulmuş ve düşman karakterler kapılarına yerleştirilmiştir. Oyunun başında iki farklı oyuncu seçeneği bulunmaktadır. Kullanıcı oyuna başlamadan önce karakterini seçecektir ve oyun seçilen karakterin özelliklerine göre başlayacaktır. Oyuncu düşmana yakalandığı an düşman giriş kapısına geri dönecek ve oyun devam edecektir. Bu 2D oyun tasarımı için nesne tabanlı programlama ve graphics kullanılmıştır.

Giriş:

Bu projede en kısa yol algoritması olarak dijkstra algoritması kullanılmıştır. Proje Java Programlama dilinde Eclipse geliştirme ortamında yazılmıştır. Projede inheritance, encapsulation ve abstraction kullanılmıştır.

Yöntem:

ARAYÜZ TASARIMI VE OYUN ALGORİTMASI

1.Görsel Tasarım

new JButton() kullanılarak gozluklu ve tembel şirin için seçim butonları oluşturuldu.

addMouseListener(new MouseListener()) kullanılarak oyun öncesi karakter seçim işlemi yaptırıldı.

Graphics kullanılarak oyun haritası çizdirilmiştir. Oyuncu ve düşman karakterler haritaya yerleştirilmiştir.Altın ve mantarlar random koordinatlarla haritaya çizdirilmiştir.

g.setFont() ile ekrana yazdırılacak skor, winner ve game over yazılarının fontu belirlendi.

g.drawString() ile ekranın sağ köşesine skor yazdırıldı.

2. Oyuncu Kontrolleri

KeyAdapter extends edilmiştir.

Public void keyPressed(KeyEvent e) fonksiyonu ile aşağı,yukarı,sağ,sol yön tuşlarına hareket atadık.

KeyEvent.VK_UP: Yukarı tuşuna hareket atandı.

KeyEvent.VK_DOWN: Aşağı tuşuna hareket atandı.

KeyEvent.VK_LEFT: Sol tuşuna hareket atandı.

KeyEvent.VK_RIGHT: Sağ tuşuna hareket atandı

3. Dosya İşlemleri

BufferedReader() fonksiyonu ile .txt dosyasından karakter isimlerini, karakterlerin hangi kapıdan gireceklerini ve harita okuması yapıldı

readLine() ile .txt içinde son satıra kadar okuma yapıldı.

4.Sınıf ve Nesne Yapısı

Lokasyon Sınıfı: x ve y koordinatlarını tutmak için oluşturulmuştur.

Map Sınıfı: .txt'den karakterler,karakterlerin giriş kapıları ve harita okuması yapıldı.Bu verileri kullanmak için getter,setter metodları oluşturuldu.

Karakter Sınıfı: Ortak özellikleri ve fonksiyonları olan classlarda kullanabilmek için abstract class olarak oluşturuldu. Lokasyon sınıfından oyuncu ve düşmanların koordinatlarını tutmak için nesne alınmıştır. Public abstract void hareketEt(int xBirim,int yBirim) metodu oluşturulmuştur.

Oyuncu Sınıfı: Abstract class olarak oluşturulmuştur. Karakter sınıfı extends edilmiştir**. Public abstract int PuanGoster(int puan)** metodu oluşturulmuştur.

GozlukluSirin ve **TembelSirin** Sınıfı: Oyuncu sınıfı extends edilmiştir. PuanGoster(int puan) metodu ve hareketEt() metodu override edilmiştir.

Dusman Sınıfı: Karakter sınıfı extends edilmiştir.

Gargamel ve **Azman** Sınıfı: Dusman sınıfı extends edilmiştir. hareketEt() metodu override edilmiştir. Dijkstra ile en kısa yol bulmak için DjikstraHedefeGitme()ve dusmanYolBul() metodları oluşturulmuştur.

Board1 Sınıfı: JPanel extends edilmiştir. Karakter seçimi için butonlar oluşturulmuştur. Map, GozlukluSirin, TembelSirin, Azman ve Gargamel sınıflarından nesne alınmıştır. Oyunun görsel tasarımı ve oyuncu kontrolleri yapılmıştır. Altin ve mantar haritaya eklenmiştir.

5. Oyunun İşleyişi

.txt dosyasından düşman karakterler, kapıları ve harita okundu. Oyun başlamadan önce karakter seçimi yapıldı.Düşman karkaterlerin oyuncuyu en kısa yoldan yakalayabilmesi için dijkstra algoritması kullanıldı. Bu algoritma kullanılarak düşman karakterin oyuncuya ulaşmak için gitmesi gereken yol çizdirildi.

Oyuncunun puanı, azman oyuncuyu yakaladığında 5 puan, gargamel oyuncuyu yakaladığında 15 puan eksildi. Yakalamayla birlikte düşman karakterler kapılarına geri döndürüldü. 5 tane altın ve mantar random koordinatlarda oluşturuldu. **Thread.sleep()** fonksiyonu **ve timerTask()** kullanılarak 5 altının aynı anda farklı konumlarda oluşturulması, mantarın random bir konumda oluşturulması ve bir süre sonra haritada görünmez olması sağlandı. Oyuncunun puanı mantarı yediğinde 50, altınların birini yediğinde 5 puan arttırıldı. Skor 0 ve altına düştüğünde oyuncu oyunu kaybeder. Şirinenin kapısına ulaşırsa oyuncu oyunu kazanır.

GELİŞTİRME ORTAMI VE DİL

Proje java dilinde, Eclipse IDE kullanılarak, Windows işletim sistemi üzerinde gerçekleştirilmiştir.

DENEYSEL SONUÇLAR

Bu projede kullanılan java dilinin nesneye yönelik programlama yapısı kullanılmıştır.

Bu projede ;

Inheritance kullanılmıştır. Inheritance miras alma işlemidir. Bir sınıfın kendisine ait özellikleri ve işlevleri bir başka sınıfa aynen aktarması ya da bazı özellik ve işlevlerini diğer sınıfların kullanmasına izin vermesidir. Inheritance, daha önce yazılmış sınıf içindeki kod parçacıklarının tekrar tekrar yazılmadan başka sınıflar için kullanılmasını sağladığından iş yükünü hafifleten ve zaman kazandıran bir yöntem olduğu görülmüştür.

Metotlar override edilerek kullanılmıştır. Override etme işlemi,bir classtaki metodu başka bir classta kullanmak için süper classtaki metodun diğer classın nesnesine uygun olarak çalışmasını sağlamaktır. Override etmek aynı metodun diğer classta tekrar yazılmamasını sağladığından override işleminin iş yükünü hafifleten ve programlamacıya zaman kazandıran bir yöntem olduğu görülmüştür.

Swing kütüphanesi kullanılmıştır. Swing ile arayüz oluşturulmuştur. Swing kütüphanesinde bulunan jButton ile seçme işleminin yapılacağı butonlar oluşturulmuştur. JPanel ile oyun paneli açılmıştır. Swing kütüphanesi ve java programlama dili kullanımıyla oyun yapımı öğrenilmiştir.

SONUÇ

Bu projeyle java programlama dilinde classlarla ilgilendik ve kendimizi inheritance, metod override gibi konularda geliştirdik. Ara yüz kullanarak algoritma kurma yeteneğimizi geliştirdik. Nesneye yönelik programlamanın mantığını daha iyi kavradık ve nesneye yönelik programlamada kendimizi geliştirdik. Ara yüz aracılığıyla java dilini kullanarak 2D oyun tasarlamayı öğrendik. Daha verimli ve işlevsel bir kod nasıl yazabiliriz gibi sorularla ilgilendik. UML diyagramı çizmeyi öğrendik. Kullanıcının etkileşimde olduğu bir kod yazdık. Bu şekilde nesneye yönelik programlamanın mantığını anlamış olduk.

KAYNAKÇA

https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/

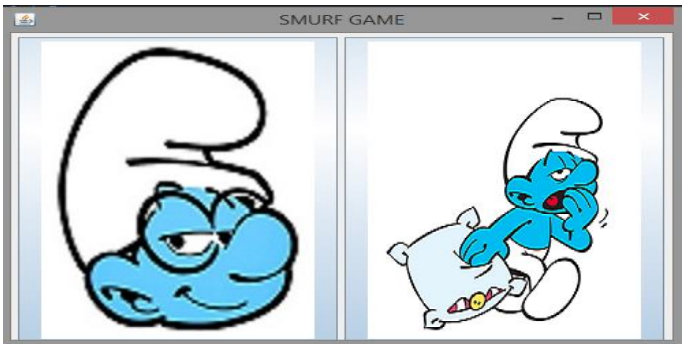
http://www.yasinhoca.com/2018/01/basit-oyun-haritas-yapm.html

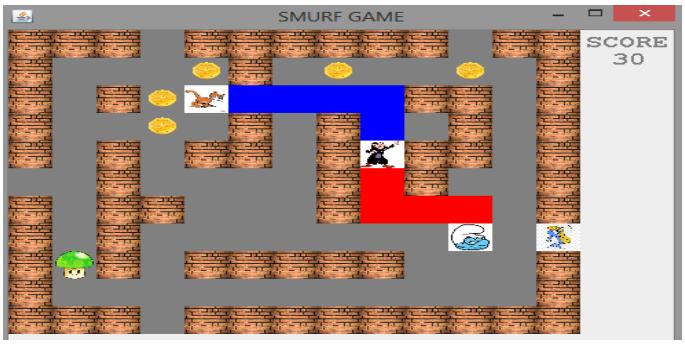
https://www.youtube.com/watch?v=bI6e6qjJ8JQ

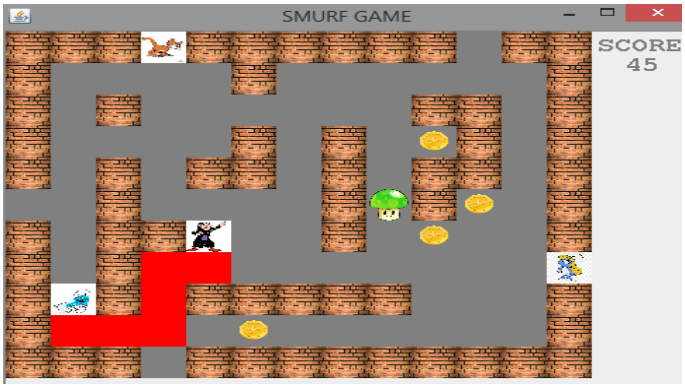
https://www.youtube.com/watch?v=SZXXnB7vSm4

https://www.youtube.com/watch?v=bImvsFNym9I

GÖRSELLER







KABA KOD:

1. .txt dosyasından oyuna giricek düşman karakterler, girecekleri kapılar ve oyunun haritası okunur.
2. JButton() ile oyuncu seçimindeki karakter butonları oluşturulur.
3. add() ile butonlar panele eklenir.
4. AddMouseListener() ile oyuncu seçimi yapılır.
5. Graphics ile oyun haritası çizilir.
6. KeyAdapter ile oyuncu yön tuşları atanır.
7. Düşman ve oyuncunun koordinatları eşit ise düşman karakterler kapıya geri döndürülür.
8. Düşman ve oyuncunun koordinatları eşit ise oyuncunun puanı azman için 5, gargamel için 15 azalır.
9. 5 tane altını random koordinatlarda oluştur ve haritaya çiz.
10. Thread.sleep ile altının ne kadar süre ekranda görüneceğini kontrol et.
11. TimerTask ile altınların kaç saniyede bir oluşturulacağını kontrol et.
12. Mantarı random koordinatlarda oluştur ve haritaya çiz.
13. Thread.sleep ile mantarın ne kadar süre ekranda görüneceğini kontrol et.
14. TimerTask ile mantarın kaç saniyede bir oluşturulacağını kontrol et.
15. Oyuncuyla mantarın koordinatları eşit ise oyuncunun puanını 50, oyuncunun altınlardan biriyle koordinatları eşit ise puanını 5 arttır.
16. Oyuncunun puanı 0 ve 0'ın altına düşerse oyuncu oyunu kaybeder.
17. Ekrana "game over" yazısı ve resmi basılır.
18. Oyuncu şirinenin kapısına ulaşırsa oyunu kazanır.
19. Ekrana "winner" yazısı ve resmi basılır.

ALGORİTMA KARMAŞIKLIĞI

1. **BELLEK KARMAŞIKLIĞI**
   Kullanılan her bir değişkenin bellekte kapladığı yeri hesaplayacak olursak ;

```
private int[][] harita = new int[11][13];
public int[][] dusmanYol = new int[100][2];
int mesafe = 0;
```

Her bir int değişkeni=4 byte;
Yukarıdaki değişkenler 4*3=12 byte yer kaplar.

Aşağıda kod parçacıkları şeklinde verilen Algoritmayı incelediğimizde 5 tane int değişkeni tanımlıdır. 5*4 =20 byte

Toplamda bu algoritma için 32 bytlık yer ayrılmıştır.

2.ZAMAN KARMAŞIKLIĞI

```
public void enKisaYol(int oyuncuY,int oyuncuX) {
    if(m.isGargamelMi()) {
    int komsuSayisi=1;
      for (int v = 0; v < komsuSayisi; v++) {


          for (int i = 0; i < 11; i++) {
              for (int j = 0; j < 13; j++) {
                  harita[i][j] = Integer.MAX_VALUE;
              }
          }
          harita[this.getLokasyon_y()][this.getLokasyon_x()] = 0;
          dusmanYolBul(this.getLokasyon_y() - 1, this.getLokasyon_x(), 1);
          dusmanYolBul(this.getLokasyon_y() + 1, this.getLokasyon_x(), 1);
          dusmanYolBul(this.getLokasyon_y(), this.getLokasyon_x() - 1, 1);
          dusmanYolBul(this.getLokasyon_y(), this.getLokasyon_x() + 1, 1);
```

İçerideki forlar sabit sayı ile döndüğü için karmaşıklığı O(1) dışarıdaki for komsuSayisi kadar döndüğü için bu kod parçacığının karmaşıklığı O(n)dir.

```
mesafe = harita[oyuncuY][oyuncuX];
dusmanYol[mesafe][0] =oyuncuY;
dusmanYol[mesafe][1] = oyuncuX;
for (int i = mesafe - 1; i > -1; i--) {
    if (harita[dusmanYol[i + 1][0] + 1][dusmanYol[i + 1][1]] == i) {
        dusmanYol[i][0] = dusmanYol[i + 1][0] + 1;
        dusmanYol[i][1] = dusmanYol[i + 1][1];
    }  else if (harita[dusmanYol[i + 1][0]][dusmanYol[i + 1][1] - 1] == i) {
        dusmanYol[i][0] = dusmanYol[i + 1][0];
        dusmanYol[i][1] = dusmanYol[i + 1][1] - 1;

    } else if (harita[dusmanYol[i + 1][0]][dusmanYol[i + 1][1] + 1] == i) {
        dusmanYol[i][0] = dusmanYol[i + 1][0];
        dusmanYol[i][1] = dusmanYol[i + 1][1] + 1;
    } else if (harita[dusmanYol[i + 1][0] - 1][dusmanYol[i + 1][1]] == i) {
        dusmanYol[i][0] = dusmanYol[i + 1][0] - 1;
        dusmanYol[i][1] = dusmanYol[i + 1][1];
    }
}
```

Bu for n-1 kere yukarıdaki ana forun içinde döndüğünden dolayı bu kod parçacığının zaman karmaşıklığı O(n^2)dir.

```
public void dusmanYolBul(int oyuncuY, int oyuncuX, int mesafe) {
    if ((oyuncuX > -1) && (oyuncuY > -1) && (oyuncuY < 10) && (oyuncuX < 12)) {

        if ((m.getMap(oyuncuY, oyuncuX)!=0) && (harita[oyuncuY][oyuncuX] > mesafe)) {
            harita[oyuncuY][oyuncuX] = mesafe-1;
            dusmanYolBul(oyuncuY - 1, oyuncuX, mesafe + 1);
            dusmanYolBul(oyuncuY + 1, oyuncuX, mesafe + 1);
            dusmanYolBul(oyuncuY, oyuncuX - 1, mesafe + 1);
            dusmanYolBul(oyuncuY, oyuncuX + 1, mesafe + 1);
        }
```

Bu fonksiyonda recursive fonksiyon kullanıldığı için zaman karmaşıklığı teleskop yöntemiyle hesaplanır aynı fonksiyon 4 defa çağrılmasına rağmen aynı zaman karmaşıklığına sahip olduklarından bu kod parçacığının zaman karmaşıklığı O(logn)dir.

Algoritmanın zaman karmaşıklığı bulunurken worst case durumuna bakıldığından dolayı yukarıda parça parça verilen algoritmanın zaman karmaşıklığı O(n^2)dir.

ZAMAN KARMAŞIKLIĞI : O(n^2)

UML DİYAGRAMI