
# Yıldız Teknik Üniversitesi

## Elektrik-Elektronik Fakültesi

## Bilgisayar Mühendisliği

**BLM1012**

**Yapısal Programlamaya Giriş**

**Gr:1**

**Prof. Dr. Mehmet Fatih Amasyalı**

**Dönem Projesi**

**İsim:** Yusuf Safa Köksal

**Numara:** 21011002

**E posta:** yusuf68815@gmail.com

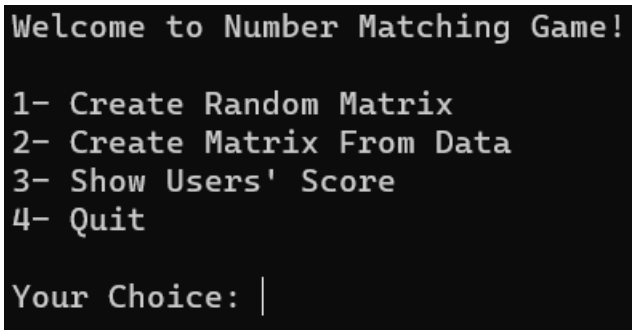
# İçindekiler

# Ön Bilgi

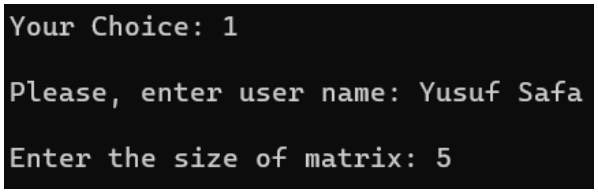
Yazılan kod otomatik modda tam olarak doğru çalışmamaktadır. Yani otomatik modda bazı matrisler için sonuç tam olarak bulunmuyor. Onun dışında istenen bütün özellikler eksiksiz bir şekilde fazlasıyla yapılmış olup doğru çalışıyor.

# Adım Adım Programın Çalışması

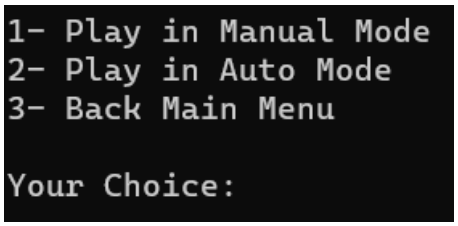
Oyun açıldığında kullanıcı böyle bir menü ekranıyla karşılaşır ve seçim yapması beklenir. Eğer uygun bir sayı girmezse uygun sayı girmesi için hata mesajı verilir ve 1 saniye o mesaj kaldıktan sonra ekran temizlenir ve yeniden seçim yapması beklenir.



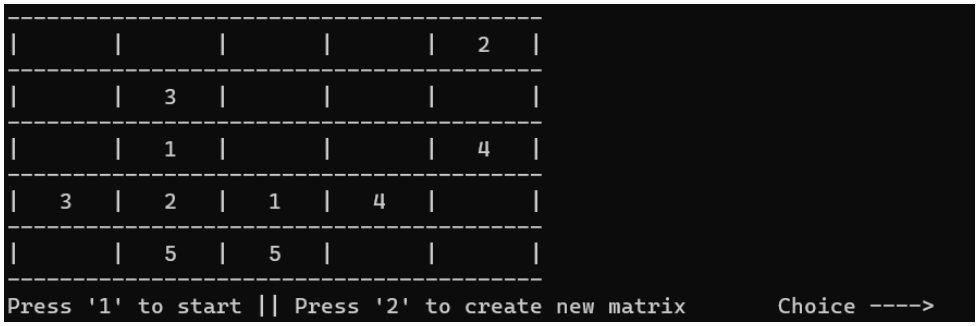

Kullanıcı eğer rastgele matris oluşturma yolunu seçerse önce kullanıcıdan adı sorulur. Daha sonra ise oluşturmak istediği matrisin boyutu istenir.



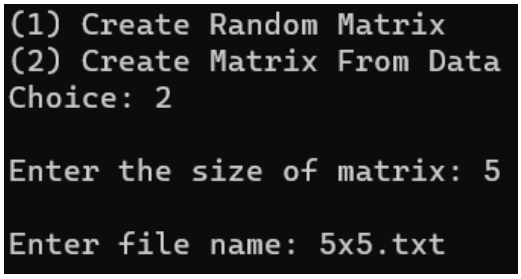

Daha sonra alt menü ekrana gelir. Kullanıcı burada oynamak istediği oyun modunu seçer veya menüye geri döner.



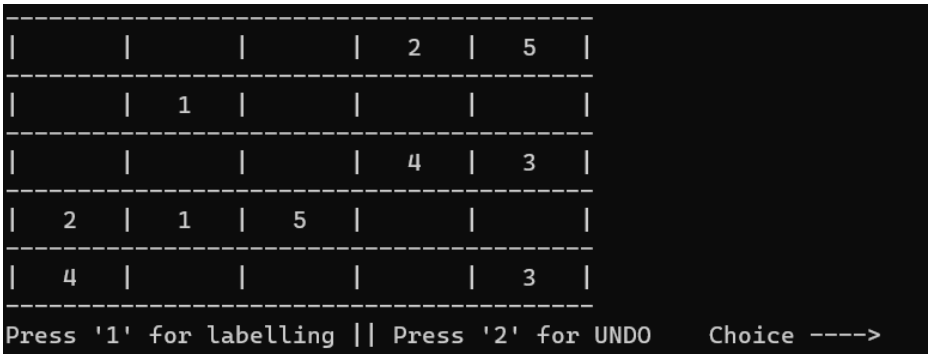
Kullanıcı eğer manuel oyun modunu seçerse önce ekrana oluşturulan matris yazdırılır ve kullanıcıya bu matris ile başlaması veya yeni bir matris oluşturması yönünde 2 farklı seçenek sunulur.



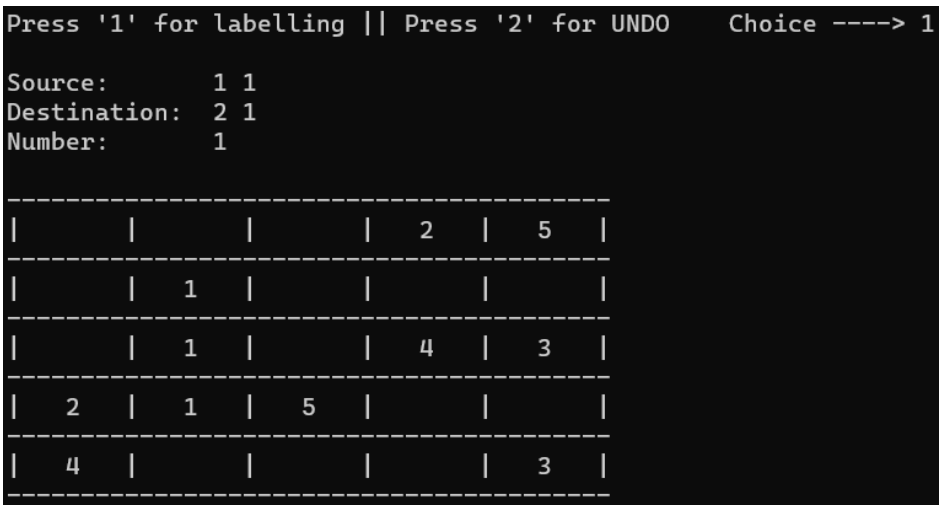

Kullanıcı eğer yeni matris oluşturmayı seçerse matrisi nasıl oluşturmak istediği sorulur. Örneğin bu kez de matrisi data'dan çekip oluşturalım. Bu durumda yine ilk olarak boyut istenir. Ardından da matrisin bilgilerini tutan text dosyasının ismi kullanıcıdan istenir.



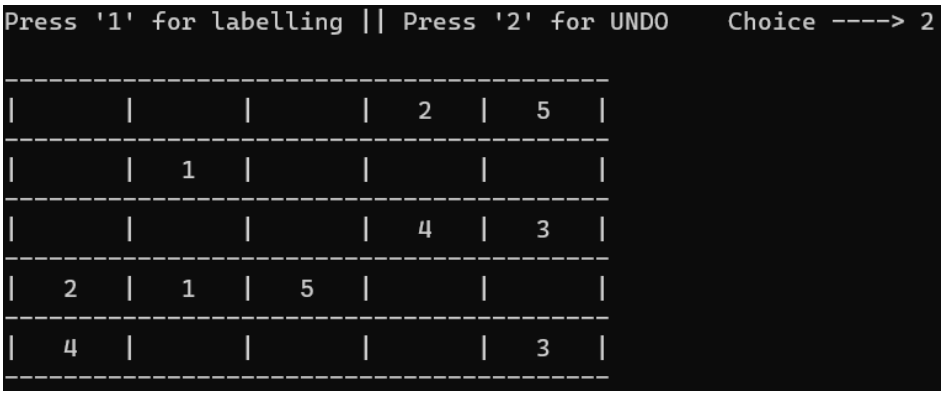

Yeni oluşturduğumuz matrisle oyuna başla seçeneğini seçip oyuna başlayalım. Bu kez kullanıcıyı yine 2 seçenek karşılar ve bu her iterasyonda tekrar sorulur. İlki eşleşmeleri yapmak için label seçeneği, ikincisi ise hata yapıldığında geri almak için undo seçeneği.



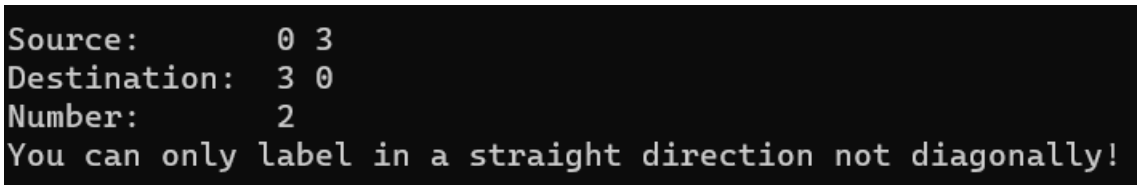
Label seçeneği seçmemiz durumunda ise bizden 3 bilgi istiyor. İlki numaralandırmanın hangi indisten başlayacağı. İkincisi numaralandırmanın hangi indiste sonlanacağı. Sonuncusu ise yerleştirilecek numara. Bilgiler girildikten sonra matris aşağıdaki hale geliyor.




Kullanıcı eğer hata yaptığını farkederse undo işlemi yapacaktır. Örneğin yukarıdaki hamleden sonra undo yapılırsa matris bir önceki haline geri getirilir.

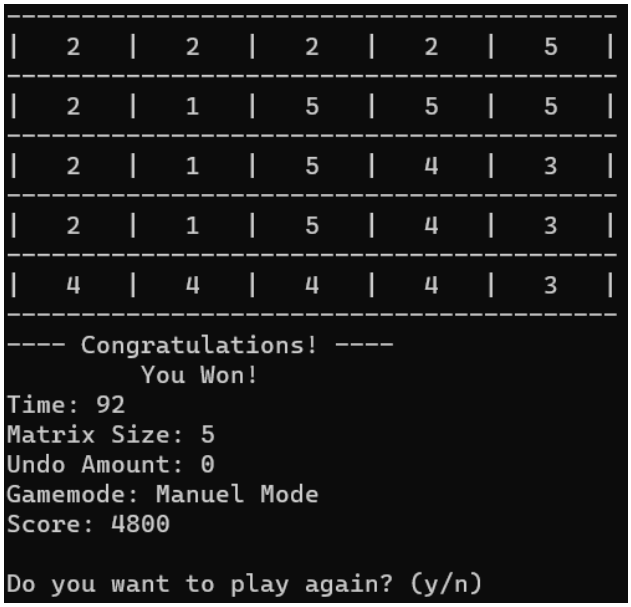



Sadece 4 yöne numaralandırma işlemi yapılacağından eğer kullanıcı uygun koordinat girmezse yalnızca düz bir yönde numaralandırma yapılacağını söyleyen hata mesajı yazdırılır.



Eğer oyun bitirilirse aşağıdaki kullanıcı aşağıdaki gibi bir ekranla karşılaşır. Oyunu bitirme süresi, matrisin boyutu, yapılan geri alma sayısı, oyun modu ve bu kriterlerden hesaplanan skor ekrana yazdırılır.

Ardından kullanıcıya başka bir oyun oynayıp oynanamamak istediği sorulur. Evet derse aynı kullanıcı üzerinden başka bir oyun başlar. Hayır derse geçerli kullanıcının oturumu kapanır ve menüye dönülür.




Eğer aynı matrisi bir de ‘Kullanıcı2’ adlı oyuncu ile otomatik modda oynamak istersek yine önce kullanıcıya geçerli matris ile başlamak isteyip istemediği sorulur.

Oyuna başlayı seçmemiz durumunda 5 ve 5'in altı boyutlu matrisler için işlemler şekildeki gibi raporlanır.

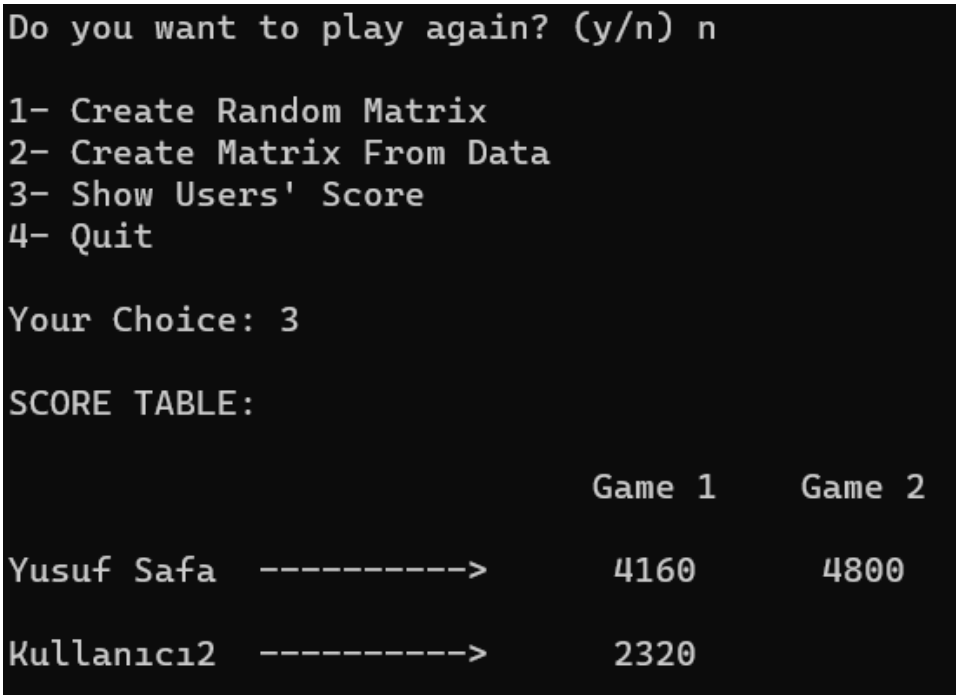
```
Source-->(1 , 1)     Destination-->(2 , 1)    Number-->1
Source-->(0 , 3)     Destination-->(1 , 3)    Number-->2
Source-->(1 , 3)     Destination-->(1 , 2)    Number-->2
Source-->(1 , 2)     Destination-->(0 , 2)    Number-->2
Source-->(0 , 2)     Destination-->(0 , 0)    Number-->2
Source-->(0 , 0)     Destination-->(2 , 0)    Number-->2
Source-->(2 , 4)     Destination-->(3 , 4)    Number-->3
Source-->(2 , 3)     Destination-->(4 , 3)    Number-->4
Source-->(4 , 3)     Destination-->(4 , 1)    Number-->4
Undo
Undo
Undo
Undo
Undo
Undo
Undo
Undo
Undo
Source-->(0 , 4)     Destination-->(1 , 4)    Number-->5
Source-->(1 , 4)     Destination-->(1 , 2)    Number-->5
Source-->(1 , 2)     Destination-->(2 , 2)    Number-->5
Source-->(0 , 3)     Destination-->(0 , 0)    Number-->2
Source-->(0 , 0)     Destination-->(2 , 0)    Number-->2
Source-->(2 , 4)     Destination-->(3 , 4)    Number-->3
Source-->(2 , 3)     Destination-->(4 , 3)    Number-->4
Source-->(4 , 3)     Destination-->(4 , 1)    Number-->4
```

Matrisin son hali yazdırılır ve manuel moddaki gibi bir bitiş ekranı kullanıcıyı karşılar. Kullanıcıya kazandınız mesajıyla birlikte ayrıntılar verilir ve başka oyun oynayıp oynamayacağı sorulur.

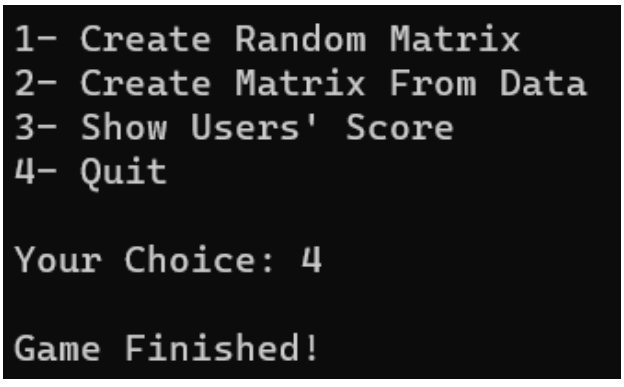
```
-----------------------------------
|  2  |  2  |  2  |  2  |  5  |
-----------------------------------
|  2  |  1  |  5  |  5  |  5  |
-----------------------------------
|  2  |  1  |  5  |  4  |  3  |
-----------------------------------
|  2  |  1  |  5  |  4  |  3  |
-----------------------------------
|  4  |  4  |  4  |  4  |  3  |
-----------------------------------
---- Congratulations! ----
        You Won!
Time: 0
Matrix Size: 5
Undo Amount: 9
Gamemode: Auto Mode
Score: 2320

Do you want to play again? (y/n)
```

Menüye döndükten sonra kullanıcıların skorlarını göster seçeneği seçilirse bir skor tablosu ekrana yazdırılır. (Skor tablosunun düzgün gözükmesi için arka planda başka oyunlar da oynanmıştır.)




Menüde oyundan çıkmayı seçersek oyun bitti mesajı yazdırılır ve program sonlanır.



# Manuel Mod Kod Analizi

Öncelikle matris yazdırılır ve kullanıcıya oyuna bu matrisle başlayıp isteyip istemediği sorulur. Eğer istemezse bu sefer yeni oluşturacağı matrisin türünü seçmesi beklenir. Seçtiği türe göre fonksiyon çalıştırılıp matris ve boyutu güncellenir ve manuel oyun modu çağırılarak yeniden başlar.

```
printGameMatrix(gameMatrix,len);
printf("\nPress '1' to start || Press '2' to create new matrix       Choice ----> ");
scanf("%d",&start);
if(start==2)
{
    printf("\n(1) Create Random Matrix\n(2) Create Matrix From Data\nChoice: ");
    scanf("%d",&newMatrixType);

    if(newMatrixType==1)
        createRandomMatrix(gameMatrix,&len);
    else if(newMatrixType==2)
        createMatrixFromData(gameMatrix,&len);

    manuelMode(gameMatrix,usersScores,len,person,whichRound);
    return;
}
```

Oyuna başlayı seçtikten sonra startTime o anki zaman ile güncellenir. Ve while döngüsüne girer. Bu döngü oyun bitene kadar döner. Ve her adımın sonunda matrisin bütün elemanları sıfırdan farklı mı diye kontrol eder ve farklıysa finishControl değişkenini 0 yapar ve oyun biter. Her adımın başında matrisin o anki güncel halini ekrana yazdırır.

```
startTime=time(NULL);

while(finishControl)
{
    printGameMatrix(gameMatrix,len);
```

While'ın içinde önce kullanıcıya ne yapmak istediği sorulur eğer 'UNDO'yu seçerse tutulan geçmiş matris dizisinin son indisindekini çeker ve yapılan undo sayısını bir arttırır. Eğer 'label' seçeneğini seçerse 'source', 'destination' ve 'number' değerleri kullanıcıdan istenir ve label işlemini yapmak için yazılan fonksiyona değerler parametre olarak gönderilir.

```
printf("\nPress '1' for labelling || Press '2' for UNDO    Choice ----> ");
scanf("%d",&choice);

if(choice==1)
{
    pushTmpGameMatrix(gameMatrix,tmpGameMatrix,len,&currStackAmount);
    printf("\nSource:       ");
    scanf("%d %d",&source[0],&source[1]);
    printf("Destination:  ");
    scanf("%d %d",&dest[0],&dest[1]);
    printf("Number:       ");
    scanf("%d",&label);
    labelGameMatrix(gameMatrix,len,source,dest,label);
}
else if(choice==2)
{
    popTmpGameMatrix(gameMatrix,tmpGameMatrix,len,&currStackAmount);
    undoAmount++;
}
else
    printf("\nUndefined choice! Please, only press '1' or '2'.");
```

'labelGameMatrix' fonksiyonu ise şöyle çalışır: Önce gönderilen source ve dest dizilerinde row değerleri mi veya column değerleri mi aynı diye kontrol edilir. Eğer ikisi de aynı değilse zaten düz bir label işlemi yapılamaz bu yüzden hata mesajı yazdırılır. Örneğin row değerleri aynı olması durumunda bu seferde hangisinde column değerinin büyük olduğu bulunur ve ona göre source'dan başlayarak sağa veya sola giderek label işlemi yapılır. Eğer column değerleri aynı olsaydı bu işlemin benzeri yapılacaktı.

```
int i;

if(source[0]==dest[0])
{
    if(source[1]>dest[1])
    {
        for(i=source[1]-1;i>=dest[1];i--)
        {
            gameMatrix[source[0]][i]=label;
        }
    }
    else if(source[1]<dest[1])
    {
        for(i=source[1]+1;i<=dest[1];i++)
        {
            gameMatrix[source[0]][i]=label;
        }
    }
}
```

İlk başta yazıldığı gibi label işlemi bittikten sonra matrisin bütün elemanları 0'dan farklı mı diye bu şekilde kontrol edilir. Eğer bir tane bile 0 olan elemanı varsa finishControl=1 yapılır yani while döngüsünden çıkamaz ve yeniden başlar

```
finishControl=0;
for(i=0;i<len;i++) // hepsi 0 dan farkli olunca oyun biter
{
    for(j=0;j<len;j++)
    {
        if(gameMatrix[i][j]==0)
            finishControl=1;
    }
}
```

While döngüsünden çıktıktan sonra önce matrisin son hali yazdırılır. Sonra arada geçen zaman 'elapsedTime' değişkenine atanır. Bu seferde kullanıcı oyunu doğru bir şekilde bitirdi mi diye kontrol yapılır.

Bu kontrolün mantığı şöyle çalışır. For döngüsüyle matriste 1'den başlayarak numaraların yerini bulur. Bulduğu numara ile direkt olarak connected component mantığı ile kendini ve komşuda bulunan aynı değerdeki numaraları matriste 0 yapar. Bu işin sonunda eğer matriste 0'ın harici bir eleman kalırsa kullanıcı oyunu doğru bitirmemiş demektir bu yüzden winControl fonksiyonundan 0 döner "Game Over" yazdırılır ve kullanıcının geçerli rounddaki oyununa skor olarak -100 atanır. Eğer 1 dönerse skorlar matrisinin geçerli kullanıcısının geçerli rounduna 'calculateScore' fonksiyonunun hesapladığı değer atanır. Bu fonksiyon içinde aynı zamanda oynanan oyun sonu bilgileri de yazdırılır.

```
printGameMatrix(gameMatrix,len);
endTime=time(NULL);
elapsedTime=difftime(endTime,startTime);

win=winControl(gameMatrix,len);
if(win)
{
    usersScores[*person][whichRound]=calculateScore(1,len,elapsedTime,undoAmount);
}
else
{
    printf("\n---- Game Over! ----");
    usersScores[*person][whichRound]=-100;
}
```

# Otomatik Mod Kod Analizi

Öncelikle matris yazdırılır ve kullanıcıya oyuna bu matrisle başlayıp isteyip istemediği sorulur. Eğer istemezse bu sefer yeni oluşturacağı matrisin türünü seçmesi beklenir. Seçtiği türe göre fonksiyon çalıştırılıp matris ve boyutu güncellenir ve otomatik oyun modu çağırılarak yeniden başlar.

```
printGameMatrix(gameMatrix,len);
printf("\nPress '1' to start || Press '2' to create new matrix       Choice ----> ");
scanf("%d",&start);
if(start==2)
{
    printf("\n(1) Create Random Matrix\n(2) Create Matrix From Data\nChoice: ");
    scanf("%d",&newMatrixType);

    if(newMatrixType==1)
        createRandomMatrix(gameMatrix,&len);
    else if(newMatrixType==2)
        createMatrixFromData(gameMatrix,&len);

    autoMode(gameMatrix,usersScores,len,person,whichRound);
    return;
}
```

Öncelikle oyun başladığı an ‘startTime’ değişkenine geçerli zaman atanır. Ve aşağıdaki while döngüsüne girer. Döngü number=1 için başlar ve len sayısına eşit oluncaya kadar devam eder. Her number değeri için oyun matrisinde o sayının bulunduğu ilk yeri ve son yerini diziye kaydeder.

```
startTime=time(NULL);

while(number<=len)
{
    for(i=0;i<len;i++)
    {
        for(j=0;j<len;j++)
        {
            if(gameMatrix[i][j]==number)
            {
                queue++;
                if(queue%2==1)
                    numberCoordinate[number][0]=i, numberCoordinate[number][1]=j;
                else
                    numberCoordinate[number][2]=i, numberCoordinate[number][3]=j;
            }
        }
    }
    number++;
}
```

Number 1’e eşitlenir ve ana işlemimizin gerçekleştiği while döngüsüne girilir. Önce ilk if’le ‘pathControl’ fonksiyonuna o anki number değerinin matriste eşleştirmesinin tamamlanıp tamamlanmadığının kontrolü yaptırılır. Eğer tamamlanmamış ise if’e girer ve matriste geçerli number değeri için yol bulmayı sağlayan ‘labelAuto’ fonksiyonu çalıştırılır. Bu fonksiyona pointer olarak verilen ‘isArrived’ değişkeni eşleştirmenin yapılıp yapılamadığını belirtir eğer yapılamadı ise 0 olarak atanır ve bu yüzden geri almak için yazılan fonksiyon çağırılır. Hedefe ulaşana kadar geri alma yapılır. Ulaştıktan sonra if’ten çıkar ve number değeri bir arttırılır. (İf’in içindeki ‘iteration’ değişkeni ise bazı matrislerde dediğim gibi otomatik mod çalışmıyor ve burada sonsuz loop’a giriyor bu yüzden bunu engellemek amacıyla bunu yaptım bu şekilde hatalı da olsa sonuç matrisini yazdırmaya çalıştım.)

```
number=1;
while(number<=len)
{
    if(!pathControl(gameMatrix,numberCoordinate,lastPaths,number,numberCoordinate[number][0],numberCoordinate[number][1],0,0))
    {
        isArrived=0;
        do{
            labelAuto(gameMatrix,path,tmpGameMatrix,numberCoordinate,len,number,numberCoordinate[number][0],numberCoordinate[number]
            if(isArrived==0)
            {
                undoAmount++;
                iteration++;
                if(iteration>=100)
                    isArrived=1;
                if(len<=5)
                    printf("\nUndo");
                popTmpGameMatrix(gameMatrix,tmpGameMatrix,len,&currStackAmount);
            }
        }while(isArrived==0);
    }
    number++;
```

Number, len değerine gelene kadar bu döngü böyle devam eder yolları bulmaya çalışır. Eğer number son değerine gelirse bir kontrol yapılır. Her number için teker teker ‘pathControl’ fonksiyonu ile eşleşmesinin doğru olarak tamamlanıp tamamlanmadığı kontrol edilir. Tamamlanmayan tespit edildiği anda number değeri o sayıya atanır i değeri ise for döngüsünden çıkması için len+1’e atanır ve ana while döngüsü baştan başlar yani oyun bitmez.(İteration sayısı 100’den fazlaysa sonsuz loop’a girmiştir bu yüzden while’den çıkıp sonucu bastırması için number değişkeni değiştiriliyor.)

```
    number++;

    if(number==len+1)
    {
        for(i=1;i<=len;i++)
        {
            if(!pathControl(gameMatrix,numberCoordinate,lastPaths,i,numberCoordinate[i][0],numberCoordinate[i][1],0,0))
            {
                number=i;
                i=len+1;
            }
        }
    }

    if(iteration>=100)
        number=len+1;
}
```

While’dan çıktıktan sonra oyunun bitmiş hali yazdırılır. ‘elapsedTime’ değişkenine geçen zaman atanır. Eğer iteration sayısı 100’den büyük ise kod doğru çalışmamıştır sonsuz loop’a girmiştir bu yüzden “Game Over” yazdırılır ve geçerli kullanıcının geçerli oyununun sırasının skoruna -100 atanır. Eğer değil ise geçerli kullanıcının geçerli oyununun sırasının skoruna ‘calculateScore’ fonksiyonunun hesapladığı değer atanır ve o fonksiyon içinde oyun sonu bilgileri ekrana yazdırılır.

```
printGameMatrix(gameMatrix,len);
endTime=time(NULL);
elapsedTime=difftime(endTime,startTime);

if(iteration>=100)
{
    usersScores[*person][whichRound]=-100;
    printf("\n---- Game Over! ----\n(Couldn't be resolved exactly)");
}
else
    usersScores[*person][whichRound]=calculateScore(2,len,elapsedTime,undoAmount);
```

## Otomatik Modda Yol Bulma Fonksiyonu

Öncelikle geçerli row ve column'a komşu olan indisler kontrol edilir. Eğer kontrol edilen noktanın koordinatları geçerli yolu bulunan sayının bitiş noktasının koordinatları ile aynıysa eşleşme bitmiş demektir. Geçerli row ve column değeri de path fonksiyonuna atanır. Burada bulduğumuz yolları path dizisine kaydediyoruz. Eğer hedefe ulaşırsak 'printPath' fonksiyonu yardımıyla koordinatları matriste number değeri ile güncelliyoruz. Path dizisinin 2. Sütununda tutulan direction değeri de o anki yola giderken kullanılan son yönü ifade ediyor. Böyle bir şeye ihtiyaç duydum çünkü hem yol bulurken geldiği tarafa geri dönmesini engellemek gerekiyordu hem de bu yollar ile matriste güncelleme yapmak için 'printPath' fonksiyonunun içinde 'labelGameMatrix' fonksiyonunu çalıştırdığımızda bizden yolları düz bir şekilde istiyor. 'direction' kullanarak bu sorunları çözdüm. (1 yönü yukarıyı, 2 yönü sağı, 3 yönü aşağıyı ve 4 yönü solu temsil ediyor.) Pointer olan 'isArrived' değeri ise hedefe başarıyla ulaştığımızı belirtiyor bu yüzden onu 1 ile güncelliyoruz. Eğer bütün yollar denenip de hedefe ulaşamazsa 'isArrived' değeri 0 olarak kalır ki bu da uygun yol bulunamadı demektir. Bu yüzden ana otomatik mod fonksiyonumuza geri döner ve 1 kere geri alma çalışır.

```
if(y-1==numberCoordinate[number][2] && x==numberCoordinate[number][3] || y+1==numberCoordinate[number][2] && x==numberCoo
{
    path[currIndex][0]=y;
    path[currIndex][1]=x;
    path[currIndex][2]=direction;
    printPath(path,gameMatrix,tmpGameMatrix,number,len,currIndex,currStackAmount);
    *isArrived=1;
    return;
}
```

Eğer o anki index sayısı 0 dan büyükse ve geçerli koordinatın matristeki değeri 0'a eşitse bu koordinatları  ve yön değerini 'path' dizisine ekler. Eğer index 0'a eşit ise fonksiyona ilk girişi demektir yani o koordinatlarda sayının başladığı değer yer alıyor bu yüzden bunu da 'path' dizisine ekler.

```
if( (gameMatrix[y][x]==0 && currIndex>0) || (gameMatrix[y][x]==number && currIndex==0) ) // gecerli koordinati path'e ekleme
{
    path[currIndex][0]=y;
    path[currIndex][1]=x;
    path[currIndex][2]=direction;
}
```

Sırasıyla yer alan aşağıdaki döngülerde ise ilk önce bir önceki geldiğimiz yön sıradaki yol için uygun mu kontrolü yapılır eğer uygunsa aynı yönden devam eder.(Bu kontrolü yapmak olasılıkları büyük ölçüde azaltıp daha hızlı sonuca götürüyor).Ardından geldiğimiz taraf hariç 3 tarafın da uygun olup olmadığı kontrol edilir eğer uygunsa o taraftan devam eden recursive fonksiyonumuz çağrılır. 'noWay' değişkeni ise her fonksiyona girdiğimizde 1 olarak atanır. Eğer bir tarafa gidebiliyorsa 0 olarak atanır. Eğer gidemiyorsa bu yol dizisinin sonlanması için return ile çıkılır. Eğer 'isArrived' pointer'ı 1 olmuşsa uygun yol bulunduğu için daha bu şartlara girmez.

```
if(direction==1 && gameMatrix[y-1][x]==0 && y-1>=0 && *isArrived!=1)
{
    noWay=0;
    labelAuto(gameMatrix,path,tmpGameMatrix,numberCoordinate,len,number,y-1,x,currIndex+1,1,currStackAmount,isArrived);
}
else if(direction==2 && gameMatrix[y][x+1]==0 && x+1<len && *isArrived!=1)
{
    noWay=0;
    labelAuto(gameMatrix,path,tmpGameMatrix,numberCoordinate,len,number,y,x+1,currIndex+1,2,currStackAmount,isArrived);
}
else if(direction==3 && gameMatrix[y+1][x]==0 && y+1<len && *isArrived!=1)
{
    noWay=0;
    labelAuto(gameMatrix,path,tmpGameMatrix,numberCoordinate,len,number,y+1,x,currIndex+1,3,currStackAmount,isArrived);
}
else if(direction==4 && gameMatrix[y][x-1]==0 && x-1>=0 && *isArrived!=1)
{
    noWay=0;
    labelAuto(gameMatrix,path,tmpGameMatrix,numberCoordinate,len,number,y,x-1,currIndex+1,4,currStackAmount,isArrived);
}
if(gameMatrix[y+1][x]==0 && direction!=1 && y+1<len && *isArrived!=1)
{
    noWay=0;
    labelAuto(gameMatrix,path,tmpGameMatrix,numberCoordinate,len,number,y+1,x,currIndex+1,3,currStackAmount,isArrived);
}
if(gameMatrix[y-1][x]==0 && direction!=3 && y-1>=0 && *isArrived!=1)
{
    noWay=0;
    labelAuto(gameMatrix,path,tmpGameMatrix,numberCoordinate,len,number,y-1,x,currIndex+1,1,currStackAmount,isArrived);
}
if(gameMatrix[y][x+1]==0 && direction!=4 && x+1<len && *isArrived!=1)
{
    noWay=0;
    labelAuto(gameMatrix,path,tmpGameMatrix,numberCoordinate,len,number,y,x+1,currIndex+1,2,currStackAmount,isArrived);
}
if(gameMatrix[y][x-1]==0 && direction!=2 && x-1>=0 && *isArrived!=1)
{
    noWay=0;
    labelAuto(gameMatrix,path,tmpGameMatrix,numberCoordinate,len,number,y,x-1,currIndex+1,4,currStackAmount,isArrived);
}
if(noWay)
    return;
```

# Manuel Mod Algoritma Karmaşıklığı

Best case için kullanıcının n (matris boyutu) hamlede oyunu bitirdiğini varsayarsak while döngüsü n defa döner. Her defasında da sonda bitmiş mi diye 2 iç içe for'dan dolayı $n^2$ olur. Toplam while döngüsünden $n^3$ karmaşıklık gelir. While'dan çıkınca kullanıcı doğru yapmış mı diye kontrol edilen 'winControl' fonksiyonundan da ilk önce bir while ve iç içe üç for'un içinde çalışan bir recursive fonksiyondan $n^4$ karmaşıklık gelir. (Recursive fonksiyonun karmaşıklığı ortalama n olarak kabul edilir) Ardından çalışan iç içe 2 for'dan da $n^2$ gelir.

Böylece manuel mod karmaşıklığı:

$$T(n) = n^3 + n^4 + n^2 \quad \text{ifadesinden}$$

$$T(n) = n^4 \quad \text{olarak bulunur.}$$

# Otomatik Mod Algoritma Karmaşıklığı

Yine best case için otomatik mod karmaşıklığını incelediğimizde öncelikle sayıların yerini bulmaktan dolayı ilk başta $n^3$ geliyor. Best case de her sayının yolunun ilk başta bulunacağını varsaydığımız için ana while döngüsü n defa döner. İf'in içindeki 'pathControl' fonksiyonu geçerli sayının herhangi bir boyama durumu olmadığı için 1 defa döner bu yüzden dikkate almayız. 'labelAuto' fonksiyonunun da best case de n defada yolu bulduğunu varsayalım. Len-1 sayı için bu döngünün karmaşıklığı (n-1).n olur. Son sayı için döndüğünde ise son if'e gireceği için $n + n^2.n!$ daha eklenir. Buradaki faktöriyelin sebebi 'pathControl' fonksiyonunun içinde geçmiş path'leri de kontrol etmek için çağırılan 'lastPathControl' fonksiyonudur.

Böylece best case için otomatik kod karmaşıklığı:

$T(n) = n^3 + (n-1)n + (n + n^2.n!)$ ifadesinden

$T(n) = n^3.(n-1)!$ olarak bulunur.

# Otomatik Mod ve Skor Tablosu Örnekleri