Yıldız Teknik Üniversitesi

Bilgisayar Bilimlerine Giriş -- BLM1011 Dönem Projesi Raporu

İSİM : YAKUP GÜLCAN

ÖĞRENCİ NUMARASI : 23011102

1.1 - Problem Tanımı

TETRİS oyunumuzda bizden oyun için önce bir menü, menü kısmının ardından oyuncuya oyun tahtasının boyutunun sorulması ve oyunun başlatılması istenmiş. Her turda rastgele parçaların üretilmesi, bu parçaların kullanıcı istediğinde döndürülebilmesi ve kullanıcıdan yerleştirilecek bloğun sol parçası için koordinat girdikten sonra verilen bloğun oyun tahtasına uygun şekilde yerleştirmemiz gerekiyordu. Ayrıca herhangi bir hat dolduğunda bu hattın kırılması ve bu kırılmalardan elde edilen puanların kaydedilmesi istenmiş. Genel olarak TETRİS oyunu mantığına uygun bir oyunu C dilinde kodlamamız gerekiyor.

1.2 - Süreç ve Genel Olarak Çözüm

Öncelikle uzun ve öğretici bir çalışma oldu. Kullanmamız gereken pek çok algoritmayı önce düşünmemiz ardından bundan somut bir çözüm üretip, bunları tasarlamamızı gerektiren bir görevdi.

Öncelikle menü kısmında oyunumuzu yazı ile tanıtıp, yapılması gerekenleri anlatan tanıtım kısmının ardından oyunu oyun tahtasının boyutu sorularak başlattık. Oyun alanımızın boyutunu aldıktan sonra bir döngü ile Oyun alanımızı '.' Sembolü ile doldurduk ve tablomuza ilk şeklini verdik. "yazdir()" fonksiyonu ile oyun alanımızın son halini ekrana bastıran, bunun yanında maximum skor ve güncel oynanan oyundaki skoru gösterecek şekilde ekrana yazdırma işlemlerimizi bu fonksiyon ile yazdık. "yeniBlokTemizle()" fonksiyonumu ile yeni bloğumuzun bilgisinin tutulacağı matrisi ' ' boşluk karakteri ile doldurarak önceden tutulan bloğu matrisimizden temizledik. Ardından her yeni blok için gereken koşula girdiğinde önce 'yeniBlok' adlı matrisimize tetriminomuzun şeklini çizdirdik ve "yeni_blok_yazdir()" fonksiyonumuz ile de matrisimize şekil çizildikten sonra matrisimizi ekrana bastırdık. Kullanıcımızdan döndürüp döndürmemişini de istedikten sonra bloğu nereye yerleştirmek istediğini soruyoruz. Blok için sütun bilgisi girildikten sonra bloğumuzun şekline uygun boş yeri Oyun Alanında i ve flag yapısını while döngüsünde kullanarak tespit etmeye çalıştık. Uygun satırın tespit edilmesinden sonra elde ettiğimiz bu bilgiyi kullanarak şeklimizi Oyun alanında belirttiğimiz yere çiziyoruz. Blok düşürme kısmından sonra ise bir bloğun tamamen dolup dolmadığını tasarladığımız algoritma ile her blok düştükten sonra kontrol ediyoruz. Eğer bir hat tamamen dolduysa algoritmamızda o satırın üstündeki satırların aşağıya birer kaydırılması yolu ile hattımızı temizlemiş oluyoruz ve Skorumuzu da 10'ar arttırıyoruz. Ardından da oyun alanımızın en üst satırını kontrol eden bir döngümüzle en üst satırda başlangıç karakteri olan '.' Dışında bir karakter yakalarsak while döngümüzü devam ettiren 'control' değişkenimizi değiştirerek bu turdaki oyunumuzu bitiriyoruz. Tur bittikten sonra maksimum skoru geçip geçmediğimizi kontrol ederek max skorumuz güncelliyoruz ve oyuncuya bir daha oynayıp oynamak istediğini soruyoruz. Eğer oyuncu oynamak isterse 1 girerek, while döngümüzü devam ettiren koşulun doğru dönmesini sağlayarak yeni bir oyun turu başlatmış oluyoruz. En sonda kullanıcı oynamak istemediğinde de programımız kaç kere oynandığını göstererek sona eriyor.

1.3 – Yazılan Kodun ve Algoritmaların Açıklamaları

1.3.1 - Global Değişkenler

TetrisV9(TamTakirKod).cpp

```
#include<stdio.h>
#include<stdlib.h> // rand fonskiyonunu kullanabilmek için ekledik.


int M,N,sutun,score[50]={0},oyunSayisi=0,max=0; // diğer fonksiyonlarda da kullanacağımız değişkenleri globalde tanımladık.

```

Bu kısımda fonksiyonumuzda kullanılacak temel fonksiyonlar için kütüphanelerimizi dahil ediyoruz. Rand fonksiyonunu kullanacağımızdan dolayı stdlib.h kütüphanesini ayrıca dahil ettik. Diğer fonksiyonlarda Oyun alanı yani tablanın boyutları olan M ve N değişkenlerini, sutunSor() ve main() fonksiyonunda kullanacağımız 'sutun' değişkeni, ekrana yazdırmada kullanacağımız ve skor bilgisinin tutulacağı score[] dizisi, kaç kere oynadığımızın bilgisini tutan ve score[] dizisinde indis olarak kullanacağımız "oyunSayisi" ve maximum skorun yakalandığı indisi tutan max değişkenlerimizi diğer fonksiyonlarla ortak kullanacağımız için globalde tanımlıyoruz.

1.3.2 – Kullanılan Fonksiyonlar

```
void yeniBlokTemizle(char matrex[][3]); // oyunumuzda yeni gelecek bloğun bilgisinin tutulacağı matrisi her yeni blok için temizleyeceğimiz fonksiyon.
void yazdir(char matris[][500]); // oyun tablamızı ekrana yazdırmamiz için oluşturduğumuz fonksiyon.
void yeni_blok_yazdir(char matris[][3]); // yeni bloğun bilgisinin kullanıcıya göstermek için kullanacağımız fonksiyon
void sutunSor(); // kullanıcıya ekrana gelecek yeni bloğun nereye yerleştirileceği bilgisini aldığımız fonksiyon.
```

Yorum satırlarından da anlaşılacağı üzere gerekli fonksiyonları kodumuzda bildiriyoruz. Şimdi bu fonksiyonların özel olarak incelenmesine geçelim.

1.3.2.1 – yazdir() Fonksiyonu

```
void yazdir(char oyunAlan[][500]){
    int i,j;
    system("cls"); // tablamızı yazdırmadan önce konsolu temizliyoruz
    printf("--OYUN TAHTANIZIN SON DURUMU -- --- MAX SKOR : %d\n\n",score[max]); // en üste rekor skorumuzu yazıyoruz
    for(i=0;i<M;i++){
        for(j=0;j<N;j++){
            printf("%2c ",oyunAlan[i][j]); // oyun alanımızın son halini ekrana bastırıyoruz.
        }
        printf("\n");
    }

    for(j=0;j<N;j++){
            printf("-- ");
        }
    printf("\n");
    for(j=0;j<N;j++){ // kullanıcı hangi sutuna yerleştirebileceğini seçebilmesi için görsellik açısından tablamızı yazdıktan sonra altına satır numaralarını gösteriyoruz.
        if(j<9){
            printf(" %d ",j+1);
        }
        else{
            printf("%d ",j+1);
        }
        }

    printf("\n\nSkorunuz : %d",score[oyunSayisi]); // güncel skorumuzu tablamızın altında gösteriyoruz.
}

```

Bu fonksiyonumuzun ana amacı Oyun alanımızı ekrana yazdırmaktır. Öncelikle ekrana yazdırmadan önceden - system("cls"); - komutumuz ile konsolumuzu temizliyoruz. En sağ köşeye Max Skor bilgimizi yazdırdıktan sonra İç içe 2 for döngüsü ile oyun alanını ekrana bastırıyrouz. Ardından görsellik açısından oyun alanımızın altına çizgiler ekleyip bu çizgilerin altına da oyunu oynamayı kolaylaştırmak için sütun numaralarını bastırıyoruz. Bu işlemlerden sonra da alta da güncel skorumuzu ekrana yazdırıyoruz.

1.3.2.2 – yeniBlokTemizle() Fonksiyonu

```
666
667 void yeniBlokTemizle(char temizBlok[][3]){
668     int i,j;
669     for(i=0;i<3;i++){
670         for(j=0;j<3;j++){
671             temizBlok[i][j]=' '; // yeni bloğu kullandıktan sonra sonraki blok için temizlediğimiz komutumuz.
672         }
673     }
674
675 }
676
```

Burada da yeni bloğumuzun bilgisinin tutulacağı matrisi iç içe iki for döngüsü ile her satır ve sütununu ‘ ‘ boşluk karakteri ile güncelliyoruz.

1.3.2.3 – yeni_blok_yazdir() Fonksiyonu

```
655
656 void yeni_blok_yazdir(char blokMatrisi[][3]){
657     int i,j;
658     printf("\nYeni Blok : \n\n");
659     for(i=0;i<3;i++){
660         for(j=0;j<3;j++){
661             printf("%2c ",blokMatrisi[i][j]); // yeni bloğumuzu ekrana yazdıran komutumuz
662             }
663         printf("\n");
664     }
665 }
666
```

Bu fonksiyonumuzda bloğun tutulduğu matrisi uygun şekilde değiştirdikten sonra program çalışırken yeni bloğu ekrana yazdırmamız için iç içe 2 for döngüsü ile ekrana yazdırma işlemimizi tamamlıyoruz.

1.3.2.4 – sutunSor() Fonksiyonu

```
677 void sutunSor(){
678     int i,j;
679     printf("\n");
680     printf("Sutun Giriniz : ");
681     scanf("%d",&sutun); // kullanıcıdan hangi sutunu isteyeceğimizi belirttiğimiz kodumuz.
682     while(sutun>N || sutun<1){ // girilen sutun tablamızın sutun sayısından fazla veya 1den küçük ise girilecek sutunu tekrar istiyoruz.
683         printf("\nGecersiz Sutun Degeri Girildi -- Tekrar Giriniz\n");
684         printf("Sutun Giriniz : ");
685         scanf("%d",&sutun);
686     }
687     sutun=sutun-1;
688 }
```

Bu fonksiyonumuzda da kullanıcıdan hangi sütuna yerleştirme işlemini yapmak istediğini soruyoruz. While döngüsü ile girilen değeri kontrol ederek geçersiz bir sütun girilip girilmediğini kontrol ediyoruz ve hatalı giriş var ise tekrardan sütun girilmesini istiyoruz. -- sutun=sutun-1; -- C’de indiler 0dan başladığı için girilen sütunu uygun indiste kullanmak için 1 azaltıyoruz.

1.3.3– int main()

```
12 int main(){
13     char dondur,yeniBlok[3][3],tabla[500][500];
14     int flag=0,i,yer,yes=1,j,h,sayac,d,k,blok,control=0,y,t;
```

İnt main fonksiyonumuzun değişken tanımlama kısmında kullanıcı döndürmek istediğinde gireceği karakter bilgisini tutacak olan ‘dondur’, yeni bloğun bilgisinin tutulacağı ‘yeniBlok[3][3]’, oyun

alanımızın tutulacağı matris olan ‘tabla[500][500]’ değişkenlerini char olarak tanımlıyoruz. Bloğumuzu oyun alanına yerleştirirken while döngüsünde uygun satır tespit edildiğinde döngüden çıkılması için ‘flag’ değişkeni, uygun satır tespit edildikten sonra bu bilgiyi uygun şekilde güncelleyerek bu bilginin tutulabilmesi için ‘yer’ değişkeni, kullanıcımızın oyunu devam ettirmeyi isteyip istemediğini belirteceği bilgisini tutacak olan ‘yes’ değişkeni, hattın dolup dolmadığını kontrol ederken bir hatta ne kadar yerin dolduğunun bilgisini tutacak olan ‘sayac’ değişkenini, rastgele blok gelebilmesi için rand fonksiyonundan çıkan değerin bilgisini tutacağımız ‘blok’ değişkeni, üst satırın dolup dolmadığını kontrol edttikten sonra bu bilginin tutulacağı ve while döngümüzün şartı olan ‘control’ değişkeni ve while-for döngülerinde iterator olarak kullanacağımız ‘i’, ‘j’, ‘h’, ‘d’, ‘k’, ’y’ ve ’t’ iteratorlarımızı int olarak tanımlıyoruz.

1.3.4 – Oyunu Başlatma

```
26  while(yes==1){ // oyunumuzu default olarak başlatıyoruz, kullanıcımız oyunu sonlandırmak istediğinde yes değişkeni değişecek.
27      oyunSayisi+=1; // oyunumuzu kaç kere oynadığımızı kaydediyoruz.
28      for(i=0;i<M;i++){
29          for(j=0;j<N;j++){
30              tabla[i][j]='.'; // tablamızı hazırlıyoruz, her oyuna başlandığından tablamız '.' işareti ile dolacak.
31          }
32      }
33      control=0; // en üst satırımızda herhangi bir parça olup olmadığının kontrolü. başlangıçta parça olmadığı için 0dan başlatıyoruz.
34      yazdir(tabla); // tablamızı ekrana yazdırıp oyunumuzu başlatıyoruz.
```

Oyunumuzu başlatıyoruz. “yes=1” değeri ile oyuncumuz programdan çıkmak isteyip, yeni bir tur oynamak istemeyene kadar devam edecek while döngümüzü başlatıyoruz.

1.3.5 – Oyunun İlk Turuna Geçiş

```
35      while(control==0){
36          blok = rand()%7; // bloğumuzu rastgele alıyoruz, alacağımız değerleri if ile kontrol ederken uygun bloğu kullanıyoruz.
37          printf("\nBLOK NO : %d",blok);
```

Oyun alanımızda başlangıçta en üstte herhangi bir parça olamayacağı için “control” değişkenimizi 0 değerinden başlatarak en üstte blok tespit edilene kadar yeni blokları indirip tutun devam ettirecek olan while döngümüzü başlatıyoruz. Ve random fonksiyonundan bize blok seçmesini istiyoruz.

1.3.6 – Blok Yerleştirme Algoritması

Burada programımız yatay 3'lü blok için algoritmamızın nasıl çalıştığı anlatılacaktır. Diğer 6 blok için de algoritma mantığı bu şekilde çalışmaktadır, sadece değişkenlerin değeri değişmektedir.

```
if(blok==0){ // 111 sekli
    yeniBlokTemizle(yeniBlok);
    yeniBlok[1][0]='X'; // burada kullanıcımıza bloğu göstermek için yeniblok adlı matrisimize seklimizi ciziyoruz.
    yeniBlok[1][1]='X';
    yeniBlok[1][2]='X';
    yeni_blok_yazdir(yeniBlok); // uygun islemlerle yeni bloğu çizdikten sonra bloğu ekrana yazdırıyoruz.
    printf("\nDondurmek icin 'w' tusuna basiniz, istemiyorsaniz baska bir tusa basiniz : ");
    scanf("%s",&dondur);
    if(dondur == 'w' || dondur == 'W'){ // | 111 dikey hali olacak
        yazdir(tabla); // tablamızı ekrana yazdırıyoruz, system("cls") komutu ile üst kısımları temizleyip döndürülmüs hali göstereceğiz.
        yeniBlokTemizle(yeniBlok);
        yeniBlok[0][1]='X';
        yeniBlok[1][1]='X';
        yeniBlok[2][1]='X';
        yeni_blok_yazdir(yeniBlok);
        sutunSor(); // kullanıcımızdan gösterilen bloğun en soldaki kısmının hangi sutuna yerleştirileceğini istiyoruz.
        i=0; // tablamızda seçilen koordinata uygun satırı bulmak için tablamızı kontrol eden kod kısmına geçiyoruz.
        flag=0;
        while(flag==0 && i<M-1){
            if(tabla[i][sutun]!='.'){ // tablamızı i ve flag yapısı ile, potansiyel yerleri kontrol ediyoruz.
                flag=1;     // eğer herhangi bir yer dolmussa döngüden çıkıp bulduğumuz yeri kaydedeceğiz.
            }
            if(tabla[i+1][sutun]!='.'){
                flag=1;
            }
            if(tabla[i+2][sutun]!='.'){
                flag=1;
            }
            i++;
        }
        yer=i-2; // döngüden çıktıktan sonra belirlenen i değerine göre olusacak değeri yer adlı değiskenine atayarak islem yapıyoruz.
        tabla[yer][sutun]='X'; // kontrolümüzden sonra bulduğumuz uygun satır ve sütunlara tablamızda seklimizi cizdiriyoruz.
        tabla[yer+1][sutun]='X';
        tabla[yer+2][sutun]='X';
    }
    else{ // kodumuz bundan sonra benzer yapıda devam etmektedir. seçilen blok ve döndürülmüs haline uygun kontroller yapılarak uygun yere seklimiz cizdirilliyor.
        sutunSor();
        i=0;
        flag=0;
        while(flag==0 && i<=M){
            if(tabla[i][sutun]!='.'){
                flag=1;
            }
            if(tabla[i][sutun+1]!='.'){
                flag=1;
            }
            if(tabla[i][sutun+2]!='.'){
                flag=1;
            }
            i++;
        }
        yer=i-2;
        tabla[yer][sutun]='X';
        tabla[yer][sutun+1]='X';
        tabla[yer][sutun+2]='X';
    }
}
```

3lü bloğumuz için kod şeklimiz eğer random fonksiyonundan 0 değeri döndürülürse bu kod bloğumuz çalışacak ve ekranda önce yatay 3 haneli bloğumuz gözükecek. Kullanıcı döndürmek isterse dikey 3lü bloğa dönüşecek ve ardından ilgili sütuna yerleştirme işlemi yapılacaktır. Şimdi parça parça inceleyelim.

Bloğun ekranda belirmesi ve kullanıcıya döndürmek isteyip istemediği sorulması

```
if(blok==0){ // 111 şekli
    yeniBlokTemizle(yeniBlok);
    yeniBlok[1][0]='X'; // burada kullanıcımıza bloğu göstermek için yeniblok adlı matrisimize şeklimizi çiziyoruz.
    yeniBlok[1][1]='X';
    yeniBlok[1][2]='X';
    yeni_blok_yazdir(yeniBlok); // uygun işlemlerle yeni bloğu çizdikten sonra bloğu ekrana yazdırıyoruz.
    printf("\nDondurmek icin 'w' tusuna basiniz, istemiyorsaniz baska bir tusa basiniz : ");
    scanf("%s",&dondur);
```

Kod bloğumuza girdikten sonra önce "yeniBlokTemizle(yeniBlok);" komutu ile yeni bloğun tutulduğu matrisi temizliyoruz.

Ardından

"

yeniBlok[1][0]='X';

yeniBlok[1][1]='X';

yeniBlok[1][2]='X';

"

komutları ile yeniBlok matrisimizde şeklimizi çizdiriyoruz. Ve “yeni_blok_yazdir(yeniBlok);” komutu ile de bloğumuzu güncelledikten sonra ekrana yazdırıyoruz. Ve son olarak da kullanıcıya döndürmek isteyip istemediğini soruyoruz.

1.3.6.1– Kullanıcı Döndürmek İsterse

```
46  if(dondur == 'w' || dondur == 'W'){ // | 111 dikey hali olacak
47      yazdir(tabla); // tablamızı ekrana yazdırıyoruz, system("cls") komutu ile üst kısımları temizleyip döndürülmüş hali göstereceğiz.
48      yeniBlokTemizle(yeniBlok);
49      yeniBlok[0][1]='X';
50      yeniBlok[1][1]='X';
51      yeniBlok[2][1]='X';
52      yeni_blok_yazdir(yeniBlok);
53      sutunSor(); // kullanıcımızdan gösterilen bloğun en soldaki kısmının hangi sutuna yerleştirileceğini istiyoruz.
```

Kullanıcı ‘w’ karakterini girerse bu kod bloğumuz çalışacak ve önce konsolu temizleyip yazdırmak için “yazdir(tabla);” ardından önceki kod bloğunda yaptığımız işlemleri tekrarlayarak dikey 3lü bloğumuzu ekrana yazdırıyoruz. Burada blok tekrar döndürüldüğünde aynı şekil elde edileceği daha fazla döndürmeye ihtiyaç duyulmayacaktır. Bu yüzden bu adımdan sonra gösterilen blok için sütun koordinatını istiyoruz.

1.3.6.2 – Kullanıcı Döndürmek İstemezse

```
73  else{ // kodumuz bunda
74      sutunSor();
```

Eğer kullanıcımız döndürmek istemez ise else bloğumuza gireceğiz ve bu adımda sütun bilgisini soracağız.

1.3.7 – Blok İçin Uygun Yerin Tespiti ve Oyun Alanına Yerleştirilmesi

```
75      i=0;
76      flag=0;
77      while(flag==0 && i<=M){
78          if(tabla[i][sutun]!='.'){
79              flag=1;
80          }
81          if(tabla[i][sutun+1]!='.'){
82              flag=1;
83          }
84          if(tabla[i][sutun+2]!='.'){
85              flag=1;
86          }
87          i++;
88      }
89      yer=i-2;
90      tabla[yer][sutun]='X';
91      tabla[yer][sutun+1]='X';
92      tabla[yer][sutun+2]='X';
```

Bu kod bloğumuzda önce while döngümüzde kullanacağımız “i” iteratorunu 0’lıyoruz ve koşulu yakaladığımızda çıkabilmek için flag değişkenimizi de uygun değerle başlatıyoruz.

Bu kısımda i satır sayısından küçük olana kadar ve flag koşulumuz sağlandığı sürece döngümüz devam edecek. Döngümüzde if blokları ile kontrol edilen satır ve sütun koordinatında parça olup

olmadığını kontrol ediyoruz. Döngümüz bittikten sonra ‘i’nin son değerini 2 azaltarak bloğumuzun yerleştirileceği satırı tespit ediyoruz. Burada “i”nin değerini azaltarak bloğumuzu parça tespit edilen yere değil de tespit edilen yerin üst satırına yerleştirme için uygun yeri buluyoruz. “yer” değişkenine uygun satırı tespit ettikten sonra

“

tabla[yer][sutun]='X';

tabla[yer][sutun+1]='X';

tabla[yer][sutun+2]='X';

“ kullanıcın girdiği sütun ve belirlediğimiz uygun satır değeri için oyun alanımızı güncelliyoruz. Ve böylece bloğu yerleştirme işlemimiz tamamlanıyor. Böylece herhangi bir bloğun yerleştirilmesi için nereye yerleştirileceğinin kontrol edilmesi ve tespit edilmesini ve oyun alanımıza yerleştirme algoritmamızı bu şekilde kodlamış oluyoruz.

1.3.8 – Herhangi Bir Satırın Dolu Olup Olmadığının Kontrolü

```
576             score[oyunSayisi]+=1; // her bir parça yerleştirildikten sonra puanımızı 1er arttırıyoruz.
577             h=0;
578             while(h<M){ // bir satırın tamamen dolup dolmadığını kontrol eden döngümüz
579                 sayac=0;
580                 d=0;
581                 while(d<N){
582                     if(tabla[h][d]=='X'){ // en üst satırdan alta kadar kontrol ediyoruz ve eğer satırda blok parçası varsa her parça için sayacı 1er arttırıyoruz.
583                         sayac++;
584                     }
585                     if(sayac==N){ // eğer sayacımız sutun sayısına eşitlendiyse o satır tamamen dolduğunu anlamış olacağız.
586                         for(y=0;y<N;y++){
587                             for(t=h;t>0;t--){
588                                 tabla[t][y]=tabla[t-1][y]; // burada da tablamızda tamamen dolu olan satırı silmek için bir döngü ile üstteki satırları aşşağıya kaydırıyoruz.
589                             }
590                         }
591                         score[oyunSayisi]+=10; // eğer bir satırı tamamen patlattıysak skorumuz 10 artıyor.
592 
593                     }
594                     d++;
595 
596                 }
597                 h++;
598             }
```

Burada herhangi bir hattın dolup dolmadığını, hat dolduysa dolan hattın silinmesi işlemlerini anlatacağız.

“score[oyunSayisi]+=1” Öncelikle bu komut ile her blok düştükten sonra güncel skorumuza 1 puan ekliyoruz. Ardından iç içe 2 while döngüsü ile en üst satırdan başlayarak tüm satır ve sütunları kontrol ediyoruz.

“ if(tabla[h][d]=='X'){.

sayac++;

} “ Bu if bloğumuz ile belirli bir satırın sütunlarını gezerken kaç tane ‘X’ yani blok parçasının olduğunu kontrol ediyoruz. Sonrasında sayacın Sütun sayısına eşit olup olmadığını kontrol ediyoruz. Eğer bir satır tamamen dolduya bu komutumuz çalışacaktır ve içindeki şu kodlar ile :

“” for(y=0;y<N;y++){

for(t=h;t>0;t--){

tabla[t][y]=tabla[t-1][y];

}

} score[oyunSayisi]+=10; // eğer bir satırı tamamen patlattıysak skorumuz 10 artıyor. “”

Bu komutumuz ile dolu olduğunu tespit ettiğimiz satırın üstündeki tüm satırları 1er aşağı kaydırarak, dolu bloğumuzu temizlemiş oluyoruz. Blok temizlendiyse bu komutların yanına ayrıca bir hattın temizlenmesinden 10 puan almasını sağlıyoruz.

1.3.9 – En Üst Satırda Parça Olup Olmadığının Kontrolü

```
600         for(i=0;i<N;i++){
601             if(tabla[0][i]=='X'){ // tablamızın en üst satırında herhangi bir blok parçası olup olmadığını kontrol ediyoruz. var ise oyun bitecektir.
602                 control=1;
603             }
604         }
605         yazdir(tabla);
```

Bu kod parçamızda tablamızın en üst satırındaki sütunları döngümüz ile geziyoruz ve eğer üstte herhangi bir blok parçası varsa control değerimizi=1e eşitliyoruz.

Tüm bu blok yerleştirme, hat silme işlemlerinden sonra tablamızın son halini ekrana yazdırıyoruz.

1.3.10 – Oyunun Tekrar Oynanması veya Oyunun Sonlandırılması

```
608         yes=0;
609         printf("\n--- GAME OVER ----\n");
610         printf("Tekrar Oynamak icin 1 giriniz : "); // oyuncu oynamaya devam etmek istiyorsa 1 girecektir ve döngü yeniden başlatılacaktır.
611         scanf("%d",&yes);
612         score[oyunSayisi+1]=0; // burada oyunSayisi+1 kullanarak bir sonraki tur oynanırsa skorun 0dan başlatılmasını sağlıyoruz.
613         if(score[oyunSayisi]>score[max]){
614             max=oyunSayisi; // eğer son oynanan oyundaki skor önceki max skordan fazla ise max değerimiz güncellenecektir.
615         }
616
617         if(yes==1){
618             printf("OYUN TAHTANIZ ICIN BOYUT GIRINIZ (Once Satir Sonra Sutun): ");
619             scanf("%d  %d",&M,&N);
620             printf("\n");
621
622         }
623     }
624     printf("Oyun Bitti --  %d kere Oynadiniz.",oyunSayisi);
625     return 0;
626 }
```

1.3.10.0 – Max Skor Bilgisi

“”

```
score[oyunSayisi+1]=0;
if(score[oyunSayisi]>score[max]){
            max=oyunSayisi;
}
```

“” Buradaki komutlarımız ile oyuncumuz yeni oyuna geçmek isterse sonraki oyunun skorunun tutulacağı dizi elemanını 0’lıyoruz. Ardından if bloğumuz ile güncel oyunumuzun skorun indisindeki değerin max skorun tutulduğu indisteki değerden yüksek olup olmadığını kontrol ediyoruz. Eğer şart sağlanıyor ise max indisimizi oynadığımız oyun sırası ile güncelliyoruz.

1.3.10.1 – Tekrar Oynama

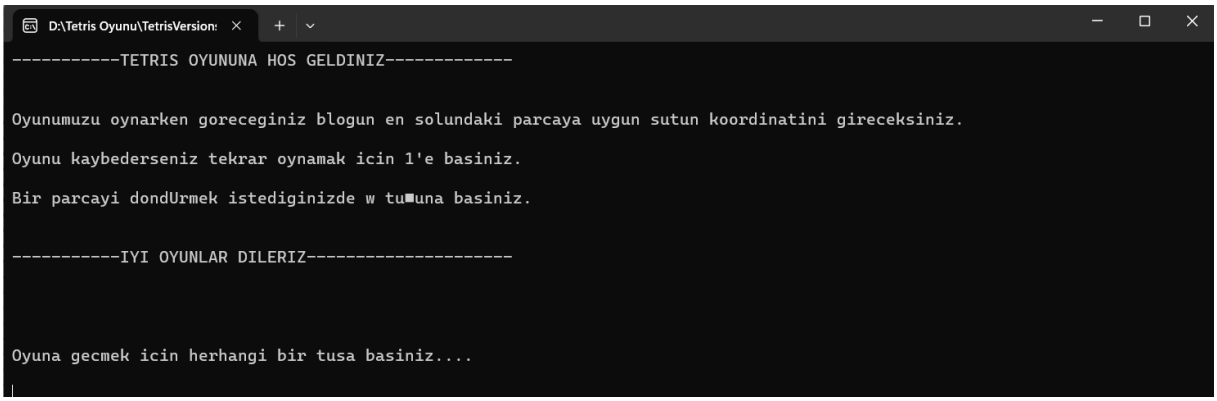
Control değişkenimiz 0dan 1e değiştiğinde içteki blokları yerleştiren while döngümüz sona erecek ve “yes=0;”dan itibaren olan kodlar çalışmaya başlayacaktır. Burada da while döngümüzün şartı “yes==1” olduğu için kullanıcımızdan tekrar oynamak istiyorsa 1 değerini girmesini istiyoruz. Eğer oyuncu 1’e basarsa oyun tahtamız için yeniden boyut soruyoruz ve oyunumuzu devam ettiriyoruz.

1.3.10.2 – Oyunu Bitirme

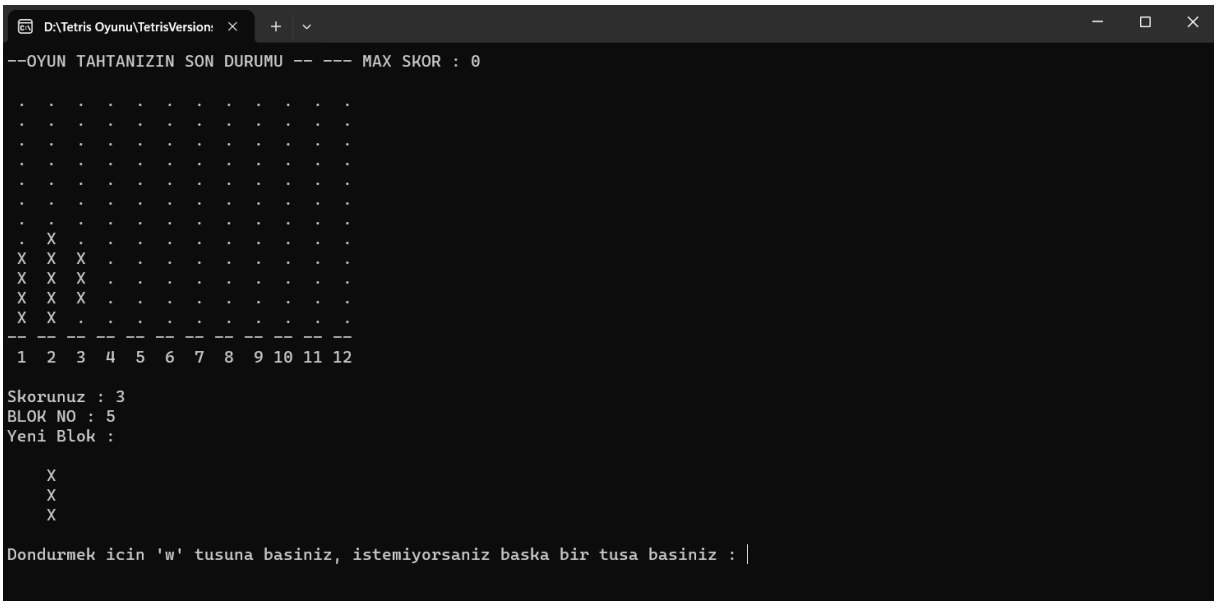
Oyuncumuz tekrar oynamak istemez ise 1 dışında bir karakter girecektir ve böylece oyunu devam ettirecek olan while döngümüz sona erecektir. Oyunumuz bittikten sonra Oyun bitti ve kaç kere oynandığının bilgisi gösterilecektir.

1.4 – Oyundan Görüntüler

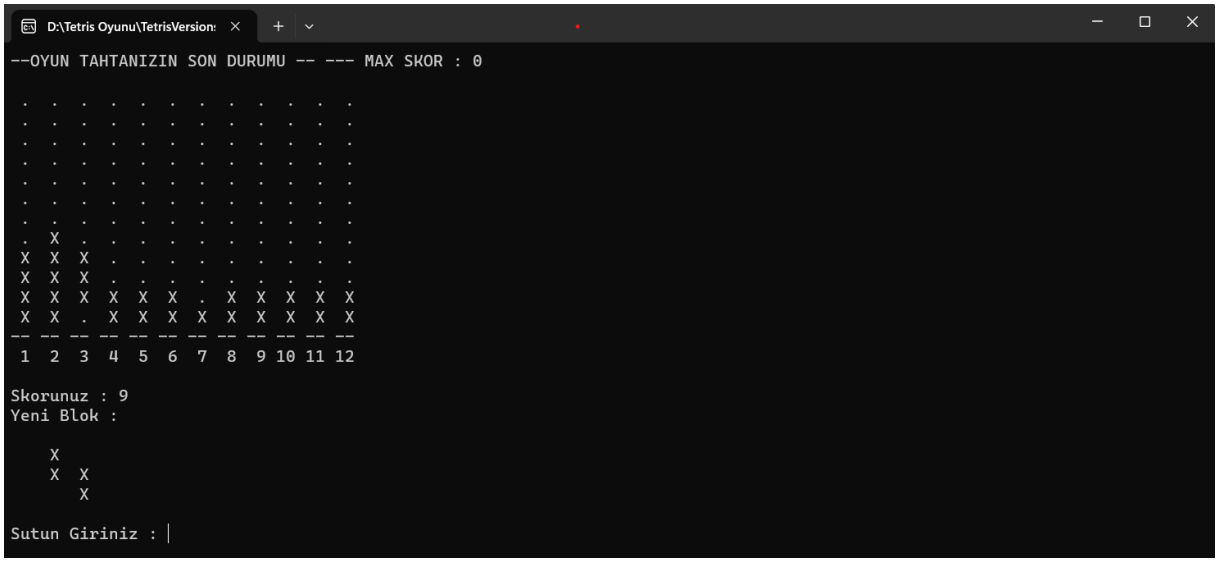
1.4.1 :



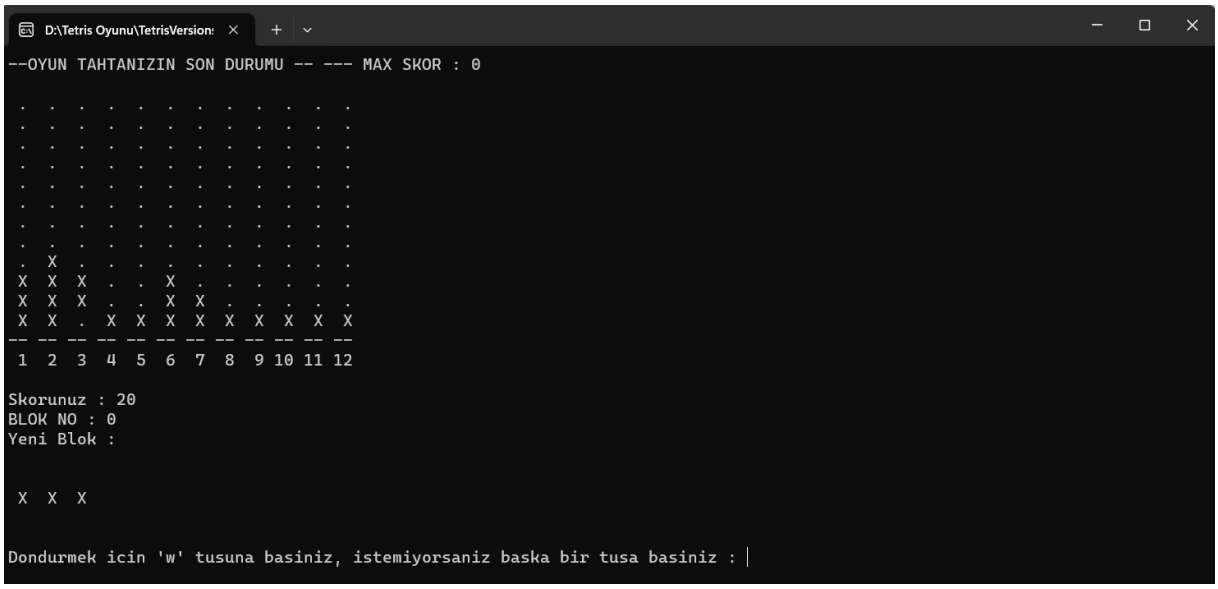
1.4.2:



1.4.3 :



1.4.4



1.5 – Sonuç

Genel olarak oyunumuzun kaynak kodları ve oyunumuzun hazırlanış aşaması böyle idi. Değerli ilginiz için teşekkürler.

Videolu anlatım linki : https://drive.google.com/file/d/1D3waZFfxHzdW5OU9fXF7bAH9qtpR-Q25/view?usp=sharing