Projektmunka - Sakk Visual Studioban, c#

Komócsin Ádám, konzulens tanár: Szeifert Gábor

Tartalom

Miért ezt választottam?

Mindenféleképpen egy olyan munkát szerettem volna készíteni, ami hozzám közel áll. Először egy weboldalra gondoltam, ami a Trónok Harcáról szólt volna, de rájöttem, hogy ez nem tartalmazna túl sok információt és nem tudnék sok speciális tartalmat beleölni az oldalba.

Majd gondoltam egy játékra, de nem tudtam milyen játék legyen. Legyen amőba vagy aknakereső? Végül mindkettő ötletet elvetettem, majd rájöttem, hogy szeretek sakkozni és a programozás sem áll annyira messze tőlem legalábbis jobban megy, mint a weblapkészítés.

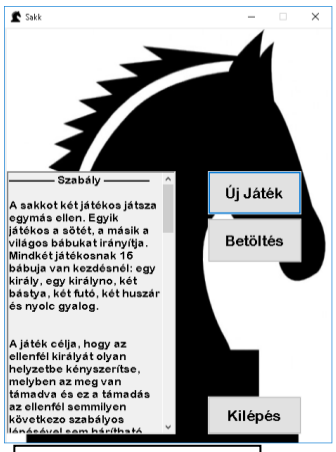
Milyen programokat használtam?

A sakkot a Visual Studio nevű programban írtam meg, viszont szükségem volt Photoshopra is.

A bábuk amiket hozzárendeltem a mezőkhöz túlságosan el voltak csúszva, ezért muszáj volt korrigálni őket egy képszerkesztő programban. Valószínűleg annyira nem lett volna zavaró, ha kihagyom ezt a lépést, de egy idő után nagyon zavarta a szememet a szétcsúszott bábuk látványa.

Menü és design felépítés

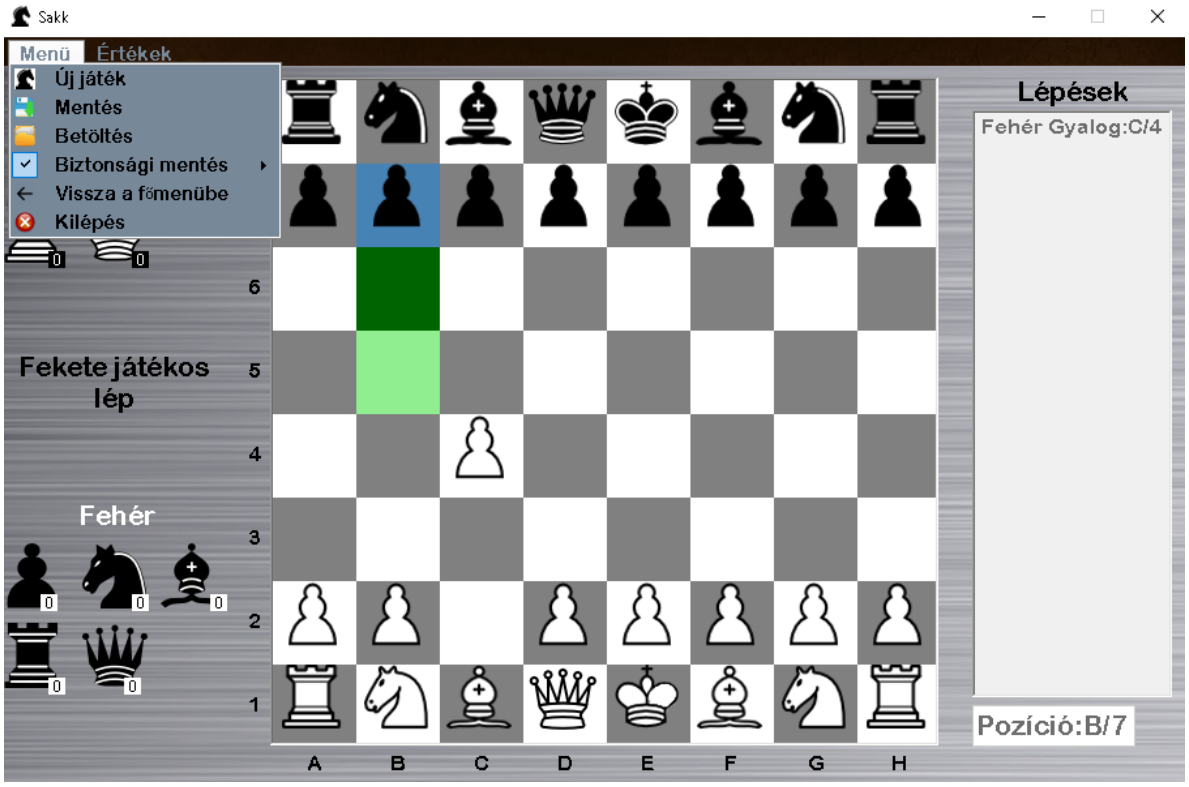
Ahogy elindítjuk a programot rögtön egy menü fogad minket. A menüben nincs sok lehetőségünk, ahogy a képen látható indíthatunk új játékot, betölthetünk egy régebbit és kiléphetünk a programból. Úgy gondoltam lesz benne egy beállítások lehetőség ahol a felhasználó be tud állítani egy-két dolgot mint például időt, játékosok nevét és színét és még pár funkció. Végül elvetettem az ötletet, mert így is elegendőnek éreztem a programot beállítások nélkül. A főmenü tartalmaz egy szabályzatot is, ami egy egyszerű *StreamReader*-el lett beolvasva a *Richtextboxba*. Új játékra kattintva behoz egy másik *Form*-ot, ami maga a táblát, a levett bábukat, a jelenlegi pozíciót és a lépést írja ki. Betöltésre kattintva a program behoz egy *OpenLoadDialog*-ot, ahol a felhasználó be tud tölteni egy .txt formátumú fájlt.



1. Menü

Új játék

Az új játék gombra kattintva ezt a felületet láthatjuk majd:



2. Játék kinézet

Balra látható a levett bábuk és, hogy eddig hányat vettek le az adott bábuból illetve, hogy melyik játékos következik. Középen látható a tábla és a bábuk ezeket később írom le hogy hogy lettek lekódolva. Jobb oldalon látható egy *richtextbox*, ami tartalmazza a megtett lépéseket, alatta pedig egy *textbox* ami a jelenlegi pozíciót írja ki. Bármelyik bábura kattintva kijelöli az adott bábut kékkel és zölddel a lehetséges lépéseit. A program tetején található egy menü fül és egy értékek fül. Az érték fül tartalmazza a bábuk értékeit. A menüben lehetőségünk van új játékot kezdeni, menteni, betölteni (természetesen olyan txt fájlt ,ami megfelel a program feltételeinek, ha nem ilyet töltünk be a program visszairányít minket a főmenübe), biztonsági mentés ki-be kapcsolása (ezt is majd később írom le), vissza lépni a főmenübe és kilépni.

A tábla, osztályok és a bábuk lerakása a táblára

A programban 2 fő osztályom van. Egy „Mezo” és egy „Babu”. A többi osztály egy-egy bábu, ami megkapja az összes olyan tulajdonságot, amivel a „Babu” rendelkezik. A „Babu” megkapja a „Mezo” tulajdonságait is egyben.

```
// ------------------------------------------------------------------------Fő classok--

public class Mezo ... //Mező

public class Babu ... // Bábu

------------------------------------------------------------------------Bábu classok--------

public class Gyalog ... // Gyalog

public class Bastya ... // Bástya

public class Lo ... // Ló

public class Futo ... // Futó

public class Kiralyno ... // Királynő

public class Kiraly ... // Király
```

3. Osztályok

A tábla

A „Mezo” egy Panel típusú osztály. A tábla egy „Mezo” típusú mátrix, tehát egy 8x8-as tömb, *„tabla”* néven lett deklarálva. A „Mezo” konstruktora kér egy színt illetve egy x és egy y értéket. Az x,y érték a *„tabla”* i illetve j-ik eleme lesz a szín, amit kér pedig a tábla színe.

Egy külön eljárásban hoztam létre a táblát *„tablaletrehozas”* néven. Itt a *„tablaszin”* lesz az a szín, amit a „Mezo” konstruktora kérni fog. A *„tabla”* tömbömre sokszor szükségem lesz, majd ezért nem magában az eljárásban hozom létre, hanem statikusan, hogy mindenhonnan elérjem majd későbbiekben. Az eljárásban létrehozok egy Panel típusú *„tabla2”*-t. Ezt a lépést szimplán csak azért tettem meg, hogy utána egyszerűbben tudjam megváltoztatni a tábla tulajdonságait (hol legyen a képernyőn, keret stb) és így nem kell két for ciklus.

```
public static List<Mezo> kek = new List<Mezo>();
public static Babu kijeloltBabu = null;
public static bool valtas = true;
public static Mezo[,] tabla = new Mezo[8, 8];
public static jatek ablak;
public static Kiraly feherkiraly;
public static Kiraly feketekiraly;
public static bool mentesdarab;
public static List<Babu> feherlevettbabuk = new List<Babu>();
public static List<Babu> feketelevettbabuk = new List<Babu>();
public static int FeherGyalog = 0;
public static int FeherLo = 0;
public static int FeherFuto = 0;
public static int FeherBastya = 0;
public static int FeherKiralyno = 0;
public static int FeketeGyalog = 0;
public static int FeketeLo = 0;
public static int FeketeFuto = 0;
public static int FeketeBastya = 0;
public static int FeketeKiralyno = 0;
```

4. A tábla deklarálása

5. A tábla létrehozása

```
public void tablaletrehozas()
{
    Panel tabla2 = new Panel();
    bool tablaszin = false;
    for (int i = 0; i < 8; i++)
    {
        for (int j = 0; j < 8; j++)
        {
            tabla[i, j] = new Mezo(tablaszin,i, j);
            tabla[i, j].SetBounds(i * 64, j * 64, 64, 64);
            tablaszin = !tablaszin;
            if (tablaszin == true)
            {
                tabla[i, j].BackColor = Color.White;
            }
            else if (tablaszin == false)
            {
                tabla[i, j].BackColor = Color.Gray;
            }
            tabla2.Controls.Add(tabla[i, j]);
        }
        tablaszin = !tablaszin;
    }
    tabla2.SetBounds(this.Width / 2 - 256, (this.Height - 39) / 2 - 256, 512, 512);
    tabla2.BorderStyle = BorderStyle.Fixed3D;
    this.Controls.Add(tabla2);
    tabla2.Anchor = AnchorStyles.Left & AnchorStyles.Right;
    ablak = this;
    ablak.debug3("Fehér játékos kezd", Color.White);
    Labelreset();
} // Tábla létrehozás eljárás
```

Bábuk lerakása

Mielőtt rátérnék a lerakásra megmutatom a „Babu” osztályt.

```
public class Babu : PictureBox
{
    public Mezo mezo;
    public bool szin;
    public Babu(Mezo mezo, bool szin)
    {
        this.mezo = mezo;
        this.szin = szin;
        mezo.Controls.Add(this);
        mezo.babu = this;
        this.MouseClick += Mezo_MouseClick;
        this.Cursor = Cursors.Hand;
    }
```

6.”Babu” konstruktora

A konstruktor tartalmaz egy *„mezo”* nevű adattagot. Ez fogja megkapni a bábuk koordinátáját és színét. A *„szin”* nevű adattagom fogja a bábuk színeit adni. Ha a „szin” igaz akkor a bábu fehér lesz, hamis esetén fekete. Ennek kivitelezése a következőképpen néz ki:

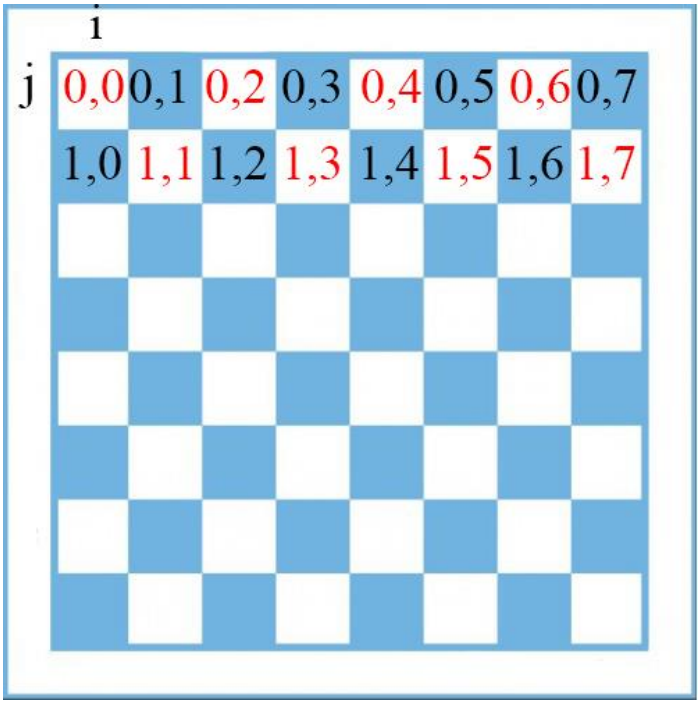
Minden egyes bábu konstruktorában megnézzük, hogy a „szin” éppen milyen értéket ad vissza. Ha a szín igaz, akkor olyan képet illeszt be a táblára ami fehér. Ha hamis, akkor feketét.

Itt egy példa:

```
public static void babulerakas()
{
    //Bábuk
    //Gyalog
    //Fehér Gyalog
    for (int i = 0; i < 8; i++)
    {
        Gyalog WPawn = new Gyalog(tabla[i, 6], true);
    }
    //Fekete Gyalog
    for (int i = 0; i < 8; i++)
    {
        Gyalog BPawn = new Gyalog(tabla[i, 1], false);
    }
```

7. Bábu lerakás

Itt éppen a *„babulerakas”* eljárásban vagyunk. Létrehozunk a fehér és fekete gyalog bábukból 8-at,8-at. Ez 2 for ciklus és a cikluson belül példányosítunk. A tabla[i,1] azt jelenti, hogy a *„mezo”* nevű adattagunk megkap 2 db számot. Kap egyszer egy 1-től nyolcig tartó számsorozatot, ez az i és egy fix 1-es értéket, ez a j. Utána megkapja azt, hogy false tehát ezek lesznek a fekete gyalogok.



8. A tábla és a bábuk „kapcsolata”

Itt most a j a sor és az i az oszlop. Megkeresi a j-t, tehát a táblán nézve a második sort és i szerint (0-tól 7-ig, ami a táblán 1-től 8-ig megy) kitölti gyalogokkal. Így működik ez a többi bábunál is, de oda nem kellettek for ciklusok.

A bábuk lépései és játékos váltás

Kijelölés

Ahhoz, hogy a bábuk tudjanak lépni kellett egy „mouse_click” esemény, ami kezeli a kattintásokat. A „Mezo”-ben és a „Babu”-ban is létrehoztam ezeket az eseményeket. A „Mezo”-ben lévő „mouse_click”-ben írtam meg, hogy a bábu tudjon lépni leütés nélkül, a „Babu”-ban pedig azt, amikor bábulevétellel ütünk.

Létrehoztam egy statikus „Babu” féle *„KijeloltBabu”*-t. Ez arra szolgál, hogy elmentsem azt a bábut, amire a felhasználó éppen kattintott. Így már le tudjuk kezelni azt, hogy mikor mit csináljon. A kijelölt bábu mezőjének a színe kék lesz így tudja a felhasználó, hogy melyik bábu van kijelölve.

Játékos váltás

Mielőtt részletezném a bábu lépését először a váltásról beszélek, mert fontos szerepe van a bábuk lépéslehetőségeikben. A váltás egy egyszerű true, false-al van megoldva. Létrehoztam egy bool típusú *„valtas”* nevű adattagot, aminek az alapértéke true. Ha true akkor a fehér bábukkal lehet lépni, ha false akkor a feketével. Ez úgy lett megoldva, hogy ha a *„valtas”* értéke megegyezik a *„szin”(bábu színe)*-el, akkor jelölje ki kékkel a bábut. Tehát, ha a *„váltás”* true, de én fekete bábut nyomogatok, akkor nem fogja kijelölni. Ez a része a „Babu”-ban van megírva. Ha a *„valtas”* és a *„szin”* megegyezik, akkor a *„kijeloltBabu”*-ban tárolom el azt a bábut, amire kattintottunk és a mezőjének a színét kékre állítom. Minden egyes lépés után a *„valtas”*-t átváltom, így cserélődnek a játékosok.

```
private void Mezo_MouseClick(object sender, EventArgs e)
{

    if (valtas == szin)
    {
        tablaszinreset();
        kek.Add(mezo);
        mezo.BackColor = Color.SteelBlue;
        zold();
        kijeloltBabu = this;
    }
    else
    {
        if (kijeloltBabu != null)
        {
            if (this.mezo.BackColor == Color.LightGreen || this.mezo.Bac
            {

                this.mezo.Controls.Remove(this);
                this.mezo.Controls.Add(kijeloltBabu);
                kijeloltBabu.mezo.babu = null;
                this.mezo.babu = kijeloltBabu;
                kijeloltBabu.mezo = this.mezo;
                mentesdarab = true;
                belepes(this.mezo.x, this.mezo.y, this.mezo.babu);
                valtas = !valtas;
                sakk();
                ablak.debug3(valtas ? "Fehér játékos lép" : "Fekete játé
                tablaszinreset();
                ablak.debug("Pozíció:");
                this.mezo.babu.Ellepeskiiras();
```

9. Váltás és a bábu szín közötti kapcsolat

Tábla színének visszaállítása eredeti színre

Azokat a mezőket, amik kékek lesznek eltárolom egy *„kek”* nevű listába, ami „Mezo” típusú. Erre szimplán csak azért van szükség, hogy ha a játékos átkattint egy másik bábura, ami szintén az övé, akkor az előző kijelölt bábu mezőjének a színét állítsa vissza az eredeti színre. Ezt igen sokszor kellett alkalmaznom, ezért ezt is egy eljárásban írtam meg, aminek a neve *„tablaszinreset()”*.

```
public static void tablaszinreset()
{
    for (int i = 0; i < kek.Count; i++)
    {
        kek[i].BackColor = kek[i].vissza ? Color.White : Color.Gray;
        kek[i].Cursor = Cursors.Default;
    }
    kek.Clear();
} //Mezőknek a színét állítja vissza
```

10. Tábla eredeti színének visszaállítása

Végig megyünk a *„kek”* lista elemeinek a színén, ami a *„vissza”*. A *„vissza”* a „Mezo” konstruktorában található, értéke megegyezik a *„tablaszin”*-el, tőle kapja meg. A lista elemeinek a színe alapján állítjuk vissza, ha igaz, akkor fehérre, ha hamis, akkor szürkére.

Virtuális eljárás

Minden egyes bábu máshogy tud lépni. Amikor valamelyik bábura rákattintunk akkor a lehetséges lépései megjelennek, a mezők színe zöld lesz. Ezt egy virtuális eljárással oldottam meg. A virtuális eljárást létrehozom a „Babu”-ban és minden egyes bábu oszttálynál átírom.

Nézzünk egy példát:

```
public virtual void zold()
{
    int ox = this.mezo.x;
    int oy = this.mezo.y;
    ablak.debug("Pozíció:" + (char)(65 + ox) + "/" + (8 - oy));
}
```

11.”Zold” virtuális eljárásom

Itt a virtuális eljárásom, aminek a neve *„zold”*.

Ez az eljárás egyelőre csak létrehoz egy ox-et és egy oy-t, ami később a bábu jelenlegi helyzetét adja meg majd nekünk. Az ablak.debug része pedig kiírja a jelenlegi pozíciót.

Ahhoz, hogy meg tudjuk határozni hova léphetünk meg kell néznünk a többi mezőt is. Meg kell nézni, hogy ahova lépne van-e valami, ha igen akkor barát vagy ellenséges bábu-e. Erre a célra hoztam létre 3 függvényt (a 4. a ló lépéseit bonyolította le nekem).

```
public static bool Tablanbelul(int x, int y)
{
    return x >= 0 && y >= 0 && x < 8 && y < 8;
} // Tablanbelul

public static bool Ellenseg(int x, int y, bool szin)
{
    return tabla[x, y].babu != null && tabla[x, y].babu.szin != szin;
} // Ellenseg

public static bool Barat(int x, int y, bool szin)
{
    return tabla[x, y].babu != null && tabla[x, y].babu.szin == szin;
} // Barat

public static bool Lepes(int ox, int oy, int x, int y, bool szin)
{
    return Tablanbelul(x + ox, y + oy) && !Barat(x + ox, y + oy, szin);
} // Lépés lónál
```

12. Fontos függvények lépéssel kapcsolatban

A *„Tablanbelul”* függvény megvizsgálja, hogy az adott lépés a táblán belülre érkezik-e. A *„Barat”* és az *„Ellenseg”* ugyanolyan visszatérési értékeket várnak, egyedül a *„szin”* értéke változik.

Miután létrehoztam ezeket és a virtuális eljárást, már csak a bábuk lépéseit kellett „kiszámolnom” és leírnom.

Gyalog

```
public override void zold()
{
    base.zold();
    List<Mezo> zoldLista = new List<Mezo>();
    int ox = this.mezo.x;
    int oy = this.mezo.y;
    int dir = szin ? -1 : 1;
    if (Tablanbelul(ox, oy + dir))
    {
        if (!Ellenseg(ox, oy + dir, szin) && !Barat(ox, oy + dir, szin))
        {
            zoldLista.Add(tabla[ox, oy + dir]);
        }

    }
    //BAL ÁTLÓ ÜTÉS
    if (Tablanbelul(ox - 1, oy + dir) && Ellenseg(ox - 1, oy + dir, szin))
    {
        zoldLista.Add(tabla[ox - 1, oy + dir]);
    }
    //JOBB ÁTLÓ ÜTÉS
    if (Tablanbelul(ox + 1, oy + dir) && Ellenseg(ox + 1, oy + dir, szin))
    {
        zoldLista.Add(tabla[ox + 1, oy + dir]);
    }
    if (!Ellenseg(ox, oy + dir *2, szin) && !Barat(ox, oy + dir * 2, szin) && !Ellenseg(ox, oy + dir, szin) && !Barat(ox, oy + dir, szin) && oy == (szin ? 6 : 1))
    {
        zoldLista.Add(tabla[ox, oy + dir * 2]);
    }
    foreach (var v in zoldLista)
    {
        v.BackColor = v.vissza ? Color.DarkGreen : Color.LightGreen;
        v.Cursor = Cursors.Hand;
    }
    kek.AddRange(zoldLista);
}
```

13. Gyalog lépései

Minden egyes osztályban máshogy alakítom át a *„zold()”* virtuális eljárásomat. Létrehozok egy listát , amibe majd belerakom azokat a mezőket, amiket zöldre kell színezni. Ez a *„zoldLista”*. Az ox, oy a bábu jelenlegi helyzetét adja majd meg nekünk, a dir pedig a gyalog esetében egy irányt. Mivel a gyalog csak egy irányba tud menni fel vagy le ezért a *„dir”* értéke a bábu színétől függ. Ha a bábu true akkor -1 ha false akkor 1. (Fölfele – értékek lefele + értékek)

Ezután ellenőriznünk kell, hogy a táblán belül van-e a lépés, ha igen akkor oy + dir-t tud lépni ami azt jelenti, hogy előre 1-et a fekete gyaloggal, tehát a felhasználó szemszögéből nézve 1-et le.

Ütéseknél azt vizsgáljuk meg, hogy átlóba van-e ellenség, ha igen akkor odaléphet.

Végül a 4.if-em, hogy, ha a gyalog a kezdőhelyén van és nincs előtte barát,ellenség , akkor tudjon 2-t lépni.

A végén foreach-el beszínezem a *„zoldLista”* elemeit világoszöldre vagy sötétzöldre, ez a tábla színétől függ. Világoszöld = fehér mezők, sötétzöld = szürke mezők.

Bástya

```
public override void zold()
{
    base.zold();
    List<Mezo> zoldLista = new List<Mezo>();
    int ox = this.mezo.x;
    int oy = this.mezo.y;
    int dir =  1;
    while(Tablanbelul(ox,oy + dir) && !Barat(ox,oy + dir,szin))
    {
        zoldLista.Add(tabla[ox, oy + dir]);
        if (Ellenseg(ox, oy + dir, szin))
        {
            break;
        }
        dir++;
    }
    dir = 1;
    while (Tablanbelul(ox, oy - dir)  && !Barat(ox, oy - dir, szin))
    {
        zoldLista.Add(tabla[ox, oy - dir]);
        if (Ellenseg(ox, oy - dir, szin))
        {
            break;
        }
        dir++;
    }
    dir = 1;
    while (Tablanbelul(ox + dir, oy) && !Barat(ox + dir, oy, szin))
    {
        zoldLista.Add(tabla[ox + dir, oy]);
        if (Ellenseg(ox + dir, oy, szin))
        {
            break;
        }
        dir++;
    }
    dir = 1;
    while (Tablanbelul(ox - dir, oy) && !Barat(ox - dir, oy, szin))
    {
        zoldLista.Add(tabla[ox - dir, oy]);
        if (Ellenseg(ox - dir, oy, szin))
        {
            break;
        }
        dir++;
    }

    foreach (var v in zoldLista)
    {
        v.BackColor = v.vissza ? Color.DarkGreen : Color.LightGreen;
        v.Cursor = Cursors.Hand;
    }
```

14. Bástya lépései

A bástyák lépéseinél már nem kell figyelembe venni az irányt, hiszen mind a 4 fele tud lépni, ezért a *„dir”* állandó 1-es értéket kap és a cikluson belül mindig növeljük, ezáltal több mező lesz kijelölve. Mindegyik irányba megvizsgáljuk, hogy táblánbelül van-e a lépés, barát vagy ellenség majd a végén hozzáadjuk a „zoldLista”-hoz, majd beszínezzük.

Ló

A ló lépéséhez létrehoztam egy másik függvényt, aminek a neve „Lepes”. Ez a függvény kér egy jelenlegi helyzetet (ox,oy) és egy másik x,y értéket, ami az ellépési érték lesz.

A *„Lepes”*-ben megvizsgáljuk, hogy táblán belül illetve, hogy nincs-e barát az adott mezőn, ahova lépni akarunk.

```
public static bool Lepes(int ox, int oy, int x, int y, bool szin)
{
    return Tablanbelul(x + ox, y + oy) && !Barat(x + ox, y + oy, szin);
} // Lépés lónál
```

15. Függvény lóhoz

Ezt a függvényt használjuk fel a felülírt virtuális függvényben:

```
public class Lo : Babu
{
    public override string ToString()
    {
        return "Ló";
    }
    public Lo(Mezo mezo, bool szin) : base(mezo, szin)
    {
        this.Image = szin ? Image.FromFile(@"feher\WKnight.png") : this.Ima


    }
    public override void zold()
    {
        base.zold();
        List<Mezo> zoldLista = new List<Mezo>();
        int ox = this.mezo.x;
        int oy = this.mezo.y;
        base.zold();
        // FÖL 1 BALRA 2
        if (Lepes(ox, oy, -2, -1, szin))
        {
            zoldLista.Add(tabla[ox - 2, oy - 1]);
        }
        // FÖL 2 BALRA 1
        if (Lepes(ox, oy, -1, -2, szin))
        {
            zoldLista.Add(tabla[ox - 1, oy - 2]);
        }
        // FÖL 2 JOBBRA 1
        if (Lepes(ox, oy, 1, -2, szin))
        {
            zoldLista.Add(tabla[ox + 1, oy - 2]);
        }
        // FÖl 1 JOBBRA 2
        if (Lepes(ox, oy, 2, -1, szin))
        {
            zoldLista.Add(tabla[ox + 2, oy - 1]);
```

16. Ló lépései

Futó

A futó lépéseit ugyanúgy kellett megírnom, mint a bástyánál csak átlósan kellett nézni az irányokat. A király és a királynő lépéseit már csak ki kellett másolnom a bástyából és a futóból.

Ellenség esetén

Bábu levételt a *„Babu”* osztályban lévő klikkelésben írtam meg. Mindegyik esetet leírtam, hogy egy bábu mikor ütheti le a másikat. Először is valahogy a klikkelésbe bele kell írni, hogy mikor van ellenségre klikkelés. Amikor azt szeretnénk, hogy a kijelölt bábumat kékkel jelölje ki, akkor a „valtas”-nak és a „szin”-nek meg kell egyeznie. Ennek az elágazásnak az ellentettjét írtam tovább, tehát amikor a „valtas” nem egyezik meg a bábu színével, akkor tudjuk, hogy ellenségre kattintott a felhasználó. Először azt kell megnéznünk, hogy van-e valami kijelölve(if(kijeloltBabu != null)). Utána meg kell vizsgálni, hogy az adott mező zöld-e. Ha igen akkor az azt jelenti, hogy a bábunk le tudja ütni az ellenséges bábut. Az ellenséges bábu mezője törli az ellenséges bábut és odarakja a kijelölt bábunkat.

```
private void Mezo_MouseClick(object sender, EventArgs e)
{

    if (valtas == szin)
    {
        tablaszinreset();
        kek.Add(mezo);
        mezo.BackColor = Color.SteelBlue;
        zold();
        kijeloltBabu = this;
    }
    else
    {
        if (kijeloltBabu != null)
        {
            if (this.mezo.BackColor == Color.LightGreen || this.mezo.BackCo
            {

                this.mezo.Controls.Remove(this);
                this.mezo.Controls.Add(kijeloltBabu);
                kijeloltBabu.mezo.babu = null;
                this.mezo.babu = kijeloltBabu;
                kijeloltBabu.mezo = this.mezo;
                mentesdarab = true;
                belepes(this.mezo.x, this.mezo.y, this.mezo.babu);
                valtas = !valtas;
                sakk();
                ablak.debug3(valtas ? "Fehér játékos lép" : "Fekete játékos
                tablaszinreset();
                ablak.debug("Pozíció:");
                this.mezo.babu.Ellepeskiiras();
```

17(9). Váltás és a bábu szín közötti

Sima lépés

A „Babu”-ban megírtam azt az esetet, amikor leütünk egy bábut. A „Mezo”-ben pedig a sima lépést, amikor egy „semleges” mezőre lépünk.

```
private void Mezo_MouseClick(object sender, EventArgs e)
{
    if (kijeloltBabu != null) // Mezőre lépés bábu levétel nélkül.
    {
        if (this.BackColor == Color.LightGreen || this.BackColor == Color.DarkGreen)
        {
            this.Controls.Add(kijeloltBabu);
            kijeloltBabu.mezo.babu = null;
            this.babu = kijeloltBabu;
            this.babu.mezo = this;
            valtas = !valtas;
            sakk();
            belepes(this.x, this.y, this.babu);
            mentesdarab = true;
            ablak.debug3(valtas ? "Fehér játékos lép" : "Fekete játékos lép",valtas ? Color.White : Color.Black);
            this.babu.Ellepeskiiras();
        }
```

18.Lépés leütés nélkül

Itt ugyanúgy megvizsgáljuk, hogy van-e kijelölve bábu, utána viszont csak azt nézzük meg, hogy zöld-e a mező. Ha igen akkor arra a mezőre „addolja” a kijelölt bábunkat, amire nyomtunk.

Mentés és betöltés, levett bábuk, biztonsági mentés

A mentésre és a betöltésre is 2 külön eljárást írtam. Az eljárások neve *„mentes”* és *„betoltes”*.

Mentés

A mentés eljárásom létrehoz egy dialog menüt, ahol a felhasználó meg tudja adni a fájl nevét és helyét. A fájl első sora tartalmazza a *„valtas”* értékét és a megtett lépéseket.

Az első sor „-„ jellel van elválasztva és kicseréltem a sortöréseket „;”-re.

A többi sor tartalmazza a bábuk elhelyezkedését a táblán és a bábuk típusát, színét.

Az utolsó sor a levett bábuk értékeit tartalmazza.(Ezt a fejezet végén fejtem ki jobban)

```
public void mentes()
{
    SaveFileDialog dialog = new SaveFileDialog();
    dialog.InitialDirectory = @"C:\";
    dialog.RestoreDirectory = true;
    dialog.FileName = "*.txt";
    dialog.DefaultExt = "txt";
    dialog.Filter = "txt files (*.txt) | *.txt";
    if (dialog.ShowDialog() == DialogResult.OK)
    {
        mentesdarab = true;
        StreamWriter file = new StreamWriter(dialog.FileName);
        file.WriteLine(valtas + "-" + richTextBox1.Text.Replace('\n', ';'));
        for (int i = 0; i < 8; i++)
        {
            for (int j = 0; j < 8; j++)
            {

                if (tabla[i, j].babu != null)
                {
                    file.WriteLine(i.ToString() + " " + j.ToString() + " " + tabla[i, j].babu + " " + tabla[i, j].babu.szin);
                }
            }
        }
        file.WriteLine(FeherGyalog.ToString() + " " + FeherLo.ToString() + " " + FeherFuto.ToString() + " " + FeherBastya.ToS
        MessageBox.Show("Játék elmentve!", "Sakk");
        file.Close();
    }
```

19.Mentés

Betöltés

Betöltésnél is létrehoztam egy dialog-ot, ahol a felhasználó kiválasztja, melyik fájlt szeretné betölteni. Itt beleírtam egy try, catch funkciót, ha esetleg véletlenül egy rossz fájlt töltene be.

Az első sor legelső eleme lesz a *„valtas”* a második eleme pedig a megtett lépések. A „-„ jel azért kellett, hogy szét tudjuk splittelni az első sort. A „;” jelt vissza kell alakítani sortöréssé, hogy a richtextbox minden egyes lépést külön sorba jelenítsen meg.

A bábuk betöltése egy switch, case funkcióban töltjük be.

Egy példán keresztül bemutatva, ha a *„segedtomb[2]”* esete Gyalog, akkor hozzon létre egy Gyalog típusú példányt és a *„segedtomb”* többi részei megadja a bábu színét és elhelyezkedését a táblán. Ugyanígy működik a többinél is.

```
if (betoltes.ShowDialog() == DialogResult.OK)
{
    babureset();
    StreamReader file = new StreamReader(betoltes.FileName);
    string[] egesz = File.ReadAllLines(betoltes.FileName);
    string utolsosor = egesz[egesz.Length - 1];
    string[] seged = utolsosor.Split();
    string sor;
    sor = file.ReadLine();
    string[] elsosor = sor.Split('-');
    valtas = bool.Parse(elsosor[0]);
    ablak.debug2(elsosor[1].Replace(';', '\n').ToString() + "\n " + " -- Betöltés utáni lépések -- " + "\n");
    ablak.debug3(valtas ? "Fehér játékos lép" : "Fekete játékos lép", valtas ? Color.White : Color.Black);
    while(!file.EndOfStream)
    {
        sor = file.ReadLine();
        string[] segedtomb = sor.Split();
        switch (segedtomb[2])
        {
            case "Gyalog":
                {
                    Gyalog Gyalog = new Gyalog(tabla[int.Parse(segedtomb[0]), int.Parse(segedtomb[1])], bool.Parse(segedtomb[3]));
                    break;
                }
            case "Bástya":
                {
                    Bastya bastya = new Bastya(tabla[int.Parse(segedtomb[0]), int.Parse(segedtomb[1])], bool.Parse(segedtomb[3]));
                    break;
                }
            case "Ló":
                {
                    Lo Lo = new Lo(tabla[int.Parse(segedtomb[0]), int.Parse(segedtomb[1])], bool.Parse(segedtomb[3]));
                    break;
                }
            case "Futó":
                {
                    Futo futo = new Futo(tabla[int.Parse(segedtomb[0]), int.Parse(segedtomb[1])], bool.Parse(segedtomb[3]));
                    break;
                }
            case "Király":
                {
                    Kiraly futo = new Kiraly(tabla[int.Parse(segedtomb[0]), int.Parse(segedtomb[1])], bool.Parse(segedtomb[3]));
                    break;
                }
            case "Királynő":
                {
                    Kiralyno futo = new Kiralyno(tabla[int.Parse(segedtomb[0]), int.Parse(segedtomb[1])], bool.Parse(segedtomb[3]));
                    break;
                }
        }
```

20.Betöltés

Levett bábuk(utolsó sor a fájlban)

A képernyőn látható bal oldalt a bábuk képei és egy szám a képek jobb alsó sarkában. Ha valamelyikből levesznek egyet, akkor ez a szám nő 1-el. Ezeket az értékeket tároltam el a fájlban, az utolsó sorban.

Amikor leveszünk egy bábut, akkor szín alapján 2 listába mentem el őket. Majd végig megyek a listán és megvizsgálom, miből mennyi van egy switch, case funkcióval.

Biztonsági mentés működése

Beleépítettem a programba egy biztonsági mentést, ami arra szolgál, hogy ha a játékos játék közben véletlenül akárhogy elhagyja a játék formját(kilépés, új játék, vissza a főmenübe, stb) akkor automatikusan mentsen a játék.

Ha a játékos elmenti a játékot és utána nem lép egyetlen bábuval sem, de egyébként a biztonsági mentés aktív (be van nyomva a fül), a játék nem ment automatikusan.

Kiírások

A játékban sok helyen láthatunk kiírásoka, mint például a jelenlegi pozíció, megtett lépések stb. Mindegyik eljárás a label szövegét változtatja meg.

```
public void debug(string s)... //Pozíció
public void debug2(string s2)... //Richtextbox
public void debug3(string s3, Color szovegszin)... // Ki jön
public void fehergyaloglevet(string s4)...//Fehergyalog
public void feherlolevet(string s5)...//Feherló
public void feherfutolevet(string s6)...//Feherfutó
public void feherbastyalevet(string s7)...//Feherbástya
public void feherkiralnyolevet(string s8)...//Feherkirálynő
public void feketegyaloglevet(string s9)...//Feketegyalog
public void feketelolevet(string s10)...//Feketeló
public void feketefutolevet(string s11)...//Feketefutó
public void feketebastyalevet(string s12)...//Feketebástya
public void feketekiralnyolevet(string s13)...//Feketekirálynő
```

21.Összes kiírás, ami a játékban megtalálható

Jelenlegi pozíció

Ezeket az eljárásokat azért kellett létrehozni, mert nem mindig értem el a labeleket.

A jelenlegi pozíciót a *„zold”* nevű függvényben állítom be:

```
public virtual void zold()
{
    int ox = this.mezo.x;
    int oy = this.mezo.y;
    ablak.debug("Pozíció:" + (char)(65 + ox) + "/" + (8 - oy));
}
```

22.Alap „zold” eljárás

Az első az X-et adja meg a második az Y-t. Mivel az X betűkből áll ezért át kell konvertálni char típusúra. Az Y azért 8-oy, mert a tábla visszafele számoz, tehát ahol nekem az egyes szám van a tábla mellett az neki a 7.sor. Így, ha rányomok az első soron lévő bábura, akkor 8-ból fog 7-et kivonni.

Megtett lépések

```
public void Ellepeskiiras()
{
    ablak.debug2((szin ? "Fehér" : "Fekete") + " " + this + ":" + (char)(65 + this.mezo.x) + "/" + (8 - this.mezo.y));
}
```

23.Megtett lépések kiírása

Erre egy külön eljárást írtam meg. Kiírja először, hogy milyen színű bábu, majd a bábu típusát és végül koordinátáját. Ahhoz, hogy ki tudjam írni a bábuk típusát minden egyes bábu osztályban csinálnom kellett egy függvényt, ami egy stringet ad vissza.

```
public override string ToString()
{
    return "Gyalog";
}
```

24.String visszaadás osztályból

Fekete vagy fehér jön kiírás

Amikor a „valtas” értéke true, akkor fehérrel írja ki, hogy „fehér játékos lép”, ha hamis, akkor fekete. Amikor lépünk a bábuval, akkor a „valtas” értéke mindig megváltozik így a kiírás is vele együtt változik.

- A többi label a levett bábukhoz tartozik

Bábuk visszahozása

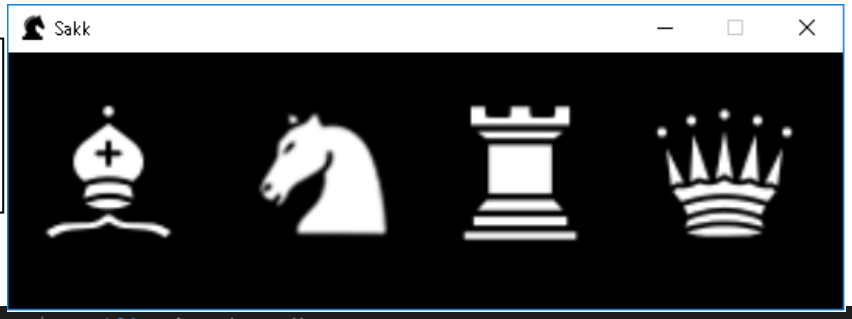
Amikor az ellenséges bábu belép a másik játékos első sorába, akkor megvizsgáljuk, hogy a bábu gyalog-e.

Ha igen akkor színtől függően 2 formot dob fel, ahol a felhasználó ki tudja választani melyik bábut szeretné visszahozni.

A „belepes” eljárásban írtam meg ezt a feltételt, ahol eldönti, hogy a bábu gyalog-e.

Ahhoz, hogy tudjuk hova kell lerakni az új bábut az eljárás kér egy x,y értéket és egy bábu típust.

Az új bábut lehelyezzük a „kijeloltBabu” x és y értékére, így oda kerül ahova a gyalog belépett.



25.Fehér bábuk visszahozása, design

```
15      public Visszahozas()
16      {
17          InitializeComponent();
18          //jatek vissza = new jatek();
19          jatek.kijeloltBabu.mezo.Controls.Remove(jatek.kijeloltBabu);
20      }
21
22      private void pictureBox1_Click(object sender, EventArgs e)//Futó
23      {
24
25          new jatek.Futo(jatek.kijeloltBabu.mezo, true);
26          this.Hide();
27      }
```

26.Koordináták átadása az új bábunak, a kijelölt bábu törlése

27.A belépett bábu gyalog-e

```
public static void belepes(int x, int y, Babu babu)
{
    if (babu is Gyalog)
    {
        if (babu.szin && y == 0)
        {
            Visszahozas feher = new Visszahozas();
            feher.Show();
        }

        if (!babu.szin && y == 7)
        {
            visszahozas2 fekete = new visszahozas2();
            fekete.Show();
        }
    }

} // Gyalog bejut a végére
```

Király és sakk

A legfontosabb és egyben legnehezebb feladat a sakk megírása volt. A sakkadás tökéletesen működik, viszont a sakkhelyzet nem. Ezalatt azt értem, hogy a király tud sakkot kapni, de ugyanúgy le lehet venni, sakkhelyzet esetén ugyanúgy el lehet lépni más bábuval akkor is, ha nem akadályozzuk meg a sakkhelyzetet illetve, ha meg is próbáljuk megakadályozni, akkor sem történik változás. Ezt még fejleszteni szeretném a jövőben.

Amikor a király sakkot kap a mezőjének színe piros lesz. Ha egy barátságos bábuval megpróbáljuk a sakkhelyzetet megszüntetni, ugyanúgy pirosan fog maradni a mező, viszont, ha egy ellenséges bábu belép a sakkot adó bábu elé (ami alapjáraton nem lehetséges), akkor a mező visszavált az eredeti színére.

Egy virtuális függvényt és két eljárást alkalmaztam arra, hogy felismerje a sakk helyzetet a játék. A királyokat elmentettem a bábuk színei alapján, *„feherkiraly”* illetve *„feketekiralyba”*.

„sakk” az eljárás neve, ami csak megvizsgálja szín alapján(fehér vagy fekete király), hogy sakk helyzet van-e, *„sakkotKeres”* eljárás csak a „Kiraly” osztályomban létezik és azt vizsgálja, hogy a különböző útvonalakon ellenséges bábu van-e, a *„sakkotTudAdni”* a virtuális függvényem, amit minden bábunál átírok.

Működésük egyszerű. A *„sakkotKeres”*-ben megnézem az útvonalakat a királyból kiindulva mindenféle lehetséges irányban. Ha ellenséget talál, akkor átad a *„SakkotTudAdni”* virtuális függvényemnek két számot.A *„SakkotTudAdni”* függvényem két értéket vár egy x-et és egy y-t. Minden más bábuban a *„SakkotTudAdni”*-t átírom. Mindig más visszatérési értéket adok meg neki, nyilvánvalóan, ami a bábura jellemző lépés. Ha *„sakkotKeres”* értéke olyan szám, ami megegyezik a *„SakkotTudAdni”* értékével, akkor tudjuk, hogy sakk van. Erre egy boolt(ennek a neve is sakk, ezért majd, ha meg kell említenem elé rakok zárójelben egy b-t) használtam, ami alapból hamison van, ha pedig teljesül amit feljebb írtam, akkor változik igazra. Ha a *” (b)sakk”* igaz, akkor a király mezőjének a színe piros lesz, ha hamis, akkor egy tábla szín visszaállítást csinál.

A *„sakkotKeres”* eljárást a sakkban hívom meg a királyok színei alapján. A *„sakk”* eljárást pedig minden egyes lépés és ütés után meghívom, tehát a „Mezo” klikkelésébe meg a „Babu” klikkelésébe is beleírom.

```
public void sakkotKeres()
{
    int dir = 1;
    int ox = this.mezo.x;
    int oy = this.mezo.y;
    bool sakk = false;
    //ELŐRE
    while (Tablanbelul(ox, oy + dir) && !Barat(ox, oy + dir, szin))
    {

        if (Ellenseg(ox, oy + dir, szin))
        {
            if(tabla[ox, oy + dir].babu.sakkotTudAdni(0, +dir))
            {
                sakk = true;
            }
            break;
        }
        dir++;
    }
    //HÁTRA
    dir = 1;
    while (Tablanbelul(ox, oy - dir) && !Barat(ox, oy - dir, szin))
    {
        if (Ellenseg(ox, oy - dir, szin))
        {
            if (tabla[ox, oy - dir].babu.sakkotTudAdni(0, -dir))
            {
                sakk = true;
            }
            break;
        }
        dir++;
    }
```

30. Sakk vizsgálása a királyban, előre és hátra

```
public virtual bool sakkotTudAdni(int x, int y)
{
    return true;
}
```

28. *”SakkotTudAdni”* alap függvény

```
public override bool sakkotTudAdni(int x, int y)
{
    return x == 0 || y == 0;
}
} // Bástya
```

29. *”SakkotTudAdni”* bástyában

Bástya vagy királynő esetén, ha a *„sakkotKeres”* a *„SakkotTudAdni”*-nak x vagy y értékére 0-t ad vissza, akkor biztos, hogy a bástya vagy a királynőnk vízszintesen vagy függőlegesen egy vonalban van a királlyal, tehát a király sakkban van.

```
public static void sakk()
{
    if (valtas)
    {
        feherkiraly.sakkotKeres();
    }
    else
    {
        feketekiraly.sakkotKeres();
    }
} // Sakk
```

31. A *„SakkotKeres”* meghívása

Fejlesztések

- A felhasználó tudjon időt beállítani
- Játékos nevek hozzáadása egyénileg a felhasználó szerint
- A sakkhelyzet tökélesítése

Tapasztalatok

Sok mindent tanultam konzulens tanáromtól mialatt a programot készítettem. Sokszor kellett segítségét kérnem egyes dolgokban és emlékszem, hogy a program előtt utáltam OOP-an programozni, de ezek után jobban megértettem és meg is szerettem. Olyan új funkciókat is tanultam, amit az iskolában még nem és valószínűleg nem is fogunk tanulni.