Logička igra Četiri u nizu s više razina težine u programskom jeziku C#

Trputec, Denis

Master's thesis / Diplomski rad

2018

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:791426

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-23**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

LOGIČKA IGRA „ČETIRI U NIZU“ S VIŠE RAZINA TEŽINE U PROGRAMSKOM JEZIKU C#

Diplomski rad

Denis Trputec

Osijek, 2018.

SADRŽAJ:

1. UVOD 1
1.1. Zadatak završnog rada 1
2. PROGRAMSKI JEZIK C# 2
2.1. Uvod u C# 2
2.2. .NET 2
3. VISUAL STUDIO 2017 4
3.1. Uvod u Visual Studio 4
4. ČETIRI U NIZU 5
4.1. Uvod u igru 5
4.2. Pravila igre 5
4.3. Taktike 5
5. PROGRAMSKI KOD 7
5.1. Klase 7
5.1.1. *Ploca* 7
5.1.2. *AI* 10
5.1.3. Ostale klase 16
5.2. Forme 16
5.2.1. *GlavniIzbornik* 16
5.2.2. *JedanIgrac* 18
5.2.3. *Igra_1P* 23
5.2.4. Ostale forme 31
ZAKLJUČAK 32
LITERATURA 33
SAŽETAK 34
ABSTRACT 35
ŽIVOTOPIS 36

1. UVOD

Logička igra „Četiri u nizu“ je komplicirana igra sa jednostavnim pravilima. Kako bi se napravila kvalitetna računalna igra potrebno je, osim poznavanja jednog od programskih jezika i korištenja programskog okruženja, razumjeti i znati igrati samu igru kako bi se onda to znanje moglo prenijeti i na umjetnu inteligenciju računala koje će igrati tu igru.

Na samom početku rada govori se općenito o tehnologijama korištenim pri izradi aplikacije. Kreće se sa upoznavanjem C# programskog jezika i .NET platforme, te odabranog razvojong okruženja Visal Studio 2017. Nadalje se čitatelj upoznaje sa samom igrom, njenim pravilima i taktikama. U zadnjem dijelu rada objašnjen je korišteni kod, te su prikazane slike na kojima se vidi izgled aplikacije.

1.1. Zadatak završnog rada

Zadatak ovog diplomskog rada je izraditi aplikaciju koristeći programski jezik C#. Aplikacija mora omogućiti korisniku igranje popularne logičke igre „Četiri u nizu“. Igra također mora sadržavati više razina težine.

2. PROGRAMSKI JEZIK C#

2.1. Uvod u C#

Programski jezik C# je moderan i u potpunosti objektno orijentiran jezik opće namjene koji je razvila tvrtka Microsoft kao dio svoje .NET platforme. Ideja za njegovo nastajanje kreće još od sredine devedesetih godina prošlog stoljeća kada je tvrtka Sun predstavila Javu. Microsoft je tada vidio sjajan potencijal u tom programskom jeziku pa su čak napravili i Java 1.1 standard, zbog čega ih je 1997. godine tvrtka Sun tužila. Microsoft se nakon toga distancirao od Jave.

Sa razvojem programskog jezika C# Microsoft je krenuo 2000. godine. Razvijao ga je mali tim ljudi predvođenih Andersom Hejlsbergom i Scottom Wiltamuthom. Hejlsberg je već tada bio poznat po tome što je razvio Turbo Pascal, a također je i vodio tim ljudi koji su dizajnirali Borland Delphi, jedno od prvih integriranih razvojnih okruženja na relaciji klijent-poslužitelj. [2]

Kreiran je s ciljem da odgovori na nedostatke koje su imali postojeći programski jezici kao što su Visual Basic, C i C++ i spoji njihove dobre strane, a ujedno bude konkurencija Javi. Da je on zapravo neka vrsta proširenja na C++ pokazuje nam i njegovo ime jer znak „#“ su zapravo četiri povezana znaka „+“.

C# je prilično jednostavan, što ga čini dobrim za početnike, ali istovremeno uključuje svu potrebnu podršku za strukturirano i objektno orijentirano programiranje koja se očekuje od jednog modernog programskog jezika. Drugim riječima namijenjen je za programiranje velikih i kompliciranih programa, a k tome je i lagan za naučiti. [1]

2.2. .NET

Ako ste učili jezike poput C-a i Jave mogli ste to učiniti bez puno brige o platformi za koju će te programirati jer ti su jezici imali inačice koje su bile kompatibilne s raznim operacijskim sustavima, bilo da se radilo o Windowsima ili o Unixu. Međutim, programski jezik C# je stvoren posebno za .NET. Postoje verzije .NET-a koje mogu biti korištene i na drugim platformama, ali za sada se slabo koriste jer je većina programa pisana s ciljem pokretanja na operacijskom sustavu

Windows. Osnovu .NET-a predstavlja .NET Framework, a najjednostavnije rečeno, to je sustav koji nadograđuje mogućnosti samog operacijskog sustava. Radi se o posebnoj infrastrukturi koja programerima nudi gotova rješenja i funkcionalnosti da bi se ubrzao i pojednostavio razvoj aplikacija svih vrsta i oblika. [2]

Kada je Microsoft najavio C# 1.0 u srpnju 2000. godine, ta objava je bila samo dio još većeg događaja, a to je objava .NET platforme. Microsoft je uložio čak 80% svog razvojnog budžeta u razvoj .NET tehnologije, a rezultati su bili impresivni. 2005. godine Microsoft je na tržište izbacio novu verziju jezika (C# 2.0), platformu i alate. Njihov cilj je bio drastično smanjiti količinu koda kojeg se mora pisati, te da se izrada *desktop* i *web* aplikacija olakša enkapsuliranjem. Razvoj se nastavio i dalje te danas koristimo .NET verziju 4.7.2, a u statusu razvoja su još i verzije 4.7.3 i 4.8.

Područje koje obuhvaća .NET je ogromno, a platforma se sastoji od triju zasebnih grupa proizvoda. Prvu čine set jezika uključujući C# i Visual Basic, set razvojnih alata uključujući Visual Studio te snažan alat za izradu aplikacija CLR (engl. *Common Language Runtime)*. Drugu čine Enterprise Server-i, a treću verzija .NET-a koja omogućuje pokretanje i na drugim uređajima od mobilnih telefona do igraćih konzola. [1]

3. VISUAL STUDIO 2017

3.1. Uvod u Visual Studio

Kada je u pitanju programiranje u C# programskom jeziku nekako je prirodno odabrati upravo njega. Visual Studio je integrirano razvojno okruženje (engl. *Integrated Development Enviroment,* IDE) koje je razvio Microsoft kako bi olakšalo izradu Windows i mrežnih aplikacija i upravo zbog te povezanosti (C# je također razvio Microsoft) je ono najbolji odabir. Najbolje alternative na tržištu su besplatna okruženja Shark Develop i Mono Develop.

Visual Studio nudi mnoge mogućnosti od kojih su najosnovnije uređivač koda, integrirani program za otklanjanje pogrešaka i dizajnere (za Windows forme, klase, baze podataka itd.). Uređivač koda je nešto što imaju sva integrirana programska okruženja. On može istaknuti sintaksu i izvršiti kompilaciju koda koristeći IntelliSense za varijable, funkcije, metode i petlje. Upravo ta pozadinska kompilacija koda u realnom vremenu omogućava korisniku brže prepravljanje grešaka koje bi inače našao tek prilikom pokretanja programa. Također može sakriti i dijelove koda kako bi cjelokupan kod bio pregledniji, a koristi i inkrementalno pretraživanje. Program za otklanjanje pogrešaka (engl. *debugger*) radi razini izvornog koda i strojnog koda. Najčešće korištena njegova mogućnost je postavljanje točaka zaustavljanja (engl. *Breakpoints*) u kodu. Njima se može odabrati linija koda gdje će se zaustaviti izvršavanje programa i omogućiti korisniku pregledavanje izvršavanja koda liniju po liniju kako bi lakše pronašao grešku. Program za otklanjanje pogrešaka također ima opciju uređivanja pomoću koje korisnik može izmijeniti programski kod za vrijeme trajanja procesa.

Visual Studio trenutno podržava trideset šest različitih programskih jezika što čini većinu postojećih programskih jezika. Jezici koji su ugrađeni su C, C++, Visual Basic, C#, F#, JavaScript, TypeScript, XML, XSLT, HTML i CSS. Ostali jezici popularni jezici poput Pythona i Rubyja su omogućeni korištenjem *plug-in*-ova. Java (iako je bila) nije podržana.

4. ČETIRI U NIZU

4.1. Uvod u igru

Četiri u nizu, poznata i pod imenom Spoji četiri (engl. *Connect 4*), je popularna logička igra za dva igrača. Igru je počeo proizvoditi Milton Bradley (današnji Hasbro) u veljači 1974. godine pod imenom „Connect 4“, ali pretpostavlja se da se igra prvi put pojavila oko početka 20. stoljeća. Igra se sastoji od postolja u kojem se nalazi dvodimenzionalno polje od sedam stupaca i šest redaka te od dvadeset jednog žetona jedne boje i dvadeset jednog žetona druge boje. Žetoni se ubacuju u postolje po stupcima, te žeton pada na dno stupca, odnosno na prvo prazno mjesto gledajući odozdo. Cilj igre je, kao što i ime govori, spojiti četiri žetona iste boje u niz. Danas također postoje i mnoge različite varijante ove igre, a razlike su najčešće u veličini polja i u broju žetona kojih se treba spojiti u niz da bi igrač pobijedio.

4.2. Pravila igre

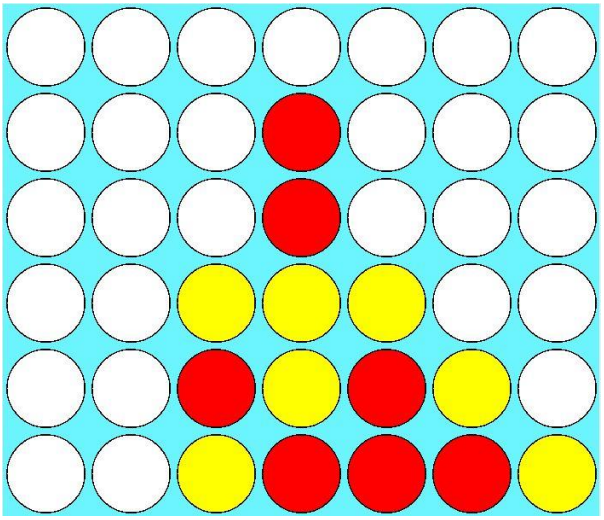
Cilj igre je spojiti četiri žetona jedne boje u liniju koja može biti horizontalna, okomita ili kosa. Na početku igre igrači odabiru boju žetona s kojom će igrati i dogovaraju se tko će prvi igrati. Nakon toga naizmjence ubacuju žetone u postolje sve dok jedan igrač ne ostvari cilj, odnosno pobijedi. Ukoliko se popune svih četrdeset dva polja u postolju, a igra još uvijek nema pobjednika, tada je igra završila neriješeno.

4.3. Taktike

Četiri u nizu je riješena igra što znači da ukoliko prvi igrač odigra svaki potez savršeno, uvijek će pobijediti bez obzira na to kako protivnik ubaci svoje žetone. Prvi kojem je to pošlo za rukom je James Dow Allen 1988. godine.

Algoritmi koje koristi računalo korišteni u ovoj aplikaciji neće biti toliko pametni jer time bi time i sama igra izgubila njenu zabavnu stranu. Stoga će računalo koristiti metode kao što su pregledavanje osnovnih poteza. To znači da će pretražiti igraću ploču i ukoliko vidi da sa sljedećim žetonom može pobijediti, ubaciti žeton u to polje. Ako ne može tada će provjeriti može li protivnik

sljedećim žetonom pobijediti i ako je tako, računalo će tu pobjedu spriječiti ubacivanjem svog žetona na to mjesto. Koristi se i metoda sprječavanja dvostrukog napada na način da kada računalo primijeti da protivnik ima svoje žetone u određenim pozicijama koje mu omogućavaju takav napad, računalo će ubacivanjem svog žetona na predviđeno mjesto taj napad spriječiti. Također postoji i taktika „sedmica“ (položaj u kojem se nalaze žetoni podsjeća na broj sedam) kojom računalo samo pokušava izvesti dvostruki napad.



Slika 4.3.1. – Napad „sedmicom“

Predzadnja taktika koju računalo koristi je ubacivanje žetona u sredinu kada za to postoji prilika. Računalo će se uvijek truditi imati žeton više od protivnika u sredini kako bi time umanjilo protivnikove šanse za pobjedu. Primjerice, ukoliko igrač nema niti jedan žeton u srednjem stupcu ploče, tada on gubi mogućnost da složi svoje žetone horizontalno i dijagonalno. Zadnji preostala taktika je da se žeton ubaci nasumično. Ali niti tada to nije u potpunosti nasumično jer će računalo uvijek težiti ubacivanju žetona bliže sredini, te će uvijek paziti da svojim žetonom ne omogući pobjedu protivniku.

5. PROGRAMSKI KOD

U ovom poglavlju biti će detaljno objašnjen programski kod korišten pri izradi aplikacije. Aplikacija se sastoji od ukupno četrnaeast klasi i formi (sedam klasi i sedam formi), pa je stoga poglavlje podijeljeno na dva dijela.

5.1. Klase

5.1.1. *Ploca*

Klasa *Ploca* je prva i najbitnija klasa u programskome kodu s obzirom da sadrži atribut *polje* u kojem se pohranjuje trenutni izgled igrače ploče. Također sadrži metode kojima se provjerava je li igra završena.

```
public class Ploca
    {
        private int[,] polje = new int[7, 6];
        private int brojacPoteza;

        public Ploca(int[,] polje, int brojacPoteza)
        {
            this.polje = polje;
            this.brojacPoteza = brojacPoteza;
        }

        public int[,] Polje
        {
            get { return polje; }
            set { polje = value; }
        }
        public int BrojacPoteza
        {
            get { return brojacPoteza; }
            set { brojacPoteza = value; }
        }
```

Slika 5.1.1. – Atributi, konstruktor i get/set metode

Na početku klase se deklariraju atributi koje klasa sadrži. Prvi atribut je već spomenuto *polje* koje je dvodimenzionalno polje tipa *int* što znači da se u njega spremaju cjelobrojne vrijednosti i veličine je 7x6. Vrijednosti koje se mogu naći u kodu su '0' koja označava prazno mjesto, '1' koja označava da se tu nalazi žeton igrača (ili prvog igrača ako je u pitanju igra za dva igrača) te '2' koja označava žeton umjetne inteligencije (odnosno drugog igrača za igru u dva igrača). Drugi atribut je *brojacPoteza* koji je tipa *int* koji govori koji je trenutni potez na redu.

Najveća vrijednost koju može postići je '42' s obzirom da toliko ima polja u igraćoj ploči. Nakon toga slijedi konstruktor koji služi za pridruživanje vrijednosti atributima prilikom kreiranja objekta te standardne *get/set* metode koji se dohvaćaju vrijednosti atributa ili pohranjuju neke nove vrijednosti. U buduće se klasični konstruktori i *get/set* metode poput ovih neće objašnjavati u daljnjem tekstu.

```
public bool ProvjeraPobjede(int a, int stupac, int redak)
        {
            //okomito
            if (redak > 2)
                if (polje[stupac, redak - 1] == a && polje[stupac, redak - 2] == a
                       && polje[stupac, redak - 3] == a)
                {
                    return true;
                }

            //horizontalno
            int br = 0;
            for (int i = stupac - 3; i <= stupac + 3; i++)
            {
                if (i >= 0 && i <= 6 && i != stupac)
                {
                    if (polje[i, redak] == a)
                        br++;
                    else
                        br = 0;
                    if (br == 3)
                    {
                        return true;
                    }
                }
            }

            //dijagonalno prema gore
            br = 0;
            int j = redak - 3;
            for (int i = stupac - 3; i <= stupac + 3; i++, j++)
            {
                if (i >= 0 && i <= 6 && i != stupac && j >= 0 && j <= 5)
                {
                    if (polje[i, j] == a)
                        br++;
                    else
                        br = 0;
                    if (br == 3)
                    {
                        return true;
                    }
                }
            }

            //dijagonalno prema dolje
            br = 0;
            j = redak + 3;
            for (int i = stupac - 3; i <= stupac + 3; i++, j--)
```

```
            {
                if (i >= 0 && i <= 6 && i != stupac && j >= 0 && j <= 5)
                {
                    if (polje[i, j] == a)
                        br++;
                    else
                        br = 0;
                    if (br == 3)
                    {
                        return true;
                    }
                }
            }

            return false;
        }
```

Slika 5.1.2. – Metoda *ProvjeraPobjede()*

Metoda *ProvjeraPobjede()* kao argumente prima tri varijable pomoću kojih određuje je li igrač koji je ubacio žeton pobijedio. Varijabla *a* ima vrijednosti '1' ili '2' ovisno želi li se provjeriti igrača (igrača 1) ili umjetnu inteligenciju (igrača 2). Varijable *stupac* i *redak* pak govore na koju poziciju u igraćoj ploči je ubačen žeton.

Prvo se provjerava je li postignuta pobjeda okomito, odnosno jesu li sljedeća tri žetona u stupcu ispod ubačenog njemu jednaka. Ako jest, tada metoda vraća vrijednost '*true*' koja označava pobjedu.

Nakon toga se provjerava je li postignuta pobjeda horizontalno. To se radi tako da se *for* petljom prođe kroz cijeli redak gdje se nalazi ubačeni žeton. Provjerava se da li je svaki žeton (osim ubačenog) u tom retku jednak ubačenom i ukoliko jest brojač *br* će se uvećati za jedan. Ukoliko nije brojač će se postaviti na nulu. Ukoliko brojač dosegne vrijednost '3' metoda vraća vrijednost '*true*'.

Slična logika kao i za horizontalni slučaj se koristi za oba dijagonalna slučaja s razlikom da se *for* petljom prolazi dijagonalno (rastuće i padajuće) u odnosu na ubačeni žeton.

```
public bool ProvjeraNerjesenog()
        {
            if (brojacPoteza == 42)
                return true;
            return false;
        }
```

Slika 5.1.3. – Atributi, konstruktor i get/set metode

Metoda *ProvjeraNerjesenog()* provjera je li *brojacPoteza* stigao do vrijednosti '42', odnosno je li cijela ploča popunjena. S obzirom da se ova metoda isključivo poziva nakon metode *ProvjeraPobjede()*, tada se u slučaju da je brojač jednak '42' može sa sigurnošću reći da je igra završila neodlučeno.

5.1.2. *AI*

Klasa *AI* je s druge strane najkompleksnija klasa u programskome kodu. Osim što sadrži tri atributa i njihove *get/set* metode i konstruktor, u njoj se još nalazi trinaest metoda. Atributi koje sadrži su *ime*, *boja* i *razina*. U atributu *razina* se nalazi broj od jedan do tri, ovisno koju je razinu težine odabrao korisnik.

```
private int VratiRedak(int[,] polje, int stupac)
        {
            for (int redak = 0; redak < 6; redak++)
                if (polje[stupac, redak] == 0)
                    return redak;
            return -1;
        }
```

Slika 5.1.4. – Metoda *VratiRedak()*

Ova metoda kao argumente prima dvodimenzionalno polje, odnosno trenutni izgled igraće ploče i indeks stupca. Vraća indeks prvog slobodnog retka u zadanom stupcu ili ukoliko nema više slobodnih mjesta vraća '-1'.

```
private int VratiStupac(Random rand)
        {
            int stupac = -1;
            int a = rand.Next(0, 22); //u a se nalazi broj u intervalu [0,21]

            if (a > 6 && a < 15) stupac = 3;  // 8/22
            else if (a > 2 && a < 7) stupac = 2;  // 4/22
            else if (a > 14 && a < 19) stupac = 4;
            else if (a > 0 && a < 3) stupac = 1;   // 2/22
            else if (a > 18 && a < 21) stupac = 5;
            else if (a == 0) stupac = 0;   // 1/22
            else if (a == 21) stupac = 6;

            return stupac;
        }
```

Slika 5.1.5. – Metoda *VratiStupac()*

Metoda *VratiStupac* nasumično vraća indeks jednog od stupaca. Vjerojatnost da će vratiti indeks srednjeg stupca iznosi 4/11 i što je stupac dalje sredini vjerojatnost da će se njegov indeks vratiti je manja, odnosno za rubne stupce iznosi 1/22. Time se forsira da računalo odabere stupac što bliže sredini.

```
private int OsnovniPotezi(Ploca ploca)
        {
            for (int stupac = 0; stupac <= 6; stupac++)
            {
                int redak = VratiRedak(ploca.Polje, stupac);
                if (redak != -1)
                {
                    if (ploca.ProvjeraPobjede(2, stupac, redak))
                        return stupac;
                    else if (ploca.ProvjeraPobjede(1, stupac, redak))
                        return stupac;
                }
            }
            return -1;
        }
```

Slika 5.1.6. – Metoda *OsnovniPotezi()*

Kao argument prima objekt tipa *Ploca*, odnosno trenutno stanje igraće ploče. Funkcija prolazi kroz sve stupce i prvo poziva metodu *VratiRedak()* da sazna gdje je prvo slobodno mjesto u stupcu ako ga ima, a zatim provjerava pomoću metode objekta *ploca* da li ubacivanjem na to mjesto ostvaruje pobjedu ili blokira protivničku pobjedu. Ukoliko da, onda vraća indeks stupca, ako ne, tada vraća '-1'.

```
private int RandomUbaci(Ploca ploca)
        {
            Random rand = new Random();
            while (true)
            {
                int stupac = VratiStupac(rand);
                if (ploca.Polje[stupac, 5] == 0)  //ako ima mjesta u stupcu
                    return stupac;
            }
        }
```

Slika 5.1.7. – Metoda *RandomUbaci()*

Ova metoda se poziva kao zadnje rješenje za umjetnu inteligenciju. Sastoji se od beskonačne petlje i nasumično traži stupac u kojem ima mjesta te vraća njegov indeks. Ova metoda

nikad neće zaglaviti u beskonačnoj petlji, jer uvijek postoji slobodno mjesto u ploči ukoliko igra nije gotova.

```
private int PametnijeRandomUbaci(Ploca ploca)
        {
            Random rand = new Random();
            int pokusaj = 0;

            while (pokusaj < 30)
            {
                int stupac = VratiStupac(rand);
                int redak = VratiRedak(ploca.Polje, stupac);
                if (redak != -1)
                {
                    if (redak > 4 || ploca.ProvjeraPobjede(1, stupac, redak + 1) ==
false)
                        return stupac;
                    else
                        pokusaj++;
                }
            }
            return -1;
        }
```

Slika 5.1.8. – Metoda *PametnijeRandomUbaci()*

Ova metoda je „pametnija“ verzija prethodne. Također nasumično traži stupac sa praznim mjestom, ali tada provjerava da li će protivnik taj potez iskoristiti u svoju korist. Odnosno provjerava hoće li korisnik ukoliko stavi svoj žeton na onaj koji želi staviti metoda pobijediti. Ako neće metoda vraća indeks stupca, ako hoće, ponovno nasumično odabire stupac. Petlja se ponavlja maksimalno trideset puta jer postoji mogućnost da niti jedan stupac nije „siguran“, pa se na taj način izbjegava ostanak u beskonačnoj petlji.

```
private int SprijeciDvostrukiNapad_Vodoravno(Ploca ploca)
        {
            for (int j = 0; j < 6; j++)
            {
                for (int i = 0; i < 3; i++)
                {
                    // kombinacije 1-3
                    if (ploca.Polje[i, j] == 0 && ploca.Polje[i + 1, j] == 0
                         && ploca.Polje[i + 2, j] == 1 && ploca.Polje[i + 3, j] == 1
                         && ploca.Polje[i + 4, j] == 0)
                    {
                        if ((j == 0) || (j > 0 && ploca.Polje[i, j - 1] != 0
                              && ploca.Polje[i + 1, j - 1] != 0
                              && ploca.Polje[i + 4, j - 1] != 0))
                            return i + 1;
                    }
```

```
                // kombinacije 4-6
                if (ploca.Polje[i, j] == 0 && ploca.Polje[i + 1, j] == 1
                      && ploca.Polje[i + 2, j] == 0 && ploca.Polje[i + 3, j] == 1
                      && ploca.Polje[i + 4, j] == 0)
                {
                    if ((j == 0) || (j > 0 && ploca.Polje[i, j - 1] != 0
                      && ploca.Polje[i + 2, j - 1] != 0
                      && ploca.Polje[i + 4, j - 1] != 0))
                        return i + 2;
                }
                // kombinacije 7-9
                if (ploca.Polje[i, j] == 0 && ploca.Polje[i + 1, j] == 1
                      && ploca.Polje[i + 2, j] == 1 && ploca.Polje[i + 3, j] == 0
                          && ploca.Polje[i + 4, j] == 0)
                {
                    if ((j == 0) || (j > 0 && ploca.Polje[i, j - 1] != 0
                          && ploca.Polje[i + 3, j - 1] != 0
                          && ploca.Polje[i + 4, j - 1] != 0))
                        return i + 3;
                }
            }
        }
        return -1;
    }
```

Slika 5.1.9. – Metoda *SprječiDvostrukiNapad_Vodoravno()*

Postoje određene situacije kada se u retku može naći prostora za dvostruki napad od strane protivnika. Do toga dolazi kada se na prostoru od pet mjesta, na tri središnja nalaze dva protivnička žetona i jedno prazno mjesto, te dva prazna mjesta na rubovima. Ova metoda prolazi kroz svih devet takvih kombinacija i vraća indeks onog stupca pomoću kojeg će spriječiti takvu situaciju. Ako takve situacije ne postoje vraća '-1'.

Ista logika je i kod sprečavanja nastajanja dvostrukog dijagonalnog napada.

```
private int UbaciUSredinu(Ploca ploca)
        {
            int br_Igrac = 0; int br_AI = 0;
            for (int j = 0; j <= 5; j++)
            {
                if (ploca.Polje[3, j] == 1)
                    br_Igrac++;
                else if (ploca.Polje[3, j] == 2)
                    br_AI++;
            }
            if (br_Igrac >= br_AI)
            {
                int redak = VratiRedak(ploca.Polje, 3);
                if (redak != -1 && (redak == 5
                    || ploca.ProvjeraPobjede(1, 3, redak + 1) == false))
                    return 3;
            }
```

```
            return -1;
        }
```

Slika 5.1.10. – Metoda *UbaciUSredinu()*

Metoda ima dva brojača pomoću kojih broji koliko žetona u srednjem stupcu ima UI, a koliko protivnik. Ukoliko protivnik ima jednak ili veći broj žetona u stupcu, metoda vraća indeks srednjeg stupca. U protivnom vraća '-1'.

```
private int StvoriSedmicu(Ploca ploca)
   {
        if (((ploca.Polje[3,1] != 1) && (ploca.Polje[3,2] != 1)
              && (ploca.Polje[2,2] != 1) && (ploca.Polje[4,2] != 1))
              && ((ploca.Polje[2,0] != 1) || ploca.Polje[4,0] != 1))
        {
                if ((ploca.Polje[3, 1] == 0) && (ploca.Polje[3, 0] != 0)
                      && !(ploca.ProvjeraPobjede(1, 3, 2)))
                    return 3;
                else if ((ploca.Polje[2, 0] == 0) && !(ploca.ProvjeraPobjede(1, 2, 1)))
                    return 2;
                else if ((ploca.Polje[4, 0] == 0) && !(ploca.ProvjeraPobjede(1, 4, 1)))
                    return 4;
                else if ((ploca.Polje[3, 2] == 0) && (ploca.Polje[3, 1] == 2)
                          && !(ploca.ProvjeraPobjede(1, 3, 3)))
                    return 3;
                else if ((ploca.Polje[2, 2] == 0) && (ploca.Polje[2, 1] != 0)
                          && !(ploca.ProvjeraPobjede(1, 2, 3)))
                    return 2;
                else if ((ploca.Polje[4, 2] == 0) && (ploca.Polje[4, 1] != 0)
                          && !(ploca.ProvjeraPobjede(1, 4, 3)))
                    return 4;
            }
            return -1;
        }

private int NapadniSedmicom(Ploca ploca)
        {
            if (ploca.Polje[3,1] == 2 && ploca.Polje[2,0] == 2 && ploca.Polje[3, 2] ==
2 && ploca.Polje[2,2] == 2  && ploca.Polje[4,2] == 2 && ploca.Polje[5,3] == 0)
            {
                return 5;
            }
            else if (ploca.Polje[3, 1] == 2 && ploca.Polje[4, 0] == 2 && ploca.Polje[3,
2] == 2 && ploca.Polje[2, 2] == 2 && ploca.Polje[4, 2] == 2 && ploca.Polje[1, 3] == 0)
            {
                return 1;
            }
            return -1;
        }
```

Slika 5.1.11. – „Sedmica“

Pomoću metode *NapadniSedmicom()* UI pokušava napraviti zamku protivniku postavivši žetone u obliku „sedmice“ jer time može stvoriti dvostruki napad. Ukoliko ne može staviti žeton na neko od potrebnih mjesta vraća '-1'. Metodom *NapadniSedmicom()* provjerava se da li je „sedmica“ složena i ako jest, vraća se indeks stupca kojim se omogućuje napad. Vraća indeks '5' ako je „sedmica“ okrenuta na pravu stranu i '1' ako je zrcaljena, a '–1' ukoliko nema potrebe za napadom.

```
public int RazinaTesko(Ploca ploca)
        {
            if (ploca.BrojacPoteza == 1) //ako je prvi na potezu, baci u sredinu
                return 3;

            int stupac = OsnovniPotezi(ploca);
            if (stupac != -1)
                return stupac;

            stupac = StvoriSedmicu(ploca);
            if (stupac != -1)
                return stupac;

            stupac = NapadniSedmicom(ploca);
            if (stupac != -1)
                return stupac;

            stupac = SprijeciDvostrukiNapad_Vodoravno(ploca);
            if (stupac != -1)
                return stupac;

            stupac = SprijeciDvostrukiNapad_Dijagonalno(ploca);
            if (stupac != -1)
                return stupac;

            stupac = UbaciUSredinu(ploca);
            if (stupac != -1)
                return stupac;

            stupac = PametnijeRandomUbaci(ploca);
            if (stupac != -1)
                return stupac;

            return RandomUbaci(ploca);
        }
```

Slika 5.1.12. – Metoda *RazinaTesko()*

Preostale su još tri glavne metode: *RazinaLako()*, *RazinaSrednje()* i *RazinaTesko()*. Poziva se jedna od njih onda kada je na potezu UI, ovisno o razini težine koju je korisnik odabrao. Na slici 5.1.12. se nalazi logika za najtežu razinu koja koristi sve prethodno objašnjene metode. Izuzev tih metoda, ukoliko UI (na najtežoj razini samo) započinje igru, uvijek će ju započeti ubacivanjem u srednji stupac.

5.1.3. Ostale klase

Jedna od važnijih klasa u programu je klasa *Igrac*, a ona je ujedno i najjednostavnija klasa u programu. Sadrži dva atributa, prvi je *ime* i tipa je *string*, a drugi je *boja* tipa *Color*. Kao što im i imena govore, u atribut *ime* se pohranjuje ime igrača, a u atribut *boja* boja njegovog žetona. U klasi se također nalazi konstruktor i *get/set* metode.

Zatim slijede klase koje služe za praćenje statistike. Prva je *HallOfFame* koja sadrži atribute koje će se pratiti u statistici. To su *ime* tipa *string*, *odigrano*, *pobjede*, *nerjesene*, *porazi* koje su tipa *int* i *postotakPobjeda* koje je *float* . Osim konstruktora i *get/set* metoda, nalazi se i metoda koja preopterećuje funkciju *ToString()* kako bi ona vraćala jedan *string* koji čine *ime*, *pobjede*, *nerjesene* i *porazi* odvojeni zarezom. To su ujedno i atributi koji će se zapisivati u tekstualnu datoteku u koju će se spremati rezultati.

Klase *Statistika_1P* i *Statistika_2P* zapravo rade istu stvar, samo što je potonja prilagođena za igru u dva igrača. One sadrže tri metode pomoću kojih se može učitavati informacije iz datoteka gdje se nalaze rezultati, spremati rezultate u datoteku i ispisivati statistiku na zaslon korisnikovog računala.

Posljednja klasa je klasa *Zvuk* koja sadrži metode pomoću kojih se reproducira glazba i zvučni efekti u igri. Audio datoteke koje se koriste su u .wav formatu i spremljene su kao resursi u Visual Studiju.

5.2. Forme

5.2.1. *GlavniIzbornik*

Kod pokretanja aplikacije prvo se pojavi glavni izbornik koji nudi četiri glavne opcije. Prve dvije opcije koje se nude su odabir želi li korisnik igrati igru sam protiv računala ili protiv drugog korisnika. Treća opcija je *Statistika* u kojoj se može pogledati statističke podatke igrača koji su igrali igru. Birajući posljednju opciju mogu se pročitati pravila igre i osnovne informacije o njoj. Postoji još jedna opcija koja služi sa uključivanje i isključivanje glazbe. Izgled glavnog izbornika može se vidjeti na slici 5.2.1.

Slika 5.2.1. Izgled glavnog izbornika

```
private void Button_1igrac_Click(object sender, EventArgs e)
        {
            JedanIgrac p1 = new JedanIgrac();

            p1.Show();
            p1.FormClosed += (a, aa) =>
            {
                Show();
                Button_Zvuk.BackgroundImage = Zvuk.UkljucenZvuk()
                     ? Resources.sound_on : Resources.sound_off;
                Button_Zvuk.BackgroundImageLayout = ImageLayout.Stretch;
            };
            Hide();
        }
```

Slika 5.2.2. *Button_1igrac_Click*

Pritiskom na tipku „1 igrač“ izvršava se programski kod na slici 5.2.2.. Prvo se kreira objekt *p1* forme *JedanIgrac* koji predstavlja sljedeću formu koja će se pokazati.. Ugrađenom metodom *Show()* se prikazuje njen zaslon, dok se sa *Hide()* trenutni zaslon sakriva. Pomoću *p1.FormClosed* linije koda opisujemo što se treba dogoditi sa trenutnom formom kada se forma *JedanIgrac* zatvori, a to je da se ona ponovno pokaže te da tipku za zvuk pokazuje ispravno je li zvuk trenutno uključen ili isključen. Identična logika se koristi prilikom pritiska tipke „2 igrača“ samo što se tada kreira objekt forme *DvaIgraca*, kao i pritiskom tipke *Statistika* koja kreira objekt forme *Statistika*.

```
private void Button_Pravila_Click(object sender, EventArgs e)
        {
            MessageBox.Show(Resources.pravila_igre, "Pravila");
        }
```

Slika 5.2.3. *Button_Pravila_Click*

Pritiskom na opciju „O igri“ izvršavaju se jednostavna linija koda koja u obliku prozora prikazuje poruku koju učitava iz resursa *pravila_igre* gdje se nalaze osnovne informacije o igri.

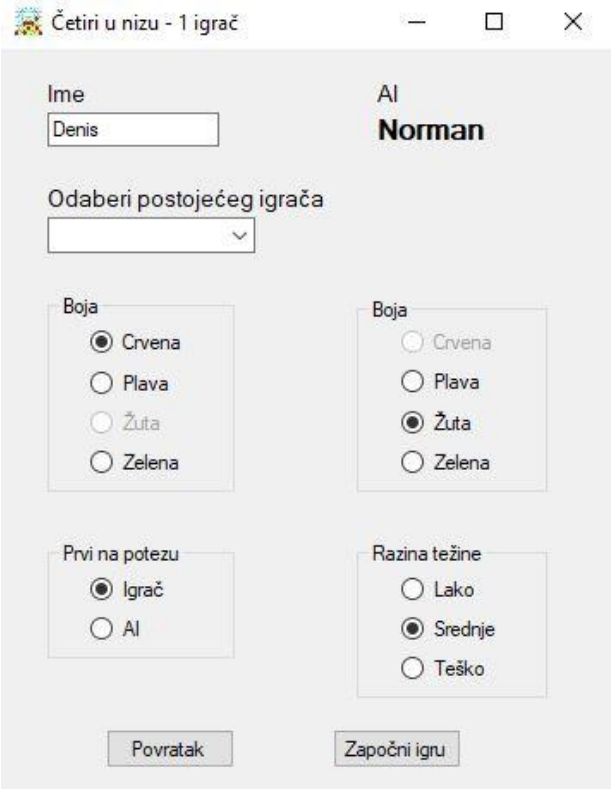
```
private void Button_Zvuk_Click(object sender, EventArgs e)
        {
            Zvuk.Ukljuci_Iskljuci();

            if (Zvuk.UkljucenZvuk())
                Zvuk.Izbornik_Pokreni();
            else
                Zvuk.Izbornik_Zaustavi();

            Button_Zvuk.BackgroundImage = Zvuk.UkljucenZvuk()
                     ? Resources.sound_on : Resources.sound_off;
            Button_Zvuk.BackgroundImageLayout = ImageLayout.Stretch;
        }
```

Slika 5.2.4. *Button_Zvuk_Click*

Pritiskom na tipku sa nacrtanim zvučnikom izvršava se programski kod sa slike 5.2.4. Prvo se poziva metoda *Ukljuci_Iskljuci()* klase *Zvuk*, a zatim metodama *Izbornik_Pokreni()* i *Izbornik_Zaustavi()* uključuje odnosno isključuje glazba glavnog izbornika ovisno o vrijednosti koju vraća metoda *UkljucenZvuk()*. Popratno se mijenja i izgled tipke u ovisnosti je li zvuk uključen ili nije.

5.2.2. *JedanIgrac*

Kod pokretanja nove igre pojavljuje se novi izbornik. Izgled izbornika će ovisiti o tome je li korisnik odlučio igrati igru protiv računala ili protiv drugog korisnika. U prvom slučaju izgled forme može se vidjeti na slici 5.2.5.



Slika 5.2.5. Izgled forme *PocetniEkran*

Korisnik može unijeti svoje ime i odabrati boju žetona s kojom će igrati. Također može odabrati i boju žetona koju će koristiti računalo. Postoji i mogućnost odabira koji će igrač prvi krenuti s igrom, te zadnja i najvažnija opcija je odabir razina težine protiv koje će korisnik igrati. U formi se još nalaze dvije tipke. S jednom tipkom se vraćamo na prethodni izbornik, a sa drugom započinjemo igru.

Forma sadrži i dvije varijable tipa *string*, *imeIgraca* u koju će se pohraniti ime koje si odabere igrač te *bojaPloce* pomoću koje se određuje koje boje će biti igrača ploča. Ona je po početnim postavkama tirkizne boje i može se promijeniti jedino korištenjem skrivenih funkcija aplikacije koje se neće objašnjavati u ovom radu.

```
private void OdabirBoje(object sender, EventArgs e)
        {
            radioButton_Crvena1.Enabled = !radioButton_Crvena2.Checked;
            radioButton_Crvena2.Enabled = !radioButton_Crvena1.Checked;
            radioButton_Plava1.Enabled = !radioButton_Plava2.Checked;
            radioButton_Plava2.Enabled = !radioButton_Plava1.Checked;
            radioButton_Zuta1.Enabled = !radioButton_Zuta2.Checked;
            radioButton_Zuta2.Enabled = !radioButton_Zuta1.Checked;
            radioButton_Zelena1.Enabled = !radioButton_Zelena2.Checked;
            radioButton_Zelena2.Enabled = !radioButton_Zelena1.Checked;
        }
```

Slika 5.2.6. Metoda *OdabirBoje()*

Ova metoda se poziva prilikom promjene boje jednog od igrača. Svaki put kad korisnik promijeni boju izvršava se osam vrlo sličnih linija koda koje se mogu vidjeti u programskom kodu na slici 5.2.6. Pomoću njih se sprječava da igrači odaberu istu boju žetona na način da se onemogući odabir one boje žetona koju je odabrao protivnik.

```
private void OdabirTezine(object sender, EventArgs e)
        {
            if (radioButton_Razina1.Checked)
            {
                label_ImeAI.Text = "Lako";
            }
            if (radioButton_Razina2.Checked)
            {
                label_ImeAI.Text = "Srednje";
            }
            if (radioButton_Razina3.Checked)
            {
                label_ImeAI.Text = "Teško";
            }
        }
```

Slika 5.2.7. Metoda *OdabirTezine()*

Metoda *OdabirTezine* se poziva kada korisnik promijeni razinu težine. Sukladno tome se mijenja i ime koje će koristiti računalo.

```
private void Button_Povratak_Click(object sender, EventArgs e)
        {
            Close();
        }
```

Slika 5.2.8. *Button_Povratak_Click()*

Pritiskom tipke „Povratak“ izvršava naredba *Close()* koja zatvara trenutnu formu.

```
private void Button_Zapocni_Click(object sender, EventArgs e)
        {
            Color bojaIgraca;
            short razinaTezine;
            Color bojaAI;
            bool prviIgra = true;
            imeIgraca = TextBox_ImeIgraca.Text;

            if (radioButton_Crvena1.Checked)
                bojaIgraca = Color.Red;
            else if (radioButton_Plava1.Checked)
                bojaIgraca = Color.Blue;
            else if (radioButton_Zuta1.Checked)
                bojaIgraca = Color.Yellow;
            else
                bojaIgraca = Color.Green;

            if (radioButton_Crvena2.Checked)
                bojaAI = Color.Red;
            else if (radioButton_Plava2.Checked)
                bojaAI = Color.Blue;
            else if (radioButton_Zuta2.Checked)
                bojaAI = Color.Yellow;
            else
                bojaAI = Color.Green;

            if (radioButton_Razina1.Checked)
                razinaTezine = 1;
            else if (radioButton_Razina2.Checked)
                razinaTezine = 2;
            else
                razinaTezine = 3;

            if (radioButton_PrviIgraAI.Checked)
                prviIgra = false;

            if (ProvjeraImena())
            {
                Igrac igrac = new Igrac(imeIgraca, bojaIgraca);
                AI ai = new AI(razinaTezine, bojaAI);
                int[,] polje = new int[7, 6];
                for (int i = 0; i < 7; i++)
                    for (int j = 0; j < 6; j++)
                    {
                        polje[i, j] = 0;
                    }
                Ploca ploca = new Ploca(polje, 0);
                Igra_1P i1p = new Igra_1P(ploca, igrac, ai, prviIgra, bojaPloce);
                i1p.FormClosed += (a, aa) => { Close(); };
                i1p.Show();
                Hide();
                Zvuk.Izbornik_Zaustavi();
            }
            else
            {
                MessageBox.Show("Ime mora sadržavati od 1 do 8 znakova!", "Pozor!");
            }
        }
```

Slika 5.2.9. ***Button_Zapocni_Click()***

Pritiskom tipke „Započni igru“ izvršava se programski kod sa slike 5.2.9. u kojem se na početku deklariraju potrebne varijable u koje će se spremati boje igrača i razina težine koju će koristiti računalo. Također se deklarira i varijabla *PrviIgra* tipa *bool* koja se postavlja u vrijednost „*true*“. U varijablu *imeIgraca* se pohranjuje *string* koji se nalazi u *textbox-u*, a u prethodno deklarirane varijable se pohranjuju one vrijednosti koje je odabrao korisnik pomoću *radiobutton-a*.

U naredbi *if* se poziva metoda *ProvjeraImena()* i ukoliko ona vrati vrijednosti *true* izvršavaju se linije koda kojima se kreiraju objekti klasa *Igrac*, *AI* i *Ploca* kao i objekt forme *Igra_1P*. Novo kreiranim objektima se kao argumenti dodaju odgovarajuće varijable. Zatim se metodom *Show()* prikazuje zaslon nove forme, a pomoću *Hide()* skriva zaslon trenutne. Zatvaranjem nove forme *Igra_1P* zatvoriti će se i forma *JedanIgrac*. Na kraju se isključuje i glazba izbornika. Ako *ProvjeraImena()* vrati *„false“* tada će se pojaviti poruka korisniku.

```
private bool ProvjeraImena()
        {
            if (imeIgraca.Length > 0 && imeIgraca.Length < 9)
                return true;
            return false;

        }
```

Slika 5.2.10. Metoda *ProvjeraImena()*

Ovom metodom se provjerava sadrži li ime igrača dozvoljenih od jedan do osam znakova. Ukoliko sadrži metoda vraća *„true“*, u suprotnom *„false“*.

```
private void Dodaj(string[] zapis)
        {
            foreach (var linija in zapis)
            {
                var strArray = linija.Split(',');
                if (!ComboBox.Items.Contains(strArray[0].ToString()))
                    ComboBox.Items.Add(strArray[0].ToString());
            }
        }
```

Slika 5.2.11. Metoda *Dodaj()*

Metode *StvoriListu()* i *Dodaj()* se koriste za popunjavanje padajućeg izbornika imenima igrača koji se nalaze u „Kući slavnih“, odnosno već su prije igrali igru. Metoda *Dodaj()* kao argument prima polje *stringova*. Svaki *string* se sastoji od četiri podatka odvojenih zarezom te se naredbom *split() stringovi* dijele na četiri dijela, a u prvom dijelu se nalazi ime igrača. Nakon toga provjerava nalazi li se to ime već u padajućem izborniku i ako ne dodaje ga u njega.

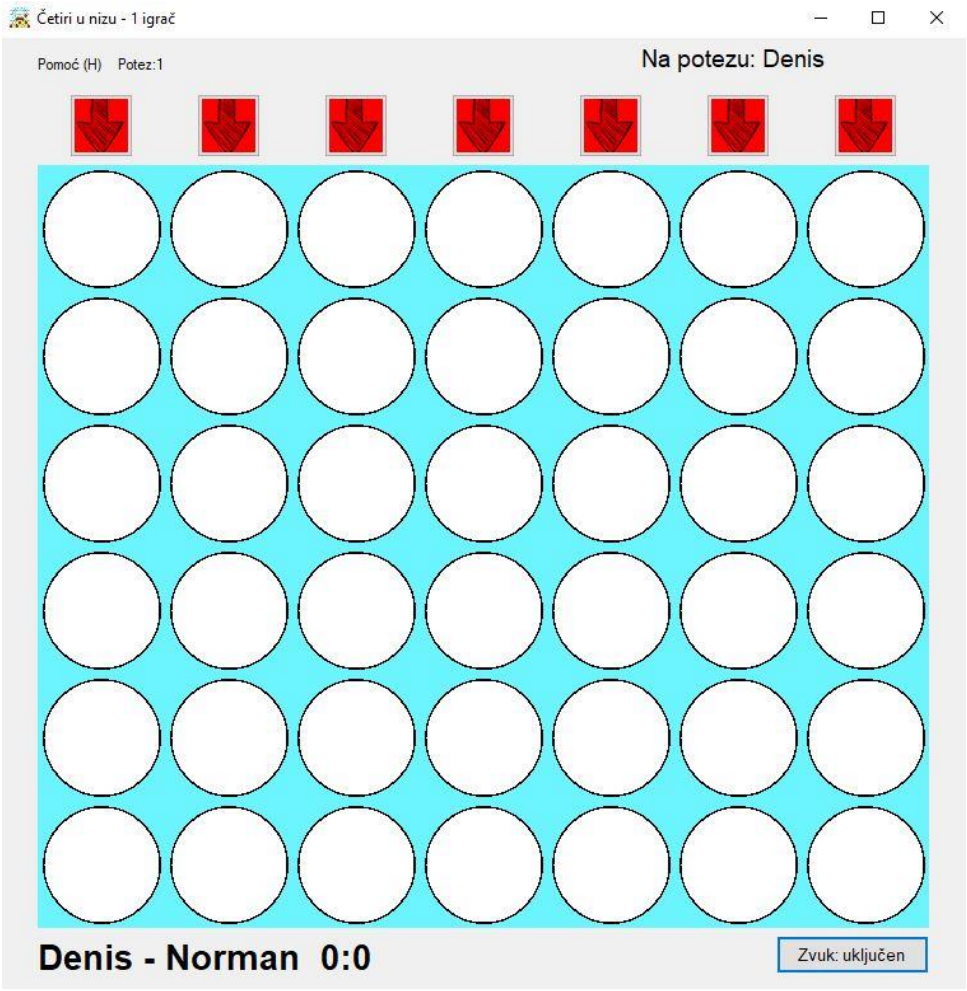
```
private void StvoriListu()
        {
            string[] zapis = File.ReadAllLines("txt/Kuca slavnih 1-Lako.txt");
            Dodaj(zapis);
            zapis = File.ReadAllLines("txt/Kuca slavnih 1-Srednje.txt");
            Dodaj(zapis);
            zapis = File.ReadAllLines("txt/Kuca slavnih 1-Tesko.txt");
            Dodaj(zapis);
        }
```

Slika 5.2.12. Metoda *StvoriListu()*

Metoda *StvoriListu()* učitava u polje *stringova* sve igrače iz „Kuće slavnih“ zapisanih u tekstualnim datotekama, a zatim poziva metodu *Dodaj()* kojoj kao argument predaju to polje. Proces se ponavlja tri puta jer svaka razina težine ima svoju „Kuću slavnih“.

5.2.3. *Igra_1P*

Nakon što korisnik pritisne tipku „Započni igru“ na zaslonu računala pojavljuje se nova forma na kojoj se odvija igra. Ona sadrži 7 tipaka pomoću kojih korisnik može ubacivati žetone u stupce, te jednu tipku pomoću koje može uključiti, odnosno isključiti zvuk. Također sadrži i nekoliko *label-a* koji korisniku pružaju korisne informacije. Izgled forme se može vidjeti na slici 5.2.13.



Slika 5.2.13. Izgled forme *Igra_1P*

```
private Ploca ploca;
private Igrac igrac;
private AI ai;
private bool naPotezu;
private string bojaPloce;

Graphics krug;

private int zadnjiX;
private int zadnjiY;
private int predzadnjiX;
private int predzadnjiY;
private bool omogucenUndo;

private int brojPobjeda1 = 0;
private int brojPobjeda2 = 0;
private bool prviJeIgrao;
private int zeton;
```

Slika 5.2.14. Varijable u formi *Igra_1P*

U programskom kodu na slici 5.2.14. mogu se vidjeti varijable i objekti koji se nalaze u ovoj formi. Objekt *krug* ugrađene klase *Graphics* će služiti za crtanje žetona i njihovo brisanje. Sljedećih pet varijabli se koristi za metodu *Undo* koja će vračati igru potez unazad. Nakon toga slijede brojači broja pobjeda pojedinog igrača, a nakon njih varijabla *prviJeIgrao* tipa *bool* koja ima vrijednost '*true*' ako je korisnik započeo trenutnu partiju, odnosno '*false*' ako je to bilo računalo. Posljednja je varijabla *zeton* i u nju se pohranjuje vrijednost '1' ili '2' ('1' za korisnika, '2' za računalo) kako bi se znalo kome pripada posljednji ubačeni žeton.

```
public Igra_1P(Ploca ploca, Igrac igrac, AI ai, bool naPotezu, string bojaPloce)
        {
            InitializeComponent();

            this.igrac = igrac;
            this.ai = ai;
            this.naPotezu = naPotezu;
            this.ploca = ploca;
            this.bojaPloce = bojaPloce;

            this.krug = pictureBox1.CreateGraphics();

            zadnjiX = -1;
            zadnjiY = -1;
            predzadnjiX = -1;
            predzadnjiY = -1;
            omogucenUndo = false;
        }
```

Slika 5.2.15. Konstruktor *Igra_1P*

Konstruktor ove forme osim standardnog dodjeljivanja vrijednosti, objektu *krug* se dodjeljuje *pictureBox1* u kojem se nalazi igraća ploča. Varijable kojima se spremaju posljednja dva poteza se postavljaju u '-1', a *omogucenUndo* se postavlja u '*false*' čime se onemogućava korištenje opcije *Undo* na početku igre.

```
private void CrtajKrug(int i, int j)
{
   if (naPotezu == true)
        krug.FillEllipse(new SolidBrush(igrac.Boja), i * 100 + 5, 505 - j * 100, 90,
90);
   else
        krug.FillEllipse(new SolidBrush(ai.Boja), i * 100 + 5, 505 - j * 100, 90, 90);
}
```

Slika 5.2.16. Metoda *CrtajKrug()*

Metodama *CrtajKrug()* i *ObrisiKrug()* se crtaju žetoni na igraćoj ploči. Obje metode zapravo crtaju krugove, samo što prva crta krugove u bojama igrača, a druga crta bijele krugove, pa korisnik stječe dojam da su polja prazna. Metode kao argument primaju indekse polja kojeg moraju obojati, a onda pomoću tog indeksa računaju gdje se nalazi ta lokacija u *PictureBox-u*. Također *CrtajKrug()* koristi naredbu *if* da odredi koji je igrač trenutno na potezu.

```
private void PostaviIgru()
        {
            label_Rezultat.Text = igrac.Ime + " - " + ai.Ime + "   "
                                   + brojPobjeda1 + ":" + brojPobjeda2;
            Button_Zvuk.Text = Zvuk.UkljucenZvuk()
                                  ? "Zvuk: uključen" : "Zvuk: isključen";

            ploca.BrojacPoteza = 0;

            for (int i = 0; i < 7; i++)
                for (int j = 0; j < 6; j++)
                {
                    ploca.Polje[i, j] = 0;
                    ObrisiKrug(i, j);
                }

            prviJeIgrao = naPotezu;
            Potez(naPotezu);
        }
```

Slika 5.2.17. Metoda *PostaviIgru()*

Nakon što se prikaže forma *Igra_1P* (korištenjem naredbe *show()* u prethodnoj formi) korištenjem događaja (eng. *event*) se poziva metoda *PostaviIgru()*. Ona postavlja brojač poteza na nulu i briše sve žetone iz igraće ploče. U varijablu *prviJeIgrao* sprema igrača koji započinje igru, kako bi iduću onda započeo onaj drugi igrač. Također zapisuje u *label_Rezultat* trenutni rezultat između igrača i mijenja tekst *Button_Zvuk*-a u ovisnosti je li zvuk bio uključen ili isključen pri pokretanju igre. Na kraju metode poziva se metoda *Potez()* koja kao argument prima igrača koji treba započeti igru.

```
private void Potez(bool naPotezu)
        {
            ploca.BrojacPoteza++;

            label_BrojPoteza.Text = "Potez:" + ploca.BrojacPoteza.ToString();
            label_BrojPoteza.Refresh();
            label_Rezultat.Refresh();
```

```
        label_NaPotezu.Text = (naPotezu) ?
                    ("Na potezu: " + igrac.Ime) : ("Na potezu: " + ai.Ime);
        label_NaPotezu.Refresh();
```

Slika 5.2.18. – Metoda *Potez()* – 1. dio

Ova metoda će se pozivati svaki put kada se promijeni igrač koji je na potezu. U prvom dijelu metode uvećava se brojač poteza, te se osvježavaju tekstovi *label*-a koji korisniku govore koliki je rezultat i tko je na potezu.

```
if (naPotezu)
      {
          Button0.BackColor = igrac.Boja;
          Button1.BackColor = igrac.Boja;
          Button2.BackColor = igrac.Boja;
          Button3.BackColor = igrac.Boja;
          Button4.BackColor = igrac.Boja;
          Button5.BackColor = igrac.Boja;
          Button6.BackColor = igrac.Boja;

          if (ploca.Polje[0, 5] == 0)
              Button0.Enabled = true;
          if (ploca.Polje[1, 5] == 0)
              Button1.Enabled = true;
          if (ploca.Polje[2, 5] == 0)
              Button2.Enabled = true;
          if (ploca.Polje[3, 5] == 0)
              Button3.Enabled = true;
          if (ploca.Polje[4, 5] == 0)
              Button4.Enabled = true;
          if (ploca.Polje[5, 5] == 0)
              Button5.Enabled = true;
          if (ploca.Polje[6, 5] == 0)
              Button6.Enabled = true;
      }
```

Slika 5.2.19. – Metoda *Potez()* – 2. dio

Ukoliko je na potezu korisnik, tada se sve tipke iznad stupaca ploče postavljaju u boju žetona koje koristi igrač. Nakon toga se provjerava ima li još mjesta u stupcima i ako ima, tipke se omogućavaju za korištenje. To se provjerava kako bi se korisniku onemogućilo ubacivanje žetona u već popunjen stupac.

```
else
            {
                Button0.BackColor = ai.Boja;
                Button1.BackColor = ai.Boja;
                Button2.BackColor = ai.Boja;
                Button3.BackColor = ai.Boja;
                Button4.BackColor = ai.Boja;
                Button5.BackColor = ai.Boja;
                Button6.BackColor = ai.Boja;

                Button0.Enabled = false;
                Button1.Enabled = false;
                Button2.Enabled = false;
                Button3.Enabled = false;
                Button4.Enabled = false;
                Button5.Enabled = false;
                Button6.Enabled = false;

                AI_Logika();
            }
```

Slika 5.2.20. – Metoda *Potez()* – 3. dio

Ukoliko je na potezu računalo, tada se sve tipke iznad stupaca ploče postavljaju u boju žetona koje koristi računalo. Nakon toga se sve tipke onemogućavaju za korištenje korisniku. Za kraj se poziva metoda *AI_logika()*.

```
private void AI_Logika()
        {
            Thread.Sleep(500);

            int stupac;
            if (ai.Razina == 1)
                stupac = ai.RazinaLako(ploca);
            else if (ai.Razina == 2)
                stupac = ai.RazinaSrednje(ploca);
            else
                stupac = ai.RazinaTesko(ploca);

            UbaciZeton(stupac);
        }
```

Slika 5.2.21. – Metoda *AI_Logika()*

Metoda *AI_logika()* najprije zaustavlja cijelu aplikaciju na pola sekunde. To se čini kako bi korisnik stekao dojam da računalo „razmišlja“ o svome potezu i time se poboljšava cjelokupno iskustvo za korisnika. Računalo je u biti toliko brzo da bi postavilo svoj žeton djelić sekunde

poslije korisnika što bi moglo dovesti i do toga da korisnik nije niti siguran gdje je računalo upravo ubacilo svoj žeton, što je razlog više za kratko zaustavljanje izvođenja programa. Zatim se pomoću *if* i *else* naredbi određuje koju razinu težine je korisnik odabrao i na osnovi toga poziva se odgovarajuća metoda klase *ai*. Svaka od tih metoda vraća indeks stupca u koji računalo želi ubaciti svoj žeton. Žeton u stupac se ubacuje metodom *UbaciZeton()* koja kao argument prima indeks stupca.

```
private void UbaciZeton(int stupac)
        {
            Zvuk.Zeton();
            int redak;
            for (redak = 0; redak < 6; redak++)
                if (ploca.Polje[stupac, redak] == 0)
                {
                    if (naPotezu)
                    {
                        ploca.Polje[stupac, redak] = 1;
                        zeton = 1;
                    }
                    else
                    {
                        ploca.Polje[stupac, redak] = 2;
                        zeton = 2;
                    }
                    break;
                }

            CrtajKrug(stupac, redak);
```

Slika 5.2.22. Metoda *UbaciZeton()* – 1.dio

Osim metode *AI_Logika()*, metoda *UbaciZeton()* se poziva i nakon što korisnik pritisne jednu od tipaka iznad stupaca. Svaka od tih tipaka ima događaj da reagira na klik i tada poziva ovu metodu i predaje joj svoj indeks, odnosno indeks stupca u koji korisnik želi ubaciti svoj žeton.

Prvi dio ove metode pokreće zvučnu datoteku (u njoj se nalazi zvuk žetona koji upada u ploču), a zatim upisuje u *polje* da je određeni žeton ubačen na odgovarajuće mjesto u dvodimenzionalnom polju. Nakon toga se poziva metoda *CrtajKrug()* koja crta na igraćoj ploči taj žeton.

```
            predzadnjiX = zadnjiX;
            predzadnjiY = zadnjiY;
            zadnjiX = stupac;
            zadnjiY = redak;
            omogucenUndo = true;

            if (ploca.ProvjeraPobjede(zeton, stupac, redak))
                KrajIgre(zeton);
            else if (ploca.ProvjeraNerjesenog())
                KrajIgre(0);
            else
            {
                naPotezu = !naPotezu;
                Potez(naPotezu);
            }
```

Slika 5.2.23. – Metoda *UbaciZeton()* – 2.dio

Drugi dio metode u varijable *predzadnjiX*/*predzadnjiY* zapisuje vrijednosti koje su se nalazile u varijablama *zadnjiX*/*zadnjiY*, a u njih pak zapisuju indekse upravo ubačenog žetona. Varijablu *omogucenUndo* postavlja u *'true'* čime omogućava korisniku da vrati svoj upravo odigrani potez. Nakon toga se provjerava je li igrač koji je ubacio žeton upravo pobijedio pozivanjem metode *ProvjeraPobjede()* koja se nalazi u klasi *ploca*. Ukoliko jest, poziva se metoda *KrajIgre()* i kao argument prima varijablu *zeton* pomoću koje će kasnije odrediti koji igrač je pobijedio. Ukoliko nije ostvarena pobjeda poziva se metoda *ProvjeraNerjesenog()* i ako je igra završila poziva se metoda *KrajIgre()* koja kao argument prima vrijednost '0'. Ukoliko igra nije još završena tada se mijenja vrijednost varijable *naPotezu* (ako je prije bila '*true*' sad će biti '*false*' i obrnuto) i poziva metoda *Potez()*.

```
private void Undo(int i, int j, int i2, int j2)
        {
            ploca.BrojacPoteza -= 2;
            label_BrojPoteza.Text = "Potez:" + ploca.BrojacPoteza.ToString();

            ploca.Polje[i, j] = 0;
            ObrisiKrug(i, j);
            ploca.Polje[i2, j2] = 0;
            ObrisiKrug(i2, j2);

            omogucenUndo = false;
        }
```

Slika 5.2.24. Metoda - *Undo()*

Metoda *Undo()* omogućava korisniku da ponovi svoj prethodni potez. Kada korisnik za vrijeme igre pritisne kombinaciju tipki *Ctrl+Z* tada se poziva preopterećena funkcija *ProcessCmdKey()* (samo ako je zastavica *omogucenUndo* postavljena u *'true'*) koja se nalazi u *Designer*-u forme *Igra_1P* i onda ona poziva metodu *Undo()* koja kao argumente prima varijable u kojima se nalazi lokacija zadnjeg i predzadnjeg ubačenog žetona. Brišu se zadnja dva žetona jer u trenutku kada se pozove ova metoda računalo je već ubacilo svoj žeton u igraću ploču, pa je korisnikov žeton zapravo predzadnje ubačen.

Na početku metode se umanjuje brojač poteza za dva, te se trenutan broj poteza prikazuje na ekranu. Nakon toga se na lokaciju u polju zadnjeg bačenog žetona upisuje '0' i poziva se metoda *ObrisiKrug()* koja crta bijeli krug na istoj lokaciji na ploči koju vidi korisnik. Isto se ponavlja za predzadnje ubačen žeton. Za kraj se zastavica *omogućenUndo* postavlja u '*false'* čime se onemogućuje korisniku da ponovno vrati potez, već će to biti moguće tek nakon što ubaci novi žeton.

5.2.4. Ostale forme

Forme *DvaIgraca* i *Igra_2P* su vrlo slične već objašnjenim formama *JedanIgrac* i *Igra_1P* s razlikom što su prilagođene igri s dva igrača kao na primjer u slučaju forme *DvaIgraca* to znači da korisnik vidi dva prostora za upisivanje imena. Interesantno je i kako kod *Igra_2P* forme više nema potrebe za pamćenjem predzadnjeg odigranog poteza s obzirom da je potrebno obrisati samo zadnji korisnikov odigran potez metodom *Undo()*.

Forma *Statistika* nudi korisniku mogućnost da odabere želi li pogledati statistiku igranja u načinu igranja ’1 igrač' ili '2 igrača'. Ukoliko želi pogledati statistiku za dva igrača, tada se poziva metoda klase *Statistika_2P Ispisi()* čime se ispisuju statistički podaci onih igrača koji su igrali taj način. Ukoliko pritisne tipku '1 igrač' tada se poziva nova forma *Statistika_1P_Razine* koja omogućava korisniku da odabere jednu od tri razine, ovisno čije statističke podatke želi pogledati, što znači da pritiskom na sve tri tipke se poziva metoda *Ispisi()* klase *Statistika_1P* samo ovisi iz koje datoteke će se čitati rezultati.

ZAKLJUČAK

U radu je opisano kako napraviti popularnu logičku igru „Četiri u nizu“ u programskom jeziku C# koristeći Windows forme i klase. Aplikacija je napravljena u integriranom razvojnom okruženju Microsoft Visual Studio 2017 jer ono ima sve mogućnosti za lak i kvalitetan rad s programskim jezikom C#.

Igra ima mogućnost igranja jednog ili dva igrača, koji mogu odabrati vlastito ime i boju. Također mogu odabrati i tko će prvi igrati, a ukoliko je odabrana igra za jednog igrača, tada postoji i mogućnost odabira jedne od tri razine težine. Igrači ubacuju žetone u polje klikom miša na tipke koje se nalaze iznad svakog stupca ili pomoću tipaka na tipkovnici. U aplikaciji je implementirana opcija *Undo* kako bi korisnik imao mogućnosti da vraćanja jednog poteza unazad. Glavni izbornik ima glazbenu podlogu koja se gasi kad igra krene, ali zato za vrijeme igre postoje zvučni efekti. Glazbu kao i zvučne efekte je moguće ugasiti. Igra također prati statistiku igrača koji su igrali igru, pa čak i nudi opciju da se prilikom pokretanja igre odaberu upravo igrači koji se već nalaze u statistici.

Iako igra nema grešaka, te je korisničko sučelje jednostavno za korištenje, postoje mogućnosti da se aplikacija nadogradi. Jedna od bitnijih nadogradnji bi bila mogućnost odabira još više razina težine, pa čak i implementacija umjetne inteligencije korištenjem minmax algoritma i njegovih varijacija. Također bi se mogla dodati opcija da si korisnik sam odabere koju boju igraće ploče želi.

LITERATURA

[1] Računalna igra „Rizik“ u C# programskom jeziku, Denis Trputec, 2016.

[2] J. Liberty, B. MacDonald, Learning C# 3.0, O’Reilly Media, Sebastopol (SAD), 2009.

[3] Rod Stephens, C# 5.0 Programmer’s Reference, John Wiley & Sons, Indianapolis, 2014.

[4] Bruce Johnson, Professional Visual Studio 2013, John Wiley & Sons, Indianapolis, 2014.

[5] Connect Four, http://web.mit.edu/sp.268/www/2010/connectFourSlides.pdf, lipanj 2018.

[6] Why learn C#?,http://www.bestprogramminglanguagefor.me/why-learn-c-sharp, rujan 2018.

SAŽETAK

Glavni zadatak ovog rada bio je napraviti aplikaciju koja bi omogućila korisniku igranje klasične logične igru „Četiri u nizu“. Aplikacija je rađena u potpunosti objektno-orijentiranom programskom jeziku C#, a korišteno je integrirano razvojno okruženje Microsoft Visual Studio 2017. Rad je podijeljen na pet cjelina. Prvo poglavlje je uvod, a u sljedeća dva poglavlja se upoznaje sa korištenom tehnologijom. Četvrto poglavlje je posvećeno igri „Četiri u nizu“, njezinim počecima, pravilima i taktikama. U posljednjem poglavlju se objašnjava korišteni programski kod. Poglavlje je podijeljeno na dva dijela, klase i forme, te su oni podijeljeni na još manjih dijelova.

Ključne riječi: objektno orijentirano programiranje, programski jezik C#, Visual Studio, „Četiri u nizu“, „Složi 4“

ABSTRACT

The main task of this thesis was to create an application that would allow user to play the classic logic game „ Four in a row “. The application was entirely build in C# object-oriented programming language while using integrated development environment Microsoft Visual Studio 2017. The thesis is divided into five chapters. The first chapter is introduction, and the next two chapters get acquainted with the technology used. The fourth chapter describes the “Four in a row” game, its beginnings, rules and tactics. The last chapter explains the program code used. The chapter is divided into two parts, classes and forms, and they are divided into even smaller parts.

Keywords: object-oriented programming, programming language C#, Visual Studio, “Four in a row”, “Connect 4”

ŽIVOTOPIS

Denis Trputec je rođen 30. Siječnja 1993. godine u Zagrebu. Od rođenja živi u Hrašću Turopoljskom, malom naselju u sklopu Grada Zagreba, a osnovnoškolsko obrazovanje stječe u OŠ Odra. Godine 2007. upisuje Tehničku školu Ruđera Boškovića, gdje sve razrede prolazi s vrlo dobrim uspjehom. Maturira 2011. godine i stječe zvanje Tehničar za računalstvo. Iste godine upisuje Fakultet elektrotehnike i računarstva u Zagrebu, ali se nakon dvije godine prebacuje na Elektrotehnički fakultet u Osijeku na preddiplomski studij računarstva. 2016. godine stječe zvanje sveučilišni prvostupnik inženjer računarstva.