Aplikacija za optimalno iskorištavanje ploča iverala

Miličić, Marko

Master's thesis / Diplomski rad

2016

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:200:175661

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-29**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

zir.nsk.hr

Image not found or type unknown

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

Diplomski studij

Aplikacija za optimalan izračun iskoristivosti ploča iverala

Diplomski rad

Marko Miličić

Osijek, 2016.

1. UVOD 1
1.1. Zadatak završnog rada 1
2. MICROSOFT VISUAL C# 2
2.1. Biblioteka klasa .NET kostura 2
2.2. Upravljani kod 3
3. IZRADA PROGRAMA U VISUAL C# 4
3.1. Dijelovi programa i njihova uloga 6
4. IZRADA APLIKACIJE ZA OPTIMALNO ISKORIŠTENJE PLOČA IVERALA 8
4.1. Problemi s velikim brojem stanja pretraživanja 8
4.2. Heurističke metode 8
4.3. Algoritam za optimalno pakiranje pravokutnika u veći pravokutnik 9
4.3.1. Sortiranje manjih ploča 9
4.3.2. Rad algoritma 9
4.3.3. SVG format datoteke 10
4.4.Izgled aplikacije i prikaz djelova aplikacije pomoću koda 11
5. ZAKLJUČAK 20
LITERATURA 21
SAŽETAK 22
ABSTRACT 22
ŽIVOTOPIS 24
PRILOG 25

1. UVOD

MS Visual Studio je programsko okruženje koje sadrži sve što je potrebno za stvaranje grafičkih aplikacija. Visual Studio pruža dva pogleda na grafičku aplikaciju: prikaz dizajna i prikaz koda aplikacije. Kada god se želi može se prebaciti s jednog prikaza na drugi. Visual Studio nudi dva predloška za izgradnju grafičkih aplikacija, a to su: Windows FormsApplication i Windows Presentationfoundation (WPF).

Iverali su ploče izrađene od iverja koje je međusobno spojeno sintetskom smolom. Iveral ploče su oplemenjene melaminskom folijom različitih dekora (jednobojni, drvni, glatki, reljefni, visoki sjaj). Jednostavni su za obradu i rezanje (režu se pravocrtno ili kružno). Nakon rezanja se rubovi prekrivaju (kantiraju) ABS trakom. Iveral ploče se najviše koriste u izradi namještaja, za oblaganje zidova i općenito pri uređenju interijera. Optimalno iskorištenje je izrazito važno zbog rezanja iverala pri kupnji. Prilikom kupovine ploča kupac nije dužan uzeti cijelu ploču iverala ako mu nije potrebna. Optimalnim slaganjem ploča kupac dobije puno veću iskoristivost ploče koju je kupio i naravno pri tome ostane jako malo ostataka.

Rad je organiziran na sljedeći način: u drugom poglavlju diplomskog rada govori se općenito o MS Visual C#, biblioteci klasa, sastavljivosti i upravljanom kodu. U trećem poglavlju prikazana je općenita izrada novog projekta, dijelovi programa i njihova uloga. U četvrtom poglavlju je prikazan i objašnjen postupak rješavanja zadatka koji je zadan u diplomskom radu. Također su prikazani kodovi i slika dizajna aplikacije. Na kraju rada dan je zaključak.

1.1. Zadatak diplomskog rada

Potrebno je napraviti aplikaciju za optimalni izračun iskoristivosti ploče iverala (dim. 280x207 cm). Aplikacija od korisnika prima informacije o veličini i broju potrebnih dijelova, izračunava optimalni raspored i kao izlazni rezultat daje CAD datoteku cijelog iverala s naznačenim linijama rezanja i mjerama. Potrebno je uračunati prostor za pilu od 1,5mm.

2. MICROSOFT VISUAL C#

Microsoft Visual C# je programski jezik koji je dio MS VisualStudia. Visual C# je proizvela tvrtka Microsoft kao odgovor na programske jezike C, C++ i VisualBasic. Izbacili su nedostatke, a istodobno koristili njihove dobre strane. Projektiran je na Microsoftovu platformu .NET. C# je namjenjen programerima koji rade na platformi Windows, a sintaksa je dobro poznata svima koji koriste C, C++, Java itd. Osim poznate sintakse nudi značajke za poboljšavanje produktivnosti: "Sakupljanje otpada oslobađa programere tiranije uobičajenih problema s upravljenjem memorije, kao što su curenje memorije i kružne reference. Povjerljiva sigurnost tipova kompajliranog koda eliminira širok raspon programskih pogrešaka i potencijalnih sigurnosnih propusta"([1], str.2.). Mnoge značajke dolaze iz .NET kostura koji pruža izvršno okruženje i biblioteke za C#, kao naprimjer biblioteka klasa.

2.1. Biblioteka klasa .NET kostura

Klase koje nudi .NET važan su dio iskustva C# programera. "Većina funkcionalnosti bliblioteke spada u jednu od tri kategorije: pomoćne značajke napisane u .NET-u, omotači oko funkcionalnosti Windowsa i kosturi"([1], str.2.).

Pomoćne značajke napisane u .NET-u sadrže pomoćne klase kao što su rječnici, popisi i druge klase kolekcija, te alate za rad s nizovima. Omotači su druga značajka u koju spadaju, naprimjer klase za pristupanje sustavu datoteka , klase za upotrebu mrežnih značajki, te klase za ispisivanje rezultata na konzolu koja je prikazana u primjeru 2.1.

```
class Program
     {
static void Main(string[] args)
         {
Console.WriteLine("Hello, world");
         }
     }
```

Primjer 2.1. *Hello World*

Kada pogledamo ovaj najjednostavniji primjer vidimo da se oslanja na klasu iz biblioteke, System.Console.

Biblioteka klasa nudi cijele kosture za izgradnju aplikacija kao što su:

1. Windows Presentation Foundation (WPF) - kostur za razvoj stolnih aplikacija za Windows

2. Windows Communication Foundation (WCF) - kostur namjenjen razvoju usluga kojima se pristupa preko mreže
3. ASP.NET - kostur za razvoj Web aplikacija

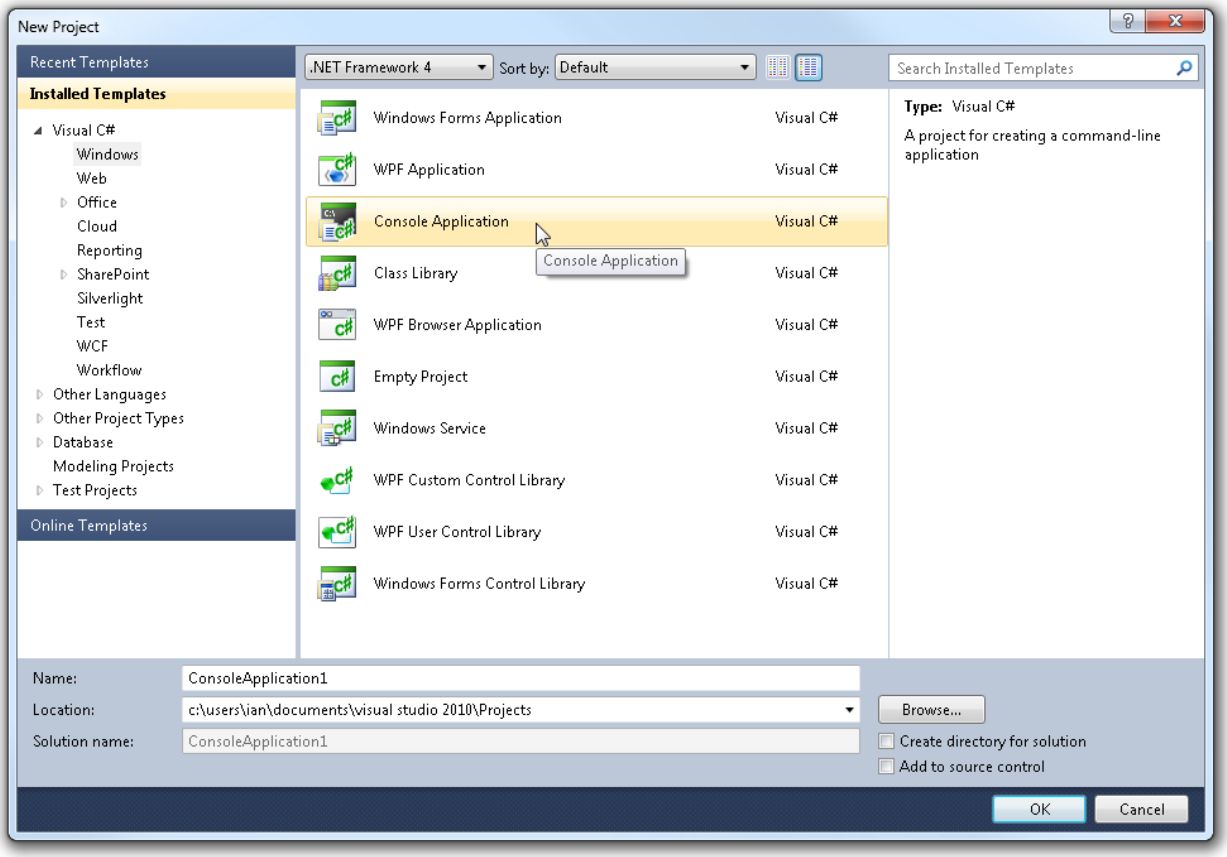
Ove katgorije nisu strogo odvojene jer je dosta klasa razvrstano u dvije. "Kao naprimjer, Windows Forms kostur za korisničko sučelje ima posebne vlastite API-je, veliki dio temeljne funkcionalnosti pružaju Win32 komponente" ([1], str.3.). Ove su kategorije samo pokazatelj što se sve može pronaći u bibliotekama klasa.

2.2. Upravljani kod

Kod koji u cijelosti ovisi o izvršnom okruženju nazivamo upravljani kod. "Upravljani kompajleri ne proizvode sirovi izvršni kod. Umjesto toga, oni daju posrednički oblik koji nazivamo IL - Intermediate Language"([1], str.5.). Jedna od prednosti je što se kompajlirani C# program može bez izmjena izvršavati na 32-bitnim i 64-bitnim sustavima. Glavni cilj upravljanog koda je programere učiniti produktivnijima. Dodatne prednosti koje pruža su: fleksibilni mehanizmi za učitavanje zajedničkih komponenata sa robusnom podrškom za usluge, optimiziranje koda tokom na temelju toga kako se kod koristi u praksi, mogućnost CLR-a da eliminira pogreške koje izazivaju sigurnosne probleme. Uglavnom C# je dosta sličan Javi, odnosno kopirano je nekoliko značajki iz Jave, no kako se razvija dalje različitosti su sve veće. Glavna razlika koja je oduvijek bila prisutna je što je C# oduvijek pružao bolji pristup značajkama temeljne Windows platforme.

3. IZRADA PROGRAMA U VISUAL C#

Prije svega mora se započeti novi projekt, a to se čini odabirom opcije izbornika File - New. Otvara se dijaloški okvir New Project (Sl.3.1.) u kojem se odabire vrsta projekta. S lijeve strane na popisu Installed Templates proširuje se Visual C# i unutar njega odabire se Windows aplikacija koja se želi izraditi.



Slika 3.1. *Dijaloški okvir New Project u VisualStudiu.*

U sljedećem koraku odabire se ConsoleApplication. ConsoleApplication se izabire jer je na njoj najlakše objasniti primjer zadatka.

Nakon toga se zadaje ime programa. Kao primjer će biti HelloWorld program. Nakon postavljenog imena pritiskom na gumb OK Visual Studio izrađuje novi projekt. "C# projekti uvijek sadrže datoteke s izvornim kodom, ali mogu sadržavati i rasterske slike. Ovaj novi projekt sadržavat će datoteku sa C# izvornim kodom po imenu Program.cs, koja je vidljiva na tekstualnom editoru VisualStudia."([1], str.12.)

```
using System;
usingSystem.Collections.Generic;
usingSystem.Linq;
```

```
usingSystem.Text;
namespace HelloWorld
{
class Program
     {
static void Main(string[] args)
          {
          }
     }
}
```

Primjer 3.1. *Izvorni kod konzolne aplikacije*

Ovo je prikaz izvornog koda i program još nije spreman za korištenje. Da bi se mogao koristiti mora se upisati kod pomoću kojeg će se izvršavati željena radnja, tj. govori se što se želi da program učini. Primjer koda je u primjeru 3.2..

```
using System;
usingSystem.Collections.Generic;
usingSystem.Linq;
usingSystem.Text;

namespace HelloWorld
{
class Program
     {
static void Main(string[] args)
         {
Console.WriteLine("Hello, world");
         }
     }
}
```

Primjer 3.2. *Hello World primjer*

Program je spreman za pokretanje. Pritiskom na izbornik Debug i odabirom Start WithoutDebugging program će se pokrenuti. Kada se pokrene u prvom redu će pisati Hello World, a ispod press anykey to continue.

3.1. Dijelovi programa i njihova uloga

Datoteka Program.cs sadrži nekoliko redova koda:

```
using System;
usingSystem.Collections.Generic;
usingSystem.Linq;
usingSystem.Text;([1], str.14.)
```

To su imenski prostori koje Visual Studio dodaje u novi projekt, a koji su korisni u mnogim aplikacijama:

1. System - sadrži usluge opće namjene, uključujući osnovne tipove podataka kao što su string i razni numerički tipovi
2. System.Collections.Generic - sadrži tipove za rad s kolekcijama, npr. popisom brojeva
3. System.Linq - tipovi koji pružaju prikladne načine za obradu kolekcija informacija
4. System.Text - sadrži tipove za rad s tekstom

Oni govore koji će vanjski kod datoteka koristiti. Da bi program koji pišemo mogao obavljati zadan posao mora se oslanjati na druge kodove. Svi C# programi se oslanjaju na biblioteku .NET kostura. Direktive using prikazuju namjeru da želimo korstiti klase iz nekih drugih biblioteka, npr. naše ili nekog drugog proizvođača. System direktiva pokazuje da se jedan dio koristi iz .NET kostura, a tekst nakon direktive using zadaje imenski prostor (eng. namespace).

Imenski prostor sadrži tipove koji najčešće predstavljaju vrstu informacije ili vrstu objekta. Svi tipovi u biblioteci klasa .NET pripadaju nekom imenskom prostoru. Njihova svrha je brže upisivanje koda jer nije potrebno ponovno upisivanje imenskog prostora kada se koristi klasa. Imenski prostori se koriste zbog korištenja imena tipova na više mjesta. Pomoću njih omogućeno je korištenje u vlastitom kodu kakva god imena želimo, bez obzira ako se ista imena koriste u dijelovima bi biblioteke klasa .NET kostura. S obzirom da .NET sadrži više od 10000 tipova imenski prostori su nam jako korisni.

Sljedeći red nakon direktiva također se bavi imenskim prostorom. "Namespace govori kompajleru koji imenski prostor planiramo pružiti - tipovi koje pišemo u programima pripadaju imenskim prostorima baš kao i tipovi u biblioteci klasa."([1], str.16.) Vitičaste zagrade označavaju sadržaj koji se nalazi u nekom imenskom prostoru.

```
namespace HelloWorld
{
...
```

```
}
```

U našem primjeru definiramo tip u C#-u pomuću klase Program.

```
class Program
{
...
}
```

Klasa definira metodu Main. "Metoda je imenovani blok koji može vratiti nekakav rezultat."([1], str.17.)

"Ključna riječ static govori da nije neophodno stvarati objekt da bi se metoda mogla koristiti."([1], str.17.)

Ključna riječ void govori da metoda ne vraća podatke nego da obavlja nekakav posao.

```
static void Main(string[] args)
{
...
}
```

4. IZRADA APLIKACIJE ZA OPTIMALNO ISKORIŠTENJE PLOČA IVERALA

Tema diplomskog rada je izrada aplikacije za optimalni raspored ploča iverala koje se trebaju izrezati iz veće ploče. Pri izradi namještaja, iz većih ploča iverala potrebno je na mjeru izrezati manje ploče, koje će se dalje koristiti kao građevni dijelovi samog namještaja koji se proizvodi. Raspored ploča koje je potrebno izrezati vrlo je teško odrediti tako da količina otpada bude minimalna. Odavde slijedi motivacija za izradom aplikacije koja će olakšati ovaj proces te koja će, unutar razumnog roka, biti u stanju predložiti plan izrezivanja velike ploče.

4.1. Problemi s velikim brojem stanja pretraživanja

Problemi s velikim brojem stanja pretraživanja su svi problemi kod kojih postoji velika količina mogućih rješenja. Ukoliko bi se pretraga najboljeg rješenja vršila grubom silom, odnosno, pretraživanjem svih mogućih rješenja problema, pretraživanje ne bi završilo u razumnom vremenu. Stoga je u takvim situacijama potrebno priskočiti raznim, nekonvencionalnim metodama kako bi se pronašlo što bolje rješenje problema. Pri rješavanju problema takvog tipa, često se niti ne traži najbolje moguće rješenje. Razlog tome je što se za dobiveno rješenje ne može ispitati je li ono zaista najbolje moguće. Nadalje, ovisno o vrsti problema, moguće je i da se ne isplati trošiti dodatno procesorsko vrijeme kako bi se došlo do neznatno boljeg rješenja nego što je već pronađeno. Problem izrezivanja ploča iverala je upravo takav problem. Naime, potrebno je smanjiti količinu otpada koji se dobije izrezivanjem, no nije nužno da se otpad svede na apsolutni minimum. Broj načina na koje se može iveral izrezati je izuzetno velik, budući da se ne radi nužno s cjelobrojnim vrijednostima. Na primjer, rez je moguće napraviti na 30 mm od početka ploče, no jednako tako je moguće napraviti rez na 30.1 mm od početka ploče. Stoga, u praksi je nemoguće pretražiti sve moguće kombinacije izrezivanja kako bi se pronašlo najbolje rješenje.

4.2. Heurističke metode

Heurističke metode predstavljaju sve one metode rješavanja problema koje su vođene nekim prethodnim znanjem o zadanom problemu. Na primjer, ukoliko je potrebno pronaći optimalan put od točke A do točke B, a zna se da je točka C između točke A i B, te da je udaljenost od A do C i od B do C relativno malena, heuristički se može zaključiti da se točka C *vjerojatno* nalazi na najkraćem putu između A i B.

U problemu izrezivanja iverala, zna se da je otpad manji ako su pravokutnici koji se režu priljubljeni jedan uz drugog. Nadalje, ako se pravokutnici većih dimenzija nalaze uz

pravokutnike manjih dimenzija, veća je vjerojatnost da će na toj razlici između pravokutnika nastati otpad. Ove i neke druge pretpostavke su iskorištene pri izradi algoritma koji će generirati nacrt izrezivanja iverala takav da će količina otpada biti relativno malena.

4.3. Algoritam za optimalno pakiranje pravokutnika u veći pravokutnik

Algoritam se pri izvođenju oslanja na određene pretpostavke koje se osiguravaju prije samog izvođenja algoritma, a to su:

- Algoritam se izvodi pri svakom dodavanju manje ploče na nacrt veće ploče
- Ploče se dodaju određenim, unaprijed zadanim redoslijedom
- Ploče koje se dodaju su uvijek transformirane tako da budu "vodoravne" (da im je širina veća ili jednaka visini)

Zbog prve dvije točke, sve manje ploče koje su unesene u program moraju prvo biti pohranjene u sortiranu listu te se nakon toga jedna po jedna ploča iz liste dodaje na nacrt velike ploče.

4.3.1. Sortiranje manjih ploča

Manje ploče se sortiraju silazno, pomoću *Comparatora*, koji prima dvije ploče te kao rezultat daje je li prva ploča *veća, manja* ili *jednaka drugoj*. Usporedba se radi prema sljedećem kriteriju:

- Ukoliko je dulja stranica prvog jednaka duljoj stranici drugog pravokutnika, veći je onaj pravokutnik koji ima veću površinu (odnosno, čija je kraća stranica veća);
- Inače, veći je pravokutnik onaj čija je dulja stranica dulja od dulje stranice drugog pravokutnika.

Nakon silaznog sortiranja po ovakvom kriteriju, jedna po jedna ploča se dodaje na nacrt, te se daljnjim tokom algoritma smješta na nacrt.

4.3.2. Rad algoritma

Algoritam u svojoj strukturi podataka sadrži:

- Točke koji su kandidati za dodavanje novih manjih ploča;
- Listu svih dosad dodanih manjih ploča;
- Dimenzije velike ploče (nacrta) na koju se dodaju manje ploče.

Inicijalno, točka-kandidat za dodavanje nove manje ploče je točka u ishodištu nacrta (0, 0). Lista koja sadrži sve točke-kandidate također je sortirana, uzlazno, prema kriteriju udaljenosti od ishodišta. Dakle, točka (2, 2) je "manja" od točke (3, 1) budući da je bliže ishodištu.

Pri dodavanju nove manje ploče na veliku ploču (nacrt) odvija se algoritam prikazan u nastavku (Algoritam 4.1.).

```
za svaku točku iz liste točaka:
      postavi poziciju ploče koja se dodaje na tu točku

      postavi ploču koja se dodaje vodoravno
      ako ploča izlazi iz nacrta:
            postavi ploču koja se dodaje okomito
            ako ploča izlazi iz nacrta:
                  postavi ploču koja se dodaje vodoravno
                  ploča nije uspješno dodana
            prijeđi na sljedeću točku

za svaku dodanu ploču iz liste dodanih ploča:
      postavi ploču koja se dodaje vodoravno
      ako se ploča ne preklapa s dodanom pločom:
            ploča je uspješno dodana
      inače:
            postavi ploču koja se dodaje okomito
            ako se ploča ne preklapa s dodanom pločom:
                  ploča je uspješno dodana
            inače:
                  postavi ploču koja se dodaje vodoravno
                  ploča nije uspješno dodana
                  prijeđi na sljedeću točku
```

Algoritam 4.1. Algoritam za pakiranje manjih pravokutnika u pravokutnu ploču

Rad ovog algoritma, općenito gledano, rezultira sljedećim izgledom dobivenog nacrta:

- Veće ploče su bliže ishodištu
- Ploče se ne dodaju niti u vodoravnom niti u okomitom nizu - zato su ploče grupirane prema ishodištu
- Veće ploče su blizu jedna drugoj
- Manje ploče su posložene u "kružni vijenac" čiji je centar u ishodištu

Zbog navedenog, donji-desni dio nacrta u pravilu ostaje prazan, a manje ploče dobro popunjavaju eventualne "rupe" nastale dodavanjem većih ploča.

4.3.3. SVG format datoteke

"Kao što mu i ime sugerira, SVG (eng. *ScalableVectorGraphics*) format je zapisa vektorskih grafika, primarno namijenjen za primjenu na webu".([11]) SVG format se u aplikaciji koristi za izvoz nacrta rezanja ploča. SVG datoteka je zapisana kao XML [9] .

Prednosti SVG prikaza :

- "mala veličina datoteke zahvaljujući vektorskom sadržaju i tekstualnom (XML) formatu zapisa
- pretraživost datoteke i mogućnost indeksiranja sadržaja putem tražilica (radi XML zapisa)
- prihvaćen u gotovo svim modernim preglednicima" [11]

Nedostaci SVG prikaza :

- "iscrtavanje SVG datoteke kompleksnog vektorskog sadržaja može potrajati neželjeno dugo, pritom ometajući rad web-preglednika
- Internet Explorer ga ne podržava nativno (potrebni dodaci - *pluginovi*)" [11]

4.4. Izgled aplikacije i prikaz djelova aplikacije pomoću koda

Pomoću Visual C# potrebno je bilo napraviti windows aplikaciju za optimalno iskorištenje ploča iverala. Potrebno je napraviti vizualan prikaz cijele aplikacije, tj. postaviti odgovarajuće panele, tekstualne okvire, labele, button-e. Nakon toga je potrebno upisati kod za svaki pojedinačni dio. Napravljena je aplikacija za optimalno iskorištenje ploča iverala. Korisnik postavlja duljinu i širinu ploča upisivanjem vrijednosti u prozore koji su za to određeni te pomoću gumba osvježi ucrtava ploče na zadanu ploču. Pokraj svakog prozora za postavljanje vrijednosti je tekst koji govori što se postavlja, duljina ili širina.

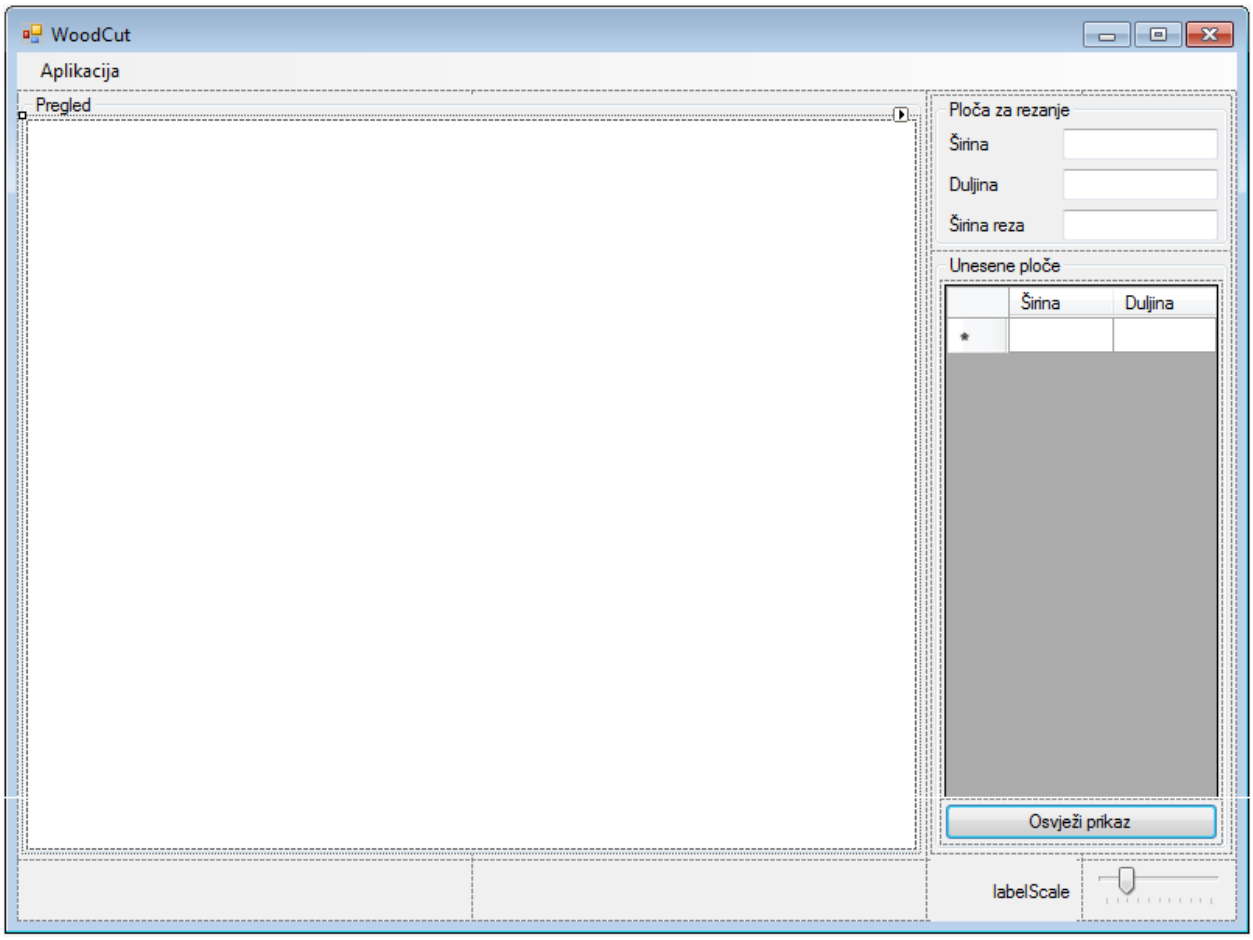
Prilikom izrade zadatka bilo je potrebno za svaku od kontrola napisati kod pomoću kojega će se zadana radnja izvršavati. Pisanje koda omogućeno je u prozoru koda, a može se ući u njega dvostrukim klikom miša u prozoru dizajna na kontrolu u koju želimo kod upisati. Tako da prije pisanja koda postavimo određenu kontrolu na obrazac forme i klikne se na njega. Postavljenje koda u zadatku prikazano je u nastavku.

Redoslijed izrade aplikacije:

- pomoću WYSIWYG (eng. What You See Is What You Get) editora unutar VisualStudia slaže se izgled aplikacije, bez funkcionalnosti
- Izrada funkcionalnosti unošenja podataka o ploči i manjim pločama kroz sučelje
    - Izrada osnovnih funkcionalnosti svih razreda unutar paketa CuttingStock
    - Izrada funkcionalnosti prikaza ploča
- Izrada algoritma (koji se nalazi isto u navedenom paketu)
    - Povezivanje rada algoritma sa sučeljem (Gumb osvježi, osvježavanje samog prikaza itd.)

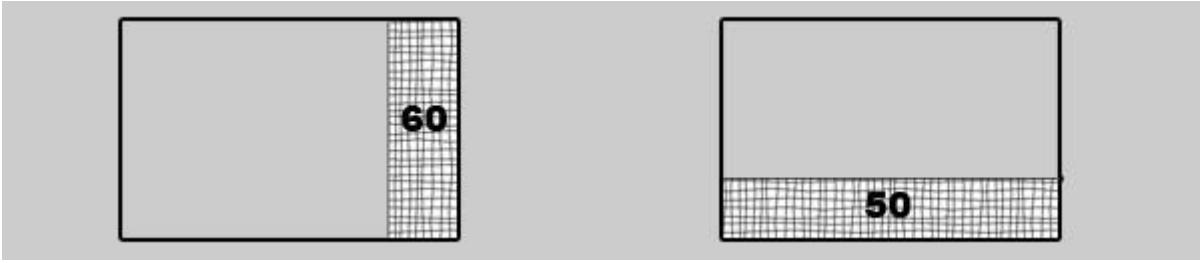
- Izvoz nacrta u SVG format
- Ispitivanje aplikacije i popravljanje grešaka

Na slici 4.1. prikazan je konačan prikaz izgleda prozora dizajna preko kojeg je izrađena windows aplikacija za optimalno iskorištenje ploča iverala.



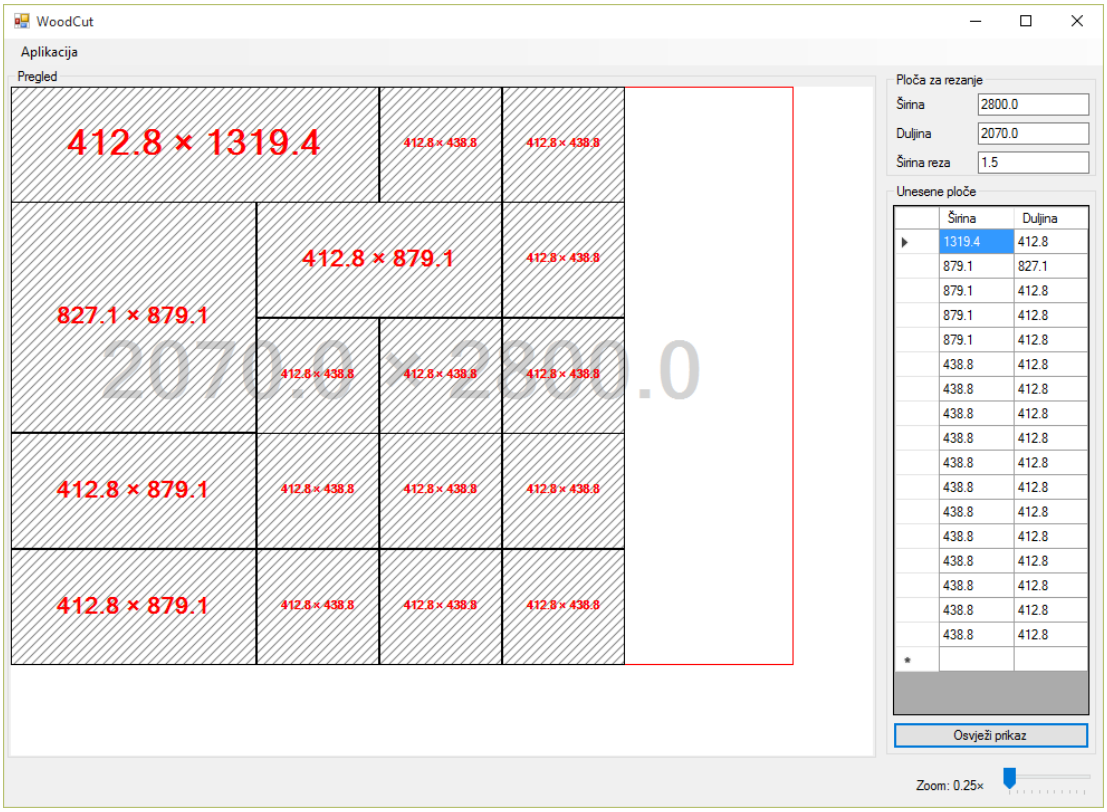
Slika 4.1. *Izgled prozora dizajna aplikacije*

Prilikom kupovine ploča iverala kupac nije dužan uzeti cijelu ploču iverala (dim. 280 x 207 cm. Ploče se prodaju ili po širini ili po dužini ili u restlovima (ostacima). Ukoliko kupac pronađe odgovarajući ostatak ploče može uzeti samo taj dio. Pri rezanju po dužini dio koji kupac ne mora uzeti iznosi 50 cm, dok pri rezanju po širini taj dio iznosi 60 cm. Slika 4.2. prikazuje pravila rezanja iverala po dužini i širini s dozvoljenim ostacima koje kupac ne mora kupiti [12].



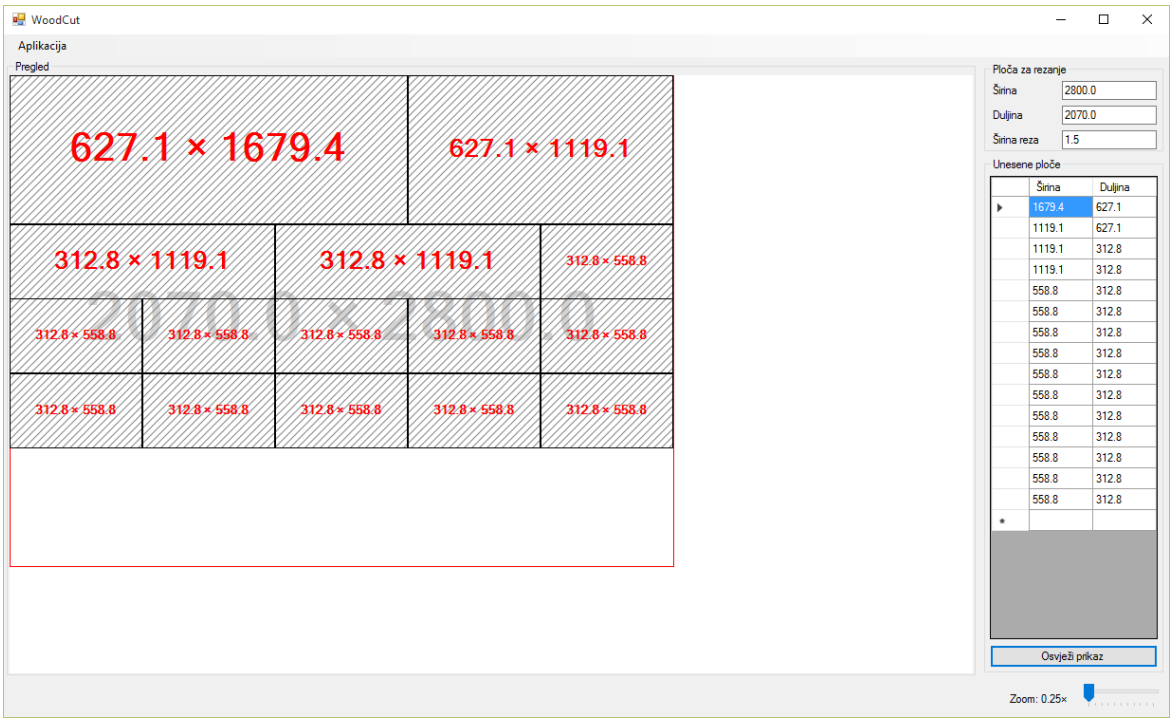
Slika 4.2. *Pravila rezanja iverala pri prodaji*

Slike 4.3. i 4.4. prikazuju primjere rada aplikacije kada su iveral ploče rezane po pravilima prodaje.



Slika 4.3. *Primjer 1*

Na slici 4.3. vidimo optimalno iskorištenje jednog dijela ploče po širini.



Slika 4.4. *Primjer 2*

Na slici 4.4. vidimo optimalno iskorištenje jednog dijela ploče po dužini.

Kod sadrži više dijelova koji su unutar njegove strukturu, a neki dijelovi koda su napisani i objašnjeni u nastavku.

Kod za gumb osvježi:

```
private void buttonOsvjezi_Click(object sender, EventArgs e)
        {
try
            {
layout.Height = Convert.ToDecimal(textBoxDuljina.Text);
layout.Width = Convert.ToDecimal(textBoxSirina.Text);
layout.CutWidth =
Convert.ToDecimal(textBoxSirinaReza.Text);
layout.clearPanels();
SortedSet<CuttingStock.Rectangle>sortedPanels = new
SortedSet<CuttingStock.Rectangle>(new
RectangleLargerDimComparer());
foreach (CuttingStock.Rectangleploca in
bindingListPloce)
```

```
                {
sortedPanels.Add(ploca);
                }
foreach (CuttingStock.Rectangleploca in sortedPanels)
                {
layout.addPanel(ploca);
                }

dataGridViewPloce.Refresh();
panelCanvas.Invalidate();
            }
catch (Exception)
            {
MessageBox.Show("Molimounesiteispravnedecimalne
vrijednosti.");
            }
     }
```

Podatke unesene u formi upisujemo u objekt layout. Uz to, layout čistimo od starih manjih pravokutnika i stavljamo nove pravokutnike koji su uneseni u Data Grid View. Prvo ih sortiramo (bitno za rad algoritma), a onda ih ubacujemo. Osvježavamo Data Grid View, a zatim iscrtavamo. Na kraju, ako je nešto neispravno uneseno ispisuje poruku o grešci.

Prikaz koda panela:

```
private void panelCanvas_Paint(object sender, PaintEventArgs e)
{
e.Graphics.TranslateTransform(panelCanvas.
AutoScrollPosition.X, panelCanvas.AutoScroll
Position.Y);
if (layout != null)
{
panelCanvas.AutoScrollMinSize = new
Size((int)(layout.Width * multiplierScale + 10),
(int)(layout.Height * multiplierScale + 10));

                Graphics g = e.Graphics;
System.Drawing.Rectanglerect;
```

```
                Pen pen = new Pen(Color.Red, 1);
HatchBrushhb = new
HatchBrush(HatchStyle.BackwardDiagonal,
Color.Gray, Color.Transparent);
                Font font;
floatfontSize;
StringFormatstringFormat = new StringFormat();
stringFormat.Alignment = StringAlignment.Center;
stringFormat.LineAlignment =
StringAlignment.Center;

rect = new System.Drawing.Rectangle(
                        (int)(layout.X0 * multiplierScale),
                        (int)(layout.Y0 * multiplierScale),
                        (int)(layout.Width * multiplierScale),
                        (int)(layout.Height * multiplierScale)
                        );

if (layout.Height>layout.Width)
                {
stringFormat.FormatFlags =
StringFormatFlags.DirectionVertical;
font = new Font(FontFamily.GenericSansSerif,
Math.Min(rect.Width / 3, rect.Height / 10),
FontStyle.Bold, GraphicsUnit.Pixel);
                }
else
                {
stringFormat.FormatFlags =
~StringFormatFlags.DirectionVertical;
font = new Font(FontFamily.GenericSansSerif,
Math.Min(rect.Height / 3, rect.Width / 10),
FontStyle.Bold, GraphicsUnit.Pixel);
                }

g.FillRectangle(Brushes.White, 0, 0,
panelCanvas.Width, panelCanvas.Height);
```

```
g.DrawRectangle(pen, rect);
g.DrawString(layout.DimensionsToString(), font,
Brushes.LightGray, rect, stringFormat);

pen.Color = Color.Black;

foreach (CuttingStock.Rectangle panel in layout)
                {
rect = new System.Drawing.Rectangle(
                        (int)(panel.X0 * multiplierScale),
                        (int)(panel.Y0 * multiplierScale),
                        (int)(panel.Width * multiplierScale),
                        (int)(panel.Height * multiplierScale)
                        );
if (panel.Height>panel.Width)
                    {
stringFormat.FormatFlags =
StringFormatFlags.DirectionVertical;
fontSize = Math.Min(rect.Width / 3,
rect.Height / 10);
                    }
else
                    {
stringFormat.FormatFlags =
~StringFormatFlags.DirectionVertical;
fontSize = Math.Min(rect.Height / 3,
rect.Width / 10);
                    }
g.FillRectangle(hb, rect);
g.DrawRectangle(pen, rect);

if (fontSize> 0)
                    {
font = new
Font(FontFamily.GenericSansSerif,
fontSize, FontStyle.Bold,
GraphicsUnit.Pixel);
```

```
g.DrawString(panel.DimensionsToString(),
font, Brushes.Red, rect, stringFormat);
                    }

                }
pen.Dispose();
            }
        }
```

Dio koda koji govori da ako je layout napravljen, definira se veličina područja pri kojoj se aktivira scroll, te se uzima Graphics objekt od baš ovog panela pomoću kojeg se onda nadalje izvodi iscrtavanje Layouta. Odvija se definiranje varijabli za pravokutnik, font, debljina olovke, šrafura i tekst. Također, stvaranje pravokutnika *za crtanje* koji će predstavljati najveći pravokutnik (koji se reže). Ako je pravokutnik okomit (uz još neke uvjete) ispisuje tekst okomito, uz to, odabire prigodnu veličinu fonta, ispunjava cijelo "platno" bijelom bojom, izvodi crtanje pravokutnika i ispis dimenzije, postavljanje boje olovke - crna. S petljom se ide kroz sve ove manje pravokutnike unutar layouta (jako slično crtanju velikog layouta). U nastavku je definiranje pravokutnika za crtanje, ispunjavanje šrafurom i crtanje pravokutnika. Ako veličina fonta koja je izračunata nije premala, ispisuje dimenzije tog pravokutnika. Na kraju uništavanje objekta za crtanje (housekeeping).

Prikaz koda za gumb otvori:

```
private void otvoriToolStripMenuItem_Click(object sender, EventArgs
e)
        {
openFileDialog1.ShowDialog();
        }

private void openFileDialog1_FileOk(object sender, CancelEventArgs e)
        {
layout =LayoutFactory.LoadFromTXT
(((OpenFileDialog)sender).FileName);
textBoxDuljina.Text = layout.Height.ToString("0.0");
textBoxSirina.Text = layout.Width.ToString("0.0");
textBoxSirinaReza.Text = layout.CutWidth.ToString("0.0");
```

```
bindingListPloce.Clear();
foreach (CuttingStock.Rectangleploca in layout)
            {
bindingListPloce.Add(ploca);
            }
panelCanvas.Invalidate()
```

Stvaranje novog layouta pomoću Factorya, popunjavanje forme novim podacima novootvorenog layouta, brisanje liste malih ploča i popunjavanje novim pločama. Naposljetku osvježavanje prikaza.

5. ZAKLJUČAK

Microsoft Visual Studio sadrži sve što je potrebno za stvaranje grafičkih aplikacije. Microsoft Visual C# je dio MS VisualStudia koji je zbog svojstva prikaza na grafičku aplikaciju u dva pogleda pogodan za izradu grafičkih aplikacija,a zbog toga svojstva je i lakši za korištenje. Mogu se koristiti prozori koda i prozori Text editora za izmjenu koda grafičke aplikacije, a prozor Design Wiew koristi se za izmjenu grafičkog sučelja.To je omogućeno zato što C# daje prednost opće namjenskim značajkama umjesto specijaliziranim. VisualC# radi s objektnim modelima, a da bi se omogućio takav pristup doneseno je niz značajki pomoću kojih se kombinirajući pristupa bazama podataka.

Windows FormsApplication je predložak pomoću kojega je napravljen diplomski rad. Korišten je prilikom izrade aplikacije diplomskog rada zbog svoje jednostavnosti i jednostavnog prikaza koda i dizajna. Pomoću Windows Forms-a u Visual Studiu može se napraviti puno toga u dizajneru, a unutar predloška su i postojeće kontrole koje nude značajke koje su potrebne za izradu sučelja.

Prilikom izrade aplikacije koristeći predložak Windows Forms napravljen je vizualan prikaz kako bi aplikacija trebala izgledati. Također kada je vizualan prikaz postavljen, prebacujući se između prikaza dizajna i prikaza koda postupno je pisan kod za kontrole koje se nalaze na formi predloška. Kada je kod završen s pisanjem potrebno je bilo zadatak provjeriti i pokrenuti. S obzirom da se ne pojavljuju greške prilikom provjere, aplikacija je spremna za korištenje. Prilikom pokretanja pojavljuje se prozor na kojemu se mogu upisati duljina, širina i širina reza ploče, kao i duljina i širina ploča koje zadajemo za rezanje. Pritiskom gumba "osvježi" prikazuje se optimalan prikaz rezanja zadane ploče.

Microsoft Visual C# se pokazao kao program u kojemu se lako i jednostavno mogu izraditi windows aplikacije. Prilikom izrade aplikacije potrebno je dobro znati programski jezik C++, a što se tiče izrade dizajna aplikacije jako je jednostavan i može se naučiti u nekoliko koraka. Dosta je sličan "Javi", osim što su neke značajke koje u "Javi" nisu bile najbolje uklonjene i dodana su neka poboljšanja.

LITERATURA

[1] I. Griffiths, M. Adams, J. Liberty, Programiranje C# 4.0, Zagreb, 2011

[2] FER-ZPM-GRZ, K. Fertalj: Visual C, 2003.

[3] Ž Gavrić, StefanTesanović, Development of C# application

[4] Visual C# Resurces, dostupno na URL adresi: "http://msdn.microsoft.com/en-us/vstudio/hh341490.aspx". - datum zadnje posjete 01.04.2016.

[5] Nikola Curčić, Programiranje C# skripta, dostupno na URL adresi: "http://sr.scribd.com/doc/27049359/C-Programiranje-skripta". - datum zadnje posjete 12.04.2016.

[6]https://www.fer.unizg.hr/_download/repository/Predavanja_1-2_[HR]_-_Uvod_u_optimizacije.pdf - datum zadnje posjete 10.04.2016.

[7] http://www.mathos.unios.hr/oui/p3.pdf - datum zadnje posjete 22.06.2016.

[8] https://bib.irb.hr/datoteka/525732.ZR.pdf - datum zadnje posjete 22.06.2016.

[9] https://www.w3.org/Graphics/SVG - datum zadnje posjete 16.06.2016.

[10] https://www.w3.org/TR/SVG/- datum zadnje posjete 16.06.2016.

[11] http://www.baotic.net/graficki-formati-na-webu/vektorski/svg/- datum zadnje posjete 16.06.2016.

[12] http://hobi-centar.hr/pravila-prodaje/ - datum zadnje posjete 25.06.2016

[13] http://www.eko-linija.hr/hrv/proizvodi/plocasti-materijali/2/iveral/105/ - datum zadnje posjete 25.06.2016.

SAŽETAK

Rad prikazuje postupak slaganja manjih ploča iverala unutar velike ploče kako bi se ploča mogla rezati po nacrtu. Aplikacija se koristi prilikom prodaje ploča iverala. Glavni problem izrade aplikacije bio je taj što postoji velika količina mogućih rješenja pri slaganju ploča. Prikazani su i objašnjeni problemi s kojima se susrećemo prilikom izrade aplikacije (npr. velik broj pretraživanja stanja). Objašnjen je postupak rješavanja problema s velikim brojem pretraživanja stanja, te su objašnjene heurističke metode koje su pomogle pri izradi algoritma. U radu je također dan i objašnjen algoritam koji se koristi u aplikaciji. Kroz razne pokušaje i istraživanja na kraju je postignut zadani cilj.

Ključne riječi : Iveral ploča, optimizacija, pretraživanje, rezanje, sortiranje

ABSTRACT

Calculation of the optimal utilization of plywood boards and its application

This paper shows the process of stacking small plywood boards into the large board so we can cut the board according to a scheme. We use the application when we selling the boards. The main problem in the making of this computer application was a large number of possible solutions. The problems that were run into during making this application are noted and explained in the work (such as the problem of multiple solution). The procedure of problem solving of the multiple solution issue is explained, and there were also explained the heuristic methods used in the making of the necessary algorithm. In this paper we also showed and explained the algorithm that we used in application. Through various attempts and research we have finally reached the default goal.

Keywords: cutting, optimization, plywood board, searching, sorting

ŽIVOTOPIS

Marko Miličić rođen 22. srpnja 1991. godine u Našicama od oca Samira i majke Marije Miličić. Otac mu je zaposlenik u Hrvatskoj elektroprivredi, a majka nezaposlena, kućanica. Odrastao je u peteročlanoj obitelji.

Osnovnu školu završio u Magadenovcu, a srednju elektrotehničku u Elektrotehničkoj i prometnoj školi u Osijeku, smjer elektrotehničar. Na stručnoj praksi i tijekom školovanja pokazao je dobre rezultate i, prema mišljenju njegovih voditelja, uspostavio dobar odnos u kolektivu. Posebice rad na računalu te u ostalim računalnim programima koji zahtijevaju samostalno razmišljanje i snalažljivost. Zavolio je rad na računalu i rad sa računalnim programima . Ima jako dobro znanje u programskim paketima MS Office-a.

Zato je nakon državne mature upisao Elektrotehnički fakultet u Osijeku. Trenutno je apsolvent nakon 5. godine istog studija, odnosno 2. godine diplomskog studija, smjer Procesno računarstvo.

Ima položen vozački ispit B i A2 kategorije. Od stranih jezika solidno se služi engleskim jezikom i ima položen B2 stupanj englekog jezika.

Vlastoručni potpis :

_____

PRILOG

Na optičkom disku koji se nalazi u prilogu ovog rada nalazi se diplomski rad u digitalnom obliku i opisana aplikacija.

Kod aplikacije :

```
using CuttingProblem.CuttingStock;
using Svg;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.IO;
using System.Windows.Forms;
namespace CuttingProblem
{
    public partial class FormMain : Form
    {
        Layout layout;
        decimal multiplierScale = 1;
        public FormMain()
        {
            InitializeComponent();
            layout = new CuttingStock.Layout(100, 100, 0);
            textBoxDuljina.Text = layout.Height.ToString("0.0");
            textBoxSirina.Text = layout.Width.ToString("0.0");
            textBoxSirinaReza.Text = layout.CutWidth.ToString("0.0");
        }
        private void FormMain_Load(object sender, EventArgs e)
        {
            setScale(2);
        }
        private void panelCanvas_Paint(object sender, PaintEventArgs
        e)
        {
```

```
e.Graphics.TranslateTransform(panelCanvas.AutoScrollPosition.X,
panelCanvas.AutoScrollPosition.Y);
            if (layout != null)
            {
                panelCanvas.AutoScrollMinSize = new
Size((int)(layout.Width * multiplierScale + 10), (int)(layout.Height *
multiplierScale + 10));
                Graphics g = e.Graphics;
                System.Drawing.Rectangle rect;
                Pen pen = new Pen(Color.Red, 1);
                HatchBrush hb = new
HatchBrush(HatchStyle.BackwardDiagonal, Color.Gray,
Color.Transparent);
                Font font;
                float fontSize;
                StringFormat stringFormat = new StringFormat();
                stringFormat.Alignment = StringAlignment.Center;
                stringFormat.LineAlignment = StringAlignment.Center;
                rect = new System.Drawing.Rectangle(
                        (int)(layout.X0 * multiplierScale),
                        (int)(layout.Y0 * multiplierScale),
                        (int)(layout.Width * multiplierScale),
                        (int)(layout.Height * multiplierScale)
                        );
                if (layout.Height > layout.Width)
                {
                    stringFormat.FormatFlags =
StringFormatFlags.DirectionVertical;
                    font = new Font(FontFamily.GenericSansSerif,
Math.Min(rect.Width / 3, rect.Height / 10), FontStyle.Bold,
GraphicsUnit.Pixel);
                }
                else
                {
                stringFormat.FormatFlags =
~StringFormatFlags.DirectionVertical;
```

```
                    font = new Font(FontFamily.GenericSansSerif,
Math.Min(rect.Height / 3, rect.Width / 10), FontStyle.Bold,
GraphicsUnit.Pixel);
                }
                g.FillRectangle(Brushes.White, 0, 0,
panelCanvas.Width, panelCanvas.Height);
                g.DrawRectangle(pen, rect);
                g.DrawString(layout.DimensionsToString(), font,
Brushes.LightGray, rect, stringFormat);
                pen.Color = Color.Black;
                foreach (CuttingStock.Rectangle panel in layout)
                {
                    rect = new System.Drawing.Rectangle(
                        (int)(panel.X0 * multiplierScale),
                        (int)(panel.Y0 * multiplierScale),
                        (int)(panel.Width * multiplierScale),
                        (int)(panel.Height * multiplierScale)
                        );
                    if (panel.Height > panel.Width)
                    {
                        stringFormat.FormatFlags =
StringFormatFlags.DirectionVertical;
                        fontSize = Math.Min(rect.Width / 3,
rect.Height / 10);
                    }
                    else
                    {
                        stringFormat.FormatFlags =
~StringFormatFlags.DirectionVertical;
                        fontSize = Math.Min(rect.Height / 3,
rect.Width / 10);
                    }
                    g.FillRectangle(hb, rect);
                    g.DrawRectangle(pen, rect);
                    if (fontSize > 0)
                    {
```

```
                        font = new Font(FontFamily.GenericSansSerif,
fontSize, FontStyle.Bold, GraphicsUnit.Pixel);
                        g.DrawString(panel.DimensionsToString(), font,
Brushes.Red, rect, stringFormat);
                    }
                }
                pen.Dispose();
            }
        }
        private void setScale(int scale)
        {
            multiplierScale = (decimal)Math.Pow(2, scale - 2);
            labelScale.Text = "Zoom: " + multiplierScale + "×";
            panelCanvas.Invalidate();
        }
        private void trackBarScale_ValueChanged(object sender,
EventArgs e)
        {
            setScale(trackBarScale.Value);
        }
        private void FormMain_Resize(object sender, EventArgs e)
        {
            //panelCanvas.Height += Height - formHeight;
            //formHeight = Height;
        }
        private void otvoriToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            openFileDialog1.ShowDialog();
        }
        private void spremiToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            saveFileDialog1.ShowDialog();
        }
        private void izlazToolStripMenuItem_Click(object sender,
EventArgs e)
```

```
        {
            Application.Exit();
        }
        private void openFileDialog1_FileOk(object sender,
CancelEventArgs e)
        {
            layout =
LayoutFactory.LoadFromTXT(((OpenFileDialog)sender).FileName);
            textBoxDuljina.Text = layout.Height.ToString("0.0");
            textBoxSirina.Text = layout.Width.ToString("0.0");
            textBoxSirinaReza.Text = layout.CutWidth.ToString("0.0");
            bindingListPloce.Clear();
            foreach (CuttingStock.Rectangle ploca in layout)
            {
                bindingListPloce.Add(ploca);
            }
            panelCanvas.Invalidate();
        }
        private void buttonOsvjezi_Click(object sender, EventArgs e)
        {
            try
            {
                layout.Height =
Convert.ToDecimal(textBoxDuljina.Text);
                layout.Width = Convert.ToDecimal(textBoxSirina.Text);
                layout.CutWidth =
Convert.ToDecimal(textBoxSirinaReza.Text);
                layout.clearPanels();
                SortedSet<CuttingStock.Rectangle> sortedPanels = new
SortedSet<CuttingStock.Rectangle>(new RectangleLargerDimComparer());
                foreach (CuttingStock.Rectangle ploca in
bindingListPloce)
                {
                    sortedPanels.Add(ploca);
                }
                foreach (CuttingStock.Rectangle ploca in sortedPanels)
                {
```

```
                    layout.addPanel(ploca);
                }
                dataGridViewPloce.Refresh();
                panelCanvas.Invalidate();
            }
            catch (Exception)
            {
                MessageBox.Show("Molimo unesite ispravne decimalne
vrijednosti.");
            }
        }
        private void bindingListPloce_ListChanged(object sender,
ListChangedEventArgs e)
        {
            foreach (DataGridViewRow redak in dataGridViewPloce.Rows)
            {
                CuttingStock.Rectangle ploca =
(CuttingStock.Rectangle)redak.DataBoundItem;
                if (ploca != null && !ploca.CanFit)
                {
                    redak.DefaultCellStyle.BackColor =
Color.LightPink;
                    redak.ErrorText = "Ploča ne može stati na glavnu
ploču.";
                }
                else
                {
                    redak.DefaultCellStyle.BackColor =
SystemColors.Window;
                    redak.ErrorText = "";
                }
            }
        }
        private void saveFileDialog1_FileOk(object sender,
CancelEventArgs e)
        {
```

```
            SvgColourServer bojaCrna = new
SvgColourServer(Color.Black);
            SvgColourServer bojaCrvena = new
SvgColourServer(Color.Red);
            SvgColourServer bojaBijela = new
SvgColourServer(Color.White);
            SvgColourServer bojaSiva = new
SvgColourServer(Color.Gray);
            SvgColourServer bojaSvijetloSiva = new
SvgColourServer(Color.LightGray);
            decimal sirinaLinije = 0.25M;
            SvgUnit sirinaLinijeMM = new
SvgUnit(SvgUnitType.Millimeter, (float)sirinaLinije);
            decimal sirinaVelikePloce = layout.Width;
            decimal duljinaVelikePloce = layout.Height;
            SvgDocument FSvgDoc = new SvgDocument
            {
                Width = new SvgUnit(SvgUnitType.Millimeter,
(float)(sirinaVelikePloce + sirinaLinije)),
                Height = new SvgUnit(SvgUnitType.Millimeter,
(float)(duljinaVelikePloce + sirinaLinije)),
            };
            FSvgDoc.ViewBox = new SvgViewBox(0, 0, FSvgDoc.Width,
FSvgDoc.Height);
            SvgPatternServer srafura = new SvgPatternServer
            {
                ID = "Srafura",
                PatternUnits = SvgCoordinateUnits.UserSpaceOnUse,
                Width = 10,
                Height = 10
            };
            srafura.Children.Add(new SvgLine
            {
                StartX = -10,
                StartY = 10,
                EndX = 0,
                EndY = 0,
```

```
                Fill = bojaSiva,
                Stroke = bojaSiva,
                StrokeLineCap = SvgStrokeLineCap.Square,
                StrokeLineJoin = SvgStrokeLineJoin.Miter,
                StrokeWidth = sirinaLinijeMM,
            });
            srafura.Children.Add(new SvgLine
            {
                StartX = 0,
                StartY = 10,
                EndX = 10,
                EndY = 0,
                Fill = bojaSiva,
                Stroke = bojaSiva,
                StrokeLineCap = SvgStrokeLineCap.Square,
                StrokeLineJoin = SvgStrokeLineJoin.Miter,
                StrokeWidth = sirinaLinijeMM,
            });
            srafura.Children.Add(new SvgLine
            {
                StartX = 10,
                StartY = 10,
                EndX = 20,
                EndY = 0,
                Fill = bojaSiva,
                Stroke = bojaSiva,
                StrokeLineCap = SvgStrokeLineCap.Square,
                StrokeLineJoin = SvgStrokeLineJoin.Miter,
                StrokeWidth = sirinaLinijeMM,
            });
            FSvgDoc.Children.Add(srafura);
            FSvgDoc.Children.Add(new SvgRectangle
            {
                X = new SvgUnit(SvgUnitType.Millimeter,
(float)(sirinaLinije / 2)),
                Y = new SvgUnit(SvgUnitType.Millimeter,
(float)(sirinaLinije / 2)),
```

```
                Width = new SvgUnit(SvgUnitType.Millimeter,
(float)(sirinaVelikePloce + sirinaLinije / 2)),
                Height = new SvgUnit(SvgUnitType.Millimeter,
(float)(duljinaVelikePloce + sirinaLinije / 2)),
                Stroke = bojaCrvena,
                StrokeWidth = sirinaLinijeMM,
                Fill = bojaBijela
            });
            SvgContentNode text = new SvgContentNode();
            text.Content = layout.DimensionsToString();
            SvgText textElement = new SvgText
            {
                FontSize = 20,
                FontFamily = FontFamily.GenericMonospace.Name,
                Fill = bojaSiva,
                TextAnchor = SvgTextAnchor.Middle,
            };
            textElement.X.Add(new SvgUnit(SvgUnitType.Millimeter,
(float)(sirinaLinije / 2 + sirinaVelikePloce / 2)));
            textElement.Y.Add(new SvgUnit(SvgUnitType.Millimeter,
(float)(sirinaLinije / 2 + duljinaVelikePloce / 2)));
            textElement.Nodes.Add(text);
            FSvgDoc.Children.Add(textElement);
            foreach (CuttingStock.Rectangle ploca in layout)
            {
                SvgRectangle svgPloca = new SvgRectangle
                {
                    X = new SvgUnit(SvgUnitType.Millimeter,
(float)(ploca.X0 + sirinaLinije / 2)),
                    Y = new SvgUnit(SvgUnitType.Millimeter,
(float)(ploca.Y0 + sirinaLinije / 2)),
                    Width = new SvgUnit(SvgUnitType.Millimeter,
(float)ploca.Width),
                    Height = new SvgUnit(SvgUnitType.Millimeter,
(float)ploca.Height),
                    Stroke = bojaCrna,
                    StrokeWidth = sirinaLinijeMM,
```

```
                    Fill = srafura
                };
                text = new SvgContentNode();
                text.Content = ploca.DimensionsToString();
                textElement = new SvgText
                {
                    FontSize = 10,
                    FontFamily = FontFamily.GenericMonospace.Name,
                    Fill = bojaCrvena,
                    TextAnchor = SvgTextAnchor.Middle,
                };
                textElement.X.Add(new SvgUnit(SvgUnitType.Millimeter,
(float)(ploca.X0 + sirinaLinije / 2 + ploca.Width / 2)));
                textElement.Y.Add(new SvgUnit(SvgUnitType.Millimeter,
(float)(ploca.Y0 + sirinaLinije / 2 + ploca.Height / 2)));
                textElement.Nodes.Add(text);
                FSvgDoc.Children.Add(svgPloca);
                FSvgDoc.Children.Add(textElement);
            }
            var stream = new MemoryStream();
            FSvgDoc.Write(stream);
            FSvgDoc.Write(saveFileDialog1.FileName);
        }
    }
}
```