Algoritme til løsning af geometriske puslespil ud fra længdematch og backtracking

Christian Skaarup Hansen, s002013

Kongens Lyngby 2007
IMM-Thesis-2007-56
2. Juni 2007

Vejleder : Jens Clausen

Danmarks Tekniske Universitet
Informatics and Mathematical Modelling
Bygning 321, DK-2800 Kongens Lyngby, Danmark
Telefon +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

Abstract

This report deals with the problematics of developing a computer programme which is able to solve geometric jigsaw puzzles correctly. The idea is that the programme can be further upgraded in such a way that it could work as a support tool in the future in industries where the demand of solving naturally generated puzzles is requested.

There are several ways to make such a programme and it is not easy to predict the way leading to the best result before the different methods have been tested. This may complicate the development of the mentioned programme and the progress has ended up in being rather drawn-out relative to the efficiency of the programme. After having developed an algorithm its functionality is tested. In view of the test results a series of optimizations of the implemented methods are carried out. The methods are tested again to see whether the rate of searching has increased in the optimized version.

The structure of the algorithm has similarities with many of the different existing solving algorithms.The most efficient approach developed to solve puzzles is the Cocktail Method combined with backtracking. Both methods exist in two versions: One where the searching process has been optimized and one where it has not. Tests have shown that the optimized version of backtracking is the one working fastest whilst for the Cocktail Method the non-optimized version is faster than the optimized version. It is not possible to generate a combined version of the algorithm in such a manner that both the Cocktail Method and the backtracking is working optimally.

Firstly the methods are tested on a simple jigsaw puzzle where each side of a piece only fits together with one other side of another piece. Subsequently there are runned tests on small jigsaw puzzles containing composite sides. Furthermore tests are carried out where the pieces are rotated in such a way that each piece has to be rotated before the actual position of the piece could be

determined. Finally tests are runned on a jigsaw puzzle consisting of 23 pieces in order to test whether the algorithm is able to handle puzzles with a larger number of pieces.

Resume

Denne rapport omhandler problematikken ved at udvikle et computerprogram, der kan løse geometriske puslespil korrekt. Ideen er, at programmet kan udvides, så det i fremtiden kan fungere som hjælpeværktøj i erhverv, hvor der er brug for at kunne samle naturligt skabte puslespil.

Der er mange fremgangsmåder, og det er ikke altid helt enkelt at overskue, hvilken vej, der vil føre til det bedste resultat, før de forskellige fremgangsmåder er afprøvet. Dette kan gøre fremstillingen af omtalte program noget problematisk, og forløbet har endt med at være noget langstrakt i forhold til, hvor effektivt programmet er. Efter at have fremstillet en algoritme, testes dens funktionalitet. På baggrund af testresultaterne gennemføres en række optimeringer af de implementerede metoder. Metoderne testes derefter igen for, om søgehastigheden er højere i den optimerede udgave.

Algoritmens opbygning ligner en del forskellige eksisterende løsningsmodeller. Den mest effektive metode, til at løse puslespil, der er udviklet, er Cocktailmetoden sammen med „Backtraking“. Begge metoder findes i to versioner; en hvor søgemetoden er optimeret og en, hvor den ikke er. Tests har vist at, den optimerede udgave af „Backtraking“ arbejder hurtigst, mens den ikke optimerede version af Cocktailmetoden, er hurtigere end den optimerede. Det kan ikke lade sig gøre at fremstille en kombineret version af algortimen, så både Cocktailmetoden og "Backtraking"fungerer optimalt.

Metoderne er først testet på et simpelt puslespil, hvor den enkelte briks sidelængder kun passer med én anden sidelængde. Der er derefter udført tests med små puslespil, som indeholder sammensatte sider. Desuden er der udført tests, hvor brikkerne er roteret, så hver enkelt brik skal drejes, før dens rette placering kan bestemmes. Som det sidste er der testet på et puslespil bestående af 23 brikker for at se, om algoritmen kan håndtere et større antal brikker.

Forord

Foreliggende projekt er udarbejdet som speciale i informatik på institut for Informatik og Matematisk Modellering (IMM) ved Danmarks Tekniske Universitet. Rapporten udgør den skriftlige del af et projekt, som er udført med det formål at fremstille et program, der hjælper mennesker til at løse geometriske puslespil med computerteknologi. Desuden er kildekoderne til algoritmen inkluderet som bilag.

I rapporten anvendes lejlighedsvis engelske termer, da det danske sprog ikke altid er tilstrækkeligt til at beskrive procesforløbet.

Jeg vil desuden benytte dette afsnit til at takke følgende personer for al den støtte og hjælp, de har bidraget med i forbindelse med udarbejdelsen af denne afhandling.

- Vejleder Jens Clausen fra IMM instititttet
- Sekretær Birgitte Meidahl
- Kontormedarbejder, Erling Veidal
- Oversætter Jeanet Grønbæk
- Fra Hovedstadens Ordblindeskole, Martin Richard
- Korrektur, Christian Grundahl Hartmann

Kongens Lyngby, 2 Juli 2007

Christian Skaarup Hansen

Indhold

1 Introduktion **1**
- 1.1 Problemstilling . . . 2
- 1.2 Opsummering . . . 2
- 1.3 Blokdiagram over systemet . . . 3
- 1.4 Problemafgrænsning . . . 4
- 1.5 Udviklingsmiljø . . . 5
- 1.6 Kapiteloversigt . . . 5

2 Referater **7**
- 2.1 Artikel 1 . . . 7
- 2.2 Artikel 2 . . . 9
- 2.3 Artikel 3 . . . 11

3 Søgemetoder på brikker **13**
- 3.1 Søgning på to sider med samme længde . . . 14
  - 3.1.1 Simpel søgning . . . 14
  - 3.1.2 Søgning uden udgangsbrik . . . 20
- 3.2 Søgning med sammensatte sider . . . 26
  - 3.2.1 Søgning med liste . . . 27
  - 3.2.2 Søgning uden liste . . . 28
- 3.3 Søgning med roterede brikker . . . 32
  - 3.3.1 Match for hver grad . . . 32
  - 3.3.2 Match uden rotation . . . 34
- 3.4 Søgning med Cocktailmetode . . . 35
- 3.5 Afslutning . . . 38

4 Optimering af Søgemetode **39**
4.1 Datastruktur . . . 39
4.1.1 Sorteringsmetode . . . 40
4.1.2 Søgemetode . . . 40
4.2 Afslutning . . . 42

5 Implementering **43**
5.1 Beskrivelse af algoritmen . . . 43
5.2 Implementering af søgning . . . 44
5.3 Implementering af optimering . . . 51
5.3.1 Søgemetode . . . 52
5.4 Delkonklusion . . . 54

6 Test **55**
6.1 Testmiljø . . . 55
6.2 Simple puslespil . . . 56
6.3 Sammensatte sider . . . 60
6.4 Roterede brikker . . . 64
6.5 Fuld test af algoritmen . . . 68
6.6 Fejltest . . . 73
6.7 Delkonklusion . . . 75

7 Diskussion **77**

8 Konklusion **79**
8.1 Forkert retning . . . 79
8.2 Opfyldelse af problemformulering . . . 79
8.3 Fremtid . . . 80

A Teori til billedanalyse **83**
A.1 Udskillelse . . . 83
A.2 Matematisk morfologi . . . 86
A.2.1 Metoder i Matematisk morfologi . . . 86
A.2.2 Matematisk morfologi i OpenCV . . . 89
A.3 Omrids . . . 90
A.3.1 Find omrids i OpenCV . . . 90
A.3.2 Fint til groft omrids . . . 91
A.4 Afrunding på teori til billedanalyse . . . 94

B Almindeligt puslespil **95**
B.1 Hvordan lægger man puslespillet? 95
B.2 Metode til analyse af brikker 96
B.2.1 Uddybende forklaring af analyseprogrammet 97
B.2.2 Resultater af analyse på brikker 100
B.3 Afrunding på almindeligt puslespil 103

C Kildekode **105**
C.1 Koden til algoritmen 105
C.1.1 PuzzleAlgo3.cs 105
C.1.2 PuzzleAlgo4.cs 144
C.1.3 Puzzle.cs 186
C.1.4 Form2.cs 202
C.1.5 Viewer.cs 205

Kapitel 1

Introduktion

Puslespil fås i forskellige udgaver. Der er dem til underholdning og dem, der bliver skabt ved uheld, f.eks. når man taber en vase på gulvet.

De fleste personer, der skal samle et almindeligt puslespil, vil som det første finde alle de brikker, der skal bruges til kanterne. De er lette at genkende, idet en af siderne altid vil være lige. Der er dog 4 brikker, hvor to af siderne er lige, nemlig hjørnebrikkerne. Når alle kant- og hjørnebrikker er sorteret fra, kan man begynde at samle rammen. Når man samler rammen, fokuseres der ikke på faconen, men istedet på farver, nuancer og motiv.

Når alle brikkerne til kanten er lagt, sorteres resten af brikkerne efter forskellige motiver. Derefter går man igang med at sætte motiverne sammen, så ser man på omgivelserne og til sidst på himlen. Undervejs er der måske ikke blevet kigget så nøjagtigt på brikkernes facon bort set fra eventuelle indhak eller tapper. Ved samling af baggrunden er det nødvendigt at se på faconen, da brikkerne i mange tilfælde har samme nuance. For lettest at samle baggrunden af puslespillet skal man se på formen af brikkernes sider. De brikker, hvor siderne skal placeres op ad hinanden, skal formerne bevæge sig på samme måde.

Almindelige puslespil er kendetegnet ved ensartede takker og indhak, som gør dem overskuelige. Det er derimod mere kompliceret at samle en vase eller en figur, som er gået i stykker. Disse puslespil kan betagtes et som geometisk puslespil. Da konservatorens tid i mange tilfælde er kostbar, kan det være en fordel, hvis der kan benyttes computerteknologi til at samle en knust vase. Ved

at benytte computerteknologi er det muligt at danne en manual over, hvordan stumperne skal lægges. Det betyder, at vasen kan samles langt hurtigere, end hvis en person skulle gøre hele arbejdet selv.

1.1 Problemstilling

Formålet med dette projekt er at udvikle en algoritme, der kan udvides til at samle et 3D-puslespil. Når algoritmen er funktionsdygtig, kan den bruges af folk, som ønsker at samle en knust genstand. Det kunne f.eks være et anvendeligt værktøj for arkæologer ved udgravninger. Ved algoritmens hjælp kan de få samlet potteskårene fra forhistorisk tid langt hurtigere end noget menneske kunne have gjort det. Potteskårene kan betragtes som 3D-puslespil. Hvis det er skabt ved naturens hjælp, eller ved et uheld kan et 3D-puslespil betagtes som et geometrisk puslespil. Et geometrisk puslespil er et puslespil, hvor man ikke tager højde for brikkernes motiv, men kun for deres facon. Disse potteskår kan i nogle tilfælde indeholde farver og mønstre. Derfor er det vigtigt, at algortimen ikke genkender på baggrund af maling eller mønstre på potteskårene.

Til at begynde med skal algoritmen kun kunne løse 2D geometriske puslespil. Den skal også kunne samle visuelle geometriske puslespil, det vil sige, puslespil der er genereret i en computer. Derved har algoritmen kun brikkernes koordinater at arbejde med. Når algoritmen er fremstillet, skal der være mulighed for at udvide den til tre dimensioner.

Udover algoritmen skal systemet indeholde et program, der kan opmåle de indscannede brikker. Programmet skal kunne opmåle og lagre hver brik som et polygon, da denne form for data lettest at arbejde med. Hvis de indscannede brikker ligger tilfældigt i forhold til hinanden, er det vigtigt at fremstille systemet sådan, at brikkerne kan flyttes og roteres.

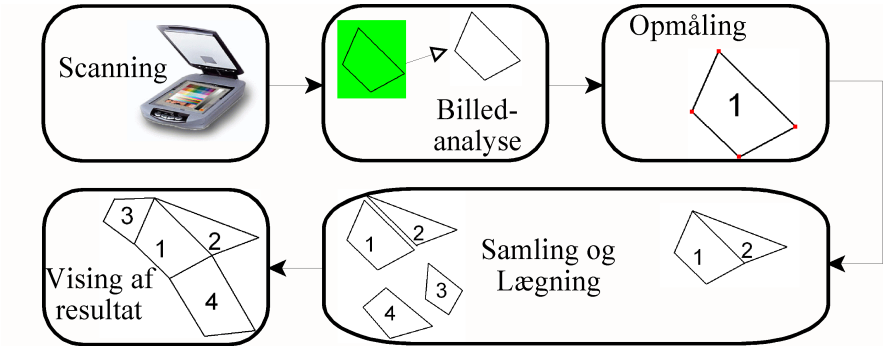
1.2 Opsummering

Under projektforløbet skal der udvikles en algoritme, som kan samle et geometrisk 2D-puslespil. Med visse udvidelser skal algoritmen være i stand til at arbejde med tre dimensioner. Når algoritmen har lagt puslespillet, skal løsningen vises på computerskærmen, så brugeren kan se, hvordan det skal lægges.

Til at begynde med skal algoritmen ved test kunne lægge et rigtigt geometrisk puslespil bestående af 50 brikker. Det rigtige geometriske puslespil kan være skabt ved blot at klippe et billede i stykker. Disse scannes ind på computeren, hvorefter algoritmen vil lægge det.

1.3 Blokdiagram over systemet

Fremgangsmåden kan deles op i mindre komponenter, som kaldes blokke. Hver blok kan indeholde en række relaterede funktioner, der skal bruges til at håndtere udfordringerne ved at samle et geometrisk puslespil. Figur 1.1 viser et blokdiagram over processerne i systemet.



Figur 1.1: Blokdiagram over processerne i systemet

I den første blok står scannes brikkerne. I forbindelse med dette skal der findes en passende opløsning. Den skal være høj nok til, at det er muligt at sammenligne brikker.

Den næste blok er billedanalysen af de indscannede brikker. Billedanalysen filtrerer al baggrund væk, så systemet kun har brikkerne at arbejde med.

Når brikkerne er klippet ud af scanningen kan de opmåles, hvilket sker i tredje blok. Opmålingerne benyttes i fjerde blok, hvor brikkerne sorteres og sammenlignes med hinanden. Når fjerde blok er færdig afslører femte blok, fremvisning af resultater, hvordan brikkerne skal lægges.

1.4 Problemafgrænsning

I dette afsnit opstilles nogle punkter, som målretter opgaverne mod løsning af projektet. Dette gøres først og fremmest for at projektet ikke skal løbe løbsk og resultere i en masse løse ender.

Projektets formål er at udvikle en algoritme, der kan løse et geometrisk puslespil. For ikke at komplicere opgaven skal algoritmen i første omgang være i stand til at lægge et visuelt geometrisk puslespil i to dimensioner. Det visuelle pulsespil bygges op af polygoner, da geometriske puslespilsbrikker kan betragtes som mangesidede figurer.

Hvis der bliver tid, vil der være mulighed for at udvide algoritmen til at kunne medtage en tredje dimension. Algoritmen skal leve op til de herunder opstillede punkter:

- Algoritmen skal kunne lægge et lille visuelt puslespil.
- Algoritmen skal kunne håndtere visuelle puslespil, hvor der kan være flere brikker med sammme sidelængde.
- Algoritmen skal kunne vise alternative løsninger, hvis sådanne findes.
- Algoritmen skal kunne håndtere sammensatte sider.
- Algoritmen skal kunne håndtere roterede brikker.
- Algoritmen skal være skrevet i C#.

Hvis der er tid tilovers, når ovenstående punkter er gennemført og testet med tilfredsstillende resultat, er der mulighed for at udbygge systemet, så algoritmen kan håndtere et ægte geometrisk puslespil. Dette geometriske puslespil kan være et indscannet billede, som er klippet i stykker. Da billedbehandlingen ikke er en del af problemafgrænsningen, skal den ikke optage for meget plads. De punkter, som projektet med fordel kunne udvides med står listet herunder:

- Udvikling af et program, som kan konvertere indscannede brikker til polygoner.
- Test af algoritmen med indscannede brikker.
- At algoritmen kan arbejde i 3D.

1.5 Udviklingsmiljø

I processens forløb vil der være en del programmer, som bruges som hjælpemidler samt udviklingsværktøj og til analyse af resultater. Herunder er listet en række programmer og biblioteker:

- Udviklingsværktøj
  - Visual Studio 2005 (Programmet bruges til at udvikle programmerne i projektet)
  - Matlab 7.0 (Programmet bruges til de analyseresultater, som kommer fra algoritmen)
- Library
  - OpenCV (Open Source Computer Vision Library, bruges til alt, hvad der omhandler billedanalyse)
- Advendte programmeringssprog
  - C/C++
  - Matlab
  - C#

1.6 Kapiteloversigt

- I kapitel 2 findes referater af de artikler, som har givet inspiration til projektet.
- Kapitel 3 omhandler opbygning af algoritmen. Der vil blive set på forskellige metoder til at opfylde de krav, som er opstillet i problemafgrænsningen.
- I kapitel 4 gennemgås muligheder for optimering af algoritmen. Der lægges vægt på at gøre søgemetoderne hurtigere.
- Kapitel 5 sætter fokus på implementering af klassediagrammer til algoritmen. Der er tabeller, som forklarer de variable og funktioner, som tilhører de forskellige klasser.
- I kapitel 6 omtales en gennemgående test af hele systemet.

- I kapitel 7 gives en omfattende diskussion af projektet.
- Kapitel 8 er en konklusion, hvor der redegøres for om projektformuleringen er opfyldt, og der reflekteres over projektforløbet.

Kapitel 2

Referater

I dette kapitel vil de artikler, der har bidraget med inspiration til opbygning og udvikling af algoritmen blive refereret. Listen herunder viser overskrifter på de refererede artikler:

1. On Solving 2D and 3D Puzzles Using Curve Matching [5]
2. Reassembling Fractured Objects by Geometric matching [3]
3. Jigsaw Puzzle Matching Using a Boundary-Centered Polar Encoding [7]

2.1 Artikel 1

Formålet med det projekt, artiklen af Kon *et al.* omhandler, er at udvikle en automatisk metode, hvor man vha. en computer kan løse 2D- og 3D-puslespil. Artiklen anskuer problemet i to stadier, lokalt og globalt. Søgemetoden anvender form matching på begge problemer ved at samle 2D- og 3D-puslespil. Forfatterne fokuserer på, at delvis kurvesammenligning er meget vigtigt, da mange af brikkerne kan være af forskellig størrelse og alligevel være naboer. Ved brug af denne metode satser de på at udvikle en algoritme, der er i stand til at lægge forskellige geometriske puslespil. Kon *et al.* har fundet ud af, at hvis puslepillet består af mere end fire brikker, så vokser antallet af muligheder eksponentielt, og derfor fokuserer de meget på den globale løsningsmodel, da den kan nedsætte antallet af potentielle løsninger.

Det første problem, de arbejder med, er at parre 2D geometriske brikker. Til dette formål anvendes curve-matching. Før curve-matching kan finde sted, er Kon *et al.* nødt til at opmåle brikkerne, så de får et polygon for hver brik. Metoden sammenligner kurverne på brikkernes små linjestykker med de originale kurver fra andre brikker. Derefter udføres 2D euklidisk transformation. Efterfølgende kontrolleres der for eventuelle overlap mellem brikkerne. Hvis disse forekommer kasseres løsningen, og derefter overføres de gyldige match til søge-algoritmen, som samler puslespillet.

I 2D curve matching metoden tester forfatterne metoden „fin-skala“. Metoden skal bidrage til at bestemme et mere præcist match. Konklusionen er, at 2D curve matching fungerer bedst, når metoden „fin-skala“ indgår. Metoden er den bedste til at finde matchende linjer, idet den forhindrer, at der sker overlap af kurver.

Efter at have udført test på 2D niveau fortsætter Kon *et al.* arbejdet med at overføre til 3D niveau. Kon *et al.* bygger en afstandsmatrice af brikkerne, som består af hastighed, krumninger og snoninger.
Den anvendte metode er dog meget følsom overfor uregelmæssigheder. For at få et bedre resultat af krumninger og snoninger benyttes metoden "ENO scheme", som står for „Essentially Non-Oscillatory". „ENO scheme“ glatter krumninger og snoninger på brikkerne, hvilket gør det lettere for algotimen at finde match.

Der udføres test vha. af computergenererede brikker, og konklusionen er, at når der er under 12 brikker, kan det antages, at brikkerne er forskellige. Hvis antallet af brikker overstiger 12, kan man antage, at metoden vil tolke nogle af brikkerne som ens. Til at lette algoritmens opgave benyttes metoden „best-first search with backtracking“. Registreres et match mellem to brikker erstattes disse med en ny, der har form som de to matchende brikker smeltet sammen. Metoden reducerer således antallet af brikker i puslepillet, efterhånden som algoritmen finder flere match mellem brikkerne. På den måde udelukkes flere og flere brikker, hvilket letter vejen til løsningen af puslespillet.
En af ulemperne ved de beskrevne metoder er, at de er forfølsomme overfor uregelmæssigheder ved brikkerne. For at undgå disse uregelmæssigheder benyttes en del funktioner til at glatte kurver og linier ud. Disse funktioner tager tid, når algoritmen skal samle et puslespil.

2.2 Artikel 2

Huang *et al.* beskæftiger sig i en artikel med, hvordan man kan samle ødelagte objekter ved geometrisk matching. Feltarkæologer har i mange år søgt efter et program, der kunne bruges til at samle 3D-objekter, der er gået i stykker. Systemet skal samle objektet ud fra geometriske mål i stedet for som mennesker at gå ud fra farver og tekstur. De mener, at problemet med samling af objektsiderne kan løses ved at bruge metoder fra Computer Grafik og Vision, da disse problemer minder meget om hinanden. I artiklen illustrerer de algoritmens automatiske samling af 3D-objekter med eksempler med digitale modeller af brudte flader. De løser problemer med henblik på arkæologi, men med få modifikationer kan algoritmen bruges til andre formål, hvor der skal samles 3D-objekter.

Målet er at samle en figur, der er gået i stykket, hvor fragmenterne er laserscannede, så de har en digital model af fragmenter fra den ødelagte figur. Forfatterne mener, at det kan skabe nogle fejlplaceringer af fragmenterne, da dele af brudfladerne kan passe sammen flere steder på andre brudflader.

Forfatterne har et fokusområde, som de deler op i fire områder:

- Ny integral invarians for overflader og 3D kurver, som er beregnet på multi-skalaer og bliver brugt i multi-level data segmentering og genkendelse af karakteristiske træk.
- Robust parvis matching, der bruger grupper af træk, som omfatter overfladeegenskaber på forskellige skalaer.
- Grafisk optimeringsmetode for den globale matching af mange fragmenter.
- Begrænset optimering for lokal registrering uden gensidig indtrængen af både to og mange fragmenter.

Den algoritme, Huang *et al.* fremstiller, får først alle fragmenterne i punktform. Deres første trin er at få skabt fragmenterne om fra punktform til kurver og skarpe kanter.

Ved at undersøge den ujævne overflade, kan algoritmen finde frem til, hvad der er brudflader, og hvad der er oprindelige flader. Da det kun er brudfladerne, som har interesse, vil det udelukke de glatte flader i matchingprocessen, og

derved vil processen gå hurtigere.

Denne segmentering øger effektiviteten af algoritmen på to måder: For det første er det kun brudflader, der kan blive matchet med/mod hinanden i begyndelsen af samlingsprocessen af et 3D-objekt, og hver overflade giver en naturlig gruppering af en række matchende træk fra fragmentets overflade. For det andet kan man øge algoritmens styrke ved at samle den originale overflade af figuren.

Fordelen ved metoden er, at algoritmen samligner overflader, når der skal samles brikker. Ved at udelukke sider, som ikke er brudflader får algoritmen færre sider at arbejde med.

2.3 Artikel 3

Problemet, som Radack *et al.* arbejder med i denne artikel, er at samle et puslespil ved hjælp af kurvegenkendelser. Radack *et al.* har ingen intention om at bruge grafisk genkendelse af brikkerne, men derimod kun kurver og former af brikkernes omkreds. De fokuserer på puslespil, hvor brikkerne passer unikt sammen, og derfor er det ikke nødvendigt at implementere backtracking (det vil sige, man lægger puslespillet, og hvis det ikke lykkes, går man tilbage og prøver igen med andre brikker) i deres løsning. Forfatterne mener, at det vil give en bedre forståelse af problemet med at samle puslespillet, hvis det undlades. Idet backtracking undlades skal sammenligningsmetoden være så præcis, at de korrekte match findes i første gennemløb.

Matchning afgrænser mellem par af kurver, der kan anses for at være opdelt i tre faser, som hver svarer til et af følgende spørgsmål.

1. "Startpunkt"Find punktet på kurven 1 og 2, som tangerer hinanden. Disse kaldes "match point."

2. "I hvilken retning"Find orienteringen af kurve 2 under hensyn til kurve 1, som tillader det bedste match. En naturlig måde at specificere dette er at rotere rundt "match point". Denne vinkel kaldes "match angle".

3. "Hvor langt skal man gå"bestemmes af de matchende udsnit af kurve 1 og 2. De matchende udsnit skal være forbundet af grænseudsnit, der indeholder matchende punkter. Længden af disse udsnit (de skal overordnet set være ens) kaldes "match length."

Matching points er en måde at sammenligne kurver på, hvor man går efter maksimums- og minimumspunkterne på kurven, som også kaldes de kritiske punkter. Derudover er det meget vigtigt, om brikkerne er opmålt med eller mod uret. Forfatterne vælger, at alle brikkerne opmåles med urets retning, og herved kan man så bestemme, at ved maksimum buer kurven udad og ved minimum buer den indad. Skarpe knæk udglattes, hvorved de som kritiske punkter forsvinder. Dette er nødvendigt, fordi metoden er meget følsom overfor støj på kurverne.

Matchvinklen sørger for at rotere brikkerne, så de to matchende kurver får den samme vinkel. Den fundne brik roteres om "match point"indtil kurven på afstanden D har den samme vinkel, som den kurve den matcher med. D er

afstanden for kurven, der skal matches.

Matchlængden er afstanden mellem start- og slutpunktet på en kurve. Denne afstand kan bruges til at finde andre kurver med samme længde, hvorefter man kan kontrollere, om kurverne er ens.

Radack *et al.* får fremstillet en algoritme, som kan samle et puslespil uden at benytte backtracking. Radack *et al.* mener, at følgende punkter kan være relevante at udbygge algoritmen med:

1. Hurtigere og bedre til at identificering kritiske punkter ved hjælp af en forbedret algoritme.
2. Forbedre match vinkel metoden efter at matchlængden er fundet, så sammenligning af kurverne bliver mere nøjagtig.
3. Udvide algoritmen til 3D.

Der er ikke de store ulemper ved metoden i denne artikel. Det mest kritisable er tidsforbruget ved at sammenligne kurver på brikkernes brudflader.

Kapitel 3

Søgemetoder på brikker

Dette kapitel vil omhandle metoder, som kan bruges i algoritmen ved samling af et visuelt geometrisk puslespil. Brikkerne, som algoritmen arbejder med, er repræsenteret som polygoner, hvor et polygon er angivet med koordinater. Ud fra koordinaterne kan polygonernes sidelængder, midtpunkter og arealer beregnes.

Herunder ses en liste, der kort fortæller, hvad algoritmen skal kunne. Listen indholder elementer, som algoritmen skal være opmærksom på:

- Algoritmen skal kunne samle et visuelt geometrisk puslespil ud fra brikkernes sidelængder.
- Algoritmen skal kunne registrere overlap.
- Algoritmen skal kunne håndtere sammensatte sider (flere brikkers sider støder op til den samme side på en anden brik)
- Algoritmen skal kunne håndtere brikker, der er roterede i forhold til hinanden.

Ovenstående punkter kan overføres til det diagram, der kan ses på figur 3.1

3.1 Søgning på to sider med samme længde

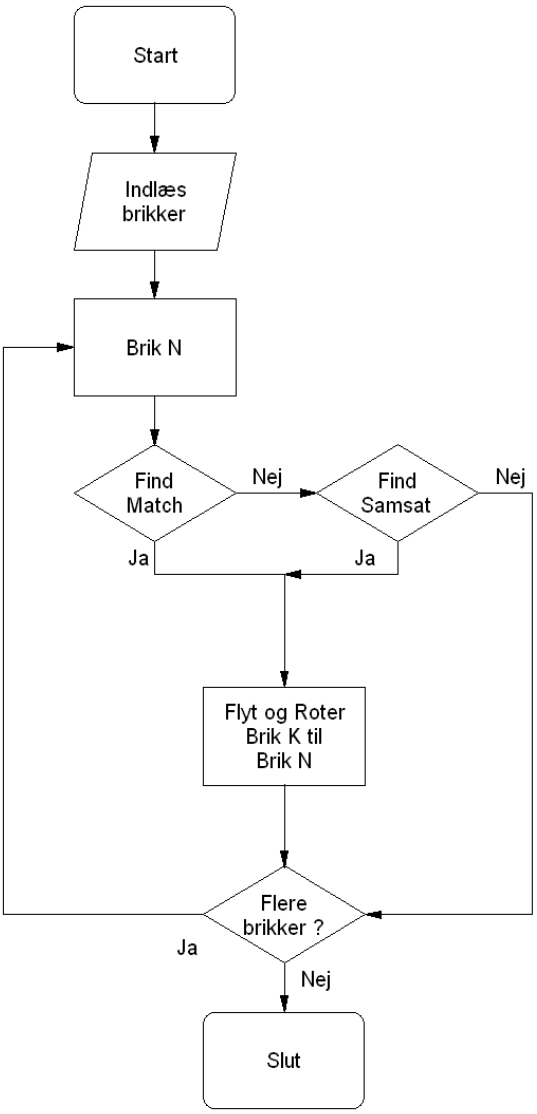
For at undgå problemer holdes algoritmen i begyndelsen på et simpelt niveau. Den første version af algoritmen skal som nævnt kun være i stand til at lægge et simpelt visuelt puslespil. Det visuelle puslespil, som er brugt her, er lavet, så der kun er en nabo til hver side.

Siderne har i dette tilfælde samme længde, som figur 3.2 nedenfor viser et eksempel på. Hvis mere end en brik støder op til samme side, kan det have stor betydning for søgningen. Dette vil blive behandlet i afsnit 3.1.2.

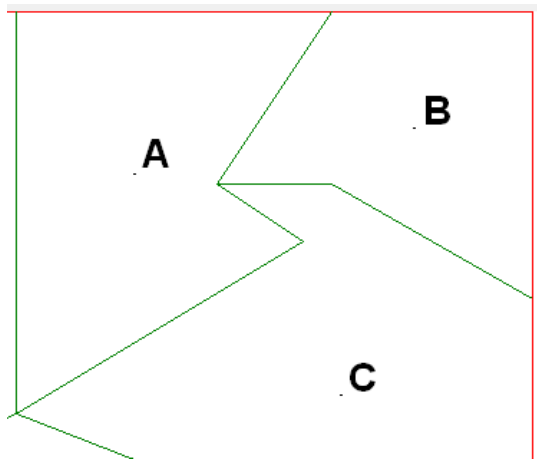
3.1.1 Simpel søgning

I begyndelsen er det et meget simpelt puslespil med kun 5 brikker, der skal løses. Alle brikkernes sider har forskellige længder. På den måde er det lettere at finde den rigtige løsning. Dog er alle brikkernes starthjørner lagt i $(x, y) = (0, 0)$ for at skabe samme udgangspunkt som ved brug af indscannede brikker. Når alle brikker er scannet ind vil det første målepunkt ligge i $(x, y) = (0, 0)$.

I dette afsnit arbejdes der ud fra de brikker, som ses i figur 3.3. Figuren viser, hvordan alle brikkerne ser ud, når de ligger korrekt.



Figur 3.1: Flow-diagram over motoren i algoritmen



Figur 3.2: Eksempel på forskellige sidelængder på brikkerne, hvor hver side er unik

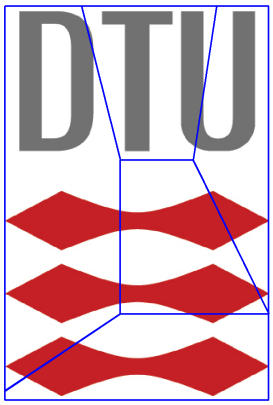
Som det første skal algoritmen kunne loade alle brikkerne ind i hukommelsen. Derefter skal alle sidelængder beregnes. Man kan finde eksempler på løsninger på dette problem i artiklerne [2,3,5]. Denne algoritmes fremgangsmåde vil være inspireret af artikel [5].

I artikel [2] arbejdes der ud fra et almindeligt puslespil, som har nogle faste former. Dog kan brikkerne have flere naboer. Dette emne vil blive behandlet senere i dette kapitel.

Metoden i artikel [2] er god til at finde naboerne. Afstanden fra midten af brikken ud til kanten på udgangsbrikken såvel som på nabobrikken betragtes. Man kan konkludere, at brikkerne er naboer, hvis linjestykkerne har samme længde. Dog kan metoden være langsom, da brikkerne muligvis skal roteres. Dette emne vil blive uddybet i afsnit 3.3.

Metoden i artikel [3] er ikke særlig anvendelig på dette stadie i udviklingen, da forfatterne har lagt fokus på problemerne med 3D.

I dette projekt vil brikkene blive genkendt på afstanden mellem opmålspunkterne. Opmålspunkterne er nogle punkter, der fortæller, om der sker en æn-



Figur 3.3: Billeder af de første testbrikker

dring i retningen på en briks side. Afstanden mellem to opmålspunkter vil passe med sidelængden på en brik.

For at få nok information om brikkernes sider skal man have et tilstrækkeligt antal punkter. Der må på den anden side heller ikke være for mange, da dette kan nedsætte hastigheden på søgningen. Derfor er de visuelle puslespil lavet, så alle opmålspunkterne ligger i brikkernes hjørner, det vil sige, alle siderne i puslespillet er rette.

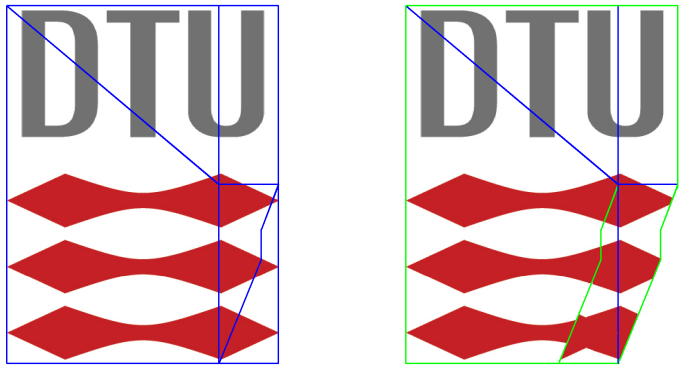
På denne måde er det kun formen på brikkerne, som skal sammenlignes. I første omgang er det altså ikke nødvendigt at kende motiverne på brikkerne.

Til at begynde med bliver der udvalgt en tilfældig brik, der danner udgangspunkt for løsningen af hele puslespillet. Det er altså ikke en bestemt brik, der skal findes først. Når der er valgt en udgangsbrik betragtes den første sidelængde. Hvis en side på en anden brik har samme længde, er der stor sandsynlighed for, at denne brik er nabo til udgangsbrikken.

Når udgangsbrikken har fået en nabo til den ene side bliver denne nye nabobriks opmålspunkter en del af udgangsbrikkens opmålspunkter. De to brikker bliver så at sige smeltet sammen til en ny udgangsbrik og sidelængderne

mellem udgangsbrikken og den fundne nabobrik bliver slettet, så der ikke er registreret sider og sidelængder midt inde i den nye udgangsbrik, hvilket ville forstyrre videre søgning.

Når der er fundet en ny nabobrik, skal det undersøges, hvilken sidelængde der passer til den. Ellers kan man risikere at ende ud med en løsning, som den der er vist på figur 3.4.



Figur 3.4: Til venstre vises, hvordan brikkerne skal ligge. Til højre kan man se problemet ved, at en brik ikke har fundet alle sine naboer. Ved at undersøge en nylagt briks naboer undgår man i mange tilfælde overlap. De grønne linier betyder, at siden er aktiv, det vil sige, der hvor det er muligt at lægge nye brikker til, mens de blå er passive sider

På figur 3.4 kan man se, hvordan algoritmen løser puslespillet, når der ikke føres kontrol med, hvilke sider, der støder op til den nye nabobrik. I eksemplet er der tre linjestykker inde i billedet. Disse linjestykker kan medføre fejl i løsning af hele puslespillet. En typisk fejl er, at en brik placeres midt på udgangsbrikken, fordi linjestykkerne passer sammen. Overlappet medfører, at det resulterende motiv bliver forkert, eftersom der ligger brikker ovenpå hinanden. Dette løses ved at undersøge, om der er andre sider på den nye nabobrik, der passer med sider på udgangsbrikken.

Samtidig kan det hjælpe algoritmen at teste, om den nye nabobrik overlapper udgangsbrikken. Algoritmen kan så hurtigt finde ud af, om den ny nabobrik er den rigtige. Metoden til at teste for overlap bliver gennemgået i afsnit 3.1.2.

For at undgå problemer med tolerance på siderne er der under udviklingen af algoritmen brugt visuelle puslespil, hvor brikkernes linjestykker har samme sidelængde som nabobrikken. Der kan være behov for en vis tolerance, når der senere anvendes indscannede brikker. Her kan det ikke garanteres, at sidelængderne er nøjagtig lige lange. Systemet skal derfor forberedes på, at der skal være en vis tolerance. Tolerancens størrelse kan ikke fastsættes på dette tidspunkt i forløbet, da senere ændringer i og udvidelser af algoritmen kan have indflydelse på toleranceværdien.

3.1.2 Søgning uden udgangsbrik

Når der indføres tolerance øges sandsynligheden for, at flere brikker passer til det samme linjestykke. Der er forskellige metoder til at omgå denne problematik. Dog har de fleste metoder det fællestræk, at der kan opstå problemer, når der findes en forkert nabobrik.

Oprindeligt var ideen, at puslespillet skulle lægges ud fra en tilfældigt udvalgt udgangsbrik. Resten af brikkerne ville så blive lagt rundt om denne udgangbrik. Dette kan give problemer, når der er sket en fejl ved lægning af puslespillet.

Da udgangsbrikken vokser efterhånden som de brikker, der matcher lægges sammen med den, bliver det også besværligt at trække en forkert lagt brik fra igen. For at gøre det lettere at rette den type fejl, bør algoritmen samle puslespillet uden udgangsbrik.

Den metode, som anvendes fra nu af kommer til at fungere på næsten samme måde. Når der er fundet en ny nabo, vil de sider, som støder op til hinanden, blive deaktiverede. Søgningen flytter automatisk videre til den nye nabo.

Denne fremgangsmåde gør det lettere at bruge rekursive kald til at lægge pulsespil. Bruger man rekursive kald, kan man nemt backtracke i tilfælde af fejl.

I tilfælde af at der findes flere brikker med samme sidelængde, kan en brik blive placeret det forkerte sted. For at kontrollere, om en brik er lagt korrekt, undersøger man, om brikken har en gyldig plads. Hvis brikken som udgangspunkt ikke er roteret, undersøges følgende:

1. Om den fundne brik er blevet lagt korrekt.

2. At start- og slutpunkt for siden stemmer overens med den fundne side, så de har samme længde.

3. At brikken ikke bliver lagt ovenpå andre brikker.

Det er vigtigt at finde ud af, om den fundne brik er lagt korrekt, så algoritmen kan tage højde for, om brikken kan flyttes, eller om den blot skal sammenligne koordinater. Når punkt 1 er opfyldt, er det muligt at gå videre til punkt 2. Hvis et match mellem to brikker skal godkendes, skal brikkens start- og slutpunkt

passe med den fundne nabobriks start- og slutpunkt og omvendt. Dette gælder naturligvis også for eventuelle andre brikker, hvis sider støder op til den fundne nabobrik. Først når punkt 1 og 2 er opfyldt, kan man gå videre til punkt 3, som ligeledes skal være opfyldt, før man kan begynde søgning efter en ny brik. Her kontrolleres det, om den nye nabobrik overlapper andre af puslespillets brikker.

For at finde overlap på brikkerne, kan disse betragtes som polygoner. Dette er muligt, fordi brikkernes position og koordinater på hjørnepunkterne er kendt. Herunder beskrives nogle metoder, som er gode til at løse problemet med overlap.

En af metoderne går ud på at undersøge, om der er skæring mellem nogle af linjerne. Dette tyder på et muligt overlap.

Ulempen ved denne metode er, at den kan være en stor tidssluger ved løsning af store pulsespil, da der kræves flere undsøgelser for overlap, jo flere brikker, der er. Derfor kan det være godt at finde simplere løsningsformer, som kun laver et gennemløb for hver brik, der bliver lagt.

Kompasmetoden

Under projektforløbet udvikles Kompasmetoden, som ligeledes skal undersøge, om der er overlap. Denne metode er udviklet som alternativ til søgning uden udgangsbrik. Metoden undersøger – ligesom søgning uden udgangsbrik – brikkernes retning i forhold til hinanden. Man finder retningen på en given brik, A, ved at tage udgangspunkt i den side på A, der støder op til en anden brik, B. For den side de to brikker har til fælles findes midtpunktet. Fra midtpunktet søges ud på midten af begge brikker. Midtpunktet på et polygon kan findes ved at bruge ligningerne 3.1 og 3.2[1].

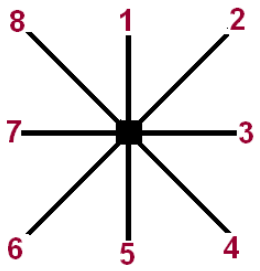
$$C_x = \frac{1}{6A} \sum_{i=0}^{N-1} (x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i) \tag{3.1}$$

$$C_y = \frac{1}{6A} \sum_{i=0}^{N-1} (y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i) \tag{3.2}$$

Når midtpunktet er fundet, kan der fastsættes en retning for, hvor brikkernes midtpunkter ligger. Retningerne deles op 1-8 hvor 1 er nord, 3 øst, 5 syd og 7 vest, jf. figur 3.5.

Når retningen på de to brikker er kendt, er det muligt at bestemme, hvorvidt brikkerne ligger ovenpå hinanden.

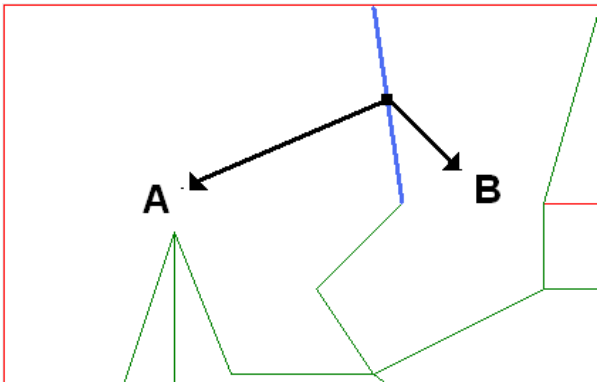
[1] http://local.wasp.uwa.edu.au/ pbourke/geometry/polyarea/



Figur 3.5: Kompasrosen viser, hvilke værdier der tillægges de forskellige retninger

Dog er der tidspunkter, hvor kompasmetoden ikke kan sige om der er overlap, fordi retningen er den samme. Dette kan f.eks. ske, hvis en brik krummer sig rundt om en anden brik. I denne situation regner man med, at brikkerne har flere sider til fælles, og kompasmetoden må frafalde.

Kompasmetoden er illustreret på figur 3.6.



Figur 3.6: Figuren viser to brikker, der støder op til hinanden, men brikkerne har hver deres retning i forhold til fællessiden. Brik A har retningen 6 mens Brik B har retningen 4.

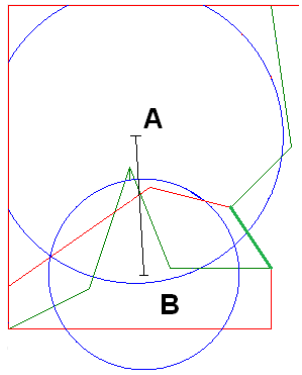
Cirkelarealmetoden

I nogle tilfælde kan kompasmetoden ikke vise om brikkerne overlapper. Dette kan f.eks. ske, hvis den ene brik er meget større end den anden. Når det sker, kan brikkerne have midtpunkter i hver deres retning og alligevel overlappe. For at undgå dette laves arealet af polygonet om til et cirkelareal. Alle polygonets koordinater skal være kendt for at kunne finde areal og centrum. Polygonet P kommer til at bestå af en række koordinater x_i, y_i. Arealet på et polygon kan findes ved at bruge ligningen[2] herunder:

$$A = \frac{1}{2} \sum_{i=0}^{N-1} (x_i y_{i+1} - x_{i+1} y_i) \tag{3.3}$$

hvor N er antal sider på polygonet.

Den beregnede radius bruges til at kontrollere, om der ligger en lille brik ovenpå en større brik. Hvis radius er større end afstanden mellem de to brikkers midtpunkter, er der tale om overlap.



Figur 3.7: Viser et eksempel på, hvordan radius kan udelukke en brik, som er godkendt ved brug af kompasmetoden. Radius for brik A er, som det fremgår af den blå cirkel, længere end afstanden mellem midtpunkterne

.

[2]http://local.wasp.uwa.edu.au/ pbourke/geometry/polyarea/

Ved at kombinere kompasmetoden og cirkelarealmetoden vil der være færre operationer end ved undersøgelse af skæring mellem linjer, fordi radius og midtpunkt bliver regnet ud ved indlæsning af brikkerne. Der vil derfor kun være tre ting, som skal undersøges, før man kan konkludere, om der er et match.

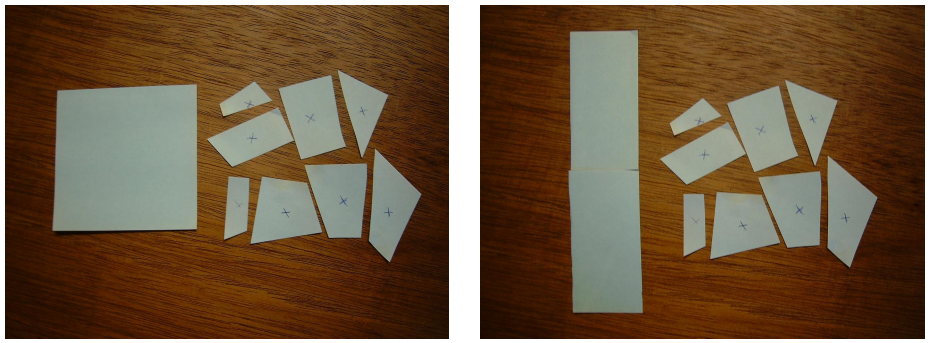
Man finder midtpunktet på linjen mellem to brikker, som muligvis passer sammen. Man finder så retningen på brikkerne, og til sidst undersøger man om alle reglerne for overlap er overholdt.

Når punkt 1 til 3 er opfyldt, kan der stadig være brikker, som kan ligge flere steder, så puslespillet kan samles på flere måder.

Nogle forkerte løsninger kan udelukkes ved at gætte på puslespillets areal. Når der fastsættes et areal, kan man observere, om nogle af brikkerne rager udenfor.

I nogle tilfælde fungerer denne metode dårligt, da der ikke kan fastsættes et areal før løsning af puslespillet. Det areal, som algoritmen kommer frem til, kan muligvis ikke indeholde nogen løsninger. I sådanne tilfælde må der udføres tests med forskellige arealformer.

Figur 3.8 illustrerer problemet med at gætte puslespillets areal.



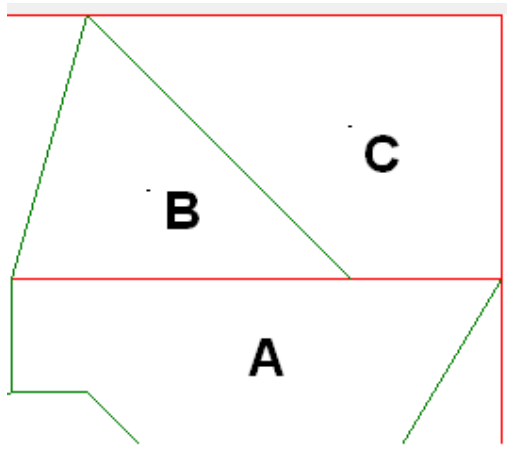
Figur 3.8: Til venstre ses det usamlede pulsespil med arealets rigtige form. Til højre har arealet en forkert form og puslespillet kan derfor ikke samles rigtigt, da de fleste at brikkerne ikke vil kunne lægges indenfor arealet

3.2 Søgning med sammensatte sider

En brik kan have to nabobrikker til samme side, fordi der er forskel på sidelængderne. Der vil også være tilfælde, hvor sidelængden på en brik er delt op i to stykker. Fænomenet optræder mest ved indscannede brikker, hvor der er stor sandsynlighed for, at opmålspunkterne ikke ligger det samme sted på de to nabobrikker. Det sker sjældent i visuelle geometriske puslespil.

Metoden fra artikel [5] omhandler tilfælde, hvor flere brikker støder op til samme side af en given brik. Problemet løses ved at sammenligne retninger. På den måde bliver det lettere at machte to brikker, selvom den ene brik har en mindre sidelængde. I tilfælde, hvor der kan være sammensatte sider skal algoritmen kende hjørnepunkternes koordinater og sidelængden. Ved hjælp af disse kan algoritmen beregne retningen på den brik, som skal være en del af den sammensatte side.

Metoden gør det mere sikkert at lave et match, hvor flere nabobrikker støder op til samme side. Mange brikker kan have en sidelængde, som passer til den længste side. Risikoen for fejl øges i og med antallet af mulige match stiger eksplosivt ved denne udvidelse.



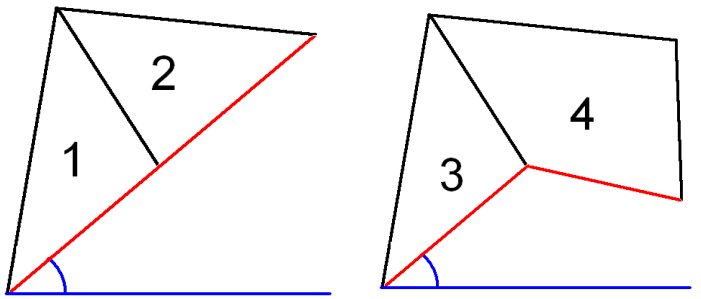
Figur 3.9: Problematikken ved at to brikker støder op til den samme flade.

3.2.1 Søgning med liste

Man kan holde styr på, hvilke brikker der til sammen kan danne en ny side, som muligvis passer til sidelængden på en tredje side, ved at undersøge alle brikkerne fra begyndelsen, så dem, der kan ligge ved siden af hinanden, danner en ny sidelængde. Informationen gemmes så i en liste, som der søges i, når der er sandsynlighed for, at man har en sammensat side.

Søgning efter sammensatte sider skal udføres før algoritmen påbegynder løsning af puslespillet, så algoritmen ikke skal søge efter de sammensatte sider undervejs i processen.

Det vil være fordelagtigt, at søgeprocessen samtidig undersøger, om brikkerne kan ligge ved siden af hinanden. På den måde undgår man, at der kommer for mange nye, sammensatte sider, som ikke kan bruges. For at man kan tale om en sammensat side, skal alle de sider, der indgår i en ny sammensat side, have samme vinkel, så de danner en ret linje. Dette er illustreret i figur 3.10.



Figur 3.10: Til venstre ses to brikker som kan bruges til sammensat side. De to brikker til højre har forskellige vinkler og kan derfor ikke anvendes til en sammensat side

Det vil være en fordel for algoritmen, at der kan lægges flere brikker på en gang. Hvis algoritmen finder et match, hvor den matchende brik er led i en sammensat side, kan algoritmen med det samme lægge de nabobrikker, som er med i den sammensatte side. Ved denne proces vil løsninger på nogle puslespil gå hurtigere, dog vil søgning og sortering af de sammensatte sider være tidskrævende.

Der benyttes metoder, som øger hastigheden på sortering og søgning i listen. Det kan være f.eks. "Quicksort"og "Binary Search Tree". Metoderne bliver beskrevet i Kapitel 4.

Ulempen ved metoden er, at det vil være begrænset, hvor stort et antal sider, der kan indgå i en sammensat side. Puslespillet skal ikke bestå af ret mange brikker, før man risikerer, at resultaterne i listen over sammensatte sider bliver meget lang.

Eksempel: Et puslespil indeholder 100 brikker, og alle brikker har 5 sider. Der tillades 4 sider til at indgå i en sammensat side.
Antallet af løsninger vil så være:

$$\prod_{n=1}^{4} 5 \cdot [100 - (n-1)] = 5,8818 \cdot 10^{10}$$

Dog vil mange løsningsforslag ikke kunne indgå i det endelige resultat. Mange vil falde bort på gund af overlap, eller fordi der ikke kan dannes en sammensat side med samme vinkel. Der vil sandsynligvis fremkomme nogle tusind anvendelige resultater. Søgeprocessen vil alligevel bruge meget tid på at undersøge alle brikkernes mulighed for at være del af en sammensat side, hvilket gør processen meget langsom.

3.2.2 Søgning uden liste

For at undgå, at der i begyndelsen bliver foretaget en søgning på alle mulige sammensatte sider, kan søgningen flyttes ind i algoritmen, så der føres løbende kontrol med de sammensatte sider. Dog vil dette give algoritmen mere at arbejde med, når der er fundet sider, som kan være en del af sammensatte sider, og der således skal findes et match.

Algoritmen kan lave en sammensat side, når der kan være en side fra en anden brik indenfor samme afstand. Den brik, som skal være en del af den sammensatte side, skal have samme vinkel som den valgte nabobrik. Når der er fundet en brik til den nye sammensatte side, skal algoritmen kunne huske alle de brikker, som indgår i den sammensatte side for at undgå, at man bruger de forkerte brikker.

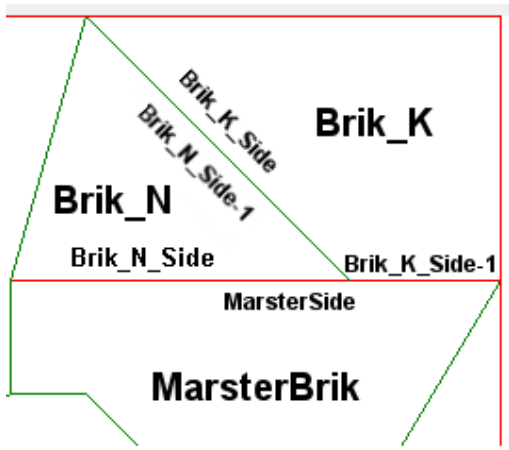
At lægge søgningen efter sammensatte sider samtidig med at puslespillet bliver lagt betyder, at mange brikker og sider på brikker kan udlukkes. Algoritmen husker, hvilke brikker der er lagt på plads i puslespillet samt de sider på brikker, der er blevet matchet. Det betyder, at algoritmen får færre brikker at vælge imellem, efterhånden som flere brikker bliver lagt.

Metoden tager ikke højde for, at brikkerne kan være roterede. Rotation kan komplicere søgningen efter sammensatte sider, da alle brikkerne skal roteres, når de undersøges. Idet man roterer en brik, kan alle sider potentielt indgå i en sammensat side. Metoden fungerer bedst, når brikkerne ikke roteres. Alternativt er den et godt supplement, hvis den benyttes til sidst, når algoritmen har lagt flest mulige brikker. Derudover har de resterende brikker sider, som kun kan bruges til få ting. Siderne kan f.eks. være kanter af puslespillet, eller de kan være dele af sammensatte sider. De få muligheder gør det nemt for metoden at finde frem til de sammensatte sider og få dem lagt rigtigt.

For at forklare, hvordan metoden virker, er der herunder en beskrivelse i form af pseudo-kode, som viser, hvordan algoritmen bør arbejde med sammensatte sider. figur 3.11 viser, hvilke sider og brikker der refereres til i pseudokoden.

```
N := 0;
find Brik_N;
Brik_N_Side = en side på Brik_N;

label:Start;

if Side < MarsterSide then
begin
  if Brik_N = Uset then
  begin
    if Brik_N_Side_Vinkel = MarsterVinkel then
    begin
      if Brik_N_Side_EndPoint = MarsterStartPoint then
      begin
        if goto:Start then
        begin
          Sæt Brik_N_Side deaktiv;
          return true;
        end;
```

```
        end;
        else
        begin
          return false;
        end;
      end;
      else
      begin
        return false;
      end;
    end;
    else
    begin
      ligge SideEndPoint til MarsterStartPoint
      if Brik_N ikke overlapper MarsterBrik then
      begin
        list_brik = brikker som kan ligge op til (
            Brik_N_Side - 1);
        label:L1;
        Brik_K = list_brik;
        Brik_K sat til Uset;
        if goto:Start (Brik_K_Side - 1) then
        begin
          Brik_K_Side deaktiver;
          return true;
        end;
        Brik_K sat til NotUset;
        if flere brikker i list_brik then
        begin
          næste brik i list_brik;
          goto:L1
        end;
      end;
    end;
  end;
  return false;
```



Figur 3.11: Figuren viser brikkerne og de sider, som indgår i pseudo-koden.

Brikkernes siderlængder opmåles i urets retning, hvilket vil sige, at alle hjørner på den enkelte brik er opmålt i rækkefølge fra midten af polygonet og med uret rundt. Man ved så, at nabobrikkens side -1 (Brik_K_Side-1), skal have den samme vinkel som Brik_N_Side. Hvis de har den samme vinkel, kan de danne en sammensat side.

Da denne metode kun virker optimalt uden rotation af brikker, kan det være en fordel at implementere den sammen med den metode, som lægger brikkerne. Det vil medføre, at metoden til de sammensatte sider ikke skal søge for at finde og lægge brikker. Det vil også medføre, at metoden kan finde sammensatte sider, som i forvejen er dannet af sammensatte sider.

3.3 Søgning med roterede brikker

Algoritmen kan ikke altid kan gå ud fra, at alle brikkerne ligger, som de skal fra begyndelsen. Nogle af brikkerne skal roteres for at kunne lægges korrekt. Nedenstående afsnit vil omhandle de metoder, algoritmen kan få brug for til at tage højde for eventuelle roterede brikker.

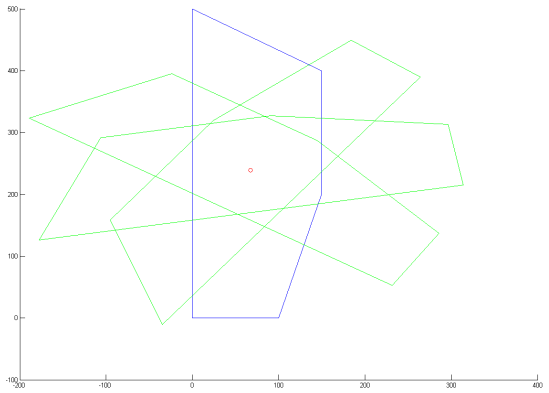
3.3.1 Match for hver grad

En ubesværet måde at løse problemet med rotation på er at rotere brikken og således kontrollere, om den kan passe på nogle leder. Alle brikker skal således roteres for at undersøge, om de kan være nabobrikker.

Når brikker roteres skal det ske i intervaller, så der ikke kommer for mange unødige trin. Ved for mange trin risikerer man, at der kommer for mange søgninger for hver brik.

Rotation vil kun ske på de brikker, der har en sidelængde, som vil kunne matche. Hver gang brikken bliver roteret, skal det kontrolles, om den kan ligge som den nye nabobrik. Hvis ikke, fortsætter algoritmen med at rotere den næste brik, der kan matche sidelængden.

På figur 3.12 ses et polygon, som er roteret tre gange. Det ses tydeligt, at der er mange vinkler, som ikke er med, så i dette tilfælde, hvor springet er på 0.75π, kan den rigtige brik let blive forbigået til fordel for en forkert.



Figur 3.12: Viser et polygon, der er roteret tre gange med 0.75π

Der findes nogle simple metoder til at rotere et 2D polygon. En af disse metoder er opskrevet herunder[3]. Polygonet P er bygget op af en række koordinater og har sit midtpunkt $Cen = (x, y)$. Den vinkel, som P skal roteres med bestemmes ved δ. Index på P viser, hvilken koordinat P roteres omkring. Denne centerkooordinat betegnes i.

$$Pn(i)_x = Cen_x + (cos(\delta) * (P(i)_x - Cen_x) - sin(\delta) * (P(i)_y - Cen_y)) \quad (3.4)$$

$$Pn(i)_y = Cen_y + (sin(\delta) * (P(i)_x - Cen_x) + cos(\delta) * (P(i)_y - Cen_y)) \quad (3.5)$$

Ved brug af denne metode vil processen blive meget langsom i større puslespil. Algoritmen vil være nødsaget til at rotere alle brikker, der er med i et match. For at finde en løsning skal rotationen foregå i mange små trin.

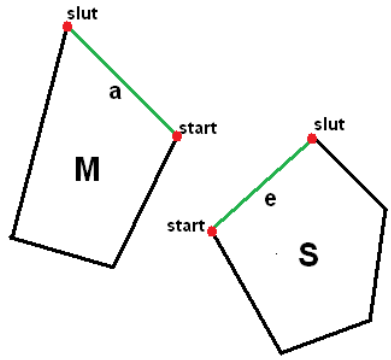
[3]http://vb-helper.com/howto_rotate_polygon_points.html

3.3.2 Match uden rotation

Hvis rotationsmetoden skal gøres hurtigere, kan det være en fordel kun at rotere brikker, når det er nødvendigt. Ved denne metode vil en brik kun blive roteret en gang, når algoritmen kontrollerer om den kan være nabobrik.

På figur 3.13 ses to polygoner: M og S. De har en sidelængde, der passer sammen. M er udgangsbrik, så det er S, der skal roteres. S bliver roteret, men det er ikke sikkert, den passer til M.

Fordelen ved denne metode er, at algorimen finder ud af, hvad vinklen på brik M's side a er. Vinklen kan bruges til at bestemme, hvor mange grader S skal roteres, så algoritmen kan finde ud af, hvad vinklen på brik S's side e er. Når algoritmen kender de to vinkler, kan den beregne, hvor stor vinkelforskel, der er mellem brik M's side a og brik S's side e. Forskellen kan bruges til at beregne brik S nye koordinater. Beregningen udføres ved hjælp af Ligning 3.4 og Ligning 3.5



Figur 3.13: To brikker der er roteret i forhold til hinanden.

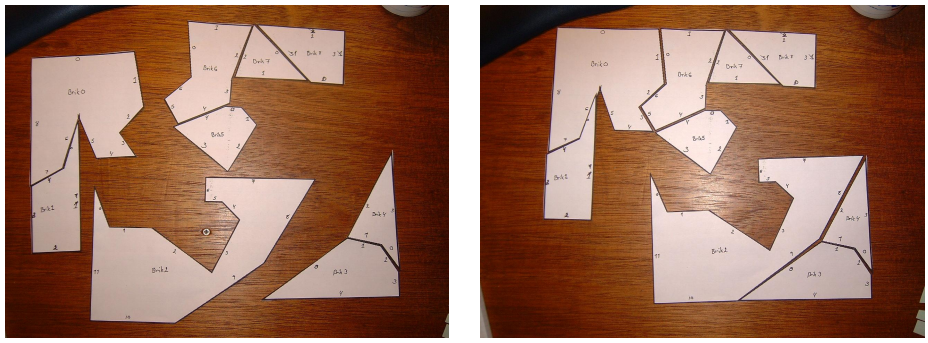
Før algoritmen kan finde den rigtige vinkel på M's side a skal a's startpunktet flyttes til $(0, 0)$. Derefter kan algoritmen beregne vinklen ud fra ligning 3.6. Idet a's startpunkt er flyttet til $(0, 0)$, er det kun slutpunktet, som skal bruges i ligning 3.6. Bemærk at hvis Sp_y er negativ, skal vinkel δ trækkes fra 2π.

$$\delta = \arccos\left(\frac{Sp_x}{\sqrt{{Sp_x}^2 + {Sp_y}^2}}\right) \tag{3.6}$$

Når vinklen på M's side a er udregnet, skal samme proces gennemføres for S's side e. Nu er det muligt for algoritmen at beregne forskellen mellem de to vinkler og finde ud af, hvor mange grader brik S skal roteres.

3.4 Søgning med Cocktailmetode

Under projektforløbet udvikles en alternativ søgemetode, som vil blive præsenteret i dette afsnit. Cocktailmetoden, som den er navngivet, er en kombination af de ovenfor beskrevne metoder. Der opstår hurtigt komplekse programmer, når de andre metoder anvendes. Hvis en brik f.eks. bliver lagt forkert i begyndelsen, tvinges algoritmen til at lægge næsten alle brikkerne igen, selvom mange af dem ligger rigtigt. I den forbindelse kan Cocktailmetoden være en stor hjælp. Ideen er, at algoritmen først finder de brikker, som kan lægges sammen – det vil sige sider, der har samme og unikke længde. Med andre ord brikkerne kan parres entydigt. Metoden er illustreret på figur 3.14



Figur 3.14: Billederne illustrerer Cocktail metodens fremgangsmåde.

Fordelen ved Cocktailmetoden er, at der er færre men større brikker, der skal lægges og det betyder, at der er mere tid tilovers til at teste, om brikkerne ligger rigtigt. Ud over at der kommer færre brikker, kan de sammensatte sider forsvinde ved at benytte Cocktailmetoden. Dette skyldes, at når brikkerne bliver smeltet sammen, kan algoritmen erstatte alle de sider, som ligger i forlængelse af hinanden – og har samme vinkel – med en ny side, der har sammme længde som disse sidelængder til sammen. På den måde bliver brikkerne meget store. Herved vil metoden også bedre kunne håndtere sammensatte sider.

Det samme gælder for roterede brikker. Fordi metoden gør det lettere at finde løsninger på sammensatte sider, betyder det ikke noget, at de brikker, der skal bruges til at danne de sammensatte sider, er roterede. Når mange af de sammensatte sider forsvinder, er det ligegyldigt, hvordan brikkerne ligger.

Samtidig med at Cocktailmetoden finder de sider, der kan danne par og smelte dem sammen, tester metoden også, om de to brikker har andre sider til fælles. Herved forsvinder mange af siderne, så algoritmen ikke har så meget at arbejde med.

Hvordan Cocktailmetoden arbejder beskrives herunder i form af pseudo-kode.

```
label:start;
if goto:SamSmalt = true then
begin
  Brik_N = næste brik;
  Brik_N_Side_M = første side på Brik_N som X antal sider;
  goto:start;
end;

goto:slut;

label:SamSmalt;

list_Side = Sider som der passer til Brik_N_Side_M;
if list_Side kun indholde en then
begin
  Aktiver de to Brikker som brugt;
  Se om de brikker har flere sider til fælles;
  Danne en ny brik af Brik_N og list_Side_Brik
  return true;
end;
```

```
else
begin
  Brik_N_Side_M = næste side på Brik_N som X antal sider;
  if der er flere side på Brik_N then
  begin
    goto:SamSmalt;
  end;
  return false;
end;

label:slut;
```

Når Cocktailmetoden er færdig med at samle de brikker, der er unikke, overtager backtracking. Backtracking prøver at samle alle de resterende brikker til den rigtig form.

Backtracking benytter de samme relger, som findes i Cocktailmetoden – dog accepterer backtrackingen, at der er flere nabosider til en side. Når backtracking er kommet frem til en løsning, gemmer den løsningen som en billedfil, som brugeren af systemet kan anvende til at se, om løsningen er rigtig. Ved at lade backtracking undersøge om alle brikker er blevet lagt, kontrollerer man, at den er kommet frem til en løsning. Idet der i så fald ikke er flere brikker, som kan algoritmen kan lægge, må der være en løsning.

3.5 Afslutning

Som sammenfatning på dette kapitel vil der blive reflekteret over, hvilke ting, der kan arbejdes videre med. Ved projektets begyndelse blev der arbejdet med en udgangsbrik, som voksede efterhånden som nabobrikkerne blev smeltet sammen med den. Ideen var god nok, men det viste sig at være problematisk at fjerne de brikker, der blev lagt forkert.

Derfor blev der arbejdet med rekursivfunktionen, som lettede arbejdet med at fjerne forkert placerede brikker.

Det er blevet diskuteret, hvordan man kan opdage fejlplaceringer af brikker. En metode går ud på, at algoritmen beregner puslespillets areal ud fra det samlede areal af brikkerne. Problemet med denne metode er, at det vil være besværligt at finde arealets form. Det vil registreres som en fejl, når algoritmen placerer brikker udenfor det beregnede areal, og disse løsninger vil blive slettet. Man risikerer at slette den korrekte løsning på puslespillet, fordi formen ikke passer ind i det beregnede areal. Metoden er ufordelagtig, da der skal udføres for mange tests, hvis algoritmen skal gætte formen på arealet.

Overlap-metoden har visse mangler. I tilfælde af, at en lille brik næsten er omkranset af en stor brik, vil der registreres fejl på nogle af siderne. Da centrum på de to brikker ligger meget tæt på hinanden, kan de synes at have samme retning ifølge kompasmetoden. Denne fejl bliver der set bort fra i begyndelsen, da det ikke er ret tit, at en stor brik omslutter en lille brik.

Rotation af brikker er en vigtig del af algoritmen. Algoritmen kan ikke regne med, at alle brikker vender rigtigt, hvis de er scannet ind. Valg af metode til at håndtere dette problem er faldet på "Match uden rotation". Denne løsning er hurtigere end den metode, hvor der testes for match for hver grad. Derudover vil det være uproblematisk at implemtere metoden sammen med den kode, som skal flytte brikkerne.

Kapitel 4

Optimering af Søgemetode

Optimering prioriteres højt i udviklingen af algoritmen, så den kan lægge et puslespil hurtigt. I det følgende vil de søgetræer, der skal indeholde information om brikkernes sidelængder, blive undersøgt. Der vil blive set på datastuktur og på de søgmetoder, som skal søge efter match mellem brikkerne.

4.1 Datastruktur

Datastukturen er en vigtig del af algoritmen. Det er vigtigt at vide, hvordan data skal gemmes, og hvordan der nemt kan søges på dem igen.

Den stuktur, der arbejdes med, har i begyndelsen af projektet en helt simpel liste, som ikke er optimeret. I stedet fokuseres der på sammenligning af sidelængderne og dermed på selve løsningen af puslespillet. Ulempen ved at bruge en liste, hvor alle felter skal kigges igennem for at finde den ønskede sidelængde, er, at det hurtigt bliver tidskrævende grundet store mængder data i listen. Det er derfor en god ide at udarbejde 'metoder, som kan gøre søgningen hurtigere.

Til dette projekt er anvendelsen af et „binary search tree“ en udmærket metode til optimering af datastukturen, da det øger søgehastigheden. Virkemåden af et „binary search tree“ kan justeres, så det passer ind i dette projekt.

Indlæst data gemmes i et objekt, som indeholder alle oplysninger om hver en-

kelt brik. Ud over objektet skal der være en liste, der indeholder sidelængderne på alle brikker. Længderne bliver gemt i en struktur, som indeholder information om længde samt om, hvilken brik og side længden hører til.

4.1.1 Sorteringsmetode

Ved indlæsning vil alle sidelængder lagres i en liste. Denne liste skal sorteres, før den kan anvendes af algortimen. Til sortering findes et stort udvalg af forskellige metoder. Den her anvendte metode ligger tæt op ad "Mergesort", fordi metoden har relation til "Quicksort"familien. "Mergesort-metoden er blot hurtigere end den normale "Quicksort"(ifølge Århus Universitet[1]).

4.1.2 Søgemetode

Når listen er sorteret, kan algoritmen arbejde med den. Algoritmen søger i listen efter de sidelængder, den skal bruge, ved hjælp af en form for „binary search tree“. Samtlige sidelængder listes med den korteste først og derefter ud fra voksende længde. I første omgang undersøger algoritmen, om den pågældende sidelængde er længere eller kortere end medianen i listen. Hvis længden er kortere end medianen, gennemføres en søgmimg på de sidelængder, som ligger indenfor intervallet fra den korteste sidelængde til medianen. Princippet er det samme, hvis sidelængden viser sig at være længere end medianen.

Tabel 4.1 illustrerer, hvordan metoden virker, når der skal findes et match. Metoden leder efter tallet 7 i listen. Først sammenlignes tallet 7 med rækkens median, som er index 4 og har værdien 9. Da 7 er mindre end 9 foretages anden søgning på elementerne fra index 0 til 4, som indeholder værdier lavere end 9. Medianen i rækken fra index 0 til 4, hvor 4 ikke er inkluderet, bliver 1,5. Midtpunktet bliver index 1, da man i sådanne tilfælde altid runder ned. Index 1 indeholder tallet 3, og da det er mindre end 7, foretages næste søgning på index 1 til 4, som skal testes på for at finde værdien 7. I tredje tilfælde bliver værdien 4 og til sidst 7. Herved bliver løsningen på søgningen index 3.

[1] http://www.daimi.au.dk/dADS1/sortering/index.html

Operator/Index	0	1	2	3	4	5	6	7	8
$<$	1	3	4	7	9	15	20	25	30
$>$	1	3	4	7	9	15	20	25	30
$>$	1	3	4	7	9	15	20	25	30
$=$	1	3	4	7	9	15	20	25	30

Tabel 4.1: Eksempel på at man ved hjælp af søgemetoden kan finde tallet 7 i listen. Resultatet bliver index = 3

I de fleste tilfælde vil der minimum være to sider med samme længde. Der vil således ofte være flere match i en søgning, og det skal metoden kunne tage højde for. For hvert match skal det undersøges, om dette kan være en del af søgningen. Hvis en eller begge naboer matcher kontrolleres deres naboer for, om de passer ind i sammenhængen.

I Tabel 4.2 ses et eksempel på, hvordan metoden finder frem til de steder, som har værdien 3. Ligesom eksemplet fra Tabel 4.1 påbegyndes søgningen fra midten af listen ved tallet 7. Da 7 er større end 3, skal der arbejdes videre med de værdier, som ligger til venstre. Medianen for indexerne mellem 0 og 3. Da der tale om heltalsdivision lander metoden på index 1. Dermed har metoden fundet frem til den efterspurgte værdi. Metoden søger dernæst efter flere tretaller på venstre side af index 1. Men da værdien på venstre side er 1, foretages en søgning på højre side af matchet i stedet for. Til højre, i index 2 findes værdien 3, så derfor fortsætter metoden søgningen mod højre. Søgningen stopper ved index 3, da den har værdien 4. De index, der skal returneres, er altså index 1 og 2.

Operator/Index	0	1	2	3	4	5	6	7	8
$<$	1	3	3	4	7	15	20	25	30
$=$	1	3	3	4	7	15	20	25	30
$!=$	1	3	3	4	7	15	20	25	30
$=$	1	3	3	4	7	15	20	25	30
$!=$	1	3	3	4	7	15	20	25	30

Tabel 4.2: Søgemetoden kan finde alle tretallerne i listen, resultatet bliver index = 1 og 2

4.2 Afslutning

Da mange testpuslespil består af mindre end 20 brikker, har anvendelsen af "binary search tree"ikke nogen bemærkelsesværdig effekt på hastigheden. Hvis der imidlertid udføres tests med 100 brikker eller flere i puslespillet, vil "binary search tree"sætte sit præg på søgehastigheden.

Cocktailmetoden vil drage stor fordel af at anvende et "binary search tree", da metoden til at begynde med søger efter sider, der kun findes to af. I stedet for at søge på 100 elmenter i en liste kan søgetiden mindst halveres med "binary search tree".

Kapitel 5

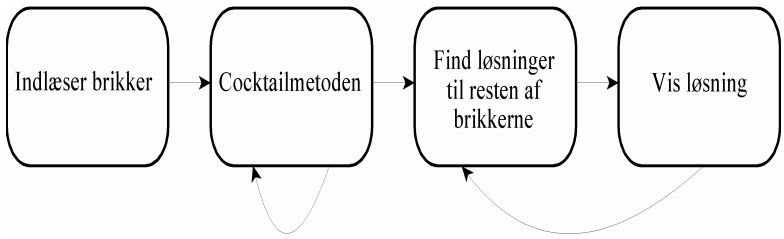
Implementering

Dette kapitel indeholder en kort gennemgang af produktet (algoritmen) i dets nuværende form. Kapitlet er opdelt i to afsnit: Et som beskriver algoritmen uden optimering og et, hvor den er optimeret. I begge afsnit vil klassediagrammer og funktionsbeskrivelse blive gennemgået.

5.1 Beskrivelse af algoritmen

I kraft af de metoder, der blev gennemgået i kapitel 3, er der skabt en funktionel algoritme. Algoritmen bygger på Cocktailmetoden, der skal fungere som drivkraft i algoritmen. Denne metode er valgt, fordi den forsimpler alle puslespil (jf. afsnit 3.4).

Blokdiagrammet på figur 5.1 illustrerer, hvordan algoritmen skal virke.
Den første blok er indlæsningerne. Ved indlæsningen beregner algoritmen areal og centrum på alle brikker og gemmer udregningerne i hver deres objekt. Derefter skønner Cocktailmetoden, hvor den muligvis kan finde brikker, som passer sammen. Hvis Cocktailmetoden finder et match mellem brikker, kører den videre. Når den ikke kan finde flere brikker, begynder algoritmen at lede efter løsninger til de brikker, der er tilbage.
Algoritmen gemmer et billede af de løsninger, den kommer frem til. Det betyder, at de brikker, der er tilbage efter at Cocktailmetoden har kørt, kan samles på flere forskellige måder.

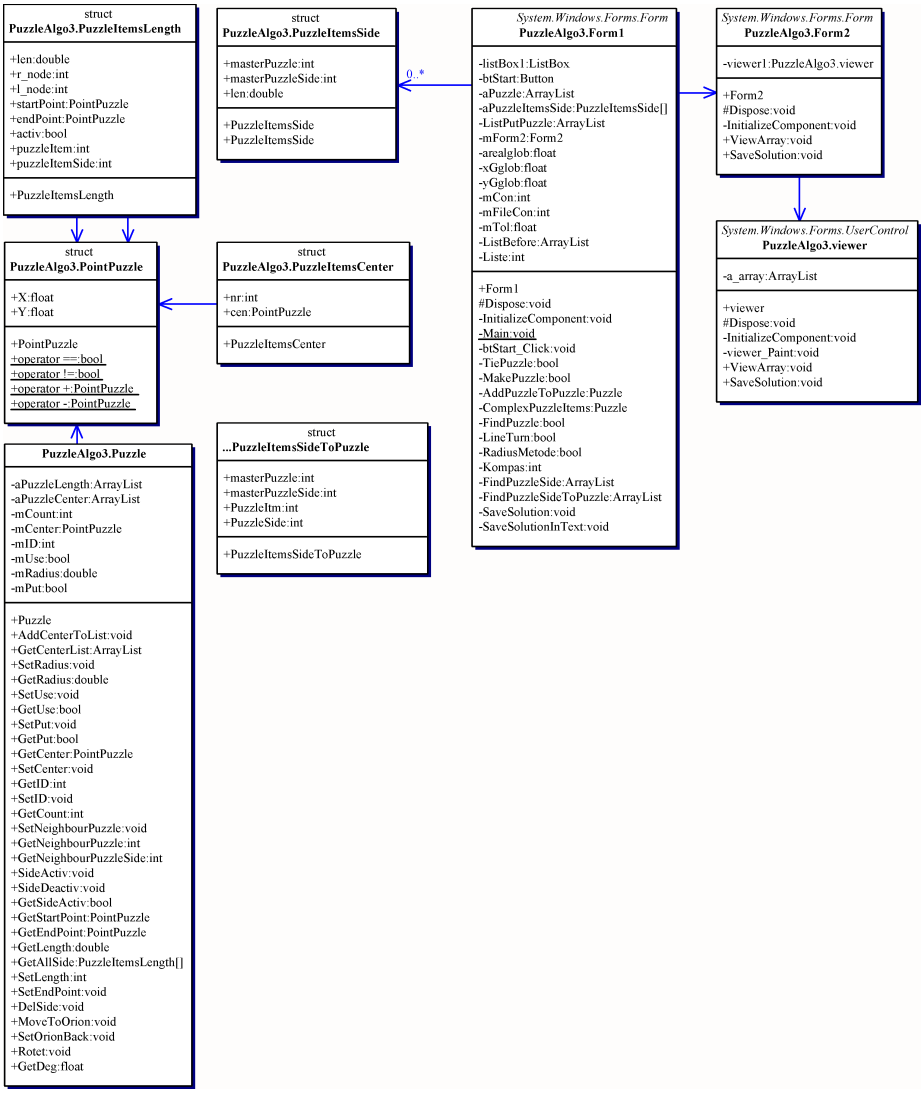


Figur 5.1: Blokdiagram over processen i systemet

5.2 Implementering af søgning

I dette afsnit skal der ses nærmere på klassebeskrivelse af algorimen ved implementering af simpel søgning. De forskellige funktioner og attributters anvendelser gennemgås og forklares.

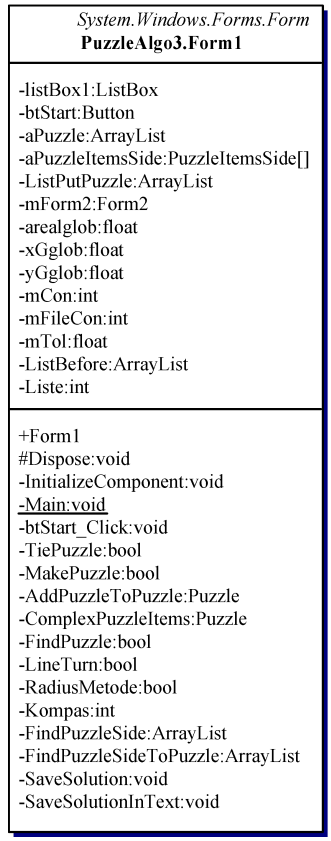
Figur 5.2 afbilleder klasseoversigten over algoritmen. De af klasserne, der er vist, vil ikke blive beskrevet i detaljer. Klasserne "Form1"og "Puzzle"gennemgås grundigt.



Figur 5.2: Klassediagram over algoritmen

Klasse Form1

Form1-klassen er hovedklassen i programmet. Den sørger for, at der bliver oprettet en liste over objekter i "puzzle-klassen". Denne liste, figur 5.3, indeholder alle de brikker, der indgår i puslespillet.



Figur 5.3: Klasse Form1

Navn	Type	Forklaring
listBox1	ListBox	listBox1 bruges til at skrive information ud til brugen af algoritmen
btStart	Button	Startknappen til at starte det hele
aPuzzle	ArrayList	Listen som indeholder alle brikkerne
aPuzzleItemsSide	ArrayList	Liste over sider, som er aktive
ListPutPuzzle	ArrayList	Listen over brikker, som er blive lagt
mForm2	Form2	Vinduet, som viser resultaterne
arealglob	float	Indeholder brikkernes samlede areal
xGglob	float	Bruges til at beregne centrum på brikkerne
yGglob	float	Bruges til at beregne centrum på brikkerne
mCon	int	Bruges til at beregne centrum på brikkerne
mFilCon	int	Nummer på filer, som indeholde løsningen
mTol	float	Tolerancen til samlingerne
ListeBefore	ArrayList	Indeholder tidligere rækkefølger af brikker, som er lagt
Liste	int	Et nummer, som fortæller rækkefølgen af brikker, som er lagt

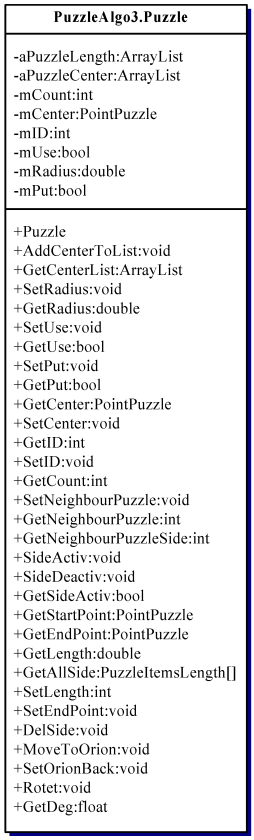
Tabel 5.1: Attributtabel

Navn	Retur type	Forklaring
btStart_Click	void	Brikkerne indlæses. Brikker, areal og centrum bliver fundet. Brikkerne bliver lagt i objeket af Puzzle. Når indlæsning er færdig, kaldes MakePuzzle ind til funktionen returnerer "false". Når det sker kaldes funktionen FindPuzzle
TiePuzzle	bool	Funktionen undersøger, om der er nogle unikke naboer til en given brik. Hvis der er en unik nabo, kaldes funktionen AddPuzzleToPuzzle.
MakePuzzke	bool	Kalder funktionen TiePuzzle angående de brikker, som ikke er blevet brugt, så der kan findes en unik nabobrik til dem.
AddPuzzleToPuzzle	Puzzle	Funktionen bliver kaldt, hvis der er to brikker, der skal smeltes sammen. Funktionen laver en ny brik af de to, som smeltes sammen, hvor de sider, de har til fælles, ikke er med.
ComplexPuzzleItem	Puzzle	Denne funktion bliver kaldt, når to brikker er smeltet sammen. Funktionen undersøger, om der er sider i den nye brik, som ligger lige efter hinanden med samme vinkel. Hvis siderne har samme vinkel, erstatter funktionen siderne med en ny og sletter de gamle
FindPuzzle	bool	Finder løsningsforslag til alle brikker, som kan lægges
LineTurn	bool	Funktionen undersøger, om der er overlap
RadiusMetode	bool	Funktionen undersøger, om brikkernes radier overlapper hinanden
Kompas	int	Funktionen undersøger, om to sider har samme retning
FindPuzzleSide	ArrayList	Funktionen finder sider med samme sidelængder
FindPuzzleSideToPuzzle	ArrayList	Funktionen finder alle de sider, som to brikker har til fælles
SaveSolution	void	Gemmer løsningen som billedfil (PNG)

Tabel 5.2: Funktionerne i Form1

Klasse Puzzle

"Puzzle"er det objekt, som brikkerne bliver lagret i. Det sørger for, at hovedklassen "Form1"kan finde alle oplysninger om samtlige brikker. Mange af funktionerne er Get/Set funktioner, så det er kun et mindre udvalg, som bliver kommenteret.



Figur 5.4: Klassen Puzzle

Navn	Type	Forklaring
aPuzzleLength	ArrayLise	Listen over alle brikkens sidelængder
aPuzzleCenter	ArrayLise	Listen over centre fra gamle brikker, der er smeltet sammen
mCount	int	Antallet af sider
mCenter	PointPuzzle	Midterkoordinater
mID	int	Id på brikken
mUse	bool	Om brikken er blevet brugt
mRadius	double	Radius på brikkens areal
mPut	bool	Fortæller om brikken er lagt

Tabel 5.3: Attributtabel

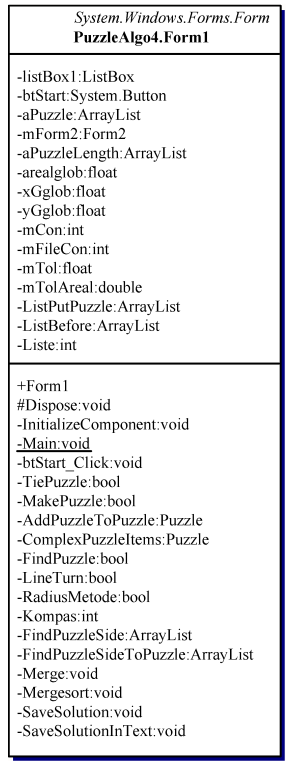
Navn	Retur type	Forklaring
GetStartPoint	PiontPuzzle	Returnerer startpunktet på en given side
GetEndPoint	PiontPuzzle	Returnerer slutpunktet på en given side
SetLength	int	Funktionen bruges til at tilføje siderne på brikken til objekterne og returnerer nummeret på siden
SetEndPoint	void	Funktionen bruges til at ændre slutpunktet på en side
DelSide	void	Slette en given side
MoveToOrion	void	Flytter hele brikken til et givet punkt
SetOrionBack	void	Flytter brikken tilbage til punktet (0,0)

Tabel 5.4: Funktionerne i Puzzle

5.3 Implementering af optimering

Dette afsnit omhandler implementering af den i kapitel 4 beskrevne optimering af søgemetoden i algoritmen. De forskellige funktioner og attributter vil ikke blive gennemgået på samme måde som i forrige afsnit. Dette skyldes, at alle funktioner og attributter er de samme, undtagen tre funktioner. Funktionen "FindPuzzleSide"vil blive uddybet nærmere i dette afsnit.

Klasse Form1, som den ser ud, når optimeringen af søgmetoderne er foretaget er afbilledet på figur 5.5.



Figur 5.5: Form1-klassen med optimeret søgemetode

5.3.1 Søgemetode

Når listen er sorteret, kan søgefunktionen søge i listen. Funktionen har navnet "FindPuzzleSide", og den medtager fire argumenter. Første argument er den sidelænge, der skal findes et macth til, og andet argument skal sørge for, at brikken ikke selv indgår i søgningen, så søgetræet ikke registrerer et match med brikken selv. Tre og fire er start- og slutværdien på listen. Herunder ses koden for funktionen "FindPuzzleSide".

```
private ArrayList FindPuzzleSide(double len, int NotPuzzle
    , int l, int r)
{
  int m = (l+r)/2;

  if (((PuzzleItemsSide)aPuzzleLength[m]).len >= len -
      mTol && ((PuzzleItemsSide)aPuzzleLength[m]).len <=
      len + mTol)
  {
    ArrayList tmpArr = new ArrayList();
    int tmpPuzzle = ((PuzzleItemsSide)aPuzzleLength[m]).
        masterPuzzle;
    int tmpPuzzleSide = ((PuzzleItemsSide)aPuzzleLength[m
        ]).masterPuzzleSide;

    if(tmpPuzzle != NotPuzzle && !((Puzzle)aPuzzle[
        tmpPuzzle]).GetUse())
        if (((Puzzle)aPuzzle[tmpPuzzle]).GetSideActiv(
            tmpPuzzleSide))
            tmpArr.Add(new PuzzleItemsSide(tmpPuzzle,
                tmpPuzzleSide));

    int tmpM = m - 1;
    while (tmpM >= 0 && ((PuzzleItemsSide)aPuzzleLength[
        tmpM]).len == len)
    {
        tmpPuzzle = ((PuzzleItemsSide)aPuzzleLength[tmpM])
            .masterPuzzle;
        tmpPuzzleSide = ((PuzzleItemsSide)aPuzzleLength[
            tmpM]).masterPuzzleSide;

        if (tmpPuzzle != NotPuzzle && !((Puzzle)aPuzzle[
            tmpPuzzle]).GetUse())
```

```
                if (((Puzzle)aPuzzle[tmpPuzzle]).GetSideActiv(
                    tmpPuzzleSide))
                    tmpArr.Add(new PuzzleItemsSide(tmpPuzzle,
                        tmpPuzzleSide));

            tmpM--;
        }

        tmpM = m + 1;
        while (tmpM < aPuzzleLength.Count && ((PuzzleItemsSide
            )aPuzzleLength[tmpM]).len == len)
        {
            tmpPuzzle = ((PuzzleItemsSide)aPuzzleLength[tmpM])
                .masterPuzzle;
            tmpPuzzleSide = ((PuzzleItemsSide)aPuzzleLength[
                tmpM]).masterPuzzleSide;

            if (tmpPuzzle != NotPuzzle && !((Puzzle)aPuzzle[
                tmpPuzzle]).GetUse())
                if (((Puzzle)aPuzzle[tmpPuzzle]).GetSideActiv(
                    tmpPuzzleSide))
                    tmpArr.Add(new PuzzleItemsSide(tmpPuzzle,
                        tmpPuzzleSide));

            tmpM++;
        }

        return tmpArr;
    }

    if (l == r || l > r)
        return new ArrayList();

    if (((PuzzleItemsSide)aPuzzleLength[m]).len > len)
        return FindPuzzleSide(len, NotPuzzle, l, m - 1);
    else
        return FindPuzzleSide(len, NotPuzzle, m+1, r);

}
```

5.4 Delkonklusion

Det har vist sig nødvendigt med en mindre udvidelse i optimeringsdelen. Problemet er, at hvis der kommer flere brikker til undervejs, skal listen sorteres igen. Dette problem kan betyde, at algoritmen kommer til at arbejde ligeså langsomt, som den ikke optimerede version. Dette skyldes Cocktailmetoden. Først finder Cocktailmetoden alle de brikker, som passer og smelter dem sammen til en brik. Denne ene brik skal tilføjes til listen over sidelængder, hvilket betyder, at listen skal sorteres igen.

Når Cocktailmetoden anvendes, kan det ske, at algoritmen samler to unikke brikker forkert. Dette sker, hvis to brikker skulle ligge hver for sig, men med sidelængder, der passer. Brikkerne bliver så smeltet sammen til en ny brik. Disse fejl kan opfanges af den menneskelige hjerne, fordi mennesket samtidig fokuserer på brikkernes motiver. Da algoritmen kun registrerer sidelængder, kan den ikke håndtere denne type fejl.

Kapitel 6

Test

De udførte tests fastsætter kravene for problemafgrænsningen. Ud over dette bliver der også udført en test, hvor der bliver fremprovokeret fejl. Herunder ses en række punkter, som der skal udføres tests på.

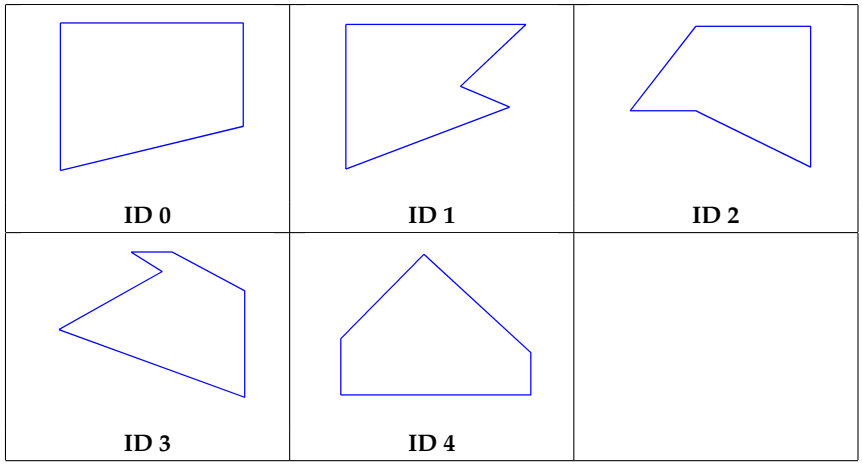
- Test med simpelt visuelt puslespil.
- Test med sammensatte sider.
- Test med roterede brikker.
- Test af hele algoritmen for at se, om metoderne kan samarbejde.

6.1 Testmiljø

Testene køres på testmiljøet, som består af helt almindeligt EDB-udstyr. Der vil ikke blive stillet store krav til udstyret. Dette testmiljø kører: PC Pentium M Centrino 1.73GHz, 2Gb Ram og med Windows XP Tablet.

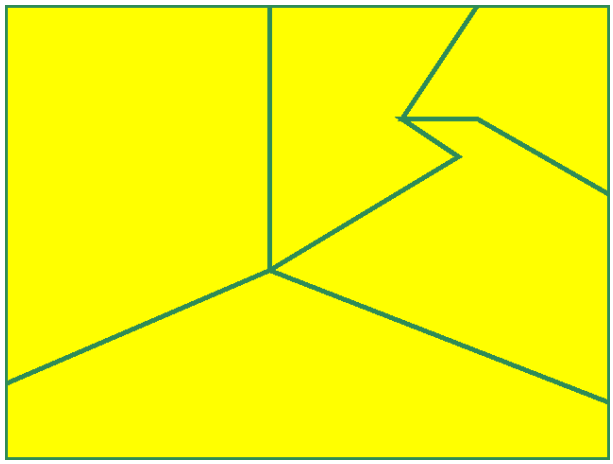
6.2 Simple puslespil

Ved testen med det simple puslespil er brikkerne lavet således, at der ikke er nogle af brikkerne, der har samme sidelængde. Herunder ses de brikker, som skal bruges til det simple puslespil.



Figur 6.1: Alle brikker, der bruges til det til den simple test. Brikkerne er skitserede i Matlab, hvorfor nogle af brikkerne er deformerede

Når alle brikkerne ligger rigtigt, skal de gerne danne det billede, som fremgår af figur 6.2.

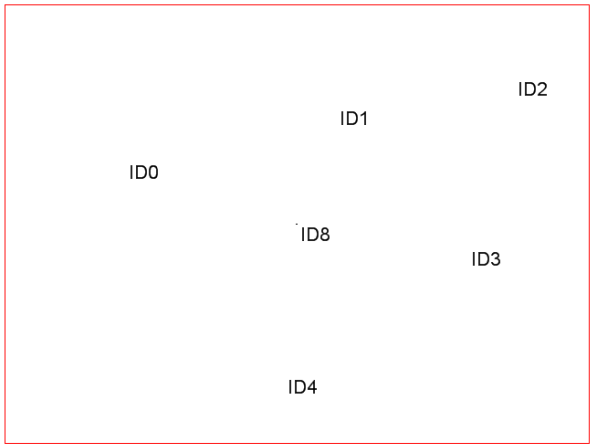


Figur 6.2: Det samlede puslespil

Den løsning, som algoritmen er kommet frem til efter at have arbejdet med brikkerne, kan ses på figur 6.3. Løsningen består af en stor brik. Hver brik har et ID, som fortæller, hvor brikken skal ligge inden for rammen. Brikkernes omkreds er ikke optegnet, fordi Cocktailmetoden fjerner linierne imellem brikkerne, når de bliver smeltet sammen.

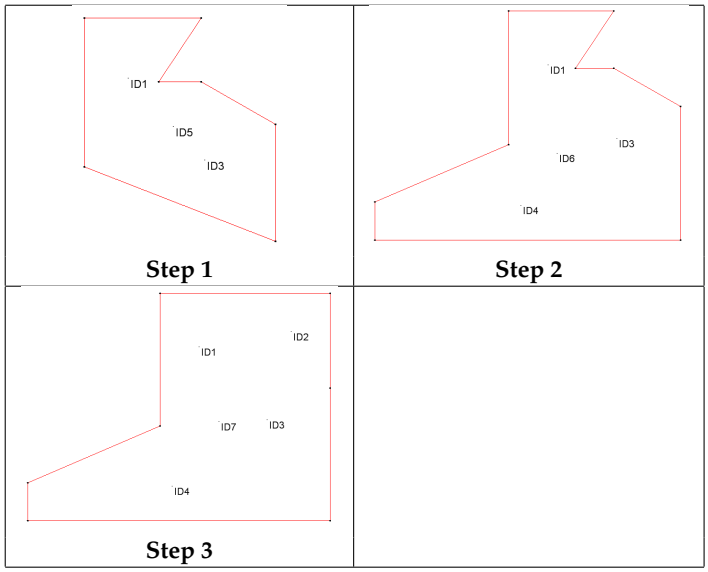
Funktionen "FindPuzzle"(backtracking) kommer ikke i brug, da der kun er én brik tilbage. Det skyldes, at Cocktailmetoden har fundet en nabo til alle brikkerne og endt op med en løsning. Det fremgår af tabel 6.1, at der kun har været et gennemløb af backtrackingfunktionen.

Resultaterne viser, at den optimerede version er langsommere end den normale algoritme. Dette kommer sig af, at listen med sidelængder skal sorteres, hver gang Cocktailmetoden danner en ny brik.



Figur 6.3: Det samlede puslespil

Herunder vises en billedserie, som illustrerer de trin, som Cocktailmetoden udfører undervejs, ind til den kommer frem til den endelig løsning. Billedserien starter med step 1, hvor Cocktailmetoden har fundet brik 1 og 3 og sammensmeltet dem til brik 5. Efter step 3 kommer Cocktailmetoden frem til den endelige løsning, der kan ses på figur 6.3.



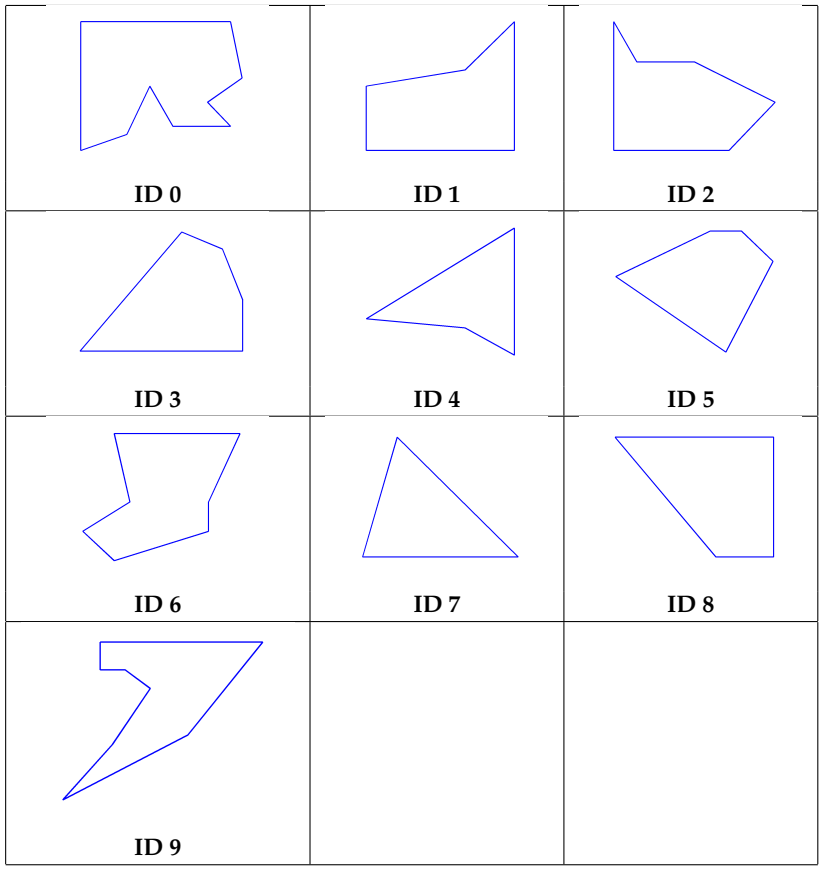
Figur 6.4: De enkelte trin, som Cocktailmetoden udfører

	Cocktailmetoden	Backtracking	Tid(ms)
Normal	20	1	93,75
Optimeret	22	1	109,38

Tabel 6.1: Tidsforbrug, gennemløb for de enkelte metoder

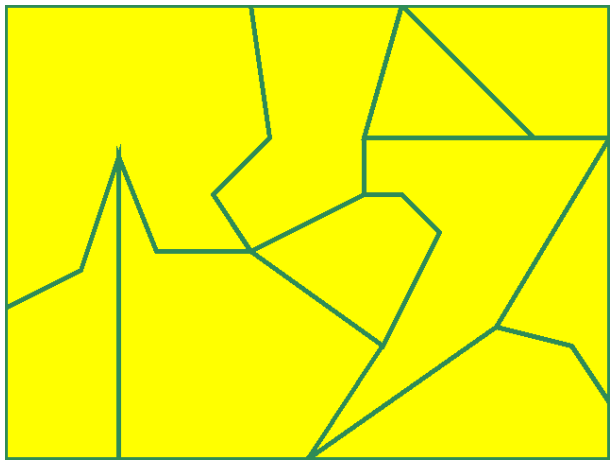
6.3 Sammensatte sider

Til at teste sammensatte sider anvendes et puslespil med 10 brikker. Der er kun to brikker, som danner en sammensat side (af brik 7 og 8). Ud over den sammensatte side findes sider, der er lige lange. Det drejer sig om sider på brik 0 og 9, 3 og 6 samt 3 og 9. Figur 6.5 viser de brikker, som skal bruges.



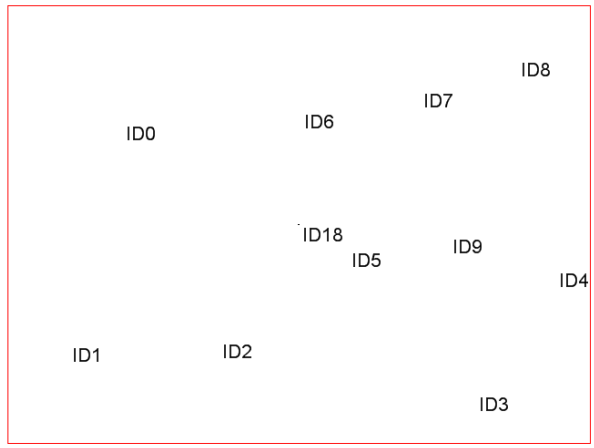
Figur 6.5: Alle brikker, som skal bruges til at teste sammensatte sider. Brikkerne er skitserede i Matlab, hvorfor nogle af brikkerne er deformerede

Det færdige resultat skulle gerne være som vist på Figur 6.6



Figur 6.6: Det samlet puslespil

Algoritmens løsning vises på figur 6.7. Løsningen består igen af en stor brik. Cocktailmetoden finder en nabo til alle brikkerne og ender op med en sammensmeltet brik. Det viser sig. at der er færre gennemløb for Cocktailmetoden i denne test end for test med simpelt puslespil, selvom dette puslespil indeholder dobbelt så mange brikker. Grunden er, at brikkerne i denne test er nummererede, hvilket gør det hurtigere for Cocktailmetoden at finde de unikke naboer. Resultatet ses i tabel 6.2. Den langsomme søgning med den optimerede algoritme skyldes den samme årsag som i testen med simpelt puslespil.

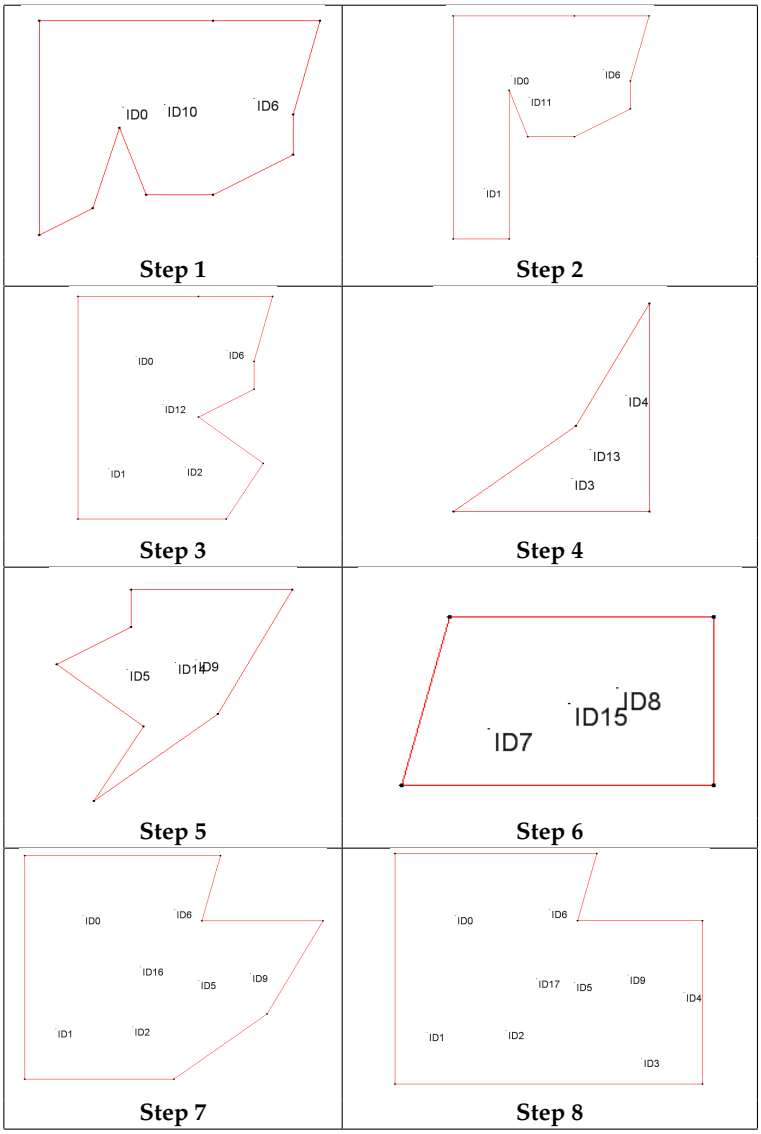


Figur 6.7: Det samlede puslespil

På figur 6.8 ses en billedserie, der viser trinnene, som Cocktailmetoden udfører undervejs, ind til den kommer frem til den endelige løsning.

	Cocktailmetoden	Backtracking	Tid(ms)
Normal	22	1	140,63
Optimeret	26	1	156,25

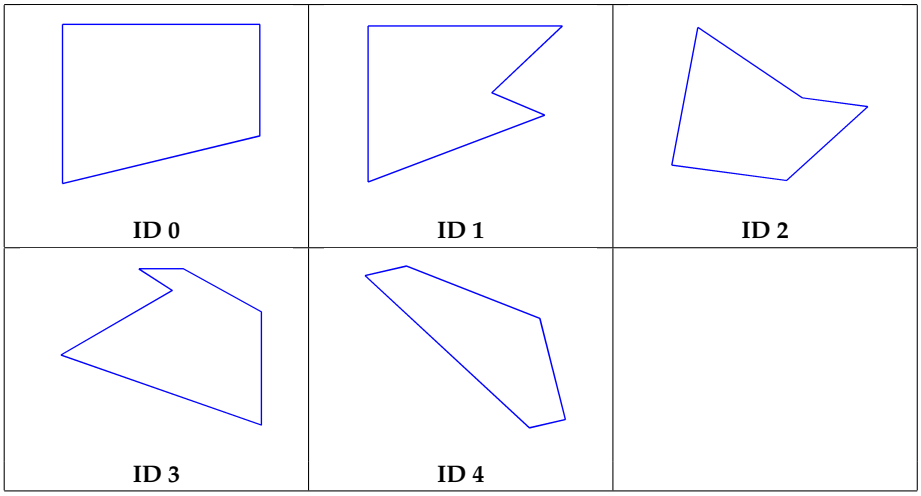
Tabel 6.2: Tidsforbrug, gennemløb for de enkelte metoder



Figur 6.8: De enkelte trin, som Cocktailmetoden udfører

6.4 Roterede brikker

Til testen med roterede brikker anvendes det samme puslespil som i første test. Dog er nogle af brikkerne roteret med tilfældige vinkler. Herunder ses de brikker, som skal bruges til det simple puslespil.

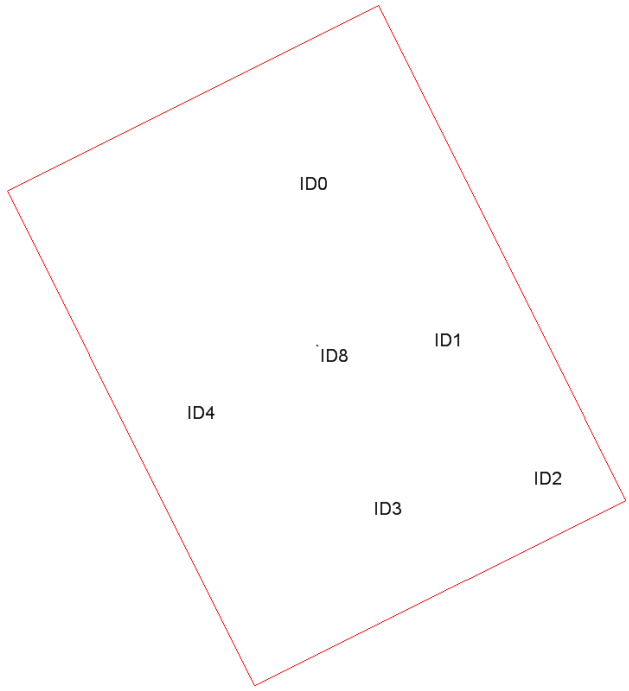


Figur 6.9: Alle brikker, der bruges til det til test med rotation. Brikkerne er skitserede i Matlab, hvorfor nogle af brikkerne er deformerede

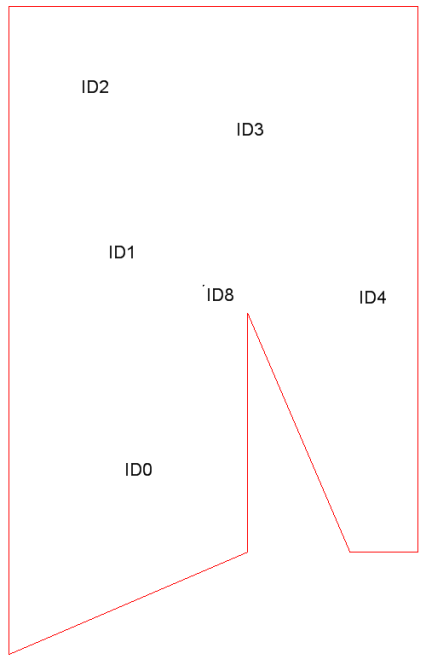
Når alle brikkerne er lagt rigtigt, skal de gerne danne det billede, som vises på figur 6.2.

Efter at algoritmen har arbejdet med brikkerne, er den kommet frem til én endelig løsning. Løsningen kan ses på figur 6.10. Løsningen ligner meget den fra testen med simpelt puslespil. Den eneste forskel er, at algoritmen har samlet puslespillet skævt, fordi brikkerne bliver roteret i forhold til masterbrikken. Hvis den sidste masterbrik ligger skævt, kommer alle brikkerne til at ligge med samme skævhed.

Anvendes den ikke optimerede algoritme fremkommer den løsning, som ses på figur 6.11. Fejlen opstår, fordi der fremkommer match, som ikke findes. Da brik 0 side 0 har samme længde som brik 1 side 4, kan de to sider være naboer. Algoritmen arbejder så ud fra dette match. Disse fejl kan algoritmen ikke tage højde for.

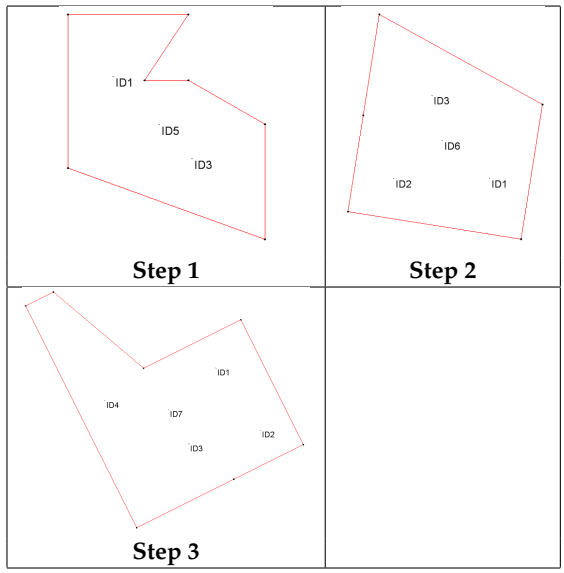


Figur 6.10: Det samlede puslespil af roterede brikker, hvor algoritmen er optimeret



Figur 6.11: Det samlede puslespil af roterede brikker, hvor algoritmen ikke er optimeret.

Herunder på figur 6.12 ses det, hvordan Cocktailmetoden arbejder med de roterede brikker i tre trin, hvor det fjerde trin er løsningen.



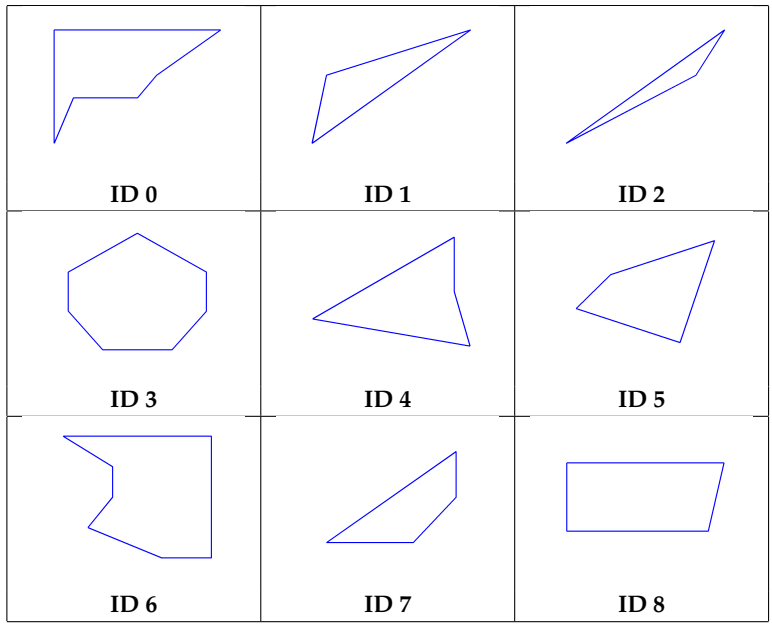
Figur 6.12: De enkelte trin som Cocktailmetoden udfører med brikker, der skal roteres

	Cocktailmetoden	Backtracking	Tid(ms)
Normal	30	1	234,38
Optimeret	24	1	296,86

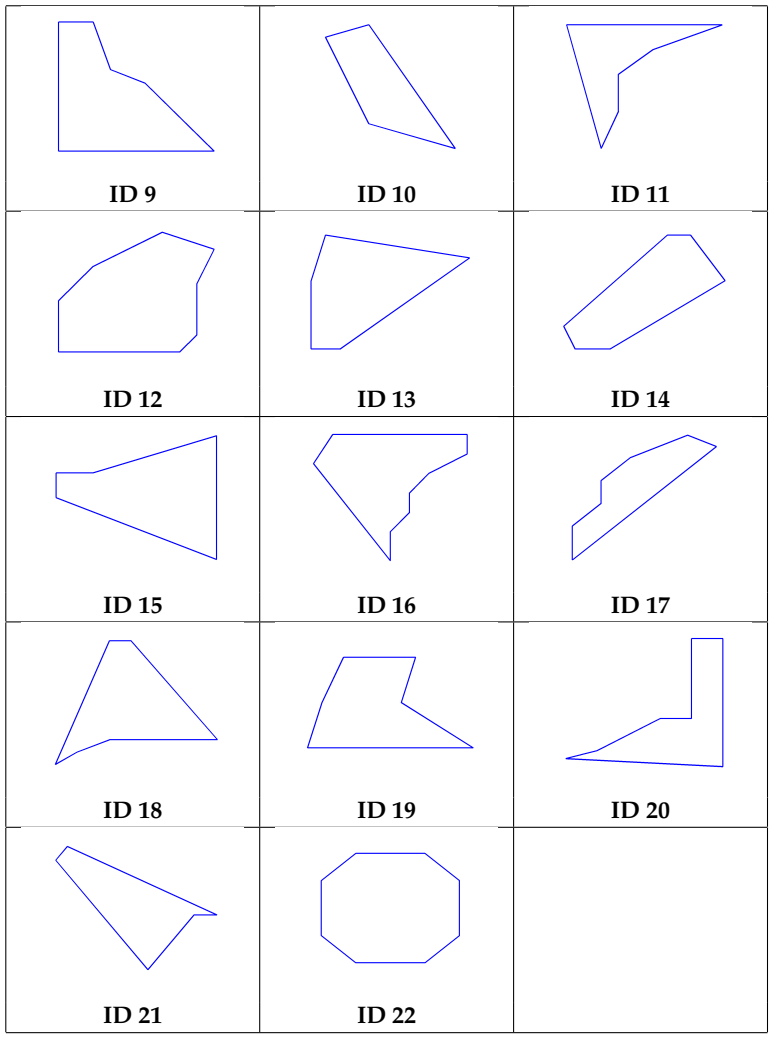
Tabel 6.3: Tidsforbrug, gennemløb for de enkelte metoder

6.5 Fuld test af algoritmen

Testen, der bliver udført i dette afsnit, ser nærmere på, om alle algoritmens metoder kan samarbejde. Der er lavet et puslespil, der indeholder 23 brikker, som er navngivet fra 0 til 22. Brikkerne kan se på figur 6.13 og figur 6.13.

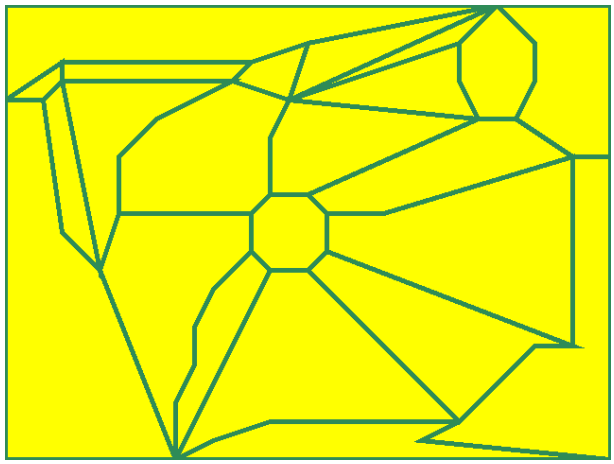


Figur 6.13: De 12 første brikker, der skal bruges til fuld test. Brikkerne er skitserede i Matlab, hvorfor nogle af brikkerne er deformerede



Figur 6.14: De 14 sidste brikker, der skal bruges til fuld test. Brikkerne er skitserede i Matlab, hvorfor nogle af brikkerne er deformerede

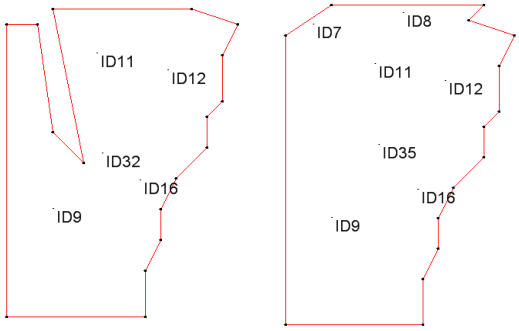
Det færdige resultat skulle gerne være som vist på figur 6.15



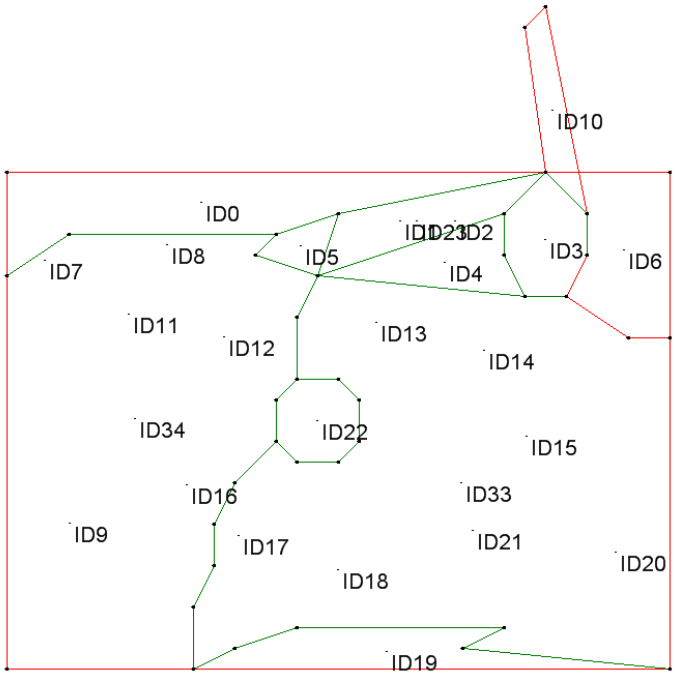
Figur 6.15: Det samlede puslespil

Algoritmernes løsning vises på figur 6.17. Løsningen er fremkommet af brikker, der ikke er roteret, da denne proces ville være meget tidskrævende.

Løsningen består af 11 store brikker, hvoraf Cocktailmetoden har dannet de tre af dem. Disse tre brikker kan ses på figur 6.18, hvor brik 23 består af brik 1 og 2, og brik 33 består af brik 13, 14, 15, 17, 18, 20 og 21 og brik 34 består af brik 7, 8, 9, 11, 12 og 16. Det var meningen, at brik 10 også skulle have været en del af brik 34. Når løsningen på figur 6.17 betragtes, kan det tilmed ses, at brik 10 lægges, hvor algoritmen kan finde plads til den. Dette skyldes at Cocktailmetoden ikke tager højde for, at der kan være et hul i en brik. Da brikkene 7 og 8 blive lagt til brik 32, dannes der en hul i midten af brik 32, og når Cocktailmetoden skal smelte brikkerne sammen til brik 34, ser den kun på yderkanten af brikken. Hermed tages hullet ikke i betragtning af den nye brik. På figur 6.16 ses brik 32 og brik 34.



Figur 6.16: Brik 32 og brik 34

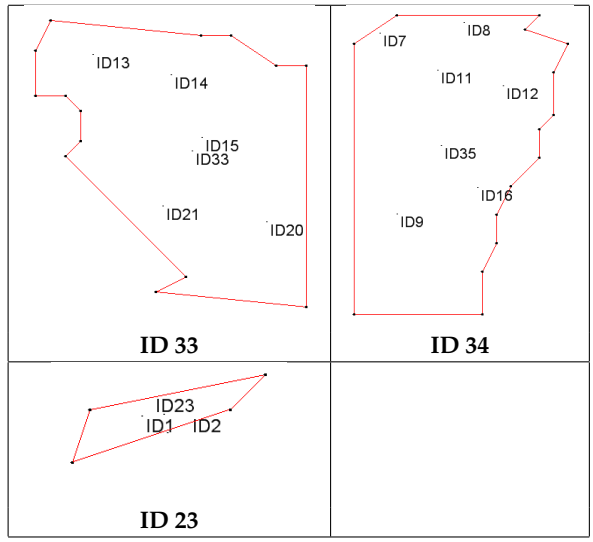


Figur 6.17: Det samlede puslespil

I tabel 6.4 ses det, hvor lang tid algoritmen har været om at løse det 23 brikker store puslespil. Den algoritme, der har optimeret søgemetode, blev stoppet efter 920 ms. Grunden til dette er, at algoritmen kommer frem til den "rigtige"løsning ved første løsningsforslag. I løbet af den tid, algoritmen har brugt, nåede den at lave 244 løsningsforslag. Derimod arbejdede den anden algoritme i over 23 timer for at komme frem til den samme løsning. Algoritmen med den optimerede søgemetode, kunne også være uheldig og starte med en anden brik og derved komme på samme arbejde som algoritmen uden optimering.

	Cocktail	Backtracking	Løsninger	Tid
Normal	234	≈2768235	544180	>23 timer
Optimeret	240	≈1205	244	920 ms

Tabel 6.4: Tidsforbrug, gennemløb for de enkelte metoder

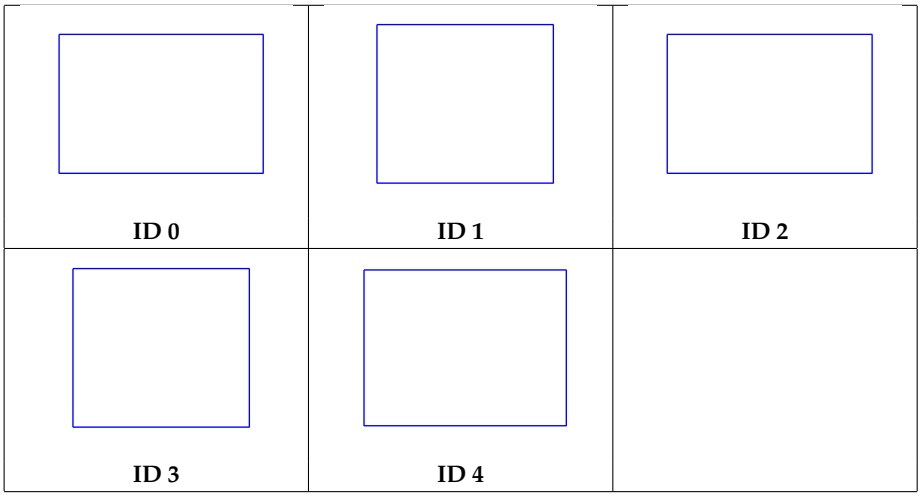


Figur 6.18: Brikker som Cocktailmetoden har smeltet sammen ud fra de 23 oprindelige brikker.

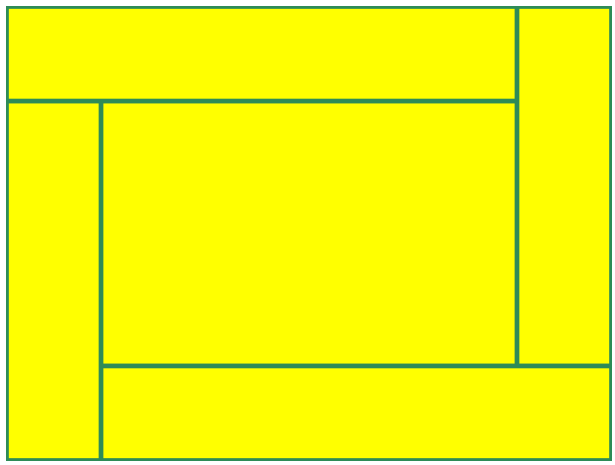
6.6 Fejltest

I fejltesten, bliver algoritmerne udsat for et puslespil, der ikke har nogle guillotinesnit, det vil sige et snit fra puslespillets ene side til siden overfor. På figur 6.20 ses et eksempel på et puslespil, der ikke har guillotinesnit.

På figur 6.19 er de brikker, som algoritmerne skal bruge til at samle puslespillet, vist.



Figur 6.19: Alle brikker der bruges i fejltesten. Brikkerne er skitserede i Matlab, hvorfor nogle af brikkerne er deformerede



Figur 6.20: Det samlede puslespil

Efter algoritmerne har forsøgt sig med at samle puslespillet, er de ikke kommet frem til nogle løsninger. Cocktailmetoden er sat ud af drift, da alle brikkerne kan have flere naboer til alle deres sider. Backtracking kan heller ikke lægge nogle af brikkerne, da det kun er Cocktailmetoden, der kan håndtere sammensatte sider.

	Cocktail	Backtracking	Løsninger	Tid(s)
Normal	20	98	0	1,703
Optimeret	20	155	0	2,07

Tabel 6.5: Tidsforbrug, gennemløb for de enkelte metoder

6.7 Delkonklusion

Da mange af testene er blevet kørt uden rotation, skyldes det, at den nuværende version arbejder meget langsomt. Hvis metoden skal kunne anvendes, skal den optimeres, så den ikke kører rotationsmetoden flere gange for at regne en vinkel ud eksakt. Derimod skal den kunne finde den rigtige vinkel med stor nøjagtighed, ved kun et gennemløb.

Der vil være en del problemer med at få algoritmen til at lægge små brikker til store brikker. Dette skyldes at Radiusmetoden, som er en del af overlapskontrollen, mange gange tror, at små brikker overlappes af store brikker. Grunden til dette er, at en brik med stort areal også har en stor radius. Når en lille brik derefter lægges ved siden af, vil denne radius i mange tilfælde rage ud over centrum på den lille brik, hvorved et overlap opstår. Hvis dette bliver forbedret, kan Cocktailmetoden blive mere effektiv, så der derved vil være færre brikke tilbage til backtrackingmetoden.

Da den fulde test at algoritmen ikke er gennemført med tilfredstillende resultat, fortsætter arbejdet med at finde bedre metoder til algoritmerne, sådan at de opnår bedre resultater.

Kapitel 7

Diskussion

Under projektets forløb er forskellige metoder forsøgt anvendt for at finde frem til den, der hurtigst kunne finde den korrekte løsning på et 2D geometrisk puslespil. De anvendte metoder er til dels inspireret af lignende projekter, som er udviklet tidligere.

Det viste sig hurtigt, at hvis metoden skulle være meget præcis, skulle der implementeres en hel del metoder. Dette medførte beklageligvis, at algoritmen var så lang tid om at lægge et forholdsvis simpelt puslespil, at et menneske ville kunne gøre det hurtigere. Det var altså nødvendigt at undersøge, hvilke optimeringsmuligheder, der var for de enkelte metoder. De fleste af metoderne er blevet optimeret undervejs i forløbet.

Algoritmen kunne være udviklet til at anvende kurvematching ved løsning af puslespil. Ved projektets begyndelse forudsattes det imidlertid, at de anvendte metoder ville være hurtigere om end lidt mindre præcise end kurvematching. Formodningen har vist sig ikke at holde stik. Det vil i hvert fald kræve en grundig optimering af de udviklede metoder at gøre algoritmen lige så hurtig.

I projektet anvendes .net som framework til programmet. Det har vist sig, at de metoder .net anvender til at gemme billedfiler er meget langsomme. En yderligere optimeringsmulighed er derfor at benytte en anden måde at gemme løsningsmulighederne på. Man kunne f.eks. gemme brikkerne som koordinater i stedet for billedfiler, hvilket ville gå meget hurtigere. Det er hurtigere blot at skrive en række tal fremfor at vente på, at .net tegner et billede og lagrer det.

Det blev desuden undersøgt, hvorvidt det var nødvendigt at brikkerne kunne roteres. Hvis algortimen skal kunne håndtere indscannede brikker, er det nødvendigt, at de kan roteres. Imidlertid var metoden til rotation ikke optimal. Rotationsmetoden "rotet"kører processen igennem adskillige gange, før vinklerne passer. Algoritmen vil således kunne arbejde mere effektivt, hvis rotationsmetoden "Rotet"videreudvikles, så det kun er nødvendigt at køre processen igennem en gang.

Cocktailmetoden er navnet på den metode, som udvikledes til at smelte nabobrikker sammen, så der er færre brikker at arbejde med. Metoden kan lave fejl, hvis der i et puslespil er flere sider med samme længde på den samme brik. Cocktailmetoden kan i sådanne tilfælde vælge at smelte de forkerte sider på brikkerne sammen, hvilket medfører, at det endelige resultat bliver forkert.

Cocktailmetoden udelukker brikker, hvis der findes mere end to brikker med samme sidelængde. Det er kun, hvis de ens sidelængder findes på samme brik, at problemet opstår. Metoden skal derfor udvides, så brikker af denne type også isoleres.

Der er således rige muligheder for at forbedre algoritmen, så den kan arbejde mere effektivt. Projektet er desværre tidsbegrænset, så det har ikke været muligt at gøre ovennævnte udvidelser til virkelighed.

Kapitel 8

Konklusion

8.1 Forkert retning

I begyndelsen tog projektet en forkert drejning i forhold til det ønskede resultat. Der blev arbejdet og udviklet flere metoder til algoritmen, som skulle benyttes til indscannede brikker. Dette viste sig imidlertid at være for ambitiøst i forhold til den tid, der var afsat til projektet. Det arbejde, der udførtes ved projektets begyndelse, er derfor henlagt til bilag A og bilag B.

8.2 Opfyldelse af problemformulering

Der har været et ønske om at udvikle en algoritme, som kan lægge et 2D geometrisk puslespil. Der er derfor stillet en række krav, som er listet under problemafgrænsningen. Disse punkter vil herunder blive gennemgået, og for hvert enkelt vurderes det, hvorvidt kravene er opfyldt.

Det første problem, der skulle løses, var, at algoritmen kunne lægge et lille visuelt puslespil. Dette lykkedes uden andre forhindringer end nogle simple problemer med Cocktailmetoden.

Algoritmen skulle derefter kunne håndtere lidt mere komplicerede puslespil, hvor der fandtes flere brikker med samme sidelængder. Som nævnt i diskussionen, kapitel 7, kan Cocktailmetoden udelukke brikker i tilfælde af, at puslespillet indeholder mere end to brikker med ens sidelængder. Hvis der kun

er to brikker med ens sidelængder, smelter Coctailmetoden dem sammen, hvorefter "backtrackingmetoden"tester for løsninger med de resterende brikker. Samtlige løsningsmuligheder gemmes i billedfiler.

En yderligere krav til algoritmen var, at den skulle kunne håndtere sammensatte sider. De metoder, der er blevet udviklet til dette formål, er imidlertid blevet overflødige, fordi Cocktailmetoden registrerer de fleste sammensatte sider. Dog er der enkelte tilfælde, som metoden ikke kan løse. Man forestiller sig, at et puslespil som, når det er sat sammen, danner et helt stykke papir. Hvis brikkernes punkter er placeret sådan, at papiret ikke kan deles i et guillotinesnit, kan det ikke løses korrekt med den eksisterende algoritme. Det ville derfor være fordelagtigt at få implementeret en metode, som kunne sætte ind i sådanne tilfælde.

Det sidste krav til algoritmen var, at den skulle kunne rotere brikkerne. Der er fundet en metode til at rotere brikkerne, og den fungerer. Metoden bruger dog uforholdsmæssigt lang tig på at spore sig ind på den rigtige vinkel. Den udviklede metode er altså ikke optimal, selvom om kravet er opfyldt. Som nævnt i diskussionen er der overvejet muligheder for at optimere rotationsmetoden.

8.3 Fremtid

Fremtidigt arbejde kunne være at genoptage den oprindelige problemstilling og udvide algoritmen, så den kan håndtere at lægge et puslespil ud fra indscannede brikker. Dette vil kræve, at arbejde i forbindelse med billedbehandling blev fuldendt, så den del af projektet, der siden blev ekskluderet, blev udført.

Derudover har der været en ambition om, at algoritmen skulle kunne udvides til at løse 3D-puslespil. Hvis denne udvidelse til algoritmen udføres, og de førnævnte metodeoptimeringer samtidig udvikles, kan programmet anvendes som værktøj til f.eks. arkæologer eller konservatorer. Som det er nu, har programmet imidlertid en del begrænsninger, og det er derfor ikke særlig anvendeligt. Erfaringen har vist, at det er langt mere effektivt at benytte kurvematching.

Litteratur

[1] Jens Michael Carstensen (Ed.), *Image analysis, vision and computer graphics*, 19 August 2002.

[2] David Goldberg, Christopher Malon and Marshall Bern, *A Global Approach to Automatic Solution of Jigsaw Puzzles*

[3] Qi-Xing Huang, Simon Flöry, Natasha Gelfand, Michael Hofer, Helmut Pottmann, *Reassembling Fractured Objects by Geometric Matching*, 2006.

[4] Robert Lafore, *Object-Oriented Programming in C++*, 1998.

[5] Weixin Kon, Benjamin B. Kimia, *On Solving 2D and 3D Puzzkes Using Curve Matching*, 2001

[6] Microsoft, *MSDN library*,
http://msdn.microsoft.com/library/default.asp

[7] Gerald M. Radack, Norman I. Badler, *Jigsaw Puzzle Matching Using a Boundry-Centered Polar Encoding*, 1981

[8] OCCS, *Binary Search Trees*,
http://occs.cs.oberlin.edu/classes/fall2004spring2005/Old cs160/lecture23.html

[9] IT University of Copenhagen, *Image and Signal Processing*,
http://www.itu.dk/courses/MBSB/F2006/Lectures/mor.pdf

[10] Paul Bourke, *Calculating the area and centroid of a polygon*,
http://local.wasp.uwa.edu.au/ pbourke/geometry/polyarea/

[11] David Eppstein, *The Geometry Junkyard*,
http://www.ics.uci.edu/ẽppstein/junkyard/all.html

[12] The Game Programming Wiki, *Polygon Collision*,
http://gpwiki.org/index.php/Polygon_Collision

[13] VB Helper, *Rotate the points in a polygon*,
http://vb-helper.com/howto_rotate_polygon_points.html

[14] Gerth Stølting Brodal, *Algoritmer og Datastrukturer 1*,
http://www.daimi.au.dk/dADS1/sortering/index.html

[15] Robert Sedgewick, *Algorihms In C++ (Third Edition)*, 1998

Bilag A

Teori til billedanalyse

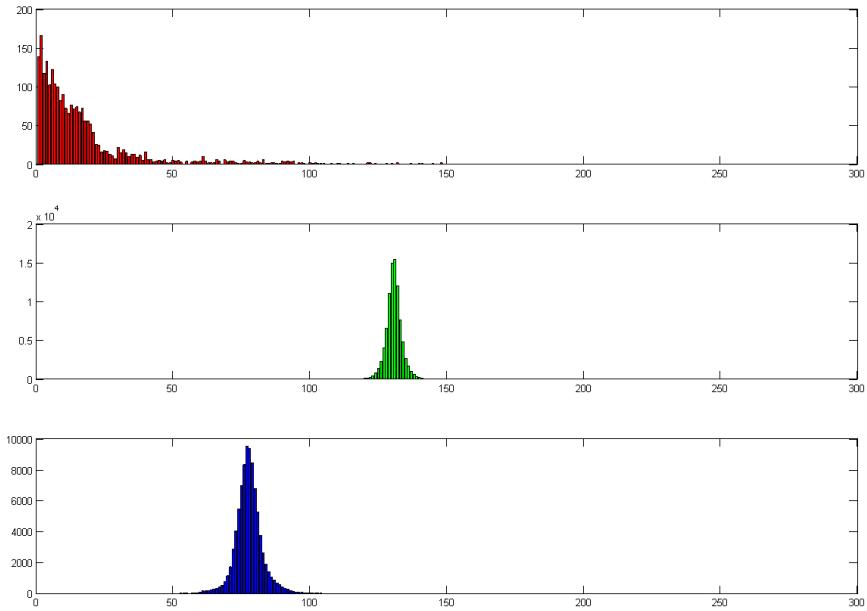
I dette kapitel om Teori til billedeanalyse vil nogle kendte metode indenfor billedeanalysens verden blive gennemgået. Også Morphological metoder og hvordan man undgår baggrunden efter scanning, så man kun skal fokusere på brikker, vil blive opridset. Udover disse metoder vil der også blive set på, hvordan man finder omridset af brikkerne og hvad man kan bruge information om omridset til i projektets videre til forløb.

A.1 Udskillelse

En af de simple analyser, der skal laves, når brikkerne er scannet ind, er at fjerne baggrunden. Dette kan gøres ved, at man tager en kendt baggrundsfarve som der ikke findes så meget af i brikkerne. Idet man kender baggrundsfarven kan man ordne det sådan, at alt der bærer farven skal være sort og alt andet skal være hvidt. Når det er sket kan der så laves en opmåling af brikkerne.

Før computeren kan finde brikkerne skal man lave en analyse på baggrunden for at finde ud, hvad RGB værdierne er, så computeren kan fjerne de pixles, der har den samme farve som baggrunden. Indscanningen kan ikke antage den samme værdi på alle pixles i baggrunden, da der kan være ujævnheder i papiret eller i den farve, som skal bruges til baggundsfarve. De fleste af pixlene vil dog have den samme værdi, men for at få så mange af de pixles, der er repiter i baggrunden, med som muligt skal man kigge indenfor et interval af farveværdier. Et sådant farveinterval kan man bedst illustrere i et histogram, som vil

vise hvilken farveværdi, der optræder oftest. Hvis man ser på Figur A.1 kan man se histogrammet for den grønne baggrund, som testbrikkerne er scannet ind med.



Figur A.1: Historgram over farve værdi i den grønne baggrund

Histogrammet på Figur A.1 er lavet ud fra et område på 300x300 pixles af den grønne baggrund. På den øverste graf kan man se antal af værdier i den røde farve, derefter kommer den grønne farve og til sidst den blå. På histogrammet kan man se, at en stor del af pixlene har værdier fra 110 til 150 med størst koncentration omkring værdien 131, det skyldes at baggrundsfarven er grøn og værdierne ligger alle indenfor det grønne interval. Dog er bagrunden ikke ren grøn, der er noget mørkt i den og dette viser sig idet der også er en del pixels, der antager blå farver indenfor intervallet 50 til 100. Ud fra disse informationer kan man lave et filter, der sørger for, at alle områder indeholdende baggrunds-

farven efter histogrammet fremstår som sorte og alle andre hvide.

Udfra histogrammet kan man fremstille et skema over det værdiområde for RGB, som gælder for baggrunden af scanningerne. Herunder i Tabel A.1 kan man se de værdier, som gælder for den grønne baggrund, der bruges til at scanne testbrikkerne ind med.

Farve	Fra	til
Rød	0	45
Grøn	110	150
Blå	50	100

Tabel A.1: Tabel over værdierne for den grønne baggrund

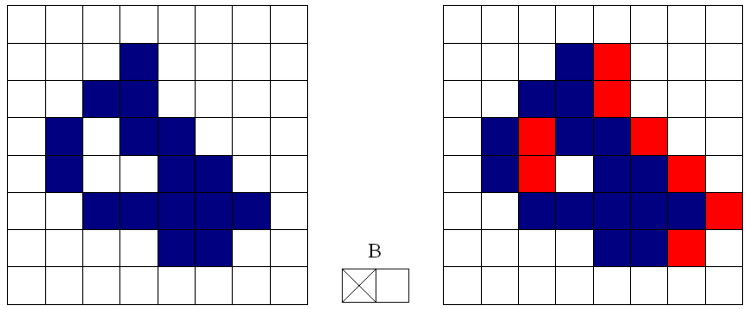
A.2 Matematisk morfologi

Når brikkerne er scannet ind er der en masse forstyrrelse i billedet i form af støvkorn og andre objekter, der kan forstyrre billeder. Disse forstyrrelser vil man gerne fjerne, da det kan forstyrre opmålingen af brikkerne. En metode man kan bruge til at fjerne mange at disse forstyrrelser er Matematisk morphologi. Dog de metoder, som man har mest interesse i Dilation, Erosion, Opening og Closing.

A.2.1 Metoder i Matematisk morfologi

Dilation giver den orginale form en ekstra grænse, så formen bliver større, størrelsen på grænsen er afhængig af, hvor stor formen er og hvor stort strukturelementet er.
Definition Form A er dilation med strukturelement B og kan stilles op som vist nedenfor:

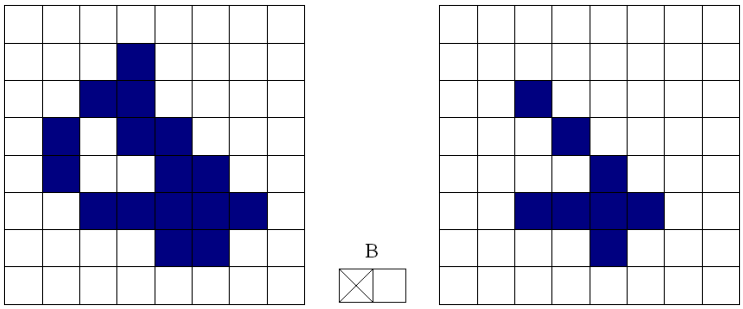
$$A \oplus B = \{x|(\widehat{B})_x \cap A \neq \emptyset\}$$



Figur A.2: Et eksempel hvor form A bliver dilation med strukturelemt B

Erosionsmetoden finder de punker for hvilke strukturelementet indeholder de originale mængder. Den yderste grænse af den originale form vil blive fjernet, når man bruger erosion.
Definition Form A er erosion med strukturelement B. Kan stilles op således:

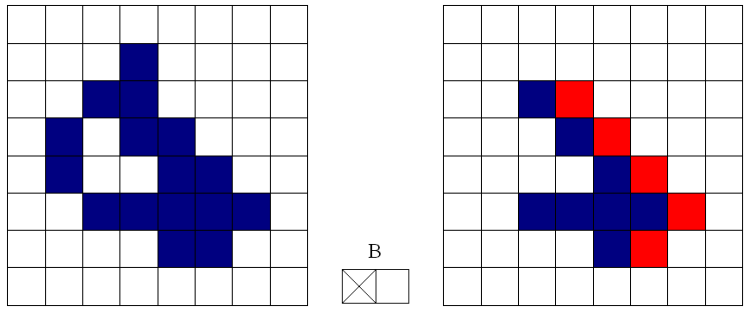
$$A \ominus B = \{x|(B)_x \subseteq A\}$$



Figur A.3: Et eksempel hvor form A bliver erosion med strukturelement B

Når man kender til disse to metoder kan man arbejde videre med Opening og Closing metoderne. Opening metoden er en blanding af erosion og derefter dilation.
Definition Form A er Opening med strukturelement B. Kan stilles op som vist nedenfor:

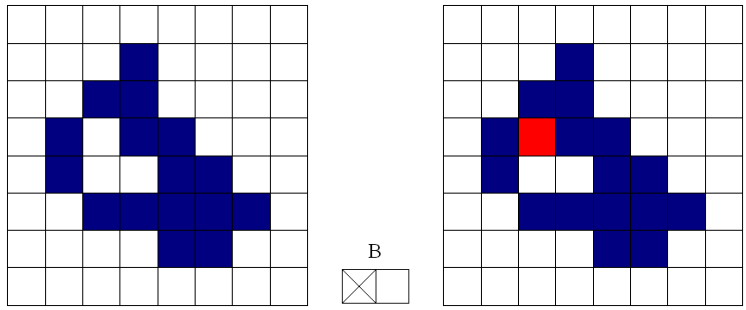
$$A \circ B = (A \ominus B) \oplus B$$



Figur A.4: Et eksempel på opening metoden

Closing metoden er næsten lige som opening metoden. Her kører man først dilation og derefter erosionsmetoden.
Definition Form A er closing med strukturelemente B. kan stilles op som i eksemplet nedenfor:

$$A \bullet B = (A \oplus B) \ominus B$$



Figur A.5: Et eksempel på closing metoden

Figur A.4 og Figur A.5 er eksempler på Opening og Closing metoderne. For at opnå fuld forståelse for disse eksempler skal man se på eksempet i Figur A.3 og på eksemplet Figur A.3. Opening metoden er den, som egner sig bedst til at rense for forstyrrelser i scanningerne, da metoden er god til at finde struktur i en store mængde pixles.

A.2.2 Matematisk morfologi i OpenCV

OpenCV Library er funktioner, som kan lave matematisk morphologi på billeder, der skal behandle det. Det første man starter med at sætte op er strukturelementet og det gøres ligesom vist herunder på kodeeksemplet:

```
IplConvKernel* tmpElement = 0;
int tmpValue = 1;
tmpElement = cvCreateStructuringElementEx( tmpValue*2+1,
     tmpValue*2+1, tmpValue, tmpValue, CV_SHAPE_CROSS,
    NULL );
```

Det føste der sker er, at man opretter et element af IplConvKernel. Derefter sætter man det op med funktionen cvCreateStructuringElementEx. Vha. denne funktion kan man finde ud af hvor stort strukturelementet skal være, i koden ovenover er det 3x3. Derefter finder man ud af, hvor holdepunktet kan være indenfor strukturelementets areal. I koden kommer holdepunktet til at være 1,1. Når areal og holdepunkt er bestemt kan man afgøre, hvordan strukturelementets form skal være. Det kan være et kryds, en firkant eller en ellipse. I koden her er det et kryds. Det sidste parameter, der kan sættes i funktion er hvis man ønsker at lave sin egen form på strukturelementet.

Når strukturelementet er sat op kan man gå videre med behandlingen af billedet og det kan se på koden herunder. Funktionen cvMorphologyEx sørger for, at strukturelementet bliver kørt igennem billedet.

```
IplImage* imTmp = cvCloneImage(images);
cvMorphologyEx(images,images,imTmp, tmpElement,
    CV_MOP_OPEN);
```

Der er nogle parametre, der skal sættes op før cvMorphologyEx kan køre behandligen af billedet igennem. Det første er input billedet og derefter er det output billedet. Den tredje parameter er et midlertidigt billede, som skal bruges til billedebehandlingen. Når disse parametre er på plads skal strukturelementet sættes ind. Til sidst afgør man, hvilken metode, man finder bedst egnet til at køre strukturelementet igennem i billedet. I eksemplet anvendes Opening metoden.

A.3 Omrids

Efter alle filterne har gjort deres arbejde kan man så finde omrise af brikkerne. Dette omris skal bruges til at lave information om de linje styker hver enkel brikkerne bestå af. Disse linje stykker skal bruges senre hen til at samline som brikkerne passer samme. Ved at bruge linje skykker længter som samlings med kan man ungå at bruge formerne på brikkerne, men der i mod kan man gøre det mere geometisk isteden.

A.3.1 Find omrids i OpenCV

OpenCV Library har nogle mmetode til at finde omris af figure i det billede. For at gøre det nemt for funktion i OpenCV til at finde figurens omris kan man gøre baggrunden helt sort og det som man vil finde omrise af kan man gøre helt hvid, her ved at der nemmer for OpenCV at finde omriset af figurene i billede. Her under i kodeeksample kan man se den funktion som der bruge til at finde omris af figuerne, funktion hedde FindContours.

```
IplImage *imageT1 = 0;
imageT1 = cvCloneImage( images );
CvMemStorage* stor;
CvPoint* PointArray;
CvSeq* cont;

stor = cvCreateMemStorage(0);
cont = cvCreateSeq(CV_SEQ_ELTYPE_POINT, sizeof(CvSeq),
    sizeof(CvPoint) , stor);

cvFindContours( imageT1, stor, &cont, sizeof(CvContour),
    CV_RETR_LIST, CV_CHAIN_APPROX_NONE, cvPoint(0,0));
```

I kode eksample her over er det første funktion “cvCreateMemStorage” gør er at alloker starten for et lagering område og retuner adressen til den plas til “stor”.

“cvCreateMemStorage” tag en parameter som fortæller hvor stor blokkene i lagering område skal være, hvis værdien er nul så bliver blok størlsen sat til at være 64Kb stor ca. Derefet bliver der lave en objokt som der holde styr på de elementer som der skal komme i “stor”. “cvCreateSeq” skal have en række parameter, første parameter er et flag som bestem hvad type rækkefølje da

denne rækkeføjle i eksample skal styre en række punkter så er det en god idé at bruge "CV_SEQ_ELTYPE_POINT" faget. Efter faget sætte objekt størlsen og derefter element størlsen, og til sidste hvilken adresse i hukommelsen lægeringe at punkter skal begynde.

Efter de to objekter er blivet sat op så kan man gå igen med at finde omris af figurene i billede. Det gøre med funktion "cvFindContours" og denne funktion har en række parameter som der skal være, første et billede som der er i 8-bit og der efter adressen hvor i hukommelsen punkterne af omrisene skal gemmes, og en pointer til det objekt som holde styr på rækkeføjle at punkter. Fjere paramere er elementet størelsen på "CvContour" da man ikke køre efter rækkenføjle størelse. Femte parameter er et flag som fortæller funktion om hvilken måde den skal gemme omrisene på, her i eksample bliver punkterne gem som en liste. Det næste parameter sætter hvilken tilnærmelse metode man vil bruge til at finde omrise med. Den sidste parameter er punktet hvor funktion skal starte fra at lede.

Når alle omrisene er blive funde så kan man begynde analysen af alle punkter som der er i hver af de omris som der høre til figurene billede. Her under kan man se et lille kode eksampel på hvordan det kan se ud, for at få hver enkel omris ud. Alle punkterne som der tilhøre til et omris bliver milertidig gemt i "PointArray", og der kan man søges alle punkter igennem for et enkel omris.

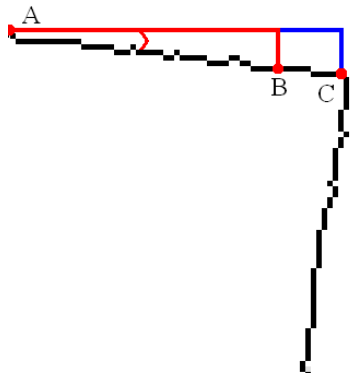
```
for(;cont;cont = cont->h_next)
{
  int count = cont->total;

  PointArray = (CvPoint*)malloc( count*sizeof(CvPoint) );
  cvCvtSeqToArray(cont, PointArray, CV_WHOLE_SEQ);

  ....
  ....

  free(PointArray);
}
```

A.3.2 Fint til groft omrids

Når alle omris er mål op kan man begynde alt lave det grove omris, det grov omris skal bruge til at samlinging processen. Da opmålingerne af omrisene be-

stå af mange punkter og mange af disse punkt ligge i en snorligerække kan man fjerne de punkter som der ligge imellem start og slut punkterne på den linje. Der kan bruge nogle forskellig motoder til at delle omriset af brikker op med, så man har en mulighed at gemme information om omrise af brikkerne.

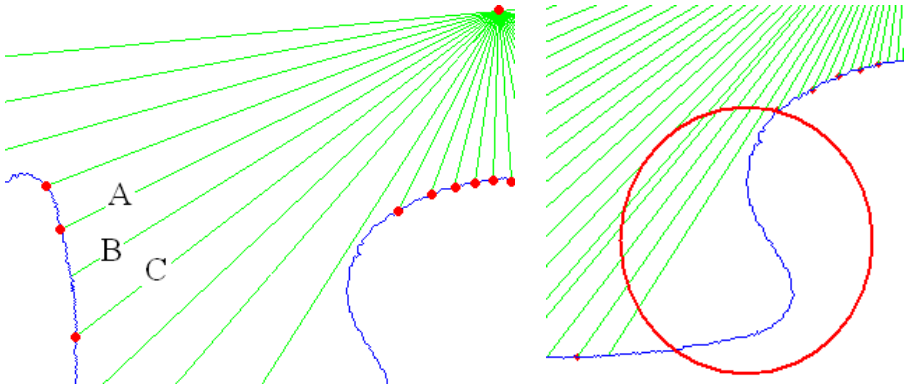
En af metoderne til at delle omrise på af brikkerne, er at man set efter de lige stykker at brikkens omris. Det gøres ved at man gå nogle punkter ud fra det første punkt så man har et linje stykke fra A til B, dette kan set elustert på Figur A.6. Dette linje stykke har en vikel og hvis det næste punk efter B, så man har et linje stykke fra A til C og giver den samme vikel ved punkte A og ved linje stykke A til B kan man indrave punkte C til linje stykke A til B. Det kan man blive ved man til vikel blive for stor, når det skår starte man så for fra med at finde det næste linje stykke.



Figur A.6: Metoden hvor man fynde linje stykkerne ved at så på vikel på punkterne

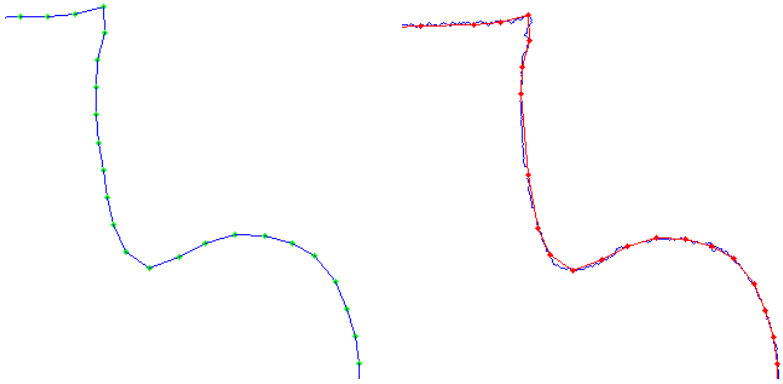
En anden metode som der kan bruge er finde linje stykker med er at man finde midten at brikken og ud fra midten lave en linje til det første punkt og der efter et linje til næste som der er forstudt vinkel Δ. Forskellen imellem Linje A og B kan bruge til at regne linje C ud med når Δ er den samme hele tiden. Hvis linje C ligge toloragsen på den berenge linje C kan man sige at de ligge på samme linje stykke på brikken omris, dette kan man også se på Figur A.7(A) hvor Linjestykke gå fra det punkt som bliver ramt af linje A til punkte som rammes af linje C. For at få alle punker med skal man køre med en lille Δ, ellers kan det give nogle problemer når der skal måles punkter i udformingerne af brikkerne

da der ikke vil være så mange punkt i disse område af omrise. Figur A.7(B) kan det ses elstert hvordan der kan mangel punkter i et stort område.



Figur A.7: (A) Metoden hvor man finde linje stykkerne ved at gå ud at midten i en fast vinkel. (B) Problemet med denne metode at der kan være for lidt punkter

Den tredje metode som der kan bruges til at finde linjestykker er man sidde et fast antal pixel alle linjestykkerne skal bestå af, i at pixel ikke ligge i en lige linje så er alle linjestykker ikke lige store. Når man har fået lave omrise af brikken om til n pixel store linjestykker har man en masse lænder på en masse linjestykker og de længter som der lige store og lægge lige efter hinanden kan man så gå hen og lave til et stort linjestykke. Hvis man ser på Figur A.8 (A) kan man se brikkens omris efter den er blive del op i 10 pixel store linjestykker, hvor på Figur A.8 (B) der er set på linjestykkernes størelse og for fjerne nogle at punkertene, man kan se det orginale brik omris nedeste i Figur A.8 (B).



Figur A.8: (A) Til venste kan man se hvor brikke er del op med 10 pixel store linjestykker. (B) Til høje hvor der er linjestykker som der er slå sammen

A.4 Afrunding på teori til billedanalyse

Efter være ind scanning skal der lige kontoller baggundes farve da den kan ender sig. Scanning har en tildens til at være mørke når man scanner store flader ind hvis man små flader, hvis man ikke joster farve filter så kan man ikke bruge morphologi metoden til noget. Ved morphologi skal der bruge Opening metoden da den er rigtig godt til at finde figuer ud fra pixel støj i billede.
Efter billede behandlingen skal omrise af brikkerne opmåles, og det vil nok være best at bruge den tredje metode hvor man dele omrise på i n pixels store linjestykker. Den anden motode er ikke god til dette formål ad den vil miste for mange vigtig punkter så man vil ikke havde den samme information om brikken. Den første metode vil ikke virke alene da den er meget følsom over for små ujenhede på omrise af brikkerne. Den første metode ville den nok havde det beder hvis man tog den trejde metode og køre den først og der efter den første metode der efter. - hvad for metode er der meste hold i.

Bilag B

Almindeligt puslespil

Dette afsnit indeholder en undersøgelse af, hvordan et puslespil lægges uden computerens hjælp. Det ønskes herved klarlagt, hvilke metoder menneskehjernen bruger til at lægge puslespil. Udover det vil kapitlet indeholde en analyse af de brikker, der skal bruges til at teste systemet med i begyndelsen.

B.1 Hvordan lægger man puslespillet?

For at få et bedre indblik i den proces det er at samle et puslespil, vil der i dette afsnit blive kigget på, hvordan man samler et uden hjælp fra computeren. Metoderne er taget ud fra en persons (Christian S. Hansen) måde at lægge et puslespil på. En anden forsøgsperson ville muligvis vælge andre metoder. Metoderne bliver vurderet udfra effektivitet og kan måske overføres til den algoritme som skal samle et geometrisk-pulsespil i sidste ende, hvis de er anvendelige.

I denne version til at lægge et pulsespil, er første skridt at finde alle de brikker, der skal bruges til kanterne. Disse brikker er lette at genkende, idet der altid er en side, som er lige. Der er dog 4 brikker, hvor to af siderne er, nemlig hjørnebrikkerne. Når alle kant- og hjørnebrikkerne er sorteret fra kan man begynde at samle rammen. Når man samler rammen fokuseres der ikke på faconen men istedet farver, nuancer og motiv.

Efter alle brikkerne til kanten er lagt sorteres resten af brikkerne efter forskel-

lige motiver, himmel og omgivelser. Efter endt sortering går man igang med at lave motiverne, derefter omgivelserne og til sidst himlen. Undervejs er der ikke blevet kigget så nøjagtigt på brikkernes facon bort set fra om der har været indhak eller tapper. Ved samling af himlen skal der dog ses på hele faconen af brikkerne da disse i mange tilfælde har samme nuance. For nemmest at samle denne del af puslespillet skal man se på formen af brikkernes sider. De brikker, hvor siderne skal placeres op af hinanden skal formerne bevæge sig på samme måde.

Nogle af disse metoder kan bruges til algoritmen. Dog vil det at vurdere efter farve og motiv ikke kunne bruges, eftersom algoritmen skal samle et geometrisk puslespil. Algoritmen vil i stedet anvende den metode, der er beskrevet til at samle himlen. Her er man ligeglad med farver men er istedet mere interesseret i brikkens facon.

B.2 Metode til analyse af brikker

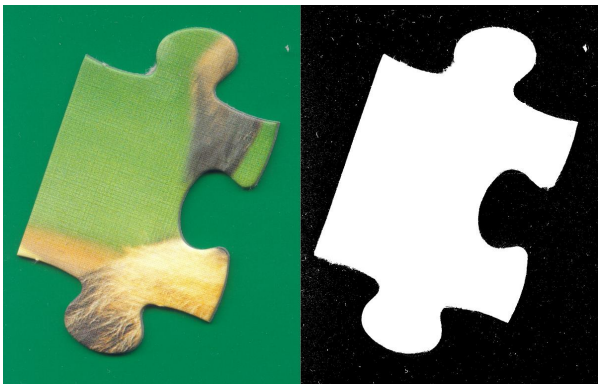
Når algoritmen skal testes skal der til at starte med benyttes et almindeligt puslespil. Det bliver et lille puslespil på 35 brikker og for at undgå at der er brikker som er ens i udformning, skal der laves en analyse af brikkerne.

Denne analyse, der skal laves af brikkerne skal gennemføres for at sørge for at der ikke findes flere brikker som har samme facon. Da algoritmen i første omgang ikke ser efter farver vil dette komme til give nogle problemer for algoritmen at samle puslespillet. For at være sikker på, at der ikke er to eller flere brikker som er ens, scannes alle brikker ind og et analyseprogram og sammenlignes med hinanden.

Når der skal laves en analyse af de brikker, som skal bruges til at teste algoritmen, skal disse først scannes ind i computeren så analyseprogrammet har nogle billeder at arbejde med. Analyseprogrammet bruger nogle filtere så programmet kan få adskilt brikkerne fra baggrunden.

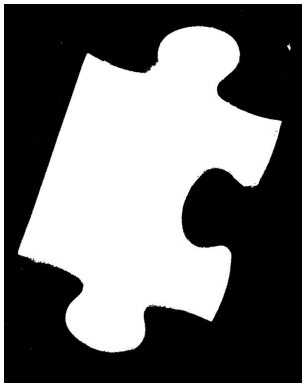
B.2.1 Uddybende forklaring af analyseprogrammet

Det første analysen skal gøre er at gemme al information om former på brikkerne. Dette gøres ved at udføre billedeanalyse på de scannede brikker. For at gøre det lidt nemmere at lave billedeanalysen, laver man scanningen med en fast farve som baggrund. Alle brikkerne i denne analyse er blevet scannet ind på en grøn baggrund. Denne grønne farve findes ikke i store mængder i puslespillet, så nu er det nemt at udelukke brikkerne fra baggunden. På Figur B.1 kan man på det første billede se, hvordan det ser ud, når man scanner det og det andet billede viser hvordan det ser ud, når man har skiftet den grønne farve ud med sort mens alle andre farvenuancer, dvs. brikkerne er skiftet ud med hvid.



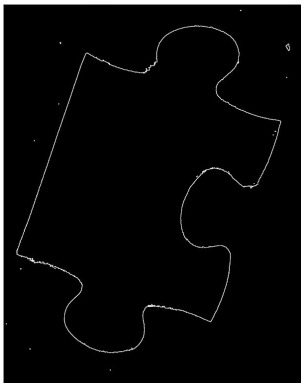
Figur B.1: Billede af brikkerne efter scanning og efter første filter

Efter det første filter er brikkerne afbidlet hvidt på sort. Dog forekommer der fejl i billedet i form at mindre og større støvkorn, disse fejl skal fjernes før man kan finde brikkerne i billedet. Dette gøres ved hjælp af „Morphological Operations“fra OpenCV. Denne funktion bevirker, at alle de små fejl forsvinder. På Figur B.2 kan man ved sammenligning med billede to i Figur B.1 se at der næsten ikke er nogen fejl tilbage.



Figur B.2: En brik som er behandlet med Morphological metoden

Efter de scannede brikker er blevet behandlet med „Morphological Operations“ fra OpenCV Library, kan man begynde at finde omridset af brikkerne. Dette omrids skal bruges til selve analysen af brikkerne. På Figur B.3 kan man se et testbillede af omridset af en brik. På billedet ses også at der er lavet omrids af nogle større fejl som Morphological metode ikke har kunnet fjerne. Fejlene kan fjernes at gå ind og se på hvor mange punkter omridset er lavet af. Billedet på Figur B.3 er scannet ind med 600 dpi så brikkerne består af 6000-7000 punkter. Derved kan man sætte en grænse på 1000 som minimum for, hvor omridset må komme med.



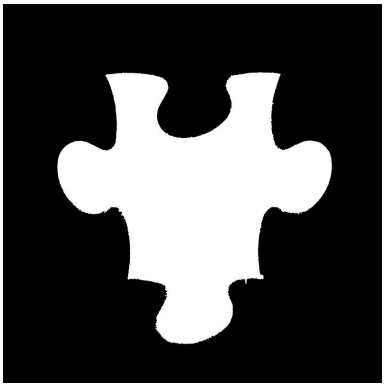
Figur B.3: Omris af en brikke med nogle små fejl

Når brikkernes omrids er fundet kan man gemme brikkerne, så de er klar til analysen. Dog skal brikkerne centreres i det nye billede, det bliver gemt i, fordi de skal rotere om deres egen akse under analysen. Centreringen sker ved at finde midtpunktet på brikken, midten af en brik er der hvor balancepunktet ligger, en tilnærmelse til dette punkt kan findes ved at tage gennemsnittet af alle punkter fra en enkelt briks omrids.

Når brikkerne har gennemgået billedebehandlingen kan analysen gå i gang med se om nogle af brikkerne er ens. Da man ikke kan regne med, at brikkerne er lagt ens i scanneren op er ikke nøvendigvis op og ned er ikke nødvendigvis ned. Derfor er man nødt til at rotere alle brikkerne under analysen. Brikkerne roteres 360 grader og hvert step er på 0,5 grad. Programmet tager den ene brik og samligner den med alle de andre brikker. Sammenligningen udføres ved først at trække omridset af den første scannede brik fra omridset af en anden. Derefter har man kun det omrids tilbage som ikke passer med det billede, man har trukket fra. Denne proces udføres med alle billeder et efter et. Når sammenligningen med det første billde er gennemført, går programmet videre til billede nummer to og sammenligner på samme måde alle billeder med dette billede et efter et. Billedet med det tilbageblevne omrids tager man så summen af. Den information kan man så plotte ind i en contur plot, hvor man så kan se, hvilke brikker der er tæt på at være ens.

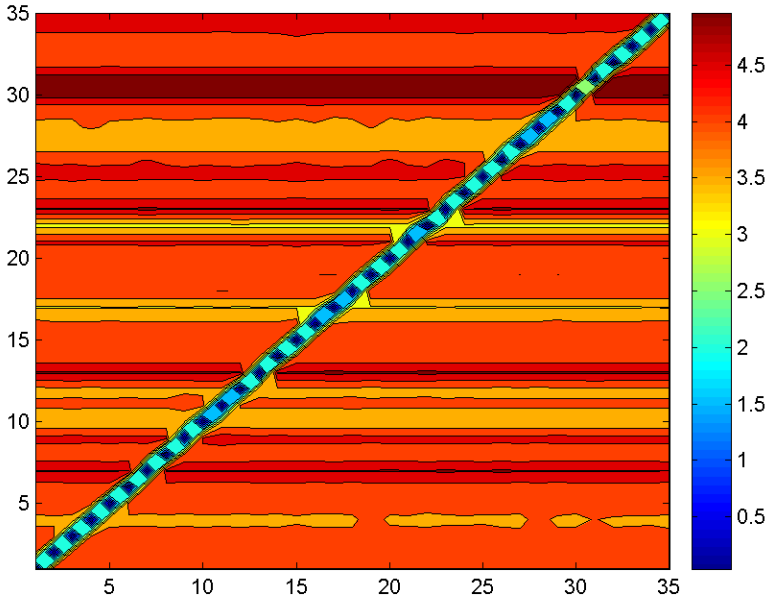
B.2.2 Resultater af analyse på brikker

Analysen bliver lavet på to måder. Den første metode, hvor omridset af brikkerne sammenlignes ses illustreret på Figur B.3 og den anden metode, hvor brikkerne er fyldt ser ud som på Figur B.4. Dette gøres for at se om man kan få bedre resultater, da omridset er et lille område at sammenligne.



Figur B.4: Billede af en brik hvor brikken er gjort hvid

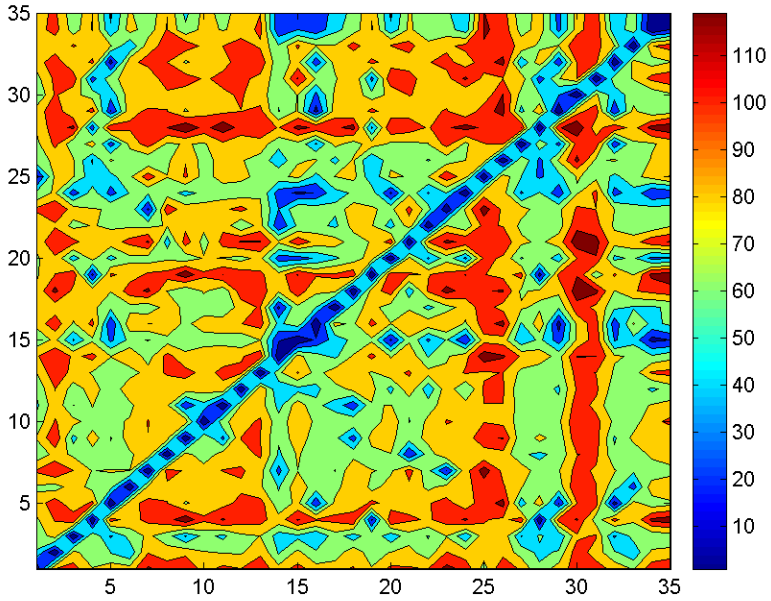
Analyseprogrammet bliver fodret med tre A4 ark, der er scannet med 600 dpi. De 35 brikker, som er på de indscannede ark bliver gemt i 35 Bitmap billede, som fylder 2000X2000 pix. Efter programmet har analyseret brikkerne har man en datafil med summen fra alle sammenligningerne. Ud fra datafilen kan man så lave en plot ligesom på Figur B.5. Da værdiernes sum er meget store er der blevet divideret med en milion for at få en pænere graf.



Figur B.5: Contour plot af analyseresultat fra sammenligninger af omrids

Det man ser på Figur B.5 er et contuor plot af resultaterne fra analysen af brikkerne i omrids. Hen af X-aksen er det de brikker man tester og op af Y-aksen er det de brikker man trækker fra med. Den blå stribe der går igennem plotteren er der hvor brikkerne bliver sammenlignet med sig selv og summen vil givetvis altid give nul.

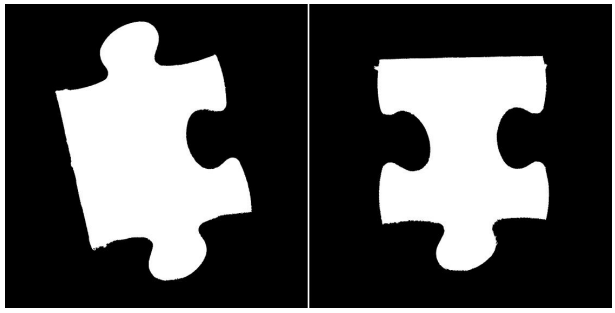
Hvis man ser på Figur B.6 ser man analysen med fyldte brikker og her kan man se stor forskel i forhold til Figur B.5. Det skyldes at man nu regner med hele fladen af brikken, så man har flere pixles at sammenligne med.



Figur B.6: Contour plot af analyseresultat fra sammenligninger af brikkernes flader

Hvis man finder det sted på plottet hvor brik 5 - brik 3 ser det ud til at værdien er nul. Det er den imidlertid ikke da værdien er 37,33. Hvis man vender den om så det er brik 3 - brik 5 får man 81,89. Dette vil ske for mange af brikkerne, de er forskellige men dog kan billedsummen komme meget langt ned.

Grunden til at billedesummen kan komme langt ned er, at når man trækker to billeder fra hinanden vil de pixles, hvor der bliver trukket værdien nul fra altid blive tildelt nul. Hvis man ser på Figur B.7 kan man se at brik 5 - til højre, har mere sort omkring sig end brik 3 og hvis man gør det omvendt vil der være mere af brik 3 tilbage og derved en større sum værdi at billedet.



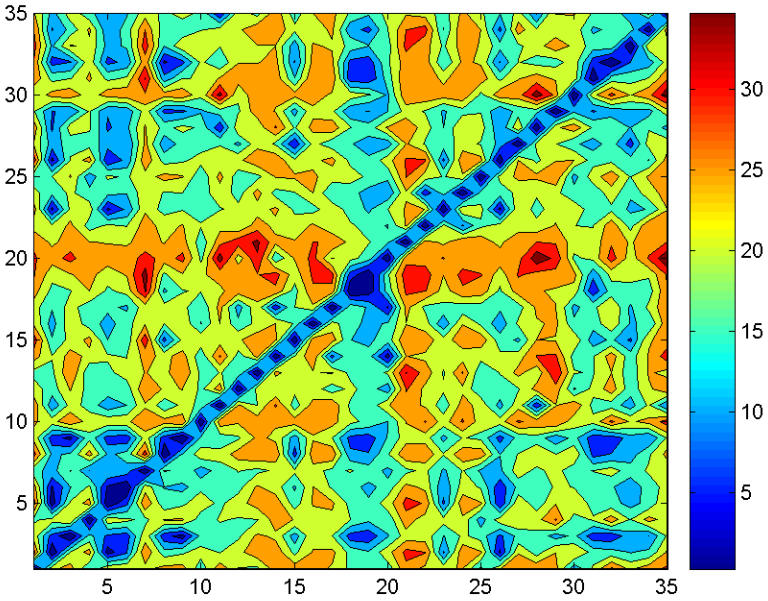
Figur B.7: Til venter brik 3 og til højer brik 5

Udover at nogle brikker kan se ens ud kan man se på plottet at brik 16, 29 og 32 er meget mere ens, da alle tre har lave sumværdier, når man trækker dem fra hinanden. Hvis man ser i Bilag **??** kan man se alle brikkerne og her fremgår det også at brik 16, 29, 32 ligner hinanden.

B.3 Afrunding på almindeligt puslespil

Hvis man ser tilbage på det, der er blevet gennemgået i dette kapital, kan man konkludere at der skal helt andre metoder til, når man skal havde computeren til at samle et puslespil. Som menneske ser man tit efter farve og ikke så meget efter facon. De fleste metoder ville ikke kunne bruges. Dog vil den metode, som anvendes til at aflæse brikkens geometriske form kunne overføres til computereprogrammet. En mulig udvidelse til programmmet kunne være at gøre algoritmen i stand til at vurdere farver og nuancer.

Analysen af de brikker, som skal bruges til at teste algoritmen undervejs, er kommet frem til det ønskede resultat. Der er mange af brikkerne, som er meget tæt på at være ens. Dog får man det bedste resultat, hvis man scanner i en højere opløsning. I så tilfælde tager det blot meget længere tid for analysen at køre alle brikkerne igennem. Resultatet fra en scanning ved 300dpi kan godt bruges til en start. På Figur B.8 herunder kan man se, at der er mange flere brikker, som er tæt på at være ens, når de er scannet ind ved 300dpi. Grunden til det er, at når der er færre pixles at arbejde med bliver billedet mindre detaljeret og derfor grovere.



Figur B.8: Contour plot af analyseresultat fra sammenligninger af brikkernes flader i 300dpi

Der kan ikke laves en sammenligning mellem Figur B.8 og Figur B.6 fordi analysen er udført efter to forskellige scanninger, så brikkerne ligger ikke det samme sted i begge scanninger.

Eftersom analysen har vist, at ingen af brikkerne er ens behøver man ikke tage højde for dette i første omgang. Plottene kan hjælpe med til at give en forklaring, hvis algoritmen ved en fejl lægger en brik forkert. Mange af de metoder, som bliver brugt til analysen af brikkerne, vil også blive brugt til den kommende algoritme. En af gengangerne vil være metoden til finde omrisdet af brikkerne, da disse data kan give en masse information om brikkerne i algoritmen.

Bilag C

Kildekode

Der er kun en fuld kode fra det første program, som ikke er optimeret. I det optimerede program er der kun filen "PuzzleAlgo4.cs", da alle andre filer, som er med i programmet, er ens med det første program.

C.1 Koden til algoritmen

C.1.1 PuzzleAlgo3.cs

```
1  using System;
2  using System.Drawing;
3  using System.Collections;
4  using System.ComponentModel;
5  using System.Windows.Forms;
6  using System.Data;
7  using System.Drawing.Imaging;
8  using System.IO;
9
10 namespace PuzzleAlgo3
11 {
12  /// <summary>
13  /// Summary description for Form1.
14  /// </summary>
15  public class Form1 : System.Windows.Forms.Form
16  {
```

```
17    private System.Windows.Forms.ListBox listBox1;
18    private System.Windows.Forms.Button btStart;
19    private ArrayList aPuzzle = new ArrayList();
20          private PuzzleItemsSide[] aPuzzleItemsSide = new
                PuzzleItemsSide[0];
21          private int muh = 0;
22          private int muh2 = 0;
23          private ArrayList ListPutPuzzle = new ArrayList();
24    private Form2 mForm2 = new Form2();
25          private float arealglob;
26          private float xGglob;
27          private float yGglob;
28          private int mCon, mFileCon = 0;
29          private DateTime mTimerStart, mTimerEnd;
30          private float mTol = 1.5f;
31          private double mAreal = (1000 * 1000);
32          private ArrayList ListBefore = new ArrayList();
33          private int Liste = 0;
34    private System.ComponentModel.Container components =
          null;
35
36    public Form1()
37    {
38     InitializeComponent();
39    }
40
41    /// <summary>
42    /// Clean up any resources being used.
43    /// </summary>
44    protected override void Dispose( bool disposing )
45    {
46     if( disposing )
47     {
48      if (components != null)
49      {
50       components.Dispose();
51      }
52     }
53     base.Dispose( disposing );
54    }
55
56    #region Windows Form Designer generated code
```

```
57      /// <summary>
58      /// Required method for Designer support - do not modify
59      /// the contents of this method with the code editor.
60      /// </summary>
61      private void InitializeComponent()
62      {
63              this.listBox1 = new System.Windows.Forms.
                    ListBox();
64              this.btStart = new System.Windows.Forms.Button
                    ();
65              this.SuspendLayout();
66              //
67              // listBox1
68              //
69              this.listBox1.Location = new System.Drawing.
                    Point(8, 16);
70              this.listBox1.Name = "listBox1";
71              this.listBox1.Size = new System.Drawing.Size
                    (373, 238);
72              this.listBox1.TabIndex = 0;
73              //
74              // btStart
75              //
76              this.btStart.Location = new System.Drawing.
                    Point(8, 264);
77              this.btStart.Name = "btStart";
78              this.btStart.Size = new System.Drawing.Size
                    (75, 23);
79              this.btStart.TabIndex = 1;
80              this.btStart.Text = "Start";
81              this.btStart.Click += new System.EventHandler(
                    this.btStart_Click);
82              //
83              // Form1
84              //
85              this.AutoScaleBaseSize = new System.Drawing.
                    Size(5, 13);
86              this.ClientSize = new System.Drawing.Size(393,
                     294);
87              this.Controls.Add(this.btStart);
88              this.Controls.Add(this.listBox1);
89              this.Name = "Form1";
```

```
90              this.Text = "Form1";
91              this.ResumeLayout(false);
92
93      }
94      #endregion
95
96      /// <summary>
97      /// The main entry point for the application.
98      /// </summary>
99      [STAThread]
100     static void Main()
101     {
102      Application.Run(new Form1());
103     }
104
105           /// <summary>
106           /// Knappen som der start hele lægningen at
                brikkerne
107           /// </summary>
108           private void btStart_Click(object sender, System.
                EventArgs e)
109           {
110               mTimerStart = DateTime.Now;
111
112               mTol = 1.5f;
113               mCon = 0;
114               mFileCon = 1;
115               string FileN = "Brik15";
116               string tmpFileName = FileN + mCon + ".csv";
117               aPuzzle.Clear();
118
119               // Indlæser alle brikkerne
120               while (File.Exists(tmpFileName))
121               {
122                   Puzzle tmpPuzzle = new Puzzle();
123                   StreamReader st = new StreamReader(
                        tmpFileName);
124                   string[] aStr = st.ReadLine().Split(',');
125
126                   aStr = st.ReadLine().Split(',');
127                   float xF = (float)Convert.ToDouble(aStr
                        [0].Replace('.', ','));
```

```
128                 float yF = (float)Convert.ToDouble(aStr
                        [1].Replace('.', ','));
129
130                 aStr = st.ReadLine().Split(',');
131                 float xE = (float)Convert.ToDouble(aStr
                        [0].Replace('.', ','));
132                 float yE = (float)Convert.ToDouble(aStr
                        [1].Replace('.', ','));
133
134                 float areal = xF * yE - xE * yF;
135
136                 float xG = (xF + xE) * (xF * yE - xE * yF)
                        ;
137                 float yG = (yF + yE) * (xF * yE - xE * yF)
                        ;
138
139                 int conG = 1;
140
141                 bool stop = false;
142
143                 int tmpSideCon = 0;
144                 // gremmer brikkernes sider i objeket og
                        regne areale ud på dem
145                 while (stop != true)
146                 {
147                     try
148                     {
149                         double tmpLen = Math.Sqrt((xF - xE
                                ) * (xF - xE) + (yF - yE) * (
                                yF - yE));
150
151                         tmpSideCon = tmpPuzzle.SetLength(
                                tmpLen, new PointPuzzle(xF, yF
                                ), new PointPuzzle(xE, yE));
152
153                         aStr = st.ReadLine().Split(',');
154                         xF = (float)Convert.ToDouble(aStr
                                [0].Replace('.', ','));
155                         yF = (float)Convert.ToDouble(aStr
                                [1].Replace('.', ','));
156
157                         aStr = st.ReadLine().Split(',');
```

```
158                         xE = (float)Convert.ToDouble(aStr
                                [0].Replace('.', ','));
159                         yE = (float)Convert.ToDouble(aStr
                                [1].Replace('.', ','));
160
161                         areal += xF * yE - xE * yF;
162
163                         xG += (xF + xE) * (xF * yE - xE *
                                yF);
164                         yG += (yF + yE) * (xF * yE - xE *
                                yF);
165                         conG += 1;
166
167
168                     }
169                     catch (Exception)
170                     {
171                         stop = true;
172                         st.Close();
173                     }
174                 }
175                 areal = Math.Abs(areal * 0.5f);
176
177                 float cenXF = (1 / (6 * areal)) * xG;
178                 float cenYF = (1 / (6 * areal)) * yG;
179
180                 int cenX = Convert.ToInt32(cenXF);
181                 int cenY = Convert.ToInt32(cenYF);
182
183                 double radius = Math.Sqrt(areal / Math.PI)
                        ;
184                 listBox1.Items.Add("Brik : " + mCon + "
                        Areal : " + areal + " (X,Y)=(" + cenX
                        + ", " + cenY + ")");
185
186                 tmpPuzzle.SetRadius(radius);
187                 tmpPuzzle.SetID(mCon);
188                 tmpPuzzle.SetCenter(cenX, cenY);
189
190                 aPuzzle.Add(tmpPuzzle);
191
192                 mCon++;
```

```
193
194                 tmpFileName = FileN + mCon + ".csv";
195             }
196
197             // Køre Coktalmetoden til den ikke kan køres
                    mere
198             while (MakePuzzle())
199             {
200                 listBox1.Items.Add("Engang til....");
201             }
202
203             // Køre backtrackingmetoden fra den første
                    brik som ikke er blive lagt.
204             for (int i = 0; i < aPuzzle.Count; i++)
205             {
206                 if (!((Puzzle)aPuzzle[i]).GetUse())
207                 {
208                     ((Puzzle)aPuzzle[i]).SetPut(true);
209                     ListPutPuzzle.Add(i);
210                     FindPuzzle(i, new PuzzleItemsSide[0],
                            0);
211                     ((Puzzle)aPuzzle[i]).SetPut(false);
212                     break;
213                 }
214             }
215
216             mTimerEnd = DateTime.Now;
217
218             listBox1.Items.Add("Timer : " + (mTimerEnd -
                    mTimerStart));
219
220             mForm2.Show();
221             mForm2.ViewArray(aPuzzle);
222
223             mForm2.SaveSolution(0);
224
225         }
226
227         /// <summary>
228         /// Funktion der finde brikker som der kan sam
                samlter brikker
229         /// </summary>
```

```
230         private bool TiePuzzle(int marsterPuzzleItem, int
                marsterPuzzleSide)
231         {
232             muh++;
233             float marsterDeg = ((Puzzle)aPuzzle[
                    marsterPuzzleItem]).GetDeg(
                    marsterPuzzleSide);
234
235             PointPuzzle marsterPuzzleItemsCenter = ((
                    Puzzle)aPuzzle[marsterPuzzleItem]).
                    GetCenter();
236             ArrayList marsterPuzzleItemsCenterList = ((
                    Puzzle)aPuzzle[marsterPuzzleItem]).
                    GetCenterList();
237
238             double marsterPuzzleSideLen = ((Puzzle)aPuzzle
                    [marsterPuzzleItem]).GetLength(
                    marsterPuzzleSide);
239             PointPuzzle marsterPuzzleItemsStartPoint = ((
                    Puzzle)aPuzzle[marsterPuzzleItem]).
                    GetStartPoint(marsterPuzzleSide);
240             PointPuzzle marsterPuzzleItemsEndPoint = ((
                    Puzzle)aPuzzle[marsterPuzzleItem]).
                    GetEndPoint(marsterPuzzleSide);
241             int marsterCon = ((Puzzle)aPuzzle[
                    marsterPuzzleItem]).GetCount();
242
243             // finde brikker som der kan lige til
                    marsterPuzzleSide
244             ArrayList list_side = new ArrayList();
245             list_side = FindPuzzleSide(false,
                    marsterPuzzleSideLen, marsterPuzzleItem);
246
247             // hvis der kun findes en så kan der måske sam
                     samlse
248             if (list_side.Count == 1)
249             {
250                 int puzzleItems = ((PuzzleItemsSide)
                        list_side[0]).masterPuzzle;
251                 int puzzleItemsSide = ((PuzzleItemsSide)
                        list_side[0]).masterPuzzleSide;
```

```
252                 int puzzleCon = ((Puzzle)aPuzzle[
                        puzzleItems]).GetCount();
253                 PointPuzzle puzzleItemsStartPoint = ((
                        Puzzle)aPuzzle[puzzleItems]).
                        GetStartPoint(puzzleItemsSide);
254                 PointPuzzle puzzleItemsEndPoint = ((Puzzle
                        )aPuzzle[puzzleItems]).GetEndPoint(
                        puzzleItemsSide);
255
256                 // roter brikken så brikken passe til
                        marsterPuzzleSide
257                 //((Puzzle)aPuzzle[puzzleItems]).Rotet(
                        marsterDeg, puzzleItemsSide);
258                 puzzleItemsStartPoint = ((Puzzle)aPuzzle[
                        puzzleItems]).GetStartPoint(
                        puzzleItemsSide);
259                 puzzleItemsEndPoint = ((Puzzle)aPuzzle[
                        puzzleItems]).GetEndPoint(
                        puzzleItemsSide);
260
261                 // flytter brikken til masterPuzzle
262                 ((Puzzle)aPuzzle[puzzleItems]).MoveToOrion
                        (marsterPuzzleItemsStartPoint,
                        puzzleItemsEndPoint);
263                 puzzleItemsStartPoint = ((Puzzle)aPuzzle[
                        puzzleItems]).GetStartPoint(
                        puzzleItemsSide);
264                 puzzleItemsEndPoint = ((Puzzle)aPuzzle[
                        puzzleItems]).GetEndPoint(
                        puzzleItemsSide);
265
266                 PointPuzzle puzzleItemsCenter = ((Puzzle)
                        aPuzzle[puzzleItems]).GetCenter();
267                 ArrayList puzzleItemsCenterList = ((Puzzle
                        )aPuzzle[puzzleItems]).GetCenterList()
                        ;
268
269                 // Ser er det to bikke passer sammen
270                 if (!LineTurn(((Puzzle)aPuzzle[puzzleItems
                        ]), ((Puzzle)aPuzzle[marsterPuzzleItem
                        ]), puzzleItemsSide, marsterPuzzleSide
                        )
```

```
271                     && ((marsterPuzzleItemsEndPoint.X >=
                            puzzleItemsStartPoint.X - mTol &&
                            marsterPuzzleItemsEndPoint.X <=
                            puzzleItemsStartPoint.X + mTol)
272                     && (marsterPuzzleItemsEndPoint.Y >=
                            puzzleItemsStartPoint.Y - mTol &&
                            marsterPuzzleItemsEndPoint.Y <=
                            puzzleItemsStartPoint.Y + mTol))
273                     && ((marsterPuzzleItemsStartPoint.X >=
                             puzzleItemsEndPoint.X - mTol &&
                            marsterPuzzleItemsStartPoint.X <=
                            puzzleItemsEndPoint.X + mTol)
274                     && (marsterPuzzleItemsStartPoint.Y >=
                            puzzleItemsEndPoint.Y - mTol &&
                            marsterPuzzleItemsStartPoint.Y <=
                            puzzleItemsEndPoint.Y + mTol)))
275                 {
276                     // finde de sider som de to brikker
                            har tilfælses
277                     ArrayList listSideAcitv = new
                            ArrayList();
278                     listSideAcitv = FindPuzzleSideToPuzzle
                            (marsterPuzzleItem, puzzleItems);
279
280                     // deaktiver de side de har til fælses
281                     for (int i = 0; i < listSideAcitv.
                            Count; i++)
282                     {
283                         int mPuzzleItems = ((
                                PuzzleItemsSideToPuzzle)
                                listSideAcitv[i]).masterPuzzle
                                ;
284                         int mPuzzleItemsSide = ((
                                PuzzleItemsSideToPuzzle)
                                listSideAcitv[i]).
                                masterPuzzleSide;
285                         int sPuzzleItems = ((
                                PuzzleItemsSideToPuzzle)
                                listSideAcitv[i]).PuzzleItm;
286                         int sPuzzleItemsSide = ((
                                PuzzleItemsSideToPuzzle)
                                listSideAcitv[i]).PuzzleSide;
```

```
287
288                         ((Puzzle)aPuzzle[mPuzzleItems]).
                                SetNeighbourPuzzle(
                                mPuzzleItemsSide, sPuzzleItems
                                , sPuzzleItemsSide);
289                         ((Puzzle)aPuzzle[mPuzzleItems]).
                                SideDeactiv(mPuzzleItemsSide);
290                         ((Puzzle)aPuzzle[sPuzzleItems]).
                                SetNeighbourPuzzle(
                                sPuzzleItemsSide, mPuzzleItems
                                , mPuzzleItemsSide);
291                         ((Puzzle)aPuzzle[sPuzzleItems]).
                                SideDeactiv(sPuzzleItemsSide);
292                     }
293
294                     ((Puzzle)aPuzzle[marsterPuzzleItem]).
                            SetUse(true);
295                     ((Puzzle)aPuzzle[puzzleItems]).SetUse(
                            true);
296
297                     try
298                     {
299                         int con = (puzzleCon + marsterCon)
                                 - (listSideAcitv.Count * 2);
300                         arealglob = 0;
301                         xGglob = 0;
302                         yGglob = 0;
303                         Puzzle tmpPuzzle = new Puzzle();
304
305                         // samlter de to brikker sammen og
                                finde ny ID til den nye brik
306                         //tmpPuzzle = AddPuzzleToPuzzle(
                                marsterPuzzleItem, 0, 0, con,
                                tmpPuzzle);
307                         tmpPuzzle = AddPuzzleToPuzzle(
                                puzzleItems, 0, 0, con,
                                tmpPuzzle);
308                         tmpPuzzle.SetID(aPuzzle.Count);
309
310                         // regne arealet ud på den nye
                                brik fra den globale variable
```

```
311                     arealglob = Math.Abs(arealglob *
                            0.5f);
312
313                     // regne centum ud på den ny brik
314                     float cenXF = (1 / (6 * arealglob)
                            ) * xGglob;
315                     float cenYF = (1 / (6 * arealglob)
                            ) * yGglob;
316                     int cenX = Convert.ToInt32(cenXF);
317                     int cenY = Convert.ToInt32(cenYF);
318
319                     double radius = Math.Sqrt(
                            arealglob / Math.PI);
320                     listBox1.Items.Add("Brik : " +
                            aPuzzle.Count + " Areal : " +
                            arealglob + " (X,Y)=(" + cenX
                            + ", " + cenY + ")");
321
322                     tmpPuzzle.SetRadius(radius);
323                     tmpPuzzle.SetCenter(cenX, cenY);
324
325                     // sam samlter sider som har den
                            samme vinkle
326                     tmpPuzzle = ComplexPuzzleItems(
                            tmpPuzzle);
327
328                     // overføre centumerne til den nye
                             brik og overføre ikke sam
                            samlte brikker centumer
329                     if (mCon > marsterPuzzleItem)
330                         tmpPuzzle.AddCenterToList(
                                marsterPuzzleItem,
                                marsterPuzzleItemsCenter);
331                     if (mCon > puzzleItems)
332                         tmpPuzzle.AddCenterToList(
                                puzzleItems,
                                puzzleItemsCenter);
333
334                     for (int i = 0; i <
                            marsterPuzzleItemsCenterList.
                            Count; i++)
335                     {
```

```
336                         tmpPuzzle.AddCenterToList(((
                                PuzzleItemsCenter)
                                marsterPuzzleItemsCenterList
                                [i]).nr,
337                              ((PuzzleItemsCenter)
                                    marsterPuzzleItemsCenterList
                                    [i]).cen);
338                     }
339
340                     for (int i = 0; i <
                            puzzleItemsCenterList.Count; i
                            ++)
341                     {
342                         tmpPuzzle.AddCenterToList(((
                                PuzzleItemsCenter)
                                puzzleItemsCenterList[i]).
                                nr,
343                              ((PuzzleItemsCenter)
                                    puzzleItemsCenterList[
                                    i]).cen);
344                     }
345                     // tilføje nye brik til listen.
346                     aPuzzle.Add(tmpPuzzle);
347
348                     listBox1.Items.Add("Brik : " +
                            marsterPuzzleItem + " side : "
                             + marsterPuzzleSide +
349                          " Lige med brik : " +
                                puzzleItems + " side : " +
                                 puzzleItemsSide);
350
351                     return true;
352                 }
353                 catch (Exception e)
354                 {
355                     for (int i = 0; i < listSideAcitv.
                            Count; i++)
356                     {
357                         int mPuzzleItems = ((
                                PuzzleItemsSideToPuzzle)
                                listSideAcitv[i]).
                                masterPuzzle;
```

```
358                             int mPuzzleItemsSide = ((
                                    PuzzleItemsSideToPuzzle)
                                    listSideAcitv[i]).
                                    masterPuzzleSide;
359                             int sPuzzleItems = ((
                                    PuzzleItemsSideToPuzzle)
                                    listSideAcitv[i]).
                                    PuzzleItm;
360                             int sPuzzleItemsSide = ((
                                    PuzzleItemsSideToPuzzle)
                                    listSideAcitv[i]).
                                    PuzzleSide;
361
362                             ((Puzzle)aPuzzle[mPuzzleItems
                                    ]).SetNeighbourPuzzle(
                                    mPuzzleItemsSide, -1, -1);
363                             ((Puzzle)aPuzzle[mPuzzleItems
                                    ]).SideActiv(
                                    mPuzzleItemsSide);
364                             ((Puzzle)aPuzzle[sPuzzleItems
                                    ]).SetNeighbourPuzzle(
                                    sPuzzleItemsSide, -1, -1);
365                             ((Puzzle)aPuzzle[sPuzzleItems
                                    ]).SideActiv(
                                    sPuzzleItemsSide);
366                         }
367
368                         ((Puzzle)aPuzzle[marsterPuzzleItem
                                ]).SetUse(false);
369                         ((Puzzle)aPuzzle[puzzleItems]).
                                SetUse(false);
370                     }
371                 }
372                 else
373                 {
374                     ((Puzzle)aPuzzle[puzzleItems]).
                            SetOrionBack();
375                 }
376             }
377
378             return false;
379         }
```

```
380
381         /// <summary>
382         /// Funktion start coktalmetoden
383         /// </summary>
384         private bool MakePuzzle()
385         {
386             bool tmpbool = false;
387
388             for (int j = 0; j < aPuzzle.Count; j++)
389             {
390
391                 for (int i = 0; i < ((Puzzle)aPuzzle[j]).
                        GetCount(); i++)
392                     if (!((Puzzle)aPuzzle[j]).GetUse())
393                     //if (((Puzzle)aPuzzle[j]).
                            GetSideActiv(i))
394                         if (TiePuzzle(j, i))
395                             tmpbool = true;
396             }
397             //Hvis der er sket en samsamlting så returnes
                    true så funktion kan gøre igen.
398             return tmpbool;
399         }
400
401         /// <summary>
402         /// Funktion som smalter brikker sammen
403         /// </summary>
404         private Puzzle AddPuzzleToPuzzle(int
                startPuzzleitm, int startSide, int con, int
                conMax, Puzzle puzzle)
405         {
406             Puzzle tmpPuzzle = puzzle;
407             int tmpCon = con;
408             int tmpMax = conMax;
409             int tmpSide = startSide;
410             int sidecon = ((Puzzle)aPuzzle[startPuzzleitm
                    ]).GetCount();
411
412             // her undersøg hvor mange sider der ikke er
                    aktive
413             while (!((Puzzle)aPuzzle[startPuzzleitm]).
                    GetSideActiv(tmpSide) && tmpSide < sidecon
```

```
                )
414             tmpSide++;
415
416         if (tmpSide >= sidecon)
417             tmpSide = 0;
418
419         // her ligge de sider der er aktive fra de to
                brikker over i det ny objekt
420         while (tmpCon < tmpMax && ((Puzzle)aPuzzle[
                startPuzzleitm]).GetSideActiv(tmpSide))
421         {
422             tmpPuzzle.SetLength(((Puzzle)aPuzzle[
                    startPuzzleitm]).GetLength(tmpSide),
423                             ((Puzzle)aPuzzle[
                                startPuzzleitm]).
                                GetStartPoint(tmpSide),
424                             ((Puzzle)aPuzzle[
                                startPuzzleitm]).
                                GetEndPoint(tmpSide));
425
426             float xF = ((Puzzle)aPuzzle[startPuzzleitm
                    ]).GetStartPoint(tmpSide).X;
427             float yF = ((Puzzle)aPuzzle[startPuzzleitm
                    ]).GetStartPoint(tmpSide).Y;
428             float xE = ((Puzzle)aPuzzle[startPuzzleitm
                    ]).GetEndPoint(tmpSide).X;
429             float yE = ((Puzzle)aPuzzle[startPuzzleitm
                    ]).GetEndPoint(tmpSide).Y;
430
431
432             arealglob += xF * yE - xE * yF;
433
434             xGglob += (xF + xE) * (xF * yE - xE * yF);
435             yGglob += (yF + yE) * (xF * yE - xE * yF);
436
437             tmpCon++;
438             tmpSide++;
439
440             if (tmpSide >= sidecon)
441                 tmpSide = 0;
442         }
443
```

```
444             if (tmpCon >= tmpMax)
445                 return tmpPuzzle;
446
447             // hvis der er flere side klades funktionen
                    igen
448             int newPuzzleItems = ((Puzzle)aPuzzle[
                    startPuzzleitm]).GetNeighbourPuzzle(
                    tmpSide);
449             int newPuzzleItemsSide = ((Puzzle)aPuzzle[
                    startPuzzleitm]).GetNeighbourPuzzleSide(
                    tmpSide)+1;
450
451             return AddPuzzleToPuzzle(newPuzzleItems,
                    newPuzzleItemsSide, tmpCon, conMax,
                    tmpPuzzle);
452         }
453
454         /// <summary>
455         /// Funktion som finde sider med samme viklen og
                lave dem om en lang side
456         /// </summary>
457         private Puzzle ComplexPuzzleItems(Puzzle puzzle)
458         {
459             Puzzle tmpPuzzle = puzzle;
460
461             PointPuzzle startPk = tmpPuzzle.GetStartPoint
                    (0);
462             PointPuzzle endPk = tmpPuzzle.GetEndPoint(0);
463             PointPuzzle tmpPk = endPk - startPk;
464
465             // Finde vinkel på den første side
466             float newDeg = (float)Math.Acos(tmpPk.X / Math
                    .Sqrt(tmpPk.X * tmpPk.X + tmpPk.Y * tmpPk.
                    Y));
467             if (tmpPk.Y > 0)
468                 newDeg = (float)((Math.PI * 2 - newDeg) /
                        Math.PI) * 180.0f;
469             else
470                 newDeg = (float)(newDeg / Math.PI) * 180.0
                        f;
471
472             float oldDeg = newDeg;
```

```
473
474             int k = 1;
475             float tmpTol = 0.5f;
476             while ( k < tmpPuzzle.GetCount()+1)
477             {
478                 int tmpK = k;
479                 if (k == tmpPuzzle.GetCount())
480                     tmpK = 0;
481                 startPk = tmpPuzzle.GetStartPoint(tmpK);
482                 endPk = tmpPuzzle.GetEndPoint(tmpK);
483                 tmpPk = endPk - startPk;
484
485                 // Finde vinkel på den næste side
486                 newDeg = (float)Math.Acos(tmpPk.X / Math.
                        Sqrt(tmpPk.X * tmpPk.X + tmpPk.Y *
                        tmpPk.Y));
487                 if (tmpPk.Y > 0)
488                     newDeg = (float)((Math.PI * 2 - newDeg
                            ) / Math.PI) * 180.0f;
489                 else
490                     newDeg = (float)(newDeg / Math.PI) *
                            180.0f;
491
492                 listBox1.Items.Add("Brik : " + tmpPuzzle.
                        GetID() + " Side : " + (k - 1) + "
                        Vinkel : "
493                         + oldDeg.ToString() + " Side : " + k +
                                " Vinkel : " + newDeg.ToString())
                            ;
494
495                 float tmp = newDeg;
496
497                 if(oldDeg > 180)
498                     tmp = 360 - oldDeg;
499                 else
500                     tmp = 180 - oldDeg;
501
502                 // Se om det to side har den samme vinkle,
                        hvis den er ligge siderne sammen.
503                 if ((newDeg > (oldDeg - tmpTol) && newDeg
                        < (oldDeg + tmpTol))
```

```
504                     || (newDeg > (tmp - tmpTol) && newDeg
                            < (tmp + tmpTol)))
505                 {
506                     tmpPuzzle.SetEndPoint(tmpPuzzle.
                            GetLength(tmpK - 1) + tmpPuzzle.
                            GetLength(tmpK), tmpPuzzle.
                            GetEndPoint(tmpK), tmpK - 1);
507                     tmpPuzzle.DelSide(tmpK);
508
509                 }
510                 else
511                 {
512                     k++;
513                 }
514
515                 oldDeg = newDeg;
516
517             }
518
519             return tmpPuzzle;
520         }
521
522         /// <summary>
523         /// Backtrackting funktionen
524         /// </summary>
525         private bool FindPuzzle(int puzzleNr,
                PuzzleItemsSide[] inLength, int startIndex)
526         {
527             muh2++;
528
529             //regne den nye liste ud
530             Liste = Liste * 10 + puzzleNr;
531             ArrayList SideActiv = new ArrayList();
532
533             //undersøger om den lige har være lavde før
534             bool stopL = false;
535             for (int i = 0; i < ListBefore.Count; i++)
536                 if (Liste == ((int)ListBefore[i]))
537                     stopL = true;
538
539             if (stopL)
540                 return false;
```

```
541
542             // ser om hvor mange brikker som der er blive
                    lagt, hvis der er alle så gemmes en
                    løsning
543             int con = 0;
544             for (int i = 1; i < aPuzzle.Count; i++)
545             {
546                 if (!((Puzzle)aPuzzle[i]).GetUse() && !((
                        Puzzle)aPuzzle[i]).GetPut())
547                 {
548                     con++;
549                 }
550             }
551
552             if (con <= 1)
553             {
554                 //SaveSolution(mFileCon);
555                 SaveSolutionInText(mFileCon);
556                 mFileCon++;
557             }
558
559             // overføre side på brikken til listen om side
                     som der kan ligges til
560             int tmpArrayCon = inLength.Length;
561             tmpArrayCon += ((Puzzle)aPuzzle[puzzleNr]).
                    GetCount();
562
563             PuzzleItemsSide[] activSideArray = new
                    PuzzleItemsSide[tmpArrayCon];
564
565             for (int i = 0; i < inLength.Length; i++)
566                 activSideArray[i] = inLength[i];
567
568             for (int i = 0; i < ((Puzzle)aPuzzle[puzzleNr
                    ]).GetCount(); i++)
569             {
570                 PuzzleItemsSide tmpPIS = new
                        PuzzleItemsSide();
571                 tmpPIS.len = ((Puzzle)aPuzzle[puzzleNr]).
                        GetLength(i);
572                 tmpPIS.masterPuzzle = puzzleNr;
573                 tmpPIS.masterPuzzleSide = i;
```

```
574
575                 activSideArray[i + inLength.Length] =
                        tmpPIS;
576             }
577
578             //-------------- Finde side som ligge og
                    passer til brikken --------
579
580             for(int i=0; i<ListPutPuzzle.Count; i++)
581             {
582                 int tmpPuzzle = ((int)ListPutPuzzle[i]);
583
584                 ArrayList listSideAcitv = new ArrayList();
585                 listSideAcitv = FindPuzzleSideToPuzzle(
                        puzzleNr, tmpPuzzle);
586
587                 for (int j = 0; j < listSideAcitv.Count; j
                        ++)
588                 {
589                     SideActiv.Add(listSideAcitv[j]);
590                     int mPuzzleItems = ((
                            PuzzleItemsSideToPuzzle)
                            listSideAcitv[j]).masterPuzzle;
591                     int mPuzzleItemsSide = ((
                            PuzzleItemsSideToPuzzle)
                            listSideAcitv[j]).masterPuzzleSide
                            ;
592                     int sPuzzleItems = ((
                            PuzzleItemsSideToPuzzle)
                            listSideAcitv[j]).PuzzleItm;
593                     int sPuzzleItemsSide = ((
                            PuzzleItemsSideToPuzzle)
                            listSideAcitv[j]).PuzzleSide;
594
595                     ((Puzzle)aPuzzle[mPuzzleItems]).
                            SetNeighbourPuzzle(
                            mPuzzleItemsSide, sPuzzleItems,
                            sPuzzleItemsSide);
596                     ((Puzzle)aPuzzle[mPuzzleItems]).
                            SideDeactiv(mPuzzleItemsSide);
597                     ((Puzzle)aPuzzle[sPuzzleItems]).
                            SetNeighbourPuzzle(
```

```
                        sPuzzleItemsSide, mPuzzleItems,
                        mPuzzleItemsSide);
598                 ((Puzzle)aPuzzle[sPuzzleItems]).
                        SideDeactiv(sPuzzleItemsSide);
599             }
600
601         }
602
603         //--------------- her start findingen at
                brikker tilsiderne --------
604
605         for (int i = startIndex; i < activSideArray.
                Length; i++)
606         {
607             int marsterPuzzle = activSideArray[i].
                    masterPuzzle;
608             int marsterPuzzleSide = activSideArray[i].
                    masterPuzzleSide;
609             float marsterDeg = ((Puzzle)aPuzzle[
                    marsterPuzzle]).GetDeg(
                    marsterPuzzleSide);
610             double marsterPuzzleSideLen = ((Puzzle)
                    aPuzzle[marsterPuzzle]).GetLength(
                    marsterPuzzleSide);
611             PointPuzzle marsterPuzzleItemsStartPoint =
                     ((Puzzle)aPuzzle[marsterPuzzle]).
                    GetStartPoint(marsterPuzzleSide);
612             PointPuzzle marsterPuzzleItemsEndPoint =
                    ((Puzzle)aPuzzle[marsterPuzzle]).
                    GetEndPoint(marsterPuzzleSide);
613
614             if (((Puzzle)aPuzzle[marsterPuzzle]).
                    GetSideActiv(marsterPuzzleSide))
615             {
616                 // finde sider til den aktive side
617                 ArrayList tmpArray = new ArrayList();
618                 tmpArray = FindPuzzleSide(false,
                        marsterPuzzleSideLen,
                        marsterPuzzle);
619
620                 // køre listen igen for hver side som
                        der er blive funde.
```

```
621                 for (int j = 0; j < tmpArray.Count; j
                        ++)
622                 {
623                     bool stop = false;
624                     int tmpPuzzle = ((PuzzleItemsSide)
                            tmpArray[j]).masterPuzzle;
625                     int tmpPuzzleSide = ((
                            PuzzleItemsSide)tmpArray[j]).
                            masterPuzzleSide;
626                     PointPuzzle
                            tmpPuzzleItemsStartPoint = ((
                            Puzzle)aPuzzle[tmpPuzzle]).
                            GetStartPoint(tmpPuzzleSide);
627                     PointPuzzle tmpPuzzleItemsEndPoint
                             = ((Puzzle)aPuzzle[tmpPuzzle
                            ]).GetEndPoint(tmpPuzzleSide);
628
629                     // hvis brikken ikke er blive lagt
                             bliver den roter og flytte
                            til marster brikken
630                     if (!((Puzzle)aPuzzle[tmpPuzzle]).
                            GetPut())
631                     {
632                         //((Puzzle)aPuzzle[tmpPuzzle])
                                .Rotet(marsterDeg,
                                tmpPuzzleSide);
633                         tmpPuzzleItemsStartPoint = ((
                                Puzzle)aPuzzle[tmpPuzzle])
                                .GetStartPoint(
                                tmpPuzzleSide);
634                         tmpPuzzleItemsEndPoint = ((
                                Puzzle)aPuzzle[tmpPuzzle])
                                .GetEndPoint(tmpPuzzleSide
                                );
635
636                         ((Puzzle)aPuzzle[tmpPuzzle]).
                                MoveToOrion(
                                marsterPuzzleItemsStartPoint
                                , tmpPuzzleItemsEndPoint);
637                         tmpPuzzleItemsStartPoint = ((
                                Puzzle)aPuzzle[tmpPuzzle])
                                .GetStartPoint(
```

```
                    tmpPuzzleSide);
638                 tmpPuzzleItemsEndPoint = ((
                        Puzzle)aPuzzle[tmpPuzzle])
                        .GetEndPoint(tmpPuzzleSide
                        );
639
640                 for (int k = 0; k <
                        ListPutPuzzle.Count; k++)
641                 {
642                     if (RadiusMetode(((Puzzle)
                            aPuzzle[tmpPuzzle]),
                            ((Puzzle)aPuzzle[((int
                            )ListPutPuzzle[k])])))
643                         stop = true;
644                 }
645             }
646             else
647                 stop = true;
648
649             // Under søger om brikken kan
                    ligge sammen.
650             if (!stop && ((Puzzle)aPuzzle[
                tmpPuzzle]).GetSideActiv(
                tmpPuzzleSide)
651             && !LineTurn(((Puzzle)aPuzzle[
                tmpPuzzle]), ((Puzzle)aPuzzle[
                marsterPuzzle]), tmpPuzzleSide
                , marsterPuzzleSide)
652             && ((marsterPuzzleItemsEndPoint.X
                >= tmpPuzzleItemsStartPoint.X
                - mTol &&
                marsterPuzzleItemsEndPoint.X
                <= tmpPuzzleItemsStartPoint.X
                + mTol)
653             && (marsterPuzzleItemsEndPoint.Y
                >= tmpPuzzleItemsStartPoint.Y
                - mTol &&
                marsterPuzzleItemsEndPoint.Y
                <= tmpPuzzleItemsStartPoint.Y
                + mTol))
654             && ((marsterPuzzleItemsStartPoint.
                X >= tmpPuzzleItemsEndPoint.X
```

```
                    - mTol &&
                    marsterPuzzleItemsStartPoint.X
                     <= tmpPuzzleItemsEndPoint.X +
                     mTol)
655                 && (marsterPuzzleItemsStartPoint.Y
                     >= tmpPuzzleItemsEndPoint.Y -
                     mTol &&
                    marsterPuzzleItemsStartPoint.Y
                     <= tmpPuzzleItemsEndPoint.Y +
                     mTol)))
656                 {
657                     //deaktiver siderne og klader
                            FindPuzzle funktion igen
                            for at finde nye brikker
658                     ((Puzzle)aPuzzle[tmpPuzzle]).
                            SideDeactiv(tmpPuzzleSide)
                            ;
659                     ((Puzzle)aPuzzle[marsterPuzzle
                            ]).SideDeactiv(
                            marsterPuzzleSide);
660
661                     bool tmpPut = ((Puzzle)aPuzzle
                            [tmpPuzzle]).GetPut();
662                     ((Puzzle)aPuzzle[tmpPuzzle]).
                            SetPut(true);
663
664                     ListPutPuzzle.Add(tmpPuzzle);
665
666                     FindPuzzle(tmpPuzzle,
                            activSideArray, i + 1);
667
668                     ListPutPuzzle.RemoveAt(
                            ListPutPuzzle.Count - 1);
669
670                     ((Puzzle)aPuzzle[tmpPuzzle]).
                            SetPut(tmpPut);
671                     ((Puzzle)aPuzzle[tmpPuzzle]).
                            SideActiv(tmpPuzzleSide);
672                     ((Puzzle)aPuzzle[marsterPuzzle
                            ]).SideActiv(
                            marsterPuzzleSide);
673                 }
```

```
674
675                         stop = false;
676                     }
677                 }
678             }
679             // Aktiver siderne igen
680             for (int i = 0; i < SideActiv.Count; i++)
681             {
682                 int mPuzzleItems = ((
                        PuzzleItemsSideToPuzzle)SideActiv[i]).
                        masterPuzzle;
683                 int mPuzzleItemsSide = ((
                        PuzzleItemsSideToPuzzle)SideActiv[i]).
                        masterPuzzleSide;
684                 int sPuzzleItems = ((
                        PuzzleItemsSideToPuzzle)SideActiv[i]).
                        PuzzleItm;
685                 int sPuzzleItemsSide = ((
                        PuzzleItemsSideToPuzzle)SideActiv[i]).
                        PuzzleSide;
686
687                 ((Puzzle)aPuzzle[mPuzzleItems]).SideActiv(
                        mPuzzleItemsSide);
688                 ((Puzzle)aPuzzle[sPuzzleItems]).SideActiv(
                        sPuzzleItemsSide);
689             }
690
691             // trækker brikken fra listen
692             ListBefore.Add(Liste);
693             Liste -= puzzleNr;
694             Liste = Liste / 10;
695
696             return false;
697         }
698
699         /// <summary>
700         /// Det er brikkerne overlapper
701         /// </summary>
702     private bool LineTurn(Puzzle p1, Puzzle p2, int sideP1,
            int sideP2)
703     {
704             PointPuzzle cenP1 = p1.GetCenter();
```

```
705             PointPuzzle cenP2 = p2.GetCenter();
706
707             PointPuzzle midP = p1.GetEndPoint(sideP1) + p1
                    .GetStartPoint(sideP1);
708             midP.X = midP.X / 2;
709             midP.Y = midP.Y / 2;
710
711             int p1_NSEW = Kompas(midP, cenP1);
712             int p2_NSEW = Kompas(midP, cenP2);
713
714             if (p1_NSEW == p2_NSEW)
715             {
716                 listBox1.Items.Add("Brik : " + p1.GetID()
                        + " Side : " + sideP1 + " / Ret : "
717                         + p1_NSEW + " og Brik : " + p2.GetID()
                             + " Side : " + sideP2 + " / Ret :
                             " + p2_NSEW);
718                 return true;
719             }
720
721             if (RadiusMetode(p1, p2))
722             {
723                 listBox1.Items.Add("Brik : " + p1.GetID()
                        + " Side : " + sideP1 + " / Ret : "
724                         + p1_NSEW + " og Brik : " + p2.GetID()
                             + " Side : " + sideP2 + " / Ret :
                             " + p2_NSEW);
725                 return true;
726             }
727
728             return false;
729     }
730
731         /// <summary>
732         /// RadiusMetode
733         /// </summary>
734         private bool RadiusMetode(Puzzle p1, Puzzle p2)
735         {
736             PointPuzzle cenP1 = p1.GetCenter();
737             PointPuzzle cenP2 = p2.GetCenter();
738
```

```
739             double tmpLen = Math.Sqrt((cenP1.X - cenP2.X)
                    * (cenP1.X - cenP2.X) + (cenP1.Y - cenP2.Y
                    ) * (cenP1.Y - cenP2.Y));
740             double r1 = p1.GetRadius();
741             double r2 = p2.GetRadius();
742
743             if (r1 > tmpLen || r2 > tmpLen)
744             {
745                 return true;
746             }
747
748             return false;
749         }
750
751         /// <summary>
752         /// KompasMetode
753         /// </summary>
754         private int Kompas(PointPuzzle mid, PointPuzzle
                cen)
755         {
756             int NSWE = 0;
757
758             if (mid.X > cen.X)
759                 NSWE = (NSWE + 7);
760             else if (mid.X < cen.X)
761                 NSWE = (NSWE + 3);
762
763             if (mid.Y < cen.Y)
764             {
765                 if (NSWE == 0)
766                     NSWE = (NSWE + 5);
767                 else
768                     NSWE = (NSWE + 5) / 2;
769             }
770             else if (mid.Y > cen.Y)
771             {
772                 if (NSWE == 0)
773                     NSWE = (NSWE + 1);
774                 else if (NSWE == 7)
775                     NSWE = (NSWE + 9) / 2;
776                 else
777                     NSWE = (NSWE + 1) / 2;
```

```
778             }
779
780             return NSWE;
781         }
782
783         /// <summary>
784         /// Finde sider om der passer til len
785         /// </summary>
786         private ArrayList FindPuzzleSide(bool small,
                double len, int NotPuzzle)
787         {
788             ArrayList tmpArr = new ArrayList();
789             float tol = 1.0f * mTol;
790
791             for (int i = 0; i < aPuzzle.Count; i++)
792             {
793                 if (NotPuzzle != i && !((Puzzle)aPuzzle[i
                        ]).GetUse())
794                 {
795                     for (int j = 0; j < ((Puzzle)aPuzzle[i
                            ]).GetCount(); j++)
796                     {
797                         double tmpError = len - ((Puzzle)
                                aPuzzle[i]).GetLength(j);
798
799                         if (small && ((Puzzle)aPuzzle[i]).
                                GetSideActiv(j) && (tmpError
                                <= tol && tmpError >= -tol))
800                             tmpArr.Add(new PuzzleItemsSide
                                    (i, j));
801                         else if (((Puzzle)aPuzzle[i]).
                                GetSideActiv(j) && (tmpError
                                <= tol && tmpError >= -tol))
802                             tmpArr.Add(new PuzzleItemsSide
                                    (i, j));
803                     }
804                 }
805
806
807             }
808
809             return tmpArr;
```

```
810         }
811
812         /// <summary>
813         /// Finde sider som brikkerne har tilfælles
814         /// </summary>
815         private ArrayList FindPuzzleSideToPuzzle(int
                puzzleItm, int marsterPuzzleItm)
816         {
817             ArrayList tmpArr = new ArrayList();
818             double tol = mTol;
819
820             for (int i = 0; i < ((Puzzle)aPuzzle[
                    marsterPuzzleItm]).GetCount(); i++)
821             {
822                 double len = ((Puzzle)aPuzzle[
                        marsterPuzzleItm]).GetLength(i);
823                 PointPuzzle mpuzzleItemsStartPoint = ((
                        Puzzle)aPuzzle[marsterPuzzleItm]).
                        GetStartPoint(i);
824                 PointPuzzle mpuzzleItemsEndPoint = ((
                        Puzzle)aPuzzle[marsterPuzzleItm]).
                        GetEndPoint(i);
825
826                 for (int j = 0; j < ((Puzzle)aPuzzle[
                        puzzleItm]).GetCount(); j++)
827                 {
828                     PointPuzzle puzzleItemsStartPoint = ((
                            Puzzle)aPuzzle[puzzleItm]).
                            GetStartPoint(j);
829                     PointPuzzle puzzleItemsEndPoint = ((
                            Puzzle)aPuzzle[puzzleItm]).
                            GetEndPoint(j);
830
831                     double tmpError = len - ((Puzzle)
                            aPuzzle[puzzleItm]).GetLength(j);
832
833                     if (((Puzzle)aPuzzle[puzzleItm]).
                            GetSideActiv(j) && tmpError <= tol
                             && tmpError >= -tol)
834                     {
835                         if (((mpuzzleItemsEndPoint.X >=
                                puzzleItemsStartPoint.X - mTol
```

```
                                && mpuzzleItemsEndPoint.X <=
                                puzzleItemsStartPoint.X + mTol
                                )
836                         && (mpuzzleItemsEndPoint.Y >=
                                puzzleItemsStartPoint.Y - mTol
                                 && mpuzzleItemsEndPoint.Y <=
                                puzzleItemsStartPoint.Y + mTol
                                ))
837                         && ((mpuzzleItemsStartPoint.X >=
                                puzzleItemsEndPoint.X - mTol
                                && mpuzzleItemsStartPoint.X <=
                                 puzzleItemsEndPoint.X + mTol)
838                         && (mpuzzleItemsStartPoint.Y >=
                                puzzleItemsEndPoint.Y - mTol
                                && mpuzzleItemsStartPoint.Y <=
                                 puzzleItemsEndPoint.Y + mTol)
                                ))
839                             tmpArr.Add(new
                                    PuzzleItemsSideToPuzzle(
                                    marsterPuzzleItm, i,
                                    puzzleItm, j));
840                     }
841                 }
842             }
843             return tmpArr;
844         }
845
846         /// <summary>
847         /// Gemmer løsning i en billede fil
848         /// </summary>
849         private void SaveSolution(int numbe)
850         {
851             try
852             {
853                 // ArgumentException is thrown because 7
                        is not an even number.
854                 //Console.WriteLine("7 divided by 2 is
                        {0}");
855                 Bitmap newBitmap = new Bitmap(1000, 1000,
                        PixelFormat.Format32bppArgb);
856                 Graphics g = Graphics.FromImage(newBitmap)
                        ;
```

```
857
858                 Font drawFont = new Font("Arial", 16);
859                 Font drawFont2 = new Font("Arial", 7);
860                 SolidBrush drawBrush = new SolidBrush(
                        Color.Black);
861                 int xt = 300, yt = 400;
862                 //float xt = 0, yt = 0;
863                 //int xt = 0, yt = 0;
864                 float aXm = 0, aYm = 0, aYM = 0, aXM = 0;
865                 for (int i = 0; i < aPuzzle.Count; i++)
866                 {
867                     if (!((Puzzle)aPuzzle[i]).GetUse() &&
                            ((Puzzle)aPuzzle[i]).GetPut())
868                     {
869                         for (int j = 0; j < ((Puzzle)
                                aPuzzle[i]).GetCount(); j++)
870                         {
871                             if (aXm > ((Puzzle)aPuzzle[i])
                                    .GetStartPoint(j).X)
872                                 aXm = ((Puzzle)aPuzzle[i])
                                        .GetStartPoint(j).X;
873                             else if (aXm > ((Puzzle)
                                    aPuzzle[i]).GetEndPoint(j)
                                    .X)
874                                 aXm = ((Puzzle)aPuzzle[i])
                                        .GetEndPoint(j).X;
875
876                             if (aXM < ((Puzzle)aPuzzle[i])
                                    .GetStartPoint(j).X)
877                                 aXM = ((Puzzle)aPuzzle[i])
                                        .GetStartPoint(j).X;
878                             else if (aXM < ((Puzzle)
                                    aPuzzle[i]).GetEndPoint(j)
                                    .X)
879                                 aXM = ((Puzzle)aPuzzle[i])
                                        .GetEndPoint(j).X;
880
881                             if (aYm > ((Puzzle)aPuzzle[i])
                                    .GetStartPoint(j).Y)
882                                 aYm = ((Puzzle)aPuzzle[i])
                                        .GetStartPoint(j).Y;
```

```
883                     else if (aYm > ((Puzzle)
                            aPuzzle[i]).GetEndPoint(j)
                            .Y)
884                         aYm = ((Puzzle)aPuzzle[i])
                                .GetEndPoint(j).Y;
885
886                     if (aYM < ((Puzzle)aPuzzle[i])
                            .GetStartPoint(j).Y)
887                         aYM = ((Puzzle)aPuzzle[i])
                                .GetStartPoint(j).Y;
888                     else if (aYM < ((Puzzle)
                            aPuzzle[i]).GetEndPoint(j)
                            .Y)
889                         aYM = ((Puzzle)aPuzzle[i])
                                .GetEndPoint(j).Y;
890
891                     if (((Puzzle)aPuzzle[i]).
                            GetSideActiv(j))
892                     {
893                         g.DrawLine(Pens.Red,
894                                 ((Puzzle)aPuzzle[i]).
                                        GetStartPoint(j).X
                                         + xt, ((Puzzle)
                                        aPuzzle[i]).
                                        GetStartPoint(j).Y
                                         + yt,
895                                 ((Puzzle)aPuzzle[i]).
                                        GetEndPoint(j).X +
                                         xt, ((Puzzle)
                                        aPuzzle[i]).
                                        GetEndPoint(j).Y +
                                         yt);
896                     }
897                     else
898                     {
899                         g.DrawLine(Pens.Green,
900                                 ((Puzzle)aPuzzle[i]).
                                        GetStartPoint(j).X
                                         + xt, ((Puzzle)
                                        aPuzzle[i]).
                                        GetStartPoint(j).Y
                                         + yt,
```

```
901                                 ((Puzzle)aPuzzle[i]).
                                        GetEndPoint(j).X +
                                         xt, ((Puzzle)
                                        aPuzzle[i]).
                                        GetEndPoint(j).Y +
                                         yt);
902                             }
903
904                             g.FillEllipse(drawBrush, ((
                                    Puzzle)aPuzzle[i]).
                                    GetStartPoint(j).X + xt -
                                    2, ((Puzzle)aPuzzle[i]).
                                    GetStartPoint(j).Y + yt -
                                    2, 4, 4);
905                             g.FillEllipse(drawBrush, ((
                                    Puzzle)aPuzzle[i]).
                                    GetEndPoint(j).X + xt - 2,
                                     ((Puzzle)aPuzzle[i]).
                                    GetEndPoint(j).Y + yt - 2,
                                     4, 4);
906
907
908                             /*//Skriver kooridater ud
909                             float xs = ((Puzzle)aPuzzle[i
                                    ]).GetStartPoint(j).X*1.0F
                                    ;
910                             float ys = ((Puzzle)aPuzzle[i
                                    ]).GetStartPoint(j).Y*1.0F
                                    ;
911                             string text = "("+ ((Puzzle)
                                    aPuzzle[i]).GetStartPoint(
                                    j).X + ","+ ((Puzzle)
                                    aPuzzle[i]).GetStartPoint(
                                    j).Y +")";
912                             e.Graphics.DrawString(text,
                                    drawFont2, drawBrush, xs,
                                    ys);
913                             */
914                         }
915
916                         PointPuzzle tmpPoint = ((Puzzle)
                                aPuzzle[i]).GetCenter();
```

```
917                     float x = tmpPoint.X * 1.0F + xt;
918                     float y = tmpPoint.Y * 1.0F + yt;
919                     float rad = (float)((Puzzle)
                            aPuzzle[i]).GetRadius();
920
921                     //g.DrawEllipse(Pens.Blue, x - rad
                            , y - rad, rad * 2, rad * 2);
922                     g.FillEllipse(drawBrush, x - 1, y
                            - 1, 1 * 2, 1 * 2);
923
924                     g.DrawString("ID" + ((Puzzle)
                            aPuzzle[i]).GetID(),
925                                                   drawFont
                                                       ,
                                                       drawBrush
                                                       ,
                                                       x,
                                                        y
                                                       );
926
927                     ArrayList tmpCenterList = ((Puzzle
                            )aPuzzle[i]).GetCenterList();
928
929                     for (int j = 0; j < tmpCenterList.
                            Count; j++)
930                     {
931                         tmpPoint = ((PuzzleItemsCenter
                                )tmpCenterList[j]).cen;
932                         x = tmpPoint.X * 1.0F + xt;
933                         y = tmpPoint.Y * 1.0F + yt;
934
935                         g.DrawString("ID" + ((
                                PuzzleItemsCenter)
                                tmpCenterList[j]).nr,
936                                                   drawFont
                                                       ,
                                                       drawBrush
                                                       ,
                                                       x,
                                                        y
                                                       );
937                     }
```

```
938                     }
939
940                 }
941                 double tmpAreal = (aXM - aXm) * (aYM - aYm
                        );
942
943                 // if (tmpAreal <= mAreal)
944                 {
945                     mAreal = tmpAreal;
946                     if (mAreal < (740 * 580))
947                         mAreal = (740 * 580);
948                     newBitmap.Save("Z" + numbe + ".png",
                            ImageFormat.Png);
949                 }
950
951             }
952             catch (ArgumentException)
953             {
954                 // Show the user that 7 cannot be divided
                        by 2.
955                 //Console.WriteLine("7 is not divided by 2
                         integrally.");
956                 MessageBox.Show("Nej den kan ikke nr : " +
                         mFileCon);
957                 //listBox1.Items.Add("Nej den kan ikke nr
                        : " + mFileCon);
958             }
959         }
960
961         /// <summary>
962         /// Gemmer løsning i en Text fil
963         /// </summary>
964         private void SaveSolutionInText(int numbe)
965         {
966             try
967             {
968                 StreamWriter fr = new StreamWriter("ZB" +
                        numbe + ".csv");
969                 StreamWriter fc = new StreamWriter("ZC" +
                        numbe + ".csv");
970                 int xt = 300, yt = 400;
971                 float xs = 0, ys = 0;
```

```
972                 fr.WriteLine("x;y;c");
973                 fc.WriteLine("x;y;n");
974
975                 for (int i = 0; i < aPuzzle.Count; i++)
976                 {
977                     if (!((Puzzle)aPuzzle[i]).GetUse() &&
                            ((Puzzle)aPuzzle[i]).GetPut())
978                     {
979                         for (int j = 0; j < ((Puzzle)
                                aPuzzle[i]).GetCount(); j++)
980                         {
981                             if (((Puzzle)aPuzzle[i]).
                                    GetSideActiv(j))
982                             {
983                                 xs = (((Puzzle)aPuzzle[i])
                                        .GetStartPoint(j).X +
                                        xt);
984                                 ys = (((Puzzle)aPuzzle[i])
                                        .GetStartPoint(j).Y +
                                        yt);
985                                 fr.WriteLine(xs+";"+ys+";r
                                        ");
986                                 xs = (((Puzzle)aPuzzle[i])
                                        .GetEndPoint(j).X + xt
                                        );
987                                 ys = (((Puzzle)aPuzzle[i])
                                        .GetEndPoint(j).Y + yt
                                        );
988                                 fr.WriteLine(xs+";"+ys+";r
                                        ");
989                             }
990                             else
991                             {
992                                 xs = (((Puzzle)aPuzzle[i])
                                        .GetStartPoint(j).X +
                                        xt);
993                                 ys = (((Puzzle)aPuzzle[i])
                                        .GetStartPoint(j).Y +
                                        yt);
994                                  fr.WriteLine(xs + ";" +
                                         ys + ";g");
```

```
995                                 xs = (((Puzzle)aPuzzle[i])
                                        .GetEndPoint(j).X + xt
                                        );
996                                 ys = (((Puzzle)aPuzzle[i])
                                        .GetEndPoint(j).Y + yt
                                        );
997                                 fr.WriteLine(xs + ";" + ys
                                         + ";g");
998                             }
999                         }
1000
1001                        PointPuzzle tmpPoint = ((Puzzle)
                                aPuzzle[i]).GetCenter();
1002                        xs = tmpPoint.X * 1.0F + xt;
1003                        ys = tmpPoint.Y * 1.0F + yt;
1004
1005                        fc.WriteLine(xs + ";" + ys + ";" +
                                 ((Puzzle)aPuzzle[i]).GetID())
                                ;
1006
1007                        ArrayList tmpCenterList = ((Puzzle
                                )aPuzzle[i]).GetCenterList();
1008
1009                        for (int j = 0; j < tmpCenterList.
                                Count; j++)
1010                        {
1011                            tmpPoint = ((PuzzleItemsCenter
                                    )tmpCenterList[j]).cen;
1012                            xs = tmpPoint.X * 1.0F + xt;
1013                            ys = tmpPoint.Y * 1.0F + yt;
1014
1015                            fc.WriteLine(xs + ";" + ys + "
                                    ;" + ((PuzzleItemsCenter)
                                    tmpCenterList[j]).nr);
1016                        }
1017                    }
1018
1019                }
1020
1021                fc.Close();
1022                fr.Close();
1023
```

```
1024                }
1025                catch (ArgumentException)
1026                {
1027                    // Show the user that 7 cannot be divided
                        by 2.
1028                    //Console.WriteLine("7 is not divided by 2
                         integrally.");
1029                    MessageBox.Show("Nej den kan ikke nr : " +
                         mFileCon);
1030                    //listBox1.Items.Add("Nej den kan ikke nr
                        : " + mFileCon);
1031                }
1032            }
1033
1034        }
1035    }
```

C.1.2 PuzzleAlgo4.cs

```
1  using System;
2  using System.Drawing;
3  using System.Collections;
4  using System.ComponentModel;
5  using System.Windows.Forms;
6  using System.Data;
7  using System.IO;
8  using System.Threading;
9  using System.Drawing.Imaging;
10
11 namespace PuzzleAlgo4
12 {
13  /// <summary>
14  /// Summary description for Form1.
15  /// </summary>
16  public class Form1 : System.Windows.Forms.Form
17  {
18   private System.Windows.Forms.ListBox listBox1;
19   private System.Windows.Forms.Button btStart;
20   private ArrayList aPuzzle = new ArrayList();
21   private Form2 mForm2 = new Form2();
22         private ArrayList aPuzzleLength = new ArrayList();
23         private float arealglob;
24         private float xGglob;
25         private float yGglob;
26         private int mCon, mFileCon;
27         private float mTol = 1.9f;
28         private float mDegTol = 0.5f;
29         private DateTime mTimerStart, mTimerEnd;
30         private double mAreal = 10000000.0;
31         private double mTolAreal = 0.0;
32         private int muh = 0;
33         private int muh2 = 0;
34         private ArrayList ListPutPuzzle = new ArrayList();
35         private ArrayList ListBefore = new ArrayList();
36         private int Liste = 0;
37
38   /// <summary>
39   /// Required designer variable.
40   /// </summary>
```

```
41    private System.ComponentModel.Container components =
          null;
42
43    public Form1()
44    {
45     InitializeComponent();
46    }
47
48          /// <summary>
49    /// Clean up any resources being used.
50    /// </summary>
51    protected override void Dispose( bool disposing )
52    {
53     if( disposing )
54     {
55      if (components != null)
56      {
57       components.Dispose();
58      }
59     }
60     base.Dispose( disposing );
61    }
62
63    #region Windows Form Designer generated code
64    /// <summary>
65    /// Required method for Designer support - do not modify
66    /// the contents of this method with the code editor.
67    /// </summary>
68    private void InitializeComponent()
69    {
70              this.listBox1 = new System.Windows.Forms.
                  ListBox();
71              this.btStart = new System.Windows.Forms.Button
                  ();
72              this.SuspendLayout();
73              //
74              // listBox1
75              //
76              this.listBox1.Location = new System.Drawing.
                  Point(8, 16);
77              this.listBox1.Name = "listBox1";
```

```
78              this.listBox1.SelectionMode = System.Windows.
                    Forms.SelectionMode.MultiExtended;
79              this.listBox1.Size = new System.Drawing.Size
                    (373, 238);
80              this.listBox1.TabIndex = 0;
81              //
82              // btStart
83              //
84              this.btStart.Location = new System.Drawing.
                    Point(8, 264);
85              this.btStart.Name = "btStart";
86              this.btStart.Size = new System.Drawing.Size
                    (75, 23);
87              this.btStart.TabIndex = 1;
88              this.btStart.Text = "Start";
89              this.btStart.Click += new System.EventHandler(
                    this.btStart_Click);
90              //
91              // Form1
92              //
93              this.AutoScaleBaseSize = new System.Drawing.
                    Size(5, 13);
94              this.ClientSize = new System.Drawing.Size(393,
                     294);
95              this.Controls.Add(this.btStart);
96              this.Controls.Add(this.listBox1);
97              this.Name = "Form1";
98              this.Text = "Form1";
99              this.ResumeLayout(false);
100
101     }
102     #endregion
103
104     /// <summary>
105     /// The main entry point for the application.
106     /// </summary>
107     [STAThread]
108     static void Main()
109     {
110      Application.Run(new Form1());
111     }
112
```

```
113         /// <summary>
114         /// Knappen som der start hele lægningen at
                brikkerne
115         /// </summary>
116         private void btStart_Click(object sender, System.
                EventArgs e)
117         {
118             mTimerStart = DateTime.Now;
119 
120             mTol = 1.5f;
121             mCon = 0;
122             mFileCon = 1;
123             string FileN = "Brik15";
124             string tmpFileName = FileN + mCon + ".csv";
125             aPuzzle.Clear();
126 
127             // Indlæser alle brikkerne
128             while (File.Exists(tmpFileName))
129             {
130                 Puzzle tmpPuzzle = new Puzzle();
131                 StreamReader st = new StreamReader(
                        tmpFileName);
132                 string[] aStr = st.ReadLine().Split(',');
133 
134                 aStr = st.ReadLine().Split(',');
135                 float xF = (float)Convert.ToDouble(aStr
                        [0].Replace('.',','));
136                 float yF = (float)Convert.ToDouble(aStr
                        [1].Replace('.', ','));
137 
138                 aStr = st.ReadLine().Split(',');
139                 float xE = (float)Convert.ToDouble(aStr
                        [0].Replace('.', ','));
140                 float yE = (float)Convert.ToDouble(aStr
                        [1].Replace('.', ','));
141 
142                 float areal = xF * yE - xE * yF;
143 
144                 float xG = (xF + xE) * (xF * yE - xE * yF)
                        ;
145                 float yG = (yF + yE) * (xF * yE - xE * yF)
                        ;
```

```
146
147                 int conG = 1;
148
149                 bool stop = false;
150
151                 int tmpSideCon = 0;
152                 // gremmer brikkernes sider i objeket og
                        regne areale ud på dem
153                 while (stop != true)
154                 {
155                     try
156                     {
157                         double tmpLen = Math.Sqrt((xF - xE
                                ) * (xF - xE) + (yF - yE) * (
                                yF - yE));
158
159                         tmpSideCon = tmpPuzzle.SetLength(
                                tmpLen, new PointPuzzle(xF, yF
                                ), new PointPuzzle(xE, yE));
160
161                         aStr = st.ReadLine().Split(',');
162                         xF = (float)Convert.ToDouble(aStr
                                [0].Replace('.', ','));
163                         yF = (float)Convert.ToDouble(aStr
                                [1].Replace('.', ','));
164
165                         aStr = st.ReadLine().Split(',');
166                         xE = (float)Convert.ToDouble(aStr
                                [0].Replace('.', ','));
167                         yE = (float)Convert.ToDouble(aStr
                                [1].Replace('.', ','));
168
169                         areal += xF * yE - xE * yF;
170
171                         xG += (xF + xE) * (xF * yE - xE *
                                yF);
172                         yG += (yF + yE) * (xF * yE - xE *
                                yF);
173                         conG += 1;
174
175                         aPuzzleLength.Add(new
                                PuzzleItemsSide(tmpLen, mCon,
```

```
                                tmpSideCon - 1));
176                         }
177                         catch (Exception)
178                         {
179                             stop = true;
180                             st.Close();
181                         }
182                     }
183                     areal = Math.Abs(areal * 0.5f);
184
185                     float cenXF = (1 / (6 * areal)) * xG;
186                     float cenYF = (1 / (6 * areal)) * yG;
187
188                     int cenX = Convert.ToInt32(cenXF);
189                     int cenY = Convert.ToInt32(cenYF);
190
191                     double radius = Math.Sqrt(areal / Math.PI)
                            ;
192
193                     mTolAreal += areal;
194
195                     listBox1.Items.Add("Brik : " + mCon + "
                            Areal : " + areal + " (X,Y)=(" + cenX
                            + ", " + cenY + ")");
196
197                     tmpPuzzle.SetRadius(radius);
198                     tmpPuzzle.SetID(mCon);
199                     tmpPuzzle.SetCenter(cenX, cenY);
200
201                     aPuzzle.Add(tmpPuzzle);
202
203                     mCon++;
204                     tmpFileName = FileN + mCon + ".csv";
205                 }
206
207                 // Sorter listen med brikker
208                 listBox1.Items.Add("Sorter.....");
209                 Mergesort(aPuzzleLength, 0, aPuzzleLength.
                        Count-1);
210
211                 // Køre Coktalmetoden til den ikke kan køres
                        mere
```

```
212             while (MakePuzzle())
213             {
214                 listBox1.Items.Add("Engang til....");
215             }
216 
217             // Køre backtrackingmetoden fra den første
                    brik som ikke er blive lagt.
218             for (int i = 0; i < aPuzzle.Count; i++)
219             {
220                 if (!((Puzzle)aPuzzle[i]).GetUse())
221                 {
222                     ((Puzzle)aPuzzle[i]).SetPut(true);
223                     ListPutPuzzle.Add(i);
224                     FindPuzzle(i, new PuzzleItemsSide[0],
                            0);
225                     ((Puzzle)aPuzzle[i]).SetPut(false);
226                     break;
227                 }
228             }
229 
230             mTimerEnd = DateTime.Now;
231 
232             listBox1.Items.Add("Timer : "+(mTimerEnd -
                    mTimerStart));
233 
234             mForm2.Show();
235             mForm2.ViewArray(aPuzzle);
236 
237             mForm2.SaveSolution(0);
238 
239         }
240 
241         /// <summary>
242         /// Funktion der finde brikker som der kan sam
                samlter brikker
243         /// </summary>
244         private bool TiePuzzle(int marsterPuzzleItem, int
                marsterPuzzleSide)
245         {
246             muh++;
247 
```

```
248             float marsterDeg = ((Puzzle)aPuzzle[
                    marsterPuzzleItem]).GetDeg(
                    marsterPuzzleSide);
249
250             PointPuzzle marsterPuzzleItemsCenter = ((
                    Puzzle)aPuzzle[marsterPuzzleItem]).
                    GetCenter();
251             ArrayList marsterPuzzleItemsCenterList = ((
                    Puzzle)aPuzzle[marsterPuzzleItem]).
                    GetCenterList();
252
253             double marsterPuzzleSideLen = ((Puzzle)aPuzzle
                    [marsterPuzzleItem]).GetLength(
                    marsterPuzzleSide);
254             PointPuzzle marsterPuzzleItemsStartPoint = ((
                    Puzzle)aPuzzle[marsterPuzzleItem]).
                    GetStartPoint(marsterPuzzleSide);
255             PointPuzzle marsterPuzzleItemsEndPoint = ((
                    Puzzle)aPuzzle[marsterPuzzleItem]).
                    GetEndPoint(marsterPuzzleSide);
256             int marsterCon = ((Puzzle)aPuzzle[
                    marsterPuzzleItem]).GetCount();
257
258             // finde brikker som der kan lige til
                    marsterPuzzleSide
259             ArrayList list_side = new ArrayList();
260             list_side = FindPuzzleSide(
                    marsterPuzzleSideLen, marsterPuzzleItem,
                    0, aPuzzleLength.Count-1);
261
262             // hvis der kun findes en så kan der måske sam
                     samlse
263             if (list_side.Count == 1)
264             {
265                 int puzzleItems = ((PuzzleItemsSide)
                        list_side[0]).masterPuzzle;
266                 int puzzleItemsSide = ((PuzzleItemsSide)
                        list_side[0]).masterPuzzleSide;
267                 int puzzleCon = ((Puzzle)aPuzzle[
                        puzzleItems]).GetCount();
268                 PointPuzzle puzzleItemsStartPoint = ((
                        Puzzle)aPuzzle[puzzleItems]).
```

```
                GetStartPoint(puzzleItemsSide);
269             PointPuzzle puzzleItemsEndPoint = ((Puzzle
                )aPuzzle[puzzleItems]).GetEndPoint(
                puzzleItemsSide);
270
271             // roter brikken så brikken passe til
                marsterPuzzleSide
272             //((Puzzle)aPuzzle[puzzleItems]).Rotet(
                marsterDeg, puzzleItemsSide);
273             puzzleItemsStartPoint = ((Puzzle)aPuzzle[
                puzzleItems]).GetStartPoint(
                puzzleItemsSide);
274             puzzleItemsEndPoint = ((Puzzle)aPuzzle[
                puzzleItems]).GetEndPoint(
                puzzleItemsSide);
275
276             // flytter brikken til masterPuzzle
277             ((Puzzle)aPuzzle[puzzleItems]).MoveToOrion
                (marsterPuzzleItemsStartPoint,
                puzzleItemsEndPoint);
278             puzzleItemsStartPoint = ((Puzzle)aPuzzle[
                puzzleItems]).GetStartPoint(
                puzzleItemsSide);
279             puzzleItemsEndPoint = ((Puzzle)aPuzzle[
                puzzleItems]).GetEndPoint(
                puzzleItemsSide);
280
281             PointPuzzle puzzleItemsCenter = ((Puzzle)
                aPuzzle[puzzleItems]).GetCenter();
282             ArrayList puzzleItemsCenterList = ((Puzzle
                )aPuzzle[puzzleItems]).GetCenterList()
                ;
283
284             // Ser er det to bikke passer sammen
285             if (!LineTurn(((Puzzle)aPuzzle[puzzleItems
                ]), ((Puzzle)aPuzzle[marsterPuzzleItem
                ]), puzzleItemsSide, marsterPuzzleSide
                )
286                 && ((marsterPuzzleItemsEndPoint.X >=
                    puzzleItemsStartPoint.X-mTol &&
                    marsterPuzzleItemsEndPoint.X <=
                    puzzleItemsStartPoint.X+mTol)
```

```
287                     && (marsterPuzzleItemsEndPoint.Y >=
                            puzzleItemsStartPoint.Y-mTol &&
                            marsterPuzzleItemsEndPoint.Y <=
                            puzzleItemsStartPoint.Y+mTol))
288                     && ((marsterPuzzleItemsStartPoint.X >=
                             puzzleItemsEndPoint.X - mTol &&
                            marsterPuzzleItemsStartPoint.X <=
                            puzzleItemsEndPoint.X + mTol)
289                     && (marsterPuzzleItemsStartPoint.Y >=
                            puzzleItemsEndPoint.Y - mTol &&
                            marsterPuzzleItemsStartPoint.Y <=
                            puzzleItemsEndPoint.Y + mTol)))
290                 {
291                     // finde de sider som de to brikker
                            har tilfælses
292                     ArrayList listSideAcitv = new
                            ArrayList();
293                     listSideAcitv = FindPuzzleSideToPuzzle
                            (marsterPuzzleItem, puzzleItems);
294
295                     // deaktiver de side de har til fælses
296                     for (int i = 0; i < listSideAcitv.
                            Count; i++)
297                     {
298                         int mPuzzleItems = ((
                                PuzzleItemsSideToPuzzle)
                                listSideAcitv[i]).masterPuzzle
                                ;
299                         int mPuzzleItemsSide = ((
                                PuzzleItemsSideToPuzzle)
                                listSideAcitv[i]).
                                masterPuzzleSide;
300                         int sPuzzleItems = ((
                                PuzzleItemsSideToPuzzle)
                                listSideAcitv[i]).PuzzleItm;
301                         int sPuzzleItemsSide = ((
                                PuzzleItemsSideToPuzzle)
                                listSideAcitv[i]).PuzzleSide;
302
303                         ((Puzzle)aPuzzle[mPuzzleItems]).
                                SetNeighbourPuzzle(
                                mPuzzleItemsSide, sPuzzleItems
```

```
                    , sPuzzleItemsSide);
304                 ((Puzzle)aPuzzle[mPuzzleItems]).
                        SideDeactiv(mPuzzleItemsSide);
305                 ((Puzzle)aPuzzle[sPuzzleItems]).
                        SetNeighbourPuzzle(
                        sPuzzleItemsSide, mPuzzleItems
                        , mPuzzleItemsSide);
306                 ((Puzzle)aPuzzle[sPuzzleItems]).
                        SideDeactiv(sPuzzleItemsSide);
307             }
308
309             ((Puzzle)aPuzzle[marsterPuzzleItem]).
                    SetUse(true);
310             ((Puzzle)aPuzzle[puzzleItems]).SetUse(
                    true);
311
312             try
313             {
314                 int con = (puzzleCon + marsterCon)
                        - (listSideAcitv.Count * 2);
315                 arealglob = 0;
316                 xGglob = 0;
317                 yGglob = 0;
318                 Puzzle tmpPuzzle = new Puzzle();
319
320                 // samlter de to brikker sammen og
                        finde ny ID til den nye brik
321                 //tmpPuzzle = AddPuzzleToPuzzle(
                        marsterPuzzleItem, 0, 0, con,
                        tmpPuzzle);
322                 tmpPuzzle = AddPuzzleToPuzzle(
                        puzzleItems, 0, 0, con,
                        tmpPuzzle);
323                 tmpPuzzle.SetID(aPuzzle.Count);
324
325                 // regne arealet ud på den nye
                        brik fra den globale variable
326                 arealglob = Math.Abs(arealglob *
                        0.5f);
327
328                 // regne centum ud på den ny brik
```

```
329                 float cenXF = (1 / (6 * arealglob)
                        ) * xGglob;
330                 float cenYF = (1 / (6 * arealglob)
                        ) * yGglob;
331                 int cenX = Convert.ToInt32(cenXF);
332                 int cenY = Convert.ToInt32(cenYF);
333
334                 double radius = Math.Sqrt(
                        arealglob / Math.PI);
335                 listBox1.Items.Add("Brik : " +
                        aPuzzle.Count + " Areal : " +
                        arealglob + " (X,Y)=(" + cenX
                        + ", " + cenY + ")");
336
337                 tmpPuzzle.SetRadius(radius);
338                 tmpPuzzle.SetCenter(cenX, cenY);
339
340                 // sam samlter sider som har den
                        samme vinkle
341                 tmpPuzzle = ComplexPuzzleItems(
                        tmpPuzzle);
342
343                 // overføre centumerne til den nye
                         brik og overføre ikke sam
                        samlte brikker centumer
344                 if (mCon > marsterPuzzleItem)
345                     tmpPuzzle.AddCenterToList(
                            marsterPuzzleItem,
                            marsterPuzzleItemsCenter);
346                 if (mCon > puzzleItems)
347                     tmpPuzzle.AddCenterToList(
                            puzzleItems,
                            puzzleItemsCenter);
348
349                 for (int i = 0; i <
                        marsterPuzzleItemsCenterList.
                        Count; i++)
350                 {
351                     tmpPuzzle.AddCenterToList(((
                            PuzzleItemsCenter)
                            marsterPuzzleItemsCenterList
                            [i]).nr,
```

```
352                         ((PuzzleItemsCenter)
                                marsterPuzzleItemsCenterList
                                [i]).cen);
353                     }
354
355                     for (int i = 0; i <
                            puzzleItemsCenterList.Count; i
                            ++)
356                     {
357                         tmpPuzzle.AddCenterToList(((
                                PuzzleItemsCenter)
                                puzzleItemsCenterList[i]).
                                nr,
358                             ((PuzzleItemsCenter)
                                puzzleItemsCenterList[
                                i]).cen);
359                     }
360
361                     // tilføje nye brik til listen.
362                     aPuzzle.Add(tmpPuzzle);
363
364                     listBox1.Items.Add("Brik : " +
                            marsterPuzzleItem + " side : "
                             + marsterPuzzleSide +
365                         " Lige med brik : " +
                                puzzleItems + " side : " +
                                 puzzleItemsSide);
366
367                     //tilføjer den nye briks sider til
                             listen og sotrer den med
                            Margesort
368                     for (int i = 0; i < tmpPuzzle.
                            GetCount(); i++)
369                     {
370                         aPuzzleLength.Add(new
                                PuzzleItemsSide(tmpPuzzle.
                                GetLength(i), tmpPuzzle.
                                GetID(), i));
371                     }
372                     listBox1.Items.Add("Sorter.....");
373                     Mergesort(aPuzzleLength, 0,
                            aPuzzleLength.Count - 1);
```

```
374
375                         return true;
376                     }
377                     catch (Exception e)
378                     {
379                         for (int i = 0; i < listSideAcitv.
                                Count; i++)
380                         {
381                             int mPuzzleItems = ((
                                    PuzzleItemsSideToPuzzle)
                                    listSideAcitv[i]).
                                    masterPuzzle;
382                             int mPuzzleItemsSide = ((
                                    PuzzleItemsSideToPuzzle)
                                    listSideAcitv[i]).
                                    masterPuzzleSide;
383                             int sPuzzleItems = ((
                                    PuzzleItemsSideToPuzzle)
                                    listSideAcitv[i]).
                                    PuzzleItm;
384                             int sPuzzleItemsSide = ((
                                    PuzzleItemsSideToPuzzle)
                                    listSideAcitv[i]).
                                    PuzzleSide;
385
386                             ((Puzzle)aPuzzle[mPuzzleItems
                                    ]).SetNeighbourPuzzle(
                                    mPuzzleItemsSide, -1, -1);
387                             ((Puzzle)aPuzzle[mPuzzleItems
                                    ]).SideActiv(
                                    mPuzzleItemsSide);
388                             ((Puzzle)aPuzzle[sPuzzleItems
                                    ]).SetNeighbourPuzzle(
                                    sPuzzleItemsSide, -1, -1);
389                             ((Puzzle)aPuzzle[sPuzzleItems
                                    ]).SideActiv(
                                    sPuzzleItemsSide);
390                         }
391
392                         ((Puzzle)aPuzzle[marsterPuzzleItem
                                ]).SetUse(false);
```

```
393                         ((Puzzle)aPuzzle[puzzleItems]).
                                SetUse(false);
394                     }
395                 }
396                 else
397                 {
398                     ((Puzzle)aPuzzle[puzzleItems]).
                            SetOrionBack();
399                 }
400             }
401
402             return false;
403         }
404
405         /// <summary>
406         /// Funktion start coktalmetoden
407         /// </summary>
408         private bool MakePuzzle()
409         {
410             bool tmpbool = false;
411
412             for (int j = 0; j < aPuzzle.Count; j++)
413             {
414                 for (int i = 0; i < ((Puzzle)aPuzzle[j]).
                        GetCount(); i++)
415                     //if (((Puzzle)aPuzzle[j]).
                            GetSideActiv(i))
416                     if (!((Puzzle)aPuzzle[j]).GetUse())
417                         if (TiePuzzle(j, i))
418                         {
419                             tmpbool = true;
420                             j = 0;
421                             break;
422                         }
423             }
424             //Hvis der er sket en samsamlting så returnes
                    true så funktion kan gøre igen.
425             return tmpbool;
426         }
427
428         /// <summary>
429         /// Funktion som smalter brikker sammen
```

```
430         /// </summary>
431         private Puzzle AddPuzzleToPuzzle(int
                startPuzzleitm, int startSide, int con, int
                conMax, Puzzle puzzle)
432         {
433             Puzzle tmpPuzzle = puzzle;
434             int tmpCon = con;
435             int tmpMax = conMax;
436             int tmpSide = startSide;
437             int sidecon = ((Puzzle)aPuzzle[startPuzzleitm
                    ]).GetCount();
438
439             // her undersøg hvor mange sider der ikke er
                    aktive
440             while (!((Puzzle)aPuzzle[startPuzzleitm]).
                    GetSideActiv(tmpSide) && tmpSide < sidecon
                    )
441                 tmpSide++;
442
443             if (tmpSide >= sidecon)
444                 tmpSide = 0;
445
446             // her ligge de sider der er aktive fra de to
                    brikker over i det ny objekt
447             while (tmpCon < tmpMax && ((Puzzle)aPuzzle[
                    startPuzzleitm]).GetSideActiv(tmpSide))
448             {
449                 tmpPuzzle.SetLength(((Puzzle)aPuzzle[
                        startPuzzleitm]).GetLength(tmpSide),
450                                 ((Puzzle)aPuzzle[
                                    startPuzzleitm]).
                                    GetStartPoint(tmpSide),
451                                 ((Puzzle)aPuzzle[
                                    startPuzzleitm]).
                                    GetEndPoint(tmpSide));
452
453                 float xF = ((Puzzle)aPuzzle[startPuzzleitm
                        ]).GetStartPoint(tmpSide).X;
454                 float yF = ((Puzzle)aPuzzle[startPuzzleitm
                        ]).GetStartPoint(tmpSide).Y;
455                 float xE = ((Puzzle)aPuzzle[startPuzzleitm
                        ]).GetEndPoint(tmpSide).X;
```

```
456                 float yE = ((Puzzle)aPuzzle[startPuzzleitm
                        ]).GetEndPoint(tmpSide).Y;
457
458
459                 arealglob += xF * yE - xE * yF;
460
461                 xGglob += (xF + xE) * (xF * yE - xE * yF);
462                 yGglob += (yF + yE) * (xF * yE - xE * yF);
463
464                 tmpCon++;
465                 tmpSide++;
466
467                 if (tmpSide >= sidecon)
468                     tmpSide = 0;
469             }
470
471             if (tmpCon >= tmpMax)
472                 return tmpPuzzle;
473
474             // hvis der er flere side klades funktionen
                    igen
475             int newPuzzleItems = ((Puzzle)aPuzzle[
                    startPuzzleitm]).GetNeighbourPuzzle(
                    tmpSide);
476             int newPuzzleItemsSide = ((Puzzle)aPuzzle[
                    startPuzzleitm]).GetNeighbourPuzzleSide(
                    tmpSide)+1;
477
478             return AddPuzzleToPuzzle(newPuzzleItems,
                    newPuzzleItemsSide, tmpCon, conMax,
                    tmpPuzzle);
479         }
480
481         /// <summary>
482         /// Funktion som finde sider med samme viklen og
                lave dem om en lang side
483         /// </summary>
484         private Puzzle ComplexPuzzleItems(Puzzle puzzle)
485         {
486             Puzzle tmpPuzzle = puzzle;
487
```

```
488             PointPuzzle startPk = tmpPuzzle.GetStartPoint
                    (0);
489             PointPuzzle endPk = tmpPuzzle.GetEndPoint(0);
490             PointPuzzle tmpPk = endPk - startPk;
491
492             // Finde vinkel på den første side
493             float newDeg = (float)Math.Acos(tmpPk.X / Math
                    .Sqrt(tmpPk.X * tmpPk.X + tmpPk.Y * tmpPk.
                    Y));
494             if (tmpPk.Y > 0)
495                 newDeg = (float)((Math.PI * 2 - newDeg) /
                        Math.PI) * 180.0f;
496             else
497                 newDeg = (float)(newDeg / Math.PI) * 180.0
                        f;
498
499             float oldDeg = newDeg;
500
501             int k = 1;
502             while ( k < tmpPuzzle.GetCount())
503             {
504                 startPk = tmpPuzzle.GetStartPoint(k);
505                 endPk = tmpPuzzle.GetEndPoint(k);
506                 tmpPk = endPk - startPk;
507
508                 // Finde vinkel på den næste side
509                 newDeg = (float)Math.Acos(tmpPk.X / Math.
                        Sqrt(tmpPk.X * tmpPk.X + tmpPk.Y *
                        tmpPk.Y));
510                 if (tmpPk.Y > 0)
511                     newDeg = (float)((Math.PI * 2 - newDeg
                            ) / Math.PI) * 180.0f;
512                 else
513                     newDeg = (float)(newDeg / Math.PI) *
                            180.0f;
514
515                 listBox1.Items.Add("Brik : " + tmpPuzzle.
                        GetID() + " Side : " + (k - 1) + "
                        Vinkel : "
516                         + oldDeg.ToString() + " Side : " + k +
                                " Vinkel : " + newDeg.ToString())
                                ;
```

```
517
518                 float tmp = newDeg;
519
520                 if(oldDeg > 180)
521                     tmp = 360 - oldDeg;
522                 else
523                     tmp = 180 - oldDeg;
524
525                 // Se om det to side har den samme vinkle,
                        hvis den er ligge siderne sammen.
526                 if ((oldDeg >= newDeg - mDegTol && oldDeg
                    <= newDeg + mDegTol) ||
527                     (tmp >= newDeg - mDegTol && tmp <=
                        newDeg + mDegTol))
528                 {
529                     tmpPuzzle.SetEndPoint(tmpPuzzle.
                        GetLength(k - 1) + tmpPuzzle.
                        GetLength(k), tmpPuzzle.
                        GetEndPoint(k), k - 1);
530                     tmpPuzzle.DelSide(k);
531
532                 }
533                 else
534                 {
535                     k++;
536                 }
537
538                 oldDeg = newDeg;
539
540             }
541
542             return tmpPuzzle;
543         }
544
545         /// <summary>
546         /// Backtrackting funktionen
547         /// </summary>
548         private bool FindPuzzle(int puzzleNr,
                PuzzleItemsSide[] inLength, int startIndex)
549         {
550             muh2++;
551
```

```
552             //regne den nye liste ud
553             Liste = Liste * 10 + puzzleNr;
554             ArrayList SideActiv = new ArrayList();
555
556             //undersøger om den lige har være lavde før
557             bool stopL = false;
558             for (int i = 0; i < ListBefore.Count; i++)
559                 if (Liste == ((int)ListBefore[i]))
560                     stopL = true;
561
562             if (stopL)
563                 return false;
564
565             // ser om hvor mange brikker som der er blive
                    lagt, hvis der er alle så gemmes en
                    løsning
566             int con = 0;
567             for (int i = 1; i < aPuzzle.Count; i++)
568             {
569                 if (!((Puzzle)aPuzzle[i]).GetUse() && !((
                        Puzzle)aPuzzle[i]).GetPut())
570                 {
571                     con++;
572                 }
573             }
574
575             if (con <= 1)
576             {
577                 //SaveSolution(mFileCon);
578                 SaveSolutionInText(mFileCon);
579                 mFileCon++;
580             }
581
582
583             // overføre side på brikken til listen om side
                     som der kan ligges til
584             int tmpArrayCon = inLength.Length;
585             tmpArrayCon += ((Puzzle)aPuzzle[puzzleNr]).
                    GetCount();
586
587             PuzzleItemsSide[] activSideArray = new
                    PuzzleItemsSide[tmpArrayCon];
```

```
588
589             for (int i = 0; i < inLength.Length; i++)
590                 activSideArray[i] = inLength[i];
591
592             for (int i = 0; i < ((Puzzle)aPuzzle[puzzleNr
                    ]).GetCount(); i++)
593             {
594                 PuzzleItemsSide tmpPIS = new
                        PuzzleItemsSide();
595                 tmpPIS.len = ((Puzzle)aPuzzle[puzzleNr]).
                        GetLength(i);
596                 tmpPIS.masterPuzzle = puzzleNr;
597                 tmpPIS.masterPuzzleSide = i;
598
599                 activSideArray[i + inLength.Length] =
                        tmpPIS;
600             }
601
602             //--------------- Finde side som ligge og
                    passer til brikken --------
603
604             for(int i=0; i<ListPutPuzzle.Count; i++)
605             {
606                 int tmpPuzzle = ((int)ListPutPuzzle[i]);
607
608                 ArrayList listSideAcitv = new ArrayList();
609                 listSideAcitv = FindPuzzleSideToPuzzle(
                        puzzleNr, tmpPuzzle);
610
611                 for (int j = 0; j < listSideAcitv.Count; j
                        ++)
612                 {
613                     SideActiv.Add(listSideAcitv[j]);
614                     int mPuzzleItems = ((
                            PuzzleItemsSideToPuzzle)
                            listSideAcitv[j]).masterPuzzle;
615                     int mPuzzleItemsSide = ((
                            PuzzleItemsSideToPuzzle)
                            listSideAcitv[j]).masterPuzzleSide
                            ;
616                     int sPuzzleItems = ((
                            PuzzleItemsSideToPuzzle)
```

```
                        listSideAcitv[j]).PuzzleItm;
617                 int sPuzzleItemsSide = ((
                        PuzzleItemsSideToPuzzle)
                        listSideAcitv[j]).PuzzleSide;
618
619                 ((Puzzle)aPuzzle[mPuzzleItems]).
                        SetNeighbourPuzzle(
                        mPuzzleItemsSide, sPuzzleItems,
                        sPuzzleItemsSide);
620                 ((Puzzle)aPuzzle[mPuzzleItems]).
                        SideDeactiv(mPuzzleItemsSide);
621                 ((Puzzle)aPuzzle[sPuzzleItems]).
                        SetNeighbourPuzzle(
                        sPuzzleItemsSide, mPuzzleItems,
                        mPuzzleItemsSide);
622                 ((Puzzle)aPuzzle[sPuzzleItems]).
                        SideDeactiv(sPuzzleItemsSide);
623             }
624
625         }
626
627         //-------------- her start findingen at
                brikker tilsiderne --------
628
629         for (int i = startIndex; i < activSideArray.
                Length; i++)
630         {
631             int marsterPuzzle = activSideArray[i].
                    masterPuzzle;
632             int marsterPuzzleSide = activSideArray[i].
                    masterPuzzleSide;
633             float marsterDeg = ((Puzzle)aPuzzle[
                    marsterPuzzle]).GetDeg(
                    marsterPuzzleSide);
634             double marsterPuzzleSideLen = ((Puzzle)
                    aPuzzle[marsterPuzzle]).GetLength(
                    marsterPuzzleSide);
635             PointPuzzle marsterPuzzleItemsStartPoint =
                     ((Puzzle)aPuzzle[marsterPuzzle]).
                    GetStartPoint(marsterPuzzleSide);
636             PointPuzzle marsterPuzzleItemsEndPoint =
                    ((Puzzle)aPuzzle[marsterPuzzle]).
```

```
                    GetEndPoint(marsterPuzzleSide);
637
638             if (((Puzzle)aPuzzle[marsterPuzzle]).
                    GetSideActiv(marsterPuzzleSide))
639             {
640                 ArrayList tmpArray = new ArrayList();
641                 tmpArray = FindPuzzleSide(
                        marsterPuzzleSideLen, puzzleNr, 0,
                         aPuzzleLength.Count - 1);
642
643                 // køre listen igen for hver side som
                        der er blive funde.
644                 for (int j = 0; j < tmpArray.Count; j
                        ++)
645                 {
646                     bool stop = false;
647                     int tmpPuzzle = ((PuzzleItemsSide)
                            tmpArray[j]).masterPuzzle;
648                     int tmpPuzzleSide = ((
                            PuzzleItemsSide)tmpArray[j]).
                            masterPuzzleSide;
649                     PointPuzzle
                            tmpPuzzleItemsStartPoint = ((
                            Puzzle)aPuzzle[tmpPuzzle]).
                            GetStartPoint(tmpPuzzleSide);
650                     PointPuzzle tmpPuzzleItemsEndPoint
                             = ((Puzzle)aPuzzle[tmpPuzzle
                            ]).GetEndPoint(tmpPuzzleSide);
651
652                     // hvis brikken ikke er blive lagt
                             bliver den roter og flytte
                            til marster brikken
653                     if (!((Puzzle)aPuzzle[tmpPuzzle]).
                            GetPut())
654                     {
655                         //((Puzzle)aPuzzle[tmpPuzzle])
                                .Rotet(marsterDeg,
                                tmpPuzzleSide);
656                         tmpPuzzleItemsStartPoint = ((
                                Puzzle)aPuzzle[tmpPuzzle])
                                .GetStartPoint(
                                tmpPuzzleSide);
```

```
657                         tmpPuzzleItemsEndPoint = ((
                                Puzzle)aPuzzle[tmpPuzzle])
                                .GetEndPoint(tmpPuzzleSide
                                );
658
659                         ((Puzzle)aPuzzle[tmpPuzzle]).
                                MoveToOrion(
                                marsterPuzzleItemsStartPoint
                                , tmpPuzzleItemsEndPoint);
660                         tmpPuzzleItemsStartPoint = ((
                                Puzzle)aPuzzle[tmpPuzzle])
                                .GetStartPoint(
                                tmpPuzzleSide);
661                         tmpPuzzleItemsEndPoint = ((
                                Puzzle)aPuzzle[tmpPuzzle])
                                .GetEndPoint(tmpPuzzleSide
                                );
662
663                         for (int k = 0; k <
                                ListPutPuzzle.Count; k++)
664                         {
665                             if (RadiusMetode(((Puzzle)
                                    aPuzzle[tmpPuzzle]),
                                    ((Puzzle)aPuzzle[((int
                                    )ListPutPuzzle[k])])))
666                                 stop = true;
667                         }
668                     }
669                     else
670                         stop = true;
671
672                     // Under søger om brikken kan
                            ligge sammen.
673                     if (!stop && ((Puzzle)aPuzzle[
                            tmpPuzzle]).GetSideActiv(
                            tmpPuzzleSide)
674                     && !LineTurn(((Puzzle)aPuzzle[
                            tmpPuzzle]), ((Puzzle)aPuzzle[
                            marsterPuzzle]), tmpPuzzleSide
                            , marsterPuzzleSide)
675                     && ((marsterPuzzleItemsEndPoint.X
                            >= tmpPuzzleItemsStartPoint.X
```

```
                    - mTol &&
                    marsterPuzzleItemsEndPoint.X
                    <= tmpPuzzleItemsStartPoint.X
                    + mTol)
676                 && (marsterPuzzleItemsEndPoint.Y
                    >= tmpPuzzleItemsStartPoint.Y
                    - mTol &&
                    marsterPuzzleItemsEndPoint.Y
                    <= tmpPuzzleItemsStartPoint.Y
                    + mTol))
677                 && ((marsterPuzzleItemsStartPoint.
                    X >= tmpPuzzleItemsEndPoint.X
                    - mTol &&
                    marsterPuzzleItemsStartPoint.X
                     <= tmpPuzzleItemsEndPoint.X +
                     mTol)
678                 && (marsterPuzzleItemsStartPoint.Y
                     >= tmpPuzzleItemsEndPoint.Y -
                     mTol &&
                    marsterPuzzleItemsStartPoint.Y
                     <= tmpPuzzleItemsEndPoint.Y +
                     mTol)))
679                 {
680                     //deaktiver siderne og klader
                        FindPuzzle funktion igen
                        for at finde nye brikker
681                     ((Puzzle)aPuzzle[tmpPuzzle]).
                        SideDeactiv(tmpPuzzleSide)
                        ;
682                     ((Puzzle)aPuzzle[marsterPuzzle
                        ]).SideDeactiv(
                        marsterPuzzleSide);
683
684                     bool tmpPut = ((Puzzle)aPuzzle
                        [tmpPuzzle]).GetPut();
685                     ((Puzzle)aPuzzle[tmpPuzzle]).
                        SetPut(true);
686
687                     ListPutPuzzle.Add(tmpPuzzle);
688
689                     FindPuzzle(tmpPuzzle,
                        activSideArray, i + 1);
```

```
690
691                             ListPutPuzzle.RemoveAt(
                                    ListPutPuzzle.Count - 1);
692
693                             ((Puzzle)aPuzzle[tmpPuzzle]).
                                    SetPut(tmpPut);
694                             ((Puzzle)aPuzzle[tmpPuzzle]).
                                    SideActiv(tmpPuzzleSide);
695                             ((Puzzle)aPuzzle[marsterPuzzle
                                    ]).SideActiv(
                                    marsterPuzzleSide);
696                         }
697
698                         stop = false;
699                     }
700                 }
701             }
702
703             // Aktiver siderne igen
704             for (int i = 0; i < SideActiv.Count; i++)
705             {
706                 int mPuzzleItems = ((
                        PuzzleItemsSideToPuzzle)SideActiv[i]).
                        masterPuzzle;
707                 int mPuzzleItemsSide = ((
                        PuzzleItemsSideToPuzzle)SideActiv[i]).
                        masterPuzzleSide;
708                 int sPuzzleItems = ((
                        PuzzleItemsSideToPuzzle)SideActiv[i]).
                        PuzzleItm;
709                 int sPuzzleItemsSide = ((
                        PuzzleItemsSideToPuzzle)SideActiv[i]).
                        PuzzleSide;
710
711                 ((Puzzle)aPuzzle[mPuzzleItems]).SideActiv(
                        mPuzzleItemsSide);
712                 ((Puzzle)aPuzzle[sPuzzleItems]).SideActiv(
                        sPuzzleItemsSide);
713             }
714
715             // trækker brikken fra listen
716             ListBefore.Add(Liste);
```

```
717             Liste -= puzzleNr;
718             Liste = Liste / 10;
719 
720             return false;
721     }
722 
723         /// <summary>
724         /// Det er brikkerne overlapper
725         /// </summary>
726         private bool LineTurn(Puzzle p1, Puzzle p2, int
                sideP1, int sideP2)
727         {
728             PointPuzzle cenP1 = p1.GetCenter();
729             PointPuzzle cenP2 = p2.GetCenter();
730 
731             PointPuzzle midP = p1.GetEndPoint(sideP1) + p1
                    .GetStartPoint(sideP1);
732             midP.X = midP.X / 2;
733             midP.Y = midP.Y / 2;
734 
735             int p1_NSEW = Kompas(midP, cenP1);
736             int p2_NSEW = Kompas(midP, cenP2);
737 
738             if (p1_NSEW == p2_NSEW)
739             {
740                 //listBox1.Items.Add("Brik : " + p1.GetID
                        () + " Side : " + sideP1 + " / Ret : "
741                 //    + p1_NSEW + " og Brik : " + p2.GetID
                        () + " Side : " + sideP2 + " / Ret : "
                         + p2_NSEW);
742                 return true;
743             }
744 
745             if (RadiusMetode(p1, p2))
746             {
747                 //listBox1.Items.Add("Brik : " + p1.GetID
                        () + " Side : " + sideP1 + " / Ret : "
748                 //    + p1_NSEW + " og Brik : " + p2.GetID
                        () + " Side : " + sideP2 + " / Ret : "
                         + p2_NSEW);
749                 return true;
750             }
```

```
751
752             //listBox1.Items.Add("retning -> " + p1_NSEW +
                    " And " + p2_NSEW);
753
754             return false;
755         }
756
757         /// <summary>
758         /// RadiusMetode
759         /// </summary>
760         private bool RadiusMetode(Puzzle p1, Puzzle p2)
761         {
762             PointPuzzle cenP1 = p1.GetCenter();
763             PointPuzzle cenP2 = p2.GetCenter();
764
765             double tmpLen = Math.Sqrt((cenP1.X - cenP2.X)
                    * (cenP1.X - cenP2.X) + (cenP1.Y - cenP2.Y
                    ) * (cenP1.Y - cenP2.Y));
766             double r1 = p1.GetRadius();
767             double r2 = p2.GetRadius();
768
769             if (r1 > tmpLen || r2 > tmpLen)
770             {
771                 return true;
772             }
773
774             return false;
775         }
776
777         /// <summary>
778         /// KompasMetode
779         /// </summary>
780         private int Kompas(PointPuzzle mid, PointPuzzle
                cen)
781         {
782             int NSWE = 0;
783
784             if (mid.X > cen.X)
785                 NSWE = (NSWE + 7);
786             else if (mid.X < cen.X)
787                 NSWE = (NSWE + 3);
788
```

```
789             if (mid.Y < cen.Y)
790             {
791                 if (NSWE == 0)
792                     NSWE = (NSWE + 5);
793                 else
794                     NSWE = (NSWE + 5) / 2;
795             }
796             else if (mid.Y > cen.Y)
797             {
798                 if (NSWE == 0)
799                     NSWE = (NSWE + 1);
800                 else if (NSWE == 7)
801                     NSWE = (NSWE + 9) / 2;
802                 else
803                     NSWE = (NSWE + 1) / 2;
804             }
805 
806             return NSWE;
807         }
808 
809         /// <summary>
810         /// Finde sider om der passer til len
811         /// </summary>
812         private ArrayList FindPuzzleSide(double len, int
                NotPuzzle, int l, int r)
813         {
814             float tol = 1.0f * mTol;
815             int m = (l+r)/2;
816             double tmpError = len - ((PuzzleItemsSide)
                    aPuzzleLength[m]).len;
817 
818             //if (((PuzzleItemsSide)aPuzzleLength[m]).len
                    >= (len - mTol) && ((PuzzleItemsSide)
                    aPuzzleLength[m]).len <= (len + mTol))
819             if (tmpError <= tol && tmpError >= -tol)
820             {
821                 ArrayList tmpArr = new ArrayList();
822                 int tmpPuzzle = ((PuzzleItemsSide)
                        aPuzzleLength[m]).masterPuzzle;
823                 int tmpPuzzleSide = ((PuzzleItemsSide)
                        aPuzzleLength[m]).masterPuzzleSide;
824 
```

```
825                 if(tmpPuzzle != NotPuzzle && !((Puzzle)
                        aPuzzle[tmpPuzzle]).GetUse())
826                     if (((Puzzle)aPuzzle[tmpPuzzle]).
                            GetSideActiv(tmpPuzzleSide))
827                         tmpArr.Add(new PuzzleItemsSide(
                                tmpPuzzle, tmpPuzzleSide));
828
829                 int tmpM = m - 1;
830                 while (tmpM >= 0)// && ((PuzzleItemsSide)
                        aPuzzleLength[tmpM]).len == len)
831                 {
832                     tmpError = len - ((PuzzleItemsSide)
                            aPuzzleLength[tmpM]).len;
833                     if (tmpError <= tol && tmpError >= -
                            tol)
834                     {
835                         tmpPuzzle = ((PuzzleItemsSide)
                                aPuzzleLength[tmpM]).
                                masterPuzzle;
836                         tmpPuzzleSide = ((PuzzleItemsSide)
                                aPuzzleLength[tmpM]).
                                masterPuzzleSide;
837
838                         if (tmpPuzzle != NotPuzzle && !((
                                Puzzle)aPuzzle[tmpPuzzle]).
                                GetUse())
839                             if (((Puzzle)aPuzzle[tmpPuzzle
                                    ]).GetSideActiv(
                                    tmpPuzzleSide))
840                                 tmpArr.Add(new
                                        PuzzleItemsSide(
                                        tmpPuzzle,
                                        tmpPuzzleSide));
841
842                         tmpM--;
843                     }
844                     else
845                         break;
846                 }
847
848                 tmpM = m + 1;
```

```
849                 while (tmpM < aPuzzleLength.Count) //&& ((
                        PuzzleItemsSide)aPuzzleLength[tmpM]).
                        len == len)
850                 {
851                     tmpError = len - ((PuzzleItemsSide)
                            aPuzzleLength[tmpM]).len;
852                     if (tmpError <= tol && tmpError >= -
                            tol)
853                     {
854                         tmpPuzzle = ((PuzzleItemsSide)
                                aPuzzleLength[tmpM]).
                                masterPuzzle;
855                         tmpPuzzleSide = ((PuzzleItemsSide)
                                aPuzzleLength[tmpM]).
                                masterPuzzleSide;
856
857                         if (tmpPuzzle != NotPuzzle && !((
                                Puzzle)aPuzzle[tmpPuzzle]).
                                GetUse())
858                             if (((Puzzle)aPuzzle[tmpPuzzle
                                    ]).GetSideActiv(
                                    tmpPuzzleSide))
859                                 tmpArr.Add(new
                                        PuzzleItemsSide(
                                        tmpPuzzle,
                                        tmpPuzzleSide));
860
861                         tmpM++;
862                     }
863                     else
864                         break;
865
866                 }
867
868                 return tmpArr;
869             }
870
871             if (l == r || l > r)
872                 return new ArrayList();
873
874             if (((PuzzleItemsSide)aPuzzleLength[m]).len >
                    len)
```

```
875                 return FindPuzzleSide(len, NotPuzzle, l, m
                        - 1);
876             else
877                 return FindPuzzleSide(len, NotPuzzle, m+1,
                        r);
878
879         }
880
881         /// <summary>
882         /// Finde sider som brikkerne har tilfælles
883         /// </summary>
884         private ArrayList FindPuzzleSideToPuzzle(int
                puzzleItm, int marsterPuzzleItm)
885         {
886             ArrayList tmpArr = new ArrayList();
887             double tol = 0.5;
888
889             for (int i = 0; i < ((Puzzle)aPuzzle[
                    marsterPuzzleItm]).GetCount(); i++)
890             {
891                 double len = ((Puzzle)aPuzzle[
                        marsterPuzzleItm]).GetLength(i);
892                 PointPuzzle mpuzzleItemsStartPoint = ((
                        Puzzle)aPuzzle[marsterPuzzleItm]).
                        GetStartPoint(i);
893                 PointPuzzle mpuzzleItemsEndPoint = ((
                        Puzzle)aPuzzle[marsterPuzzleItm]).
                        GetEndPoint(i);
894
895                 for (int j = 0; j < ((Puzzle)aPuzzle[
                        puzzleItm]).GetCount(); j++)
896                 {
897                     PointPuzzle puzzleItemsStartPoint = ((
                            Puzzle)aPuzzle[puzzleItm]).
                            GetStartPoint(j);
898                     PointPuzzle puzzleItemsEndPoint = ((
                            Puzzle)aPuzzle[puzzleItm]).
                            GetEndPoint(j);
899
900                     double tmpError = len - ((Puzzle)
                            aPuzzle[puzzleItm]).GetLength(j);
901
```

```
902                     //if (((PuzzleItemsSide)aPuzzleLength[
                            m]).len >= (len - mTol) && ((
                            PuzzleItemsSide)aPuzzleLength[m]).
                            len <= (len + mTol))
903                     if (((Puzzle)aPuzzle[puzzleItm]).
                            GetSideActiv(j) && tmpError <= tol
                             && tmpError >= -tol)
904                     //if (((Puzzle)aPuzzle[puzzleItm]).
                            GetSideActiv(j) && len == ((Puzzle
                            )aPuzzle[puzzleItm]).GetLength(j))
905                     {
906                         //if (puzzleItemsStartPoint ==
                                mpuzzleItemsEndPoint &&
                                mpuzzleItemsStartPoint ==
                                puzzleItemsEndPoint)
907                         if (((mpuzzleItemsEndPoint.X >=
                                puzzleItemsStartPoint.X - mTol
                                 && mpuzzleItemsEndPoint.X <=
                                puzzleItemsStartPoint.X + mTol
                                )
908                     && (mpuzzleItemsEndPoint.Y >=
                            puzzleItemsStartPoint.Y - mTol &&
                            mpuzzleItemsEndPoint.Y <=
                            puzzleItemsStartPoint.Y + mTol))
909                     && ((mpuzzleItemsStartPoint.X >=
                            puzzleItemsEndPoint.X - mTol &&
                            mpuzzleItemsStartPoint.X <=
                            puzzleItemsEndPoint.X + mTol)
910                     && (mpuzzleItemsStartPoint.Y >=
                            puzzleItemsEndPoint.Y - mTol &&
                            mpuzzleItemsStartPoint.Y <=
                            puzzleItemsEndPoint.Y + mTol)))
911                             tmpArr.Add(new
                                    PuzzleItemsSideToPuzzle(
                                    marsterPuzzleItm, i,
                                    puzzleItm, j));
912                     }
913                 }
914             }
915             return tmpArr;
916         }
917
```

```
918         /// <summary>
919         /// Funktion som Mergesort skal bruge
920         /// </summary>
921         private void Merge(ArrayList a, int l, int m, int
                r)
922         {
923             int i, j;
924
925             ArrayList aux = new ArrayList();
926             for (int n = 0; n < a.Count; n++)
927                 aux.Add(new PuzzleItemsSide());
928
929
930                 for (i = m + 1; i > l; i--)
931                     aux[i - 1] = a[i - 1];
932
933             for (j = m; j < r; j++)
934                 aux[r + m - j] = a[j + 1];
935
936             for (int k = l; k <= r; k++)
937                 if (((PuzzleItemsSide)aux[j]).len < ((
                        PuzzleItemsSide)aux[i]).len)
938                     a[k] = aux[j--];
939                 else
940                     a[k] = aux[i++];
941
942         }
943
944         /// <summary>
945         /// Funktion som soter listen med
946         /// </summary>
947         private void Mergesort(ArrayList a, int l, int r)
948         {
949             if (r <= l)
950                 return;
951
952             int m = (r + l) / 2;
953
954             Mergesort(a, l, m);
955
956             Mergesort(a, m+1, r);
957
```

```
958             Merge(a, l, m, r);
959
960         }
961
962         /// <summary>
963         /// Gemmer løsning i en billede fil
964         /// </summary>
965         private void SaveSolution()
966         {
967             int numbe = mFileCon;
968             ArrayList tmpAPuzzle = aPuzzle;
969             try
970             {
971                 // ArgumentException is thrown because 7
                        is not an even number.
972                 Console.WriteLine("7 divided by 2 is {0}")
                        ;
973                 Bitmap newBitmap = new Bitmap(1000, 1000,
                        PixelFormat.Format32bppArgb);
974                 Graphics g = Graphics.FromImage(newBitmap)
                        ;
975
976                 Font drawFont = new Font("Arial", 16);
977                 Font drawFont2 = new Font("Arial", 7);
978                 SolidBrush drawBrush = new SolidBrush(
                        Color.Black);
979                 int xt = 300, yt = 400;
980                 //float xt = 0, yt = 0;
981                 //int xt = 0, yt = 0;
982                 float aXm = 0, aYm = 0, aYM = 0, aXM = 0;
983                 for (int i = 0; i < tmpAPuzzle.Count; i++)
984                 {
985                     if (!((Puzzle)tmpAPuzzle[i]).GetUse()
                            && ((Puzzle)tmpAPuzzle[i]).GetPut
                            ())
986                     {
987                         for (int j = 0; j < ((Puzzle)
                                tmpAPuzzle[i]).GetCount(); j
                                ++)
988                         {
989                             if (aXm > ((Puzzle)tmpAPuzzle[
                                    i]).GetStartPoint(j).X)
```

```
990                         aXm = ((Puzzle)tmpAPuzzle[
                                i]).GetStartPoint(j).X
                                ;
991                     else if (aXm > ((Puzzle)
                            tmpAPuzzle[i]).GetEndPoint
                            (j).X)
992                         aXm = ((Puzzle)tmpAPuzzle[
                                i]).GetEndPoint(j).X;
993
994                     if (aXM < ((Puzzle)tmpAPuzzle[
                            i]).GetStartPoint(j).X)
995                         aXM = ((Puzzle)tmpAPuzzle[
                                i]).GetStartPoint(j).X
                                ;
996                     else if (aXM < ((Puzzle)
                            tmpAPuzzle[i]).GetEndPoint
                            (j).X)
997                         aXM = ((Puzzle)tmpAPuzzle[
                                i]).GetEndPoint(j).X;
998
999                     if (aYm > ((Puzzle)tmpAPuzzle[
                            i]).GetStartPoint(j).Y)
1000                        aYm = ((Puzzle)tmpAPuzzle[
                                i]).GetStartPoint(j).Y
                                ;
1001                    else if (aYm > ((Puzzle)
                            tmpAPuzzle[i]).GetEndPoint
                            (j).Y)
1002                        aYm = ((Puzzle)tmpAPuzzle[
                                i]).GetEndPoint(j).Y;
1003
1004                    if (aYM < ((Puzzle)tmpAPuzzle[
                            i]).GetStartPoint(j).Y)
1005                        aYM = ((Puzzle)tmpAPuzzle[
                                i]).GetStartPoint(j).Y
                                ;
1006                    else if (aYM < ((Puzzle)
                            tmpAPuzzle[i]).GetEndPoint
                            (j).Y)
1007                        aYM = ((Puzzle)tmpAPuzzle[
                                i]).GetEndPoint(j).Y;
1008
```

```
1009                    if (((Puzzle)tmpAPuzzle[i]).
                            GetSideActiv(j))
1010                    {
1011                        g.DrawLine(Pens.Red,
1012                            ((Puzzle)tmpAPuzzle[i
                                ]).GetStartPoint(j
                                ).X + xt, ((Puzzle
                                )tmpAPuzzle[i]).
                                GetStartPoint(j).Y
                                 + yt,
1013                            ((Puzzle)tmpAPuzzle[i
                                ]).GetEndPoint(j).
                                X + xt, ((Puzzle)
                                tmpAPuzzle[i]).
                                GetEndPoint(j).Y +
                                 yt);
1014                    }
1015                    else
1016                    {
1017                        g.DrawLine(Pens.Green,
1018                            ((Puzzle)tmpAPuzzle[i
                                ]).GetStartPoint(j
                                ).X + xt, ((Puzzle
                                )tmpAPuzzle[i]).
                                GetStartPoint(j).Y
                                 + yt,
1019                            ((Puzzle)tmpAPuzzle[i
                                ]).GetEndPoint(j).
                                X + xt, ((Puzzle)
                                tmpAPuzzle[i]).
                                GetEndPoint(j).Y +
                                 yt);
1020                    }
1021
1022                    g.FillEllipse(drawBrush, ((
                        Puzzle)tmpAPuzzle[i]).
                        GetStartPoint(j).X + xt -
                        4, ((Puzzle)tmpAPuzzle[i])
                        .GetStartPoint(j).Y + yt -
                         4, 4, 4);
1023                    g.FillEllipse(drawBrush, ((
                        Puzzle)tmpAPuzzle[i]).
```

```
                                GetEndPoint(j).X + xt - 4,
                                 ((Puzzle)tmpAPuzzle[i]).
                                GetEndPoint(j).Y + yt - 4,
                                 4, 4);
1024
1025                        /*//Skriver kooridater ud
1026                        float xs = ((Puzzle)aPuzzle[i
                                ]).GetStartPoint(j).X*1.0F
                                ;
1027                        float ys = ((Puzzle)aPuzzle[i
                                ]).GetStartPoint(j).Y*1.0F
                                ;
1028                        string text = "("+ ((Puzzle)
                                aPuzzle[i]).GetStartPoint(
                                j).X + ","+ ((Puzzle)
                                aPuzzle[i]).GetStartPoint(
                                j).Y +")";
1029                        e.Graphics.DrawString(text,
                                drawFont2, drawBrush, xs,
                                ys);
1030                        */
1031                    }
1032
1033                    PointPuzzle tmpPoint = ((Puzzle)
                            tmpAPuzzle[i]).GetCenter();
1034                    float x = tmpPoint.X * 1.0F + xt;
1035                    float y = tmpPoint.Y * 1.0F + yt;
1036                    float rad = (float)((Puzzle)
                            tmpAPuzzle[i]).GetRadius();
1037
1038                    //g.DrawEllipse(Pens.Blue, x - rad
                            , y - rad, rad * 2, rad * 2);
1039                    g.FillEllipse(drawBrush, x - 1, y
                            - 1, 1 * 2, 1 * 2);
1040
1041                    g.DrawString("ID" + ((Puzzle)
                            tmpAPuzzle[i]).GetID(),
1042                                                drawFont
                                                    ,
                                                drawBrush
                                                    ,
                                                x,
```

```
                                                                    y
                                                                   );
1043
1044                        ArrayList tmpCenterList = ((Puzzle
                                )tmpAPuzzle[i]).GetCenterList
                                ();
1045
1046                        for (int j = 0; j < tmpCenterList.
                                Count; j++)
1047                        {
1048                            tmpPoint = ((PuzzleItemsCenter
                                    )tmpCenterList[j]).cen;
1049                            x = tmpPoint.X * 1.0F + xt;
1050                            y = tmpPoint.Y * 1.0F + yt;
1051
1052                            g.DrawString("ID" + ((
                                    PuzzleItemsCenter)
                                    tmpCenterList[j]).nr,
1053                                                        drawFont
                                                            ,
                                                            drawBrush
                                                            ,
                                                            x,
                                                             y
                                                            );
1054                        }
1055                    }
1056
1057                }
1058                double tmpAreal = (aXM - aXm) * (aYM - aYm
                        );
1059
1060                if (tmpAreal <= mAreal)
1061                {
1062                    mAreal = tmpAreal;
1063                    if (mAreal < (770 * 560))
1064                        mAreal = (770 * 560);
1065                    newBitmap.Save("Z" + numbe + ".png",
                            ImageFormat.Png);
1066                }
1067
1068            }
```

```
1069            catch (ArgumentException)
1070            {
1071                // Show the user that 7 cannot be divided
                        by 2.
1072                Console.WriteLine("7 is not divided by 2
                        integrally.");
1073                MessageBox.Show("Nej den kan ikke nr : " +
                         mFileCon);
1074            }
1075        }
1076
1077        /// <summary>
1078        /// Gemmer løsning i en Text fil
1079        /// </summary>
1080        private void SaveSolutionInText(int numbe)
1081        {
1082            try
1083            {
1084                StreamWriter fr = new StreamWriter("ZB" +
                        numbe + ".csv");
1085                StreamWriter fc = new StreamWriter("ZC" +
                        numbe + ".csv");
1086                int xt = 300, yt = 400;
1087                float xs = 0, ys = 0;
1088                fr.WriteLine("x;y;c");
1089                fc.WriteLine("x;y;n");
1090
1091                for (int i = 0; i < aPuzzle.Count; i++)
1092                {
1093                    if (!((Puzzle)aPuzzle[i]).GetUse() &&
                            ((Puzzle)aPuzzle[i]).GetPut())
1094                    {
1095                        for (int j = 0; j < ((Puzzle)
                                aPuzzle[i]).GetCount(); j++)
1096                        {
1097                            if (((Puzzle)aPuzzle[i]).
                                    GetSideActiv(j))
1098                            {
1099                                xs = (((Puzzle)aPuzzle[i])
                                        .GetStartPoint(j).X +
                                        xt);
```

```
1100                        ys = (((Puzzle)aPuzzle[i])
                                .GetStartPoint(j).Y +
                                yt);
1101                        fr.WriteLine(xs + ";" + ys
                                 + ";r");
1102                        xs = (((Puzzle)aPuzzle[i])
                                .GetEndPoint(j).X + xt
                                );
1103                        ys = (((Puzzle)aPuzzle[i])
                                .GetEndPoint(j).Y + yt
                                );
1104                        fr.WriteLine(xs + ";" + ys
                                 + ";r");
1105                    }
1106                    else
1107                    {
1108                        xs = (((Puzzle)aPuzzle[i])
                                .GetStartPoint(j).X +
                                xt);
1109                        ys = (((Puzzle)aPuzzle[i])
                                .GetStartPoint(j).Y +
                                yt);
1110                        fr.WriteLine(xs + ";" + ys
                                 + ";g");
1111                        xs = (((Puzzle)aPuzzle[i])
                                .GetEndPoint(j).X + xt
                                );
1112                        ys = (((Puzzle)aPuzzle[i])
                                .GetEndPoint(j).Y + yt
                                );
1113                        fr.WriteLine(xs + ";" + ys
                                 + ";g");
1114                    }
1115                }
1116
1117                PointPuzzle tmpPoint = ((Puzzle)
                    aPuzzle[i]).GetCenter();
1118                xs = tmpPoint.X * 1.0F + xt;
1119                ys = tmpPoint.Y * 1.0F + yt;
1120
1121                fc.WriteLine(xs + ";" + ys + ";" +
                     ((Puzzle)aPuzzle[i]).GetID())
```

```
                                    ;
1122
1123                                ArrayList tmpCenterList = ((Puzzle
                                        )aPuzzle[i]).GetCenterList();
1124
1125                                for (int j = 0; j < tmpCenterList.
                                        Count; j++)
1126                                {
1127                                    tmpPoint = ((PuzzleItemsCenter
                                            )tmpCenterList[j]).cen;
1128                                    xs = tmpPoint.X * 1.0F + xt;
1129                                    ys = tmpPoint.Y * 1.0F + yt;
1130
1131                                    fc.WriteLine(xs + ";" + ys + "
                                            ;" + ((PuzzleItemsCenter)
                                            tmpCenterList[j]).nr);
1132                                }
1133                            }
1134
1135                        }
1136
1137                        fc.Close();
1138                        fr.Close();
1139
1140                    }
1141                    catch (ArgumentException)
1142                    {
1143                        // Show the user that 7 cannot be divided
                                by 2.
1144                        //Console.WriteLine("7 is not divided by 2
                                integrally.");
1145                        MessageBox.Show("Nej den kan ikke nr : " +
                                mFileCon);
1146                        //listBox1.Items.Add("Nej den kan ikke nr
                                : " + mFileCon);
1147                    }
1148                }
1149
1150
1151            }
1152        }
```

C.1.3 Puzzle.cs

```
1  using System;
2  using System.Collections;
3  using System.ComponentModel;
4  using System.Data;
5
6  namespace PuzzleAlgo3
7  {
8   public struct PointPuzzle
9   {
10   public PointPuzzle(float inX, float inY)
11   {
12    this.X = inX;
13    this.Y = inY;
14   }
15
16   public static bool operator ==(PointPuzzle mat1,
         PointPuzzle mat2)
17   {
18    if(mat1.X == mat2.X && mat1.Y == mat2.Y)
19     return true;
20    else
21     return false;
22   }
23
24   public static bool operator !=(PointPuzzle mat1,
         PointPuzzle mat2)
25   {
26    if(mat1.X == mat2.X && mat1.Y == mat2.Y)
27     return false;
28    else
29     return true;
30   }
31
32   public static PointPuzzle operator +(PointPuzzle mat1,
         PointPuzzle mat2)
33   {
34    return new PointPuzzle(mat1.X + mat2.X, mat1.Y + mat2.Y
         );
35   }
36
```

```
37    public static PointPuzzle operator -(PointPuzzle mat1,
          PointPuzzle mat2)
38    {
39     return new PointPuzzle(mat1.X - mat2.X, mat1.Y - mat2.Y
           );
40    }
41
42
43    public float X;
44    public float Y;
45   }
46
47   public struct PuzzleItemsLength
48   {
49    public PuzzleItemsLength(double inLen, int inR, int inL,
          PointPuzzle inStP, PointPuzzle inElP)
50    {
51     this.len = inLen;
52     this.r_node = inR;
53     this.l_node = inL;
54
55     this.startPoint = inStP;
56     this.endPoint = inElP;
57
58     this.activ = true;
59     this.puzzleItem = -1;
60     this.puzzleItemSide = -1;
61    }
62
63    public double len;
64    public int r_node;
65    public int l_node;
66    public PointPuzzle startPoint;
67    public PointPuzzle endPoint;
68
69    public bool activ;
70    public int puzzleItem;
71    public int puzzleItemSide;
72
73   }
74
75     public struct PuzzleItemsSide
```

```
76      {
77          public PuzzleItemsSide(int masterPuzzle, int
                masterPuzzleSide)
78          {
79              this.masterPuzzle = masterPuzzle;
80              this.masterPuzzleSide = masterPuzzleSide;
81              this.len = 0;
82          }
83
84          public PuzzleItemsSide(double len, int
                masterPuzzle, int masterPuzzleSide)
85          {
86              this.len = len;
87              this.masterPuzzle = masterPuzzle;
88              this.masterPuzzleSide = masterPuzzleSide;
89          }
90
91          public int masterPuzzle;
92          public int masterPuzzleSide;
93          public double len;
94
95      }
96
97      public struct PuzzleItemsSideToPuzzle
98      {
99          public PuzzleItemsSideToPuzzle(int masterPuzzle,
                int masterPuzzleSide, int PuzzleItm, int
                PuzzleSide)
100         {
101             this.masterPuzzle = masterPuzzle;
102             this.masterPuzzleSide = masterPuzzleSide;
103             this.PuzzleItm = PuzzleItm;
104             this.PuzzleSide = PuzzleSide;
105         }
106
107         public int masterPuzzle;
108         public int masterPuzzleSide;
109         public int PuzzleItm;
110         public int PuzzleSide;
111     }
112
113     public struct PuzzleItemsCenter
```

```
114     {
115         public PuzzleItemsCenter(int nr, PointPuzzle cen)
116         {
117             this.cen = cen;
118             this.nr = nr;
119         }
120 
121         public int nr;
122         public PointPuzzle cen;
123 
124     }
125 
126 /// <summary>
127 /// Summary description for Puzzle.
128 /// </summary>
129 public class Puzzle
130 {
131  private ArrayList aPuzzleLength;
132         private ArrayList aPuzzleCenter;
133         private int mCount;
134  private PointPuzzle mCenter;
135  private int mID;
136  private bool mUse = false;
137         private double mRadius;
138         private bool mPut = false;
139 
140 
141  public Puzzle()
142  {
143   mCount = 0;
144 
145   mCenter  = new PointPuzzle(0,0);
146   aPuzzleLength = new ArrayList();
147             aPuzzleCenter = new ArrayList();
148 
149             mID = -1;
150             mRadius = 0.0;
151  }
152 
153         //-------- Set/Get Center -------------
154 
```

```
155         public void AddCenterToList(int nr, PointPuzzle
                cen)
156         {
157             aPuzzleCenter.Add(new PuzzleItemsCenter(nr,
                    cen));
158         }
159
160         public ArrayList GetCenterList()
161         {
162             return aPuzzleCenter;
163         }
164
165         //-------- Set/Get Radius -------------
166
167         public void SetRadius(double inR)
168         {
169             mRadius = inR;
170         }
171
172         public double GetRadius()
173         {
174             return mRadius;
175         }
176
177
178     //-------- Set/Get Uset -------------
179
180     public void SetUse(bool use)
181     {
182      mUse = use;
183     }
184
185         public bool GetUse()
186         {
187             return mUse;
188         }
189
190         //-------- Set/Get Put -------------
191
192         public void SetPut(bool put)
193         {
194             mPut = put;
```

```
195         }
196
197         public bool GetPut()
198         {
199             return mPut;
200         }
201
202     //-------- Set/Get Center -------------
203
204     public PointPuzzle GetCenter()
205     {
206      return mCenter;
207     }
208
209     public void SetCenter(int x, int y)
210     {
211      mCenter = new PointPuzzle(x,y);
212     }
213
214     //-------- Set/Get ID -------------
215
216     public int GetID()
217     {
218      return mID;
219     }
220
221     public void SetID(int id)
222     {
223      mID = id;
224     }
225
226     //-------- Get count -------------
227
228     public int GetCount()
229     {
230      return mCount;
231     }
232
233     //-------- Get/Set Side to Side -------------
234
235     public void SetNeighbourPuzzle(int pos, int puzzle, int
            puzzleSide)
```

```
236     {
237             int tmpInt = pos;
238
239             if (pos < 0)
240                 tmpInt = mCount - 1;
241
242             if (pos >= mCount)
243                 tmpInt = pos - mCount;
244
245             PuzzleItemsLength tmpPuzzleItemsLength = ((
                    PuzzleItemsLength)aPuzzleLength[tmpInt]);
246      tmpPuzzleItemsLength.puzzleItem = puzzle;
247      tmpPuzzleItemsLength.puzzleItemSide = puzzleSide;
248             aPuzzleLength[tmpInt] = tmpPuzzleItemsLength;
249     }
250
251     public int GetNeighbourPuzzle(int pos)
252     {
253             int tmpInt = pos;
254
255             if (pos < 0)
256                 tmpInt = mCount - 1;
257
258             if (pos >= mCount)
259                 tmpInt = pos - mCount;
260
261             return ((PuzzleItemsLength)aPuzzleLength[
                    tmpInt]).puzzleItem;
262     }
263
264     public int GetNeighbourPuzzleSide(int pos)
265     {
266             int tmpInt = pos;
267
268             if (pos < 0)
269                 tmpInt = mCount - 1;
270
271             if (pos >= mCount)
272                 tmpInt = pos - mCount;
273
274             return ((PuzzleItemsLength)aPuzzleLength[
                    tmpInt]).puzzleItemSide;
```

```
275     }
276
277
278     //-------- Get/Set Side activ -------------
279           public void SideActiv(int pos)
280     {
281               int tmpInt = pos;
282
283               if (pos < 0)
284                   tmpInt = mCount - 1;
285
286               PuzzleItemsLength tmpPuzzleItemsLength = ((
                      PuzzleItemsLength)aPuzzleLength[tmpInt]);
287      tmpPuzzleItemsLength.activ = true;
288               aPuzzleLength[tmpInt] = tmpPuzzleItemsLength;
289     }
290
291     public void SideDeactiv(int pos)
292     {
293               int tmpInt = pos;
294
295               if (pos < 0)
296                   tmpInt = mCount - 1;
297
298               PuzzleItemsLength tmpPuzzleItemsLength = ((
                      PuzzleItemsLength)aPuzzleLength[tmpInt]);
299      tmpPuzzleItemsLength.activ = false;
300               aPuzzleLength[tmpInt] = tmpPuzzleItemsLength;
301     }
302
303     public bool GetSideActiv(int pos)
304     {
305               int tmpInt = pos;
306
307               if (pos < 0)
308                   tmpInt = mCount - 1;
309
310               if (pos >= mCount)
311                   tmpInt = pos - mCount;
312
313               return ((PuzzleItemsLength)aPuzzleLength[
                      tmpInt]).activ;
```

```
314     }
315
316
317     //-------- Get Start End Point -------------
318
319     public PointPuzzle GetStartPoint(int pos)
320     {
321               int tmpInt = pos;
322
323               if (pos < 0)
324                   tmpInt = mCount - 1;
325
326               if (pos >= mCount)
327                   tmpInt = pos - mCount;
328
329               return ((PuzzleItemsLength)aPuzzleLength[
                      tmpInt]).startPoint;
330     }
331
332     public PointPuzzle GetEndPoint(int pos)
333     {
334               int tmpInt = pos;
335
336               if (pos < 0)
337                   tmpInt = mCount - 1;
338
339               if (pos >= mCount)
340                   tmpInt = pos - mCount;
341
342               return ((PuzzleItemsLength)aPuzzleLength[
                      tmpInt]).endPoint;
343     }
344
345
346     //-------- Get Length -------------
347
348     public double GetLength(int pos)
349     {
350               int tmpInt = pos;
351
352               if (pos < 0)
353                   tmpInt = mCount - 1;
```

```
354
355             if (pos >= mCount)
356                 tmpInt = pos - mCount;
357
358             return ((PuzzleItemsLength)aPuzzleLength[
                    tmpInt]).len;
359     }
360
361         public PuzzleItemsLength[] GetAllSide()
362         {
363             PuzzleItemsLength[] tmpArray = new
                    PuzzleItemsLength[mCount];
364
365             for (int i = 0; i < mCount; i++)
366                 if (((PuzzleItemsLength)aPuzzleLength[i]).
                        activ)
367                     tmpArray[i] = ((PuzzleItemsLength)
                            aPuzzleLength[i]);
368
369             return tmpArray;
370         }
371
372
373     //-------- Set the lenght --------------
374
375     public int SetLength(double length, PointPuzzle startP,
            PointPuzzle slutP)
376     {
377      mCount++;
378
379      aPuzzleLength.Add(new PuzzleItemsLength(length,mCount
            -1, 0, startP, slutP));
380      PuzzleItemsLength tmpPuzzleItemsLength = ((
            PuzzleItemsLength)aPuzzleLength[mCount-1]);
381      tmpPuzzleItemsLength.r_node = mCount;
382              tmpPuzzleItemsLength.l_node = mCount-2;
383      aPuzzleLength[mCount-1] = tmpPuzzleItemsLength;
384
385              return mCount;
386     }
387
388
```

```
389         //-------- Set the end point --------------
390         public void SetEndPoint(double length, PointPuzzle
                endPoint, int pos)
391         {
392             PuzzleItemsLength tmpPuzzleItemsLength = ((
                    PuzzleItemsLength)aPuzzleLength[pos]);
393             tmpPuzzleItemsLength.len = length;
394             tmpPuzzleItemsLength.endPoint = endPoint;
395             aPuzzleLength[pos] = tmpPuzzleItemsLength;
396         }
397
398
399         //-------- Sletter den siden på pos plas
                --------------
400         public void DelSide(int pos)
401         {
402             mCount--;
403             aPuzzleLength.RemoveAt(pos);
404         }
405
406
407     //-------- Flytter brikken til Orion ----------------
408     public void MoveToOrion(PointPuzzle orion, PointPuzzle
            point)
409         {
410             PointPuzzle tmpPoint = orion - point;
411
412             for (int i = 0; i < aPuzzleLength.Count; i++)
413             {
414                 PuzzleItemsLength tmpPuzzleItemsLength =
                        ((PuzzleItemsLength)aPuzzleLength[i]);
415                 tmpPuzzleItemsLength.startPoint = (
                        tmpPuzzleItemsLength.startPoint +
                        tmpPoint);
416                 tmpPuzzleItemsLength.endPoint = (
                        tmpPuzzleItemsLength.endPoint +
                        tmpPoint);
417                 aPuzzleLength[i] = tmpPuzzleItemsLength;
418             }
419
420             mCenter = (mCenter + tmpPoint);
421
```

```
422             for (int i = 0; i < aPuzzleCenter.Count; i++)
423             {
424                 PuzzleItemsCenter tmpPuzzleItemsCen = ((
                        PuzzleItemsCenter)aPuzzleCenter[i]);
425                 tmpPuzzleItemsCen.cen = (tmpPuzzleItemsCen
                        .cen + tmpPoint);
426
427                 aPuzzleCenter[i] = tmpPuzzleItemsCen;
428             }
429         }
430
431
432         //-------- Set Orion Back----------------
433         public void SetOrionBack()
434         {
435             PointPuzzle tmpBack = ((PuzzleItemsLength)
                    aPuzzleLength[0]).startPoint;
436             for (int i = 0; i < aPuzzleLength.Count; i++)
437             {
438                 PuzzleItemsLength tmpPuzzleItemsLength =
                        ((PuzzleItemsLength)aPuzzleLength[i]);
439                 tmpPuzzleItemsLength.startPoint =
                        tmpPuzzleItemsLength.startPoint -
                        tmpBack;
440                 tmpPuzzleItemsLength.endPoint =
                        tmpPuzzleItemsLength.endPoint -
                        tmpBack;
441                 aPuzzleLength[i] = tmpPuzzleItemsLength;
442             }
443
444             mCenter = mCenter - tmpBack;
445
446             for (int i = 0; i < aPuzzleCenter.Count; i++)
447             {
448                 PuzzleItemsCenter tmpPuzzleItemsCen = ((
                        PuzzleItemsCenter)aPuzzleCenter[i]);
449                 tmpPuzzleItemsCen.cen = (tmpPuzzleItemsCen
                        .cen - tmpBack);
450
451                 aPuzzleCenter[i] = tmpPuzzleItemsCen;
452             }
453         }
```

```
454
455
456         //-------- Roter brikken ----------------
457
458         public void Rotet(float deg, int side)
459         {
460             PointPuzzle startPk = ((PuzzleItemsLength)
                    aPuzzleLength[side]).startPoint;
461             PointPuzzle endPk = ((PuzzleItemsLength)
                    aPuzzleLength[side]).endPoint;
462             PointPuzzle tmpPk = startPk - endPk;
463
464             float newDeg = (float)Math.Acos(tmpPk.X / Math
                    .Sqrt(tmpPk.X * tmpPk.X + tmpPk.Y * tmpPk.
                    Y));
465             if (tmpPk.Y > 0.0f)
466                 newDeg = (float)(Math.PI * 2 - newDeg);
467
468             newDeg = deg - newDeg;
469
470             //if (newDeg > -0.01 && newDeg < 0.01)
471             //    return;
472             int rCon = 0;
473
474             // bliver ved med at roter så vikel passer
475             while ((newDeg < -0.001 || newDeg > 0.001) &&
                    rCon < 10000)
476             {
477                 rCon++;
478                 // flytte alle sidene med den nye vinkle
479                 for (int i = 0; i < aPuzzleLength.Count; i
                        ++)
480                 {
481                     PuzzleItemsLength tmpPuzzleItemsLength
                            = ((PuzzleItemsLength)
                            aPuzzleLength[i]);
482
483                     PointPuzzle tmpStart =
                            tmpPuzzleItemsLength.startPoint -
                            mCenter;
484                     PointPuzzle tmpEnd =
                            tmpPuzzleItemsLength.endPoint -
```

```
                    mCenter;
485                 PointPuzzle tmpPointS = new
                        PointPuzzle(0, 0);
486                 PointPuzzle tmpPointE = new
                        PointPuzzle(0, 0);
487
488                 tmpPointS.X = (float)(Math.Cos(newDeg)
                        * tmpStart.X - Math.Sin(newDeg) *
                        tmpStart.Y);
489                 tmpPointS.Y = (float)(Math.Sin(newDeg)
                        * tmpStart.X + Math.Cos(newDeg) *
                        tmpStart.Y);
490
491                 tmpPointE.X = (float)(Math.Cos(newDeg)
                        * tmpEnd.X - Math.Sin(newDeg) *
                        tmpEnd.Y);
492                 tmpPointE.Y = (float)(Math.Sin(newDeg)
                        * tmpEnd.X + Math.Cos(newDeg) *
                        tmpEnd.Y);
493
494
495                 tmpPuzzleItemsLength.startPoint =
                        tmpPointS + mCenter;
496                 tmpPuzzleItemsLength.endPoint =
                        tmpPointE + mCenter;
497                 aPuzzleLength[i] =
                        tmpPuzzleItemsLength;
498             }
499
500             // flytte alle centumerne med den nye
                    vinkle
501             for (int i = 0; i < aPuzzleCenter.Count; i
                    ++)
502             {
503                 PuzzleItemsCenter tmpPuzzleItemsCen =
                        ((PuzzleItemsCenter)aPuzzleCenter[
                        i]);
504                 PointPuzzle tmpCen = tmpPuzzleItemsCen
                        .cen - mCenter;
505                 PointPuzzle tmpPoint = new PointPuzzle
                        (0, 0);
506
```

```
507                     tmpPoint.X = (float)(Math.Cos(newDeg)
                            * tmpCen.X - Math.Sin(newDeg) *
                            tmpCen.Y);
508                     tmpPoint.Y = (float)(Math.Sin(newDeg)
                            * tmpCen.X + Math.Cos(newDeg) *
                            tmpCen.Y);
509
510                     tmpPuzzleItemsCen.cen = (tmpPoint +
                            mCenter);
511
512                     aPuzzleCenter[i] = tmpPuzzleItemsCen;
513                 }
514
515                 // finde den nye vinkle
516                 startPk = ((PuzzleItemsLength)
                        aPuzzleLength[side]).startPoint;
517                 endPk = ((PuzzleItemsLength)aPuzzleLength[
                        side]).endPoint;
518                 tmpPk = startPk - endPk;
519
520                 newDeg = (float)Math.Acos(tmpPk.X / Math.
                        Sqrt(tmpPk.X * tmpPk.X + tmpPk.Y *
                        tmpPk.Y));
521                 if (tmpPk.Y > 0.0f)
522                     newDeg = (float)(Math.PI * 2 - newDeg)
                            ;
523
524                 newDeg = deg - newDeg;
525             }
526
527         }
528
529         //------------ Get angle ---------
530         public float GetDeg(int side)
531         {
532             PointPuzzle startPk = ((PuzzleItemsLength)
                    aPuzzleLength[side]).startPoint;
533             PointPuzzle endPk = ((PuzzleItemsLength)
                    aPuzzleLength[side]).endPoint;
534             PointPuzzle tmpPk = endPk - startPk;
535
```

```
536                 float newDeg = (float)Math.Acos(tmpPk.X / Math
                        .Sqrt(tmpPk.X * tmpPk.X + tmpPk.Y * tmpPk.
                        Y));
537                 if (tmpPk.Y > 0)
538                     newDeg = (float)(Math.PI * 2 - newDeg);
539
540                 return newDeg;
541             }
542
543     }
544 }
```

C.1.4 Form2.cs

```
1  using System;
2  using System.Drawing;
3  using System.Collections;
4  using System.ComponentModel;
5  using System.Windows.Forms;
6  using System.Data;
7
8  namespace PuzzleAlgo4
9  {
10  /// <summary>
11  /// Summary description for Form2.
12  /// </summary>
13  public class Form2 : System.Windows.Forms.Form
14  {
15   private PuzzleAlgo4.viewer viewer1;
16   /// <summary>
17   /// Required designer variable.
18   /// </summary>
19   private System.ComponentModel.Container components =
         null;
20
21   public Form2()
22   {
23    //
24    // Required for Windows Form Designer support
25    //
26    InitializeComponent();
27
28    //
29    // TODO: Add any constructor code after
          InitializeComponent call
30    //
31   }
32
33   /// <summary>
34   /// Clean up any resources being used.
35   /// </summary>
36   protected override void Dispose( bool disposing )
37   {
38    if( disposing )
```

```
39      {
40       if(components != null)
41       {
42        components.Dispose();
43       }
44      }
45      base.Dispose( disposing );
46     }
47
48     #region Windows Form Designer generated code
49     /// <summary>
50     /// Required method for Designer support - do not modify
51     /// the contents of this method with the code editor.
52     /// </summary>
53     private void InitializeComponent()
54     {
55               this.viewer1 = new PuzzleAlgo4.viewer();
56               this.SuspendLayout();
57               //
58               // viewer1
59               //
60               this.viewer1.BackColor = System.Drawing.
                     SystemColors.Window;
61               this.viewer1.Location = new System.Drawing.
                     Point(10, 10);
62               this.viewer1.Name = "viewer1";
63               this.viewer1.Size = new System.Drawing.Size
                     (800, 600);
64               this.viewer1.TabIndex = 0;
65               //
66               // Form2
67               //
68               this.AutoScaleBaseSize = new System.Drawing.
                     Size(5, 13);
69               this.ClientSize = new System.Drawing.Size(827,
                      626);
70               this.Controls.Add(this.viewer1);
71               this.Name = "Form2";
72               this.Text = "Form2";
73               this.ResumeLayout(false);
74
75     }
```

```
76      #endregion
77
78    }
79  }
```

C.1.5 Viewer.cs

```
1  using System;
2  using System.Collections;
3  using System.ComponentModel;
4  using System.Drawing;
5  using System.Drawing.Imaging;
6  using System.Data;
7  using System.Windows.Forms;
8
9  namespace PuzzleAlgo4
10 {
11  /// <summary>
12  /// Summary description for viewer.
13  /// </summary>
14  public class viewer : System.Windows.Forms.UserControl
15  {
16   /// <summary>
17   /// Required designer variable.
18   /// </summary>
19   private System.ComponentModel.Container components =
         null;
20   private ArrayList a_array = new ArrayList();
21
22   public viewer()
23   {
24    // This call is required by the Windows.Forms Form
         Designer.
25    InitializeComponent();
26
27    // TODO: Add any initialization after the
         InitializeComponent call
28
29   }
30
31   /// <summary>
32   /// Clean up any resources being used.
33   /// </summary>
34   protected override void Dispose( bool disposing )
35   {
36    if( disposing )
37    {
```

```
38       if(components != null)
39       {
40        components.Dispose();
41       }
42      }
43      base.Dispose( disposing );
44     }
45
46     #region Component Designer generated code
47     /// <summary>
48     /// Required method for Designer support - do not modify
49     /// the contents of this method with the code editor.
50     /// </summary>
51     private void InitializeComponent()
52     {
53      //
54      // viewer
55      //
56      this.Name = "viewer";
57      this.Paint += new System.Windows.Forms.
            PaintEventHandler(this.viewer_Paint);
58
59     }
60     #endregion
61
62     private void viewer_Paint(object sender, System.Windows.
           Forms.PaintEventArgs e)
63     {
64      Font drawFont = new Font("Arial", 16);
65      Font drawFont2 = new Font("Arial", 7);
66      SolidBrush drawBrush = new SolidBrush(Color.Black);
67               //int xt = 500, yt = 400;
68               float xt = 0, yt = 0;
69               //int xt = 0, yt = 0;
70      for(int i=0; i<a_array.Count; i++)
71      {
72                   if (!((Puzzle)a_array[i]).GetUse())
73                   {//     ((Puzzle)a_array[i]).SetOrion(new
                         PointPuzzle(0, 0), new PointPuzzle
                         (640, 480));
74                       for (int j = 0; j < ((Puzzle)a_array[i
                             ]).GetCount(); j++)
```

```
75                  {
76                      if (xt > ((Puzzle)a_array[i]).
                            GetStartPoint(j).X)
77                          xt = ((Puzzle)a_array[i]).
                                GetStartPoint(j).X;
78
79                      if (yt > ((Puzzle)a_array[i]).
                            GetStartPoint(j).Y)
80                          yt = ((Puzzle)a_array[i]).
                                GetStartPoint(j).Y;
81                  }
82                  xt = 20 + (xt * -1);
83                  yt = 20 + (yt * -1);
84
85                  for (int j = 0; j < ((Puzzle)a_array[i
                        ]).GetCount(); j++)
86                  {
87                      if (((Puzzle)a_array[i]).
                            GetSideActiv(j))
88                      {
89                          e.Graphics.DrawLine(Pens.Red,
90                              ((Puzzle)a_array[i]).
                                    GetStartPoint(j).X +
                                    xt, ((Puzzle)a_array[i
                                    ]).GetStartPoint(j).Y
                                    + yt,
91                              ((Puzzle)a_array[i]).
                                    GetEndPoint(j).X + xt,
                                     ((Puzzle)a_array[i]).
                                    GetEndPoint(j).Y + yt)
                                    ;
92                      }
93                      else
94                      {
95                          e.Graphics.DrawLine(Pens.Green
                                ,
96                              ((Puzzle)a_array[i]).
                                    GetStartPoint(j).X +
                                    xt, ((Puzzle)a_array[i
                                    ]).GetStartPoint(j).Y
                                    + yt,
```

```
97                              ((Puzzle)a_array[i]).
                                    GetEndPoint(j).X + xt,
                                     ((Puzzle)a_array[i]).
                                    GetEndPoint(j).Y + yt)
                                    ;
98                          }
99
100                         /*//Skriver kooridater ud
101                         float xs = ((Puzzle)a_array[i]).
                                GetStartPoint(j).X*1.0F;
102                         float ys = ((Puzzle)a_array[i]).
                                GetStartPoint(j).Y*1.0F;
103                         string text = "("+ ((Puzzle)
                                a_array[i]).GetStartPoint(j).X
                                 + ","+ ((Puzzle)a_array[i]).
                                GetStartPoint(j).Y +")";
104                         e.Graphics.DrawString(text,
                                drawFont2, drawBrush, xs, ys);
105                         */
106                     }
107
108                     PointPuzzle tmpPoint = ((Puzzle)
                            a_array[i]).GetCenter();
109                     float x = tmpPoint.X * 1.0F + xt;
110                     float y = tmpPoint.Y * 1.0F + yt;
111                     float rad = (float)((Puzzle)a_array[i
                            ]).GetRadius();
112
113                     e.Graphics.DrawEllipse(Pens.Blue, x -
                            rad, y - rad, rad * 2, rad * 2);
114                     e.Graphics.FillEllipse(drawBrush, x -
                            1, y - 1, 1 * 2, 1 * 2);
115
116                     e.Graphics.DrawString("ID" + ((Puzzle)
                            a_array[i]).GetID(),
117                                                 drawFont,
                                                        drawBrush
                                                        , x, y
                                                        );
118
119                     ArrayList tmpCenterList = ((Puzzle)
                            a_array[i]).GetCenterList();
```

```
120
121                     for (int j = 0; j < tmpCenterList.
                            Count; j++)
122                     {
123                         tmpPoint = ((PuzzleItemsCenter)
                                tmpCenterList[j]).cen;
124                         x = tmpPoint.X * 1.0F + xt;
125                         y = tmpPoint.Y * 1.0F + yt;
126
127                         e.Graphics.DrawString("ID" + ((
                                PuzzleItemsCenter)
                                tmpCenterList[j]).nr,
128                                                     drawFont,
                                                        drawBrush
                                                        , x, y
                                                        );
129                     }
130                 }
131
132     }
133    }
134
135    public void ViewArray(ArrayList array)
136    {
137     a_array = array;
138     this.Refresh();
139    }
140
141   }
142  }
```