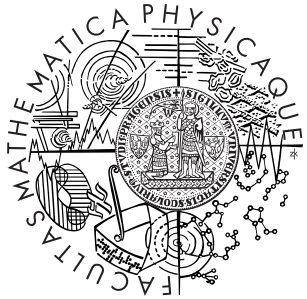
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Ondřej Dvořák

Generování rozmístění nábytku

Katedra aplikované matematiky

Vedoucí bakalářské práce: Mgr. Eva Jelínková
Studijní program: Informatika
Studijní obor: Programování (IP)

2008

Rád bych poděkoval Mgr. Evě Jelínkové za ochotu a cenné rady při zpracování této bakalářské práce.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne Ondřej Dvořák

Obsah

1 Úvod **6**

2 Existující rozvrhovací programy **8**
2.1 Volně dostupné produkty 8
2.2 Komerční produkty 9

3 Genetické algoritmy **10**
3.1 Úvod 10
3.2 Biologické pozadí 10
3.3 Základní tvar algoritmu 11
3.4 Genetické operátory 11

4 Algoritmus programu Spaceout **13**
4.1 Slovníček pojmů 13
4.2 Upřesnění problému 14
4.3 Organizační podrobnosti algoritmu 17
4.4 Rozbor algoritmu 21
4.4.1 Vstup a výstup algoritmu 21
4.4.2 Kostra algoritmu 21
4.4.3 Algoritmus prvotní generace 21
4.4.4 Algoritmus dalších generací 25

5 Uživatelská dokumentace programu Spaceout **32**
5.1 Instalace 32
5.1.1 Obecné pokyny a závislosti 32
5.1.2 Instalace programu 32
5.2 Popis uživatelského rozhraní 33
5.2.1 Pokoj 33
5.2.2 Objekty 34
5.2.3 Omezující podmínky 37
5.2.4 Projekt 40
5.2.5 Algoritmus 41
5.2.6 Nastavení programu 42

5.2.7 Scéna . . . 43
5.2.8 Výstup programu . . . 43

6 Programátorská dokumentace programu Spaceout **44**
6.1 Implementační informace . . . 44
6.2 Struktura programu . . . 44
6.2.1 Grafický modul . . . 45
6.2.2 Algoritmický modul . . . 45
6.2.3 Organizační modul . . . 45
6.2.4 Konfigurační modul . . . 47
6.2.5 GUI modul . . . 47

7 Závěr **49**
7.1 Problémy . . . 49
7.2 Další vývoj . . . 49

8 Přílohy **51**
8.1 Instalační CD . . . 51

Literatura **52**

Název práce: Generování rozmístění nábytku
Autor: Ondřej Dvořák
Katedra (ústav): Katedra aplikované matematiky
Vedoucí bakalářské práce: Mgr. Eva Jelínková, Katedra aplikované matematiky
e-mail vedoucího: eva@kam.mff.cuni.cz

Abstrakt: Cílem práce je vytvořit program, který na základě stanovených omezujících podmínek generuje různé návrhy rozestavení nábytku v místnosti. Tento program se jmenuje Spaceout a jeho základem jsou genetické algoritmy. Práce zároveň podává stručný přehled existujících programů pro vytváření návrhů interiérů a exteriérů. Její součástí je i programátorská a uživatelská dokumentace programu Spaceout.

Klíčová slova: Rozmísťování nábytku, Genetický algoritmus, Návrh pokoje

Title: Room Arrangment Generation
Author: Ondřej Dvořák
Department: Department of applied mathematics (KAM)
Supervisor: Mgr. Eva Jelínková, Department of applied mathematics (KAM)
Supervisor's e-mail address: eva@kam.mff.cuni.cz

Abstract: The aim of this thesis is to create a program which generates layouts of furniture in a room on the basis of specific constraints. This program is called Spaceout and it is based on genetic algorithms. The thesis also gives a basic overview of existing programs for creating projects of interiors and exteriors. The thesis contains a programmer and user documentation of Spaceout.

Keywords: Space layout, Genetic algorithm, Room arrangement

Kapitola 1

Úvod

V současné době je k dispozici celá řada komerčních, ale i volně dostupných programů určených k vytváření návrhů pokojů, bytů a někdy i celých domů. Tyto programy se zpravidla vyznačují velmi propracovanou grafikou a zároveň poměrně snadným uživatelským rozhraním, aby bylo možné v relativně krátkém čase vytvořit velmi působivý nákres. Důvodem vzniku práce je fakt, že žádný z těchto programů nemá vlastní inteligenci, která by generovala návrhy na základě stanovených požadavků a současně zohledňovala vkus uživatele, čímž by vytvářela návrhy pro uživatele jistým způsobem přínosné.

Program Spaceout, který je předmětem této práce, se ovšem od všech těchto produktů zásadně liší. Nesnaží se doplnit sbírku programů výše uvedeného typu, které jsou založeny na vizuálním dojmu, ale zaměřuje se na jejich nedostatky. Začleňuje do rozvrhovacího algoritmu inteligenci, která vytváří ve spolupráci s uživatelem návrhy splňující jak předem dané požadavky, tak jistý vyhraněný vkus uživatele. Z důvodu poměrně velké složitosti problému, který Spaceout řeší, není kladen příliš velký důraz na grafickou stránku programu, ale spíše na funkčnost genetického algoritmu, který je v programu implementován.

Podobným problémem se zabývali i autoři Kuroda a Yamagata v práci [2], kde představili i aplikaci, která tento problém řeší. Bohužel je text dostupný pouze v japonštině, a proto si můžeme význam obrázků v textu jen domýšlet. Práce částečně čerpá inspiraci z textu [1], ve kterém je popsán evoluční přístup k problému rozvrhování umístění kanceláří v několikapatrové budově. Vývoj algoritmu programu Spaceout byl také ovlivněn prací [3], která popisuje aplikaci genetických algoritmů na problém optimálního rozmístění dané množiny obrazců v rovině tak, aby oblast, do které obrazce zasahují, byla co nejmenší. Autoři použitím genetického algoritmu docílili velmi zajímavých výsledků. Bylo nalezeno řešení optimálního rozestavení 50 obrazců již ve 400. generaci. K rozhodnutí použít v programu Spaceout genetický algoritmus vedl především fakt, že genetické algoritmy hledají optimální řešení na základě kvalit jedinců. Tyto kvality si může v programu určovat sám uživatel, a tím může ovlivnit výsledek rozvrhování.

V úvodní kapitole rozeberu již existující programy určené na vytváření návrhů po-

kojů a bytů. V následujících kapitolách se budu věnovat programu Spaceout. Protože genetické algoritmy tvoří základ tohoto programu, bude druhá kapitola určena k představení genetických algoritmů. Kapitola následující pak přinese podrobnější popis algoritmu použitého v programu Spaceout, za který je připojena uživatelská a programátorská dokumentace k programu.

Kapitola 2

Existující rozvrhovací programy

Předtím, než se zaměřím na důkladný popis programu Spaceout, bych se rád věnoval právě typu programů popsaným výše.

2.1 Volně dostupné produkty

- **Google SketchUp**

  Google SketchUp je sharewarový program na vytváření 3D modelů. Program začal být vyvíjen firmou **@Last Software**, která nyní spadá pod společnost **Google**. Na rozdíl od jiných programů umožňuje v plné verzi uživateli svůj návrh exportovat do formátu CAD. Program disponuje velice snadným ovládáním, díky němuž zvládne vytvoření 3D modelu prakticky každý.

  Program je dostupný na webu [8].

- **Sweet Home 3D**

  Sweet Home 3D je volně šiřitelná aplikace sloužící pro návrh rozmístění nábytku a dalšího vybavení v interiérech. Aplikace disponuje rozsáhlou knihovnou nábytku, která je rozšiřitelná o další modely stažitelné z internetu nebo o modely vlastní. Navrhování interiéru se realizuje v 2D okně s možností zobrazení 3D náhledu.

  Program je dostupný na webu [9].

- **Furnish**

  Další volně dostupná aplikace je vyvíjená dánskou firmou **Bo Concept**. Firma se zabývá výrobou moderního nábytku a prostřednictvím programu Furnish prezentuje právě sortiment svých výrobků. Program umožňuje změnu detailů nábytku, a tím dává uživateli prostor využít všechny možnosti, které tento

ucelený sortiment nabízí. Výsledný návrh si lze prohlédnout v 3D podobě, a to ze všech možných úhlů. Jedním z větších nedostatků programu jsou poněkud vyšší hardwarové požadavky.

Program je dostupný na webu [10].

2.2 Komerční produkty

- **Room Arranger**

  Room Arranger je český komerčně vyvíjený program. Disponuje možností vytvářet návrhy několika místností celého domu, které mohou být umístěny hned v několika patrech. Součástí programu je rozsáhlá knihovna uživatelsky nastavitelných objektů. Zajímavou funkcí je možnost virtuálního průchodu pokoje pomocí prohlížeče VRML.

  Program je dostupný na webu [11].

V současné době není znám žádný program, který by generoval návrhy pokojů za pomoci některého rozvrhovacího algoritmu. Jak již bylo zmíněno v úvodu, pouze v Japonsku byla vyvinuta aplikace, která vytváří návrhy pomocí genetického algoritmu, ale ta je bohužel nedostupná. Je pouze popsána v článku [2], který ovšem existuje jen v japonské verzi.

Kapitola 3

Genetické algoritmy

3.1 Úvod

Genetické algoritmy se řadí mezi klasické představitele evolučních výpočetních postupů. Kořeny jejich vzniku sahají až do 60. let minulého století, kdy je představil americký vědec a profesor psychologie John Holland. V následující kapitole podám základní pohled na filosofii genetických algoritmů. Podrobnější informace je pak možné nalézt například v knize [4].

3.2 Biologické pozadí

Základní myšlenka genetických algoritmů je založena na Darwinově teorii o vývoji druhů. Stejně jako v Darwinově teorii, podle které v přírodě přežívají silnější jedinci a mají tak šanci předat svoji genetickou informaci do dalších generací, je i problém řešený pomocí genetického algoritmu převeden na problém objektů, které se snaží udržet ve specifikovaném prostředí co nejdéle. Kvalita objektu je zde vyjádřena fitness funkcí, jejíž hodnota dává objektu větší či menší šance na přežití do následující generace.

Jak je již z předchozího odstavce zřejmé, nepracují genetické algoritmy pouze s jedním kandidátem na řešení daného problému, nýbrž s celými množinami jedinců, které se označují jako populace. V přírodě se mezi sebou jedinci kříží a zároveň podléhají jistým genetickým mutacím. Tímto procesem pak postupně vznikají nové a nové populace jedinců. O vzniklé posloupnosti populací se pak často hovoří jako o generacích. V analogii s přírodním procesem jsou genetické algoritmy postaveny na vytváření generací pomocí selekce, operátorů křížení a mutace. Operátory křížení a mutace se řadí mezi takzvané rekombinační operátory.

V přírodě jsou vlastnosti jedinců kódovány do chromozomů, jejichž každý gen nese jistou genetickou informaci o konkrétní vlastnosti jedince. Analogicky i genetické algoritmy používají pro specifikování vlastnosti daného objektu strukturu řetězců podobnou chromozomu, jejíž každý člen označuje určitou vlastnost objektu.

Nad těmito chromozomy pak pracují právě zmíněné rekombinační operátory, kterým bude později věnováno několik odstavců.

3.3 Základní tvar algoritmu

Algoritmus běží v několika krocích. Na začátku je důležitý krok **inicializace**, v kterém se vygeneruje počáteční generace jedinců. Tento krok by měl být navržen tak, aby hledal prvotní generaci jistým inteligentním způsobem na základě znalostí problému, který má algoritmus řešit.

Po vytvoření prvotní generace následuje krok **vyhodnocení**, kdy se vzniklá prvotní populace ohodnotí a každému objektu populace se určí hodnota jeho fitness funkce.

Třetí krok algoritmu běží v cyklu, kdy se postupně provede **selekce** nad aktuální generací, **mutace** a **křížení** nad právě vybranou množinou objektů a následné vyhodnocení vzniklé populace.

3.4 Genetické operátory

- **Selekce**

  Cílem selekce je upřednostnění kvalitnějších jedinců před těmi méně kvalitními na základě fitness funkce, kterou jsou všichni jedinci ohodnoceni. Základ selekce je tvořen myšlenkou náhodného výběru, kdy je pravděpodobnost přežití jedince do další generace přímo úměrná jeho kvalitě. Při implementaci selekce se často používá takzvaný **mechanismus ruletového kola**, který je popsán v knize [4].

- **Mutace**

  Prvním z příkladů rekombinačních operátorů použitých v této práci je mutace. Tento operátor má za následek změnu určité vlastnosti jedince, na kterého je operátor použit. Při mutaci je postupně nad každým genem chromozomu mutovaného jedince proveden náhodný pokus s jistou, předem danou pravděpodobností úspěchu. V případě, že je pokus úspěšný, provede se mutace genu, tedy jistá změna hodnoty mutovaného genu. Tento operátor je aplikován na každého jedince mutované generace.

- **Křížení**

  Druhým příkladem rekombinačních operátorů je křížení. Výsledkem operátoru křížení je kombinace dvou jedinců, z nichž se vytvoří dva noví potomci. Nad každým jedincem celé populace se provede náhodný pokus s jistou,

předem danou pravděpodobností úspěchu. V případě, že je pokus úspěšný, je jedinec zařazen mezi kandidáty na křížení. V případě, že je velikost množiny kandidátů na křížení lichá, odstraníme jeden prvek a kandidáty rozdělíme do dvojic. Na každou takovou dvojici pak aplikujeme operátor křížení. Je možné použít například takzvané **jednobodové křížení**. Při jednobodovém křížení se může například zvolit dělící gen a všechny hodnoty ostatních genů ležící v chromozomu od tohoto dělícího genu vpravo se mezi sebou prohodí. Implementace křížení však výrazně závisí na konkrétní situaci, v níž je operátor použit.

Podrobnější popis implementace křížení je popsán v knize [4].

Kapitola 4

Algoritmus programu Spaceout

V úvodu této kapitoly představím slovníček pojmů, které budu používat nejen v celé této kapitole, ale i v programátorské dokumentaci. V další části kapitoly se zaměřím na detailnější popis algoritmu programu Spaceout, který doplním o jeho zápis v pseudokódu.

4.1 Slovníček pojmů

- **Objekt**

  Jeden kus nábytku v pokoji.

- **Gen**

  Struktura, která kóduje veškeré vlastnosti a pozici daného objektu.

- **Jedinec**

  Jedinec se skládá z objektů a představuje jeden aktuální návrh rozmístění pokoje. Každému objektu jedince přísluší právě jeden gen z chromozomu.

- **Chromozom**

  Řetězec genů. Každému jedinci přísluší právě jeden chromozom.

- **Generace**

  Množina jedinců ohodnocených fitness funkcí.

- **Křížení**

  Proces, při kterém dojde k vytvoření nových dvou jedinců ze dvou vybraných rodičovských jedinců.

- **Mutace**

  Proces, při kterém dojde k vytvoření nového jedince z jednoho jiného jedince změnou některých z genů chromozomu, který přísluší danému jedinci.

- **Selekce**

  Proces, při kterém je z populace na základě kvality jedinců náhodně vybrána nová populace shodné velikosti. Jeden jedinec může být přitom vybrán i víckrát a proto může být v nové populaci přítomen hned v několika exemplářích. Na tuto novou populaci se pak aplikují rekombinační operátory a populace se stane novou generací.

- **Kvalita jedince**

  Kvalita jedince označuje míru šance, že se jedinec dostane selekcí do následující generace. Tuto kvalitu určuje uživatel dle vlastního vkusu.

V analogii s evolučním vývojem vystupuje v programu celý pokoj jako živočišný jedinec vyvíjející se vlivem nahodilých změn, které zde představuje změna pozice některého kusu nábytku. Jeho vývoj je také ovlivněn selekcí, která daného jedince, na základě hodnoty jeho fitness funkce, před ostatními upřednostňuje, nebo naopak znevýhodňuje. Hodnotu fitness funkce zadává uživatel, který ji určí na základě toho, jak se mu daný návrh pokoje líbí.

4.2 Upřesnění problému

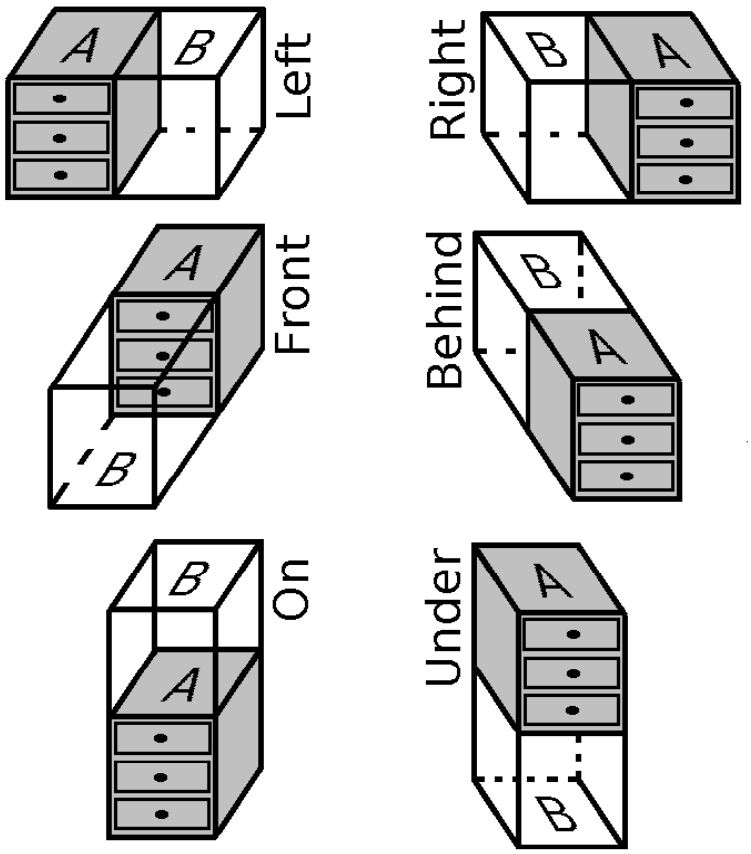
Nyní upřesním problém, který má vůbec celý algoritmus řešit.

Vstupem algoritmu jsou objekty, kterým odpovídají kusy nábytku v pokoji, a omezující podmínky mezi objekty, které představují vztahy mezi kusy nábytku. Vztah mezi dvěma objekty určuje postavení, rotaci a zarovnání jednoho objektu vůči druhému objektu. Mezi dvěma objekty existuje buď právě jedna dvojice navzájem inverzních vztahů, nebo mezi objekty není žádný vztah. Podrobnější popis vztahů a jejich vlastností je uveden níže. Problémem, který má algoritmus řešit, je způsob, jak navrhnout rozmístění nábytku v místnosti tak, aby žádná požadovaná omezení kladená na rozmístění nábytku nebyla porušena.

V genetické terminologii řeší algoritmus problém jak nastavit jednotlivé geny chromozomu jedince, aby požadované vztahy mezi žádnými dvěma geny nebyly porušeny.

Pro další popis definuji **rozvrhovací graf** jako orientovaný graf, kde každému vrcholu přísluší právě jeden objekt z daného pokoje a každá orientovaná hrana označuje vztah mezi dvěma objekty. Z předpokladu, že mezi dvěma objekty existuje nejvýše jedna dvojice vztahů, vyplývá, že v rozvrhovacím grafu mezi dvěma vrcholy existuje buď jedna dvojice hran s opačnou orientací, nebo žádná hrana.

Algoritmus tedy řeší problém nalezení takové pozice objektů v místnosti, aby pro každý objekt platilo, že žádný objekt, který je s ním spojen hranou, neporušuje vztah touto hranou definovaný.

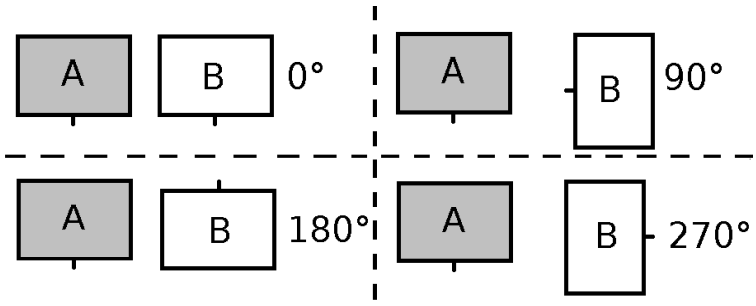


Obrázek 4.1: Vztahový typ

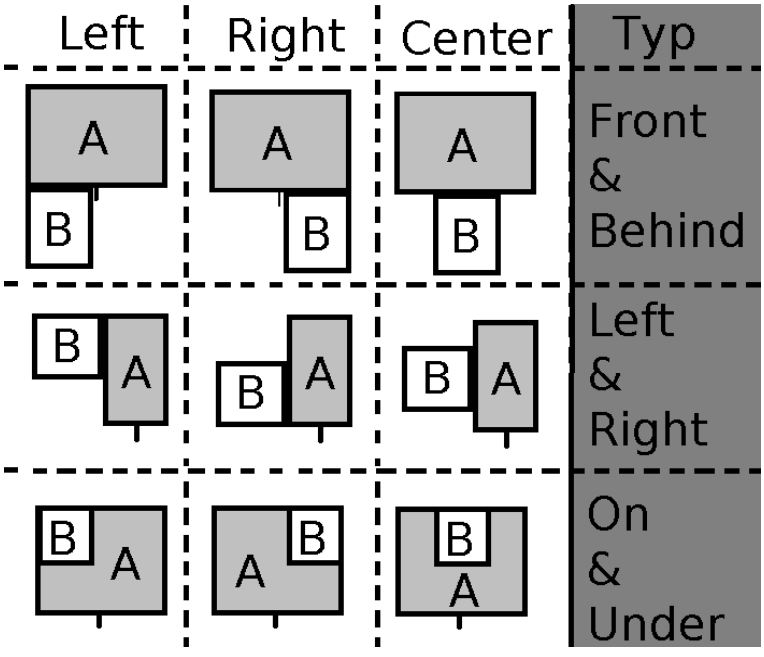
Vztahem rozumíme v programu trojici **(typ, rotace, zarovnání)**, kde **typ** je prvkem množiny **{left, right, front, behind, on, under}**, **rotace** je prvkem množiny **{0, 90, 180, 270}** a **zarovnání** je prvkem množiny **{left, right, center}**. Nechť **A** a **B** jsou objekty, mezi kterými existuje v rozvrhovacím grafu orientovaná hrana z **A** do **B** označující vztah **V**. Pak **typ** ve vztahu **V** určuje pozici objektu **B** vůči objektu **A**. Význam možných hodnot **typu** je zakreslen v obrázku 4.1. **Rotace** určuje natočení objektu **B** vůči objektu **A**. Význam možných hodnot **rotace** je zakreslen v obrázku 4.2. **Zarovnání** určuje zarovnání objektu **B** vůči objektu **A**. Význam jednotlivých hodnot při daném zarovnání je zakreslen v obrázku 4.3.

Inverzní vztah B ke vztahu **A** se určí následovně:

Typ inverzního vztahu **B** se určí na základě **rotace** a **typu** vztahu **A** pomocí této převodní tabulky:



Obrázek 4.2: Vztahová rotace



Obrázek 4.3: Vztahové zarovnání

	0	**90**	**180**	**270**
Front	Behind	Right	Front	Left
Left	Right	Front	Left	Behind
Behind	Front	Left	Behind	Right
Right	Left	Behind	Right	Front
On	Under	Under	Under	Under
Under	On	On	On	On

Rotace inverzního vztahu **B** se určí na základě rotace vztahu **A** podle následujícího předpisu:

rotace inverzního vztahu B = (360 - (**rotace vztahu A**))

Zarovnání inverzního vztahu **B** se určí na základě **typu** a **rotace** vztahu **A**. Pro další popis zavedu substituční tabulku:

vztah A	**vztah B**
Left	Right
Center	Center
Right	Left

Je-li **typ** vztahu prvkem množiny **{right, left}** a **rotace** je prvkem z množiny **{180, 270}**, pak vznikne hodnota **zarovnání** inverzního vztahu **B** dle uvedené substituční tabulky.

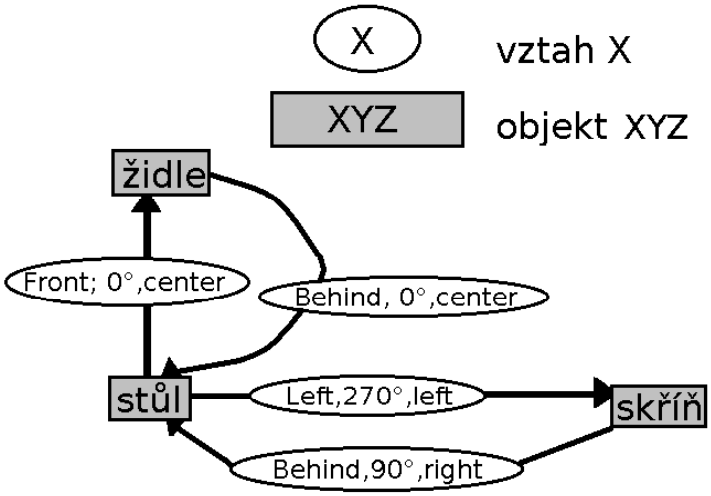
Je-li **typ** vztahu prvkem množiny **{right, left}** a **rotace** je prvkem z množiny **{0, 90}**, pak je hodnota **zarovnání** inverzního vztahu **B** stejná jako hodnota vztahu **A**

Je-li **typ** vztahu prvkem množiny **{front, behind}** a **rotace** je prvkem z množiny **{90, 180}**, pak vznikne hodnota **zarovnání** inverzního vztahu **B** dle uvedené substituční tabulky.

Je-li **typ** vztahu prvkem množiny **{front, behind}** a **rotace** je prvkem z množiny **{180, 270}**, pak je hodnota **zarovnání** inverzního vztahu **B** shodná s hodnotou vztahu **A**.

4.3 Organizační podrobnosti algoritmu

Prvotním objektem budu v následujícím textu rozumět objekt, pro který algoritmus aktuálně hledá vyhovující pozici.



Obrázek 4.4: Rozvrhovací graf

Příbuznými objekty objektu A budu chápat takové objekty, do jejichž vrcholu v rozvrhovacím grafu vede cesta z vrcholu odpovídajícímu objektu A. Protože z definice rozvrhovacího grafu jsou všechny hrany symetrické, vede z těchto objektů také cesta zpět do A. Příbuzné objekty objektu A jsou tedy všechny objekty, příslušející vrcholům komponenty souvislosti, ve které leží vrchol objektu A.

Fixovaným objektem se rozumí objekt, jehož poloha se v průběhu rozvrhování již nebude měnit. Fixované objekty mají v každém návrhu stejnou pozici. Implementačně to znamená, že genu, který přísluší fixovanému objektu, se v chromozomu nastaví položka **fix** na **true**.

Viditelné versus neviditelné objekty

Rozvrhovací algoritmus pracuje zejména s objekty, které odpovídají jednotlivým kusům nábytku. Označme je jako viditelné objekty. Kromě nich pracuje algoritmus ještě s neviditelnými objekty. Na rozdíl od viditelných objektů, kterým v programu odpovídá určitý kus nábytku, neviditelné objekty fyzicky neexistují a v programu vystupují pouze jejich abstrakce, takzvané InvisibleGens. Tato pomocná struktura slouží k tomu, aby mohl uživatel určit, kde nesmí v pokoji stát žádný objekt. Tato oblast může být určena buď absolutně v rámci celého pokoje, nebo relativně vůči specifickému objektu. Tvar neviditelné oblasti je vždy kvádr. Informace o požadované pozici a velikosti abstraktního objektu je uložena přímo v genu. Při hledání vhodné pozice viditelného objektu v pokoji se tento objekt společně s jeho příbuznými objekty mimo testu na kolize s ostatními viditelnými objekty testuje i na kolize s neviditelnými objekty, a tím se zamezí možnému umístění objektu do oblasti, kde je to nežádoucí.

Iniciální chromozom

Iniciální chromozom nad množinou **X** je prázdný chromozom, který má velikost shodnou s počtem viditelných objektů v množině **X**. Tato množina je složena ze všech objektů v pokoji. Žádný z genů iniciálního chromozomu není fixován, a tedy není určena pozice žádného objektu v pokoji.

Utilita IntelligentLayout

Utilita IntelligentLayout na přání uživatele zaručí, že všechny vygenerované návrhy budou splňovat jisté užitečné vlastnosti. Tyto vlastnosti jsou následující:

1. Bude možné plnohodnotně otevřít dveře skříně tak, aby nedošlo ke kolizi s jiným objektem.
2. Bude možné plnohodnotně vysunout šuplíky komody tak, aby nedošlo ke kolizi s jiným objektem.
3. Žádný z objektů nebude umístěn na stole.

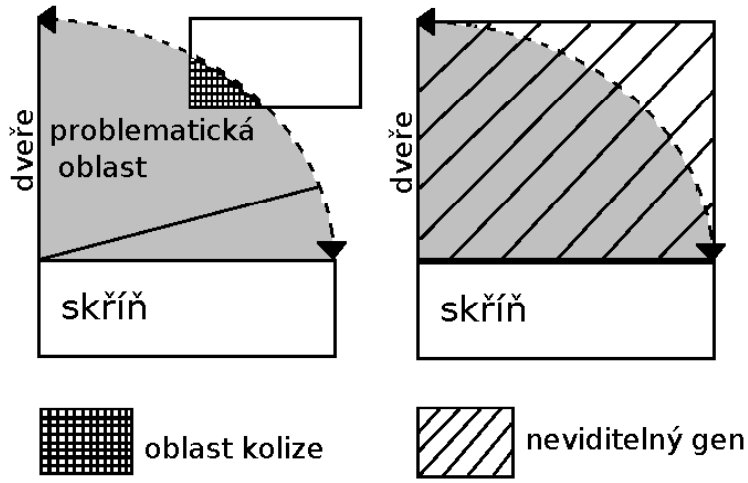
Pro další popis definuji **problematickou oblast** jako kvádrovou oblast v okolí objektu, do které zasahuje objekt jistou svojí funkční složkou. Tedy v případě skříně, respektive komody, je to minimální kvádrová oblast před objektem, do které zasahují dveře, respektive šuplíky, v průběhu otevírání. V případě stolu je to kvádrová oblast nad stolem, která má stejné rozměry jako stůl a výšku danou konstantou. **Problematickým objektem** budu rozumět objekt, který svoji funkčností může vyvolat určitou kolizi v problematické oblasti (viz. obrázek 4.5). To znamená, že může existovat návrh, v němž nebude tento objekt splňovat některou z vlastností uvedených výše.

Tato funkce je implementována tak, že projde všechny geny iniciálního chromozomu, a jestliže objekt (označme ho A), který danému genu přísluší, patří mezi **problematické** objekty, je do problematické oblasti vložen neviditelný objekt, jehož velikost je shodná s velikostí problematické oblasti. Nachází-li se nově vložený objekt před objektem A, je s ním objekt A spojen podmínkou typu **MUST(Front, 0,Left)**. Nachází-li se nově vložený objekt na objektu A, je s ním objekt A spojen podmínkou **MUST(On, 0, Left)**. Pro první případ je situace ilustrována na obrázku 4.5.

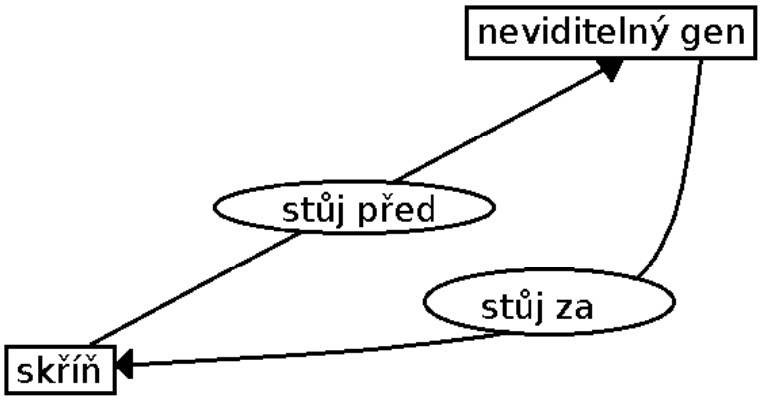
Grafově je tato situace zachycena na obrázku 4.6. Utilita IntelligentLayout je volána hned v úvodu rozvrhovacího algoritmu. Po skončení algoritmu jsou všechny objekty přidané touto utilitou smazány.

Sekvence

Sekvence označuje libovolnou permutaci identifikačních čísel všech objektů, které nejsou fixovány.



Obrázek 4.5: Problematická situace a její řešení



Obrázek 4.6: Grafová interpretace řešení problematické situace

Sekvence plní v algoritmu funkci náhodného elementu, na jehož základě vznikají v prvotní generaci různé návrhy. Určuje pořadí, ve kterém budou objekty do pokoje vkládány.

4.4 Rozbor algoritmu

4.4.1 Vstup a výstup algoritmu

Vstupem algoritmu je množina objektů pokoje, množina vztahů mezi nimi a velikost generace. Výstupem algoritmu je množina jedinců splňujících zadané vstupní podmínky. Velikost množiny je shodná s velikostí generace dané na vstupu.

4.4.2 Kostra algoritmu

Celý algoritmus nejprve vygeneruje prvotní generaci pomocí takzvaného **algoritmu prvotní generace**. Vyžádá-li si uživatel vygenerování nové sady návrhů, je na poslední vygenerovanou generaci aplikován **algoritmus dalších generací**. V následujících dvou sekcích je uveden popis každé ze dvou částí algoritmu, který je doplněn příslušným pseudokódem.

4.4.3 Algoritmus prvotní generace

Vytvoření iniciálního chromozomu

Nejprve se vytvoří prvotní chromozom nad vstupní množinou. Velikost počátečního chromozomu označme jako **IndividuumSize**.

Přidání neviditelných objektů utilitou IntelligentLayout

Pokud je zapnutá utilita **IntelligentLayout**, připojí se ke každému objektu v iniciálním chromozomu užitečný neviditelný objekt.

Fixování objektů

Úkolem tohoto kroku algoritmu je fixování takových objektů, které mají na vstupu specifikovány všechny souřadnice, a tudíž je jejich poloha přesně určena. Všechny geny příslušející fixovaným objektům budou mít po tomto kroku již přesně stanovenou pozici.

Fixování příbuzných objektů

V tomto kroku se pomocí prohledávání do šířky fixují pozice všech objektů, které jsou ve stejné komponentě souvislosti jako některý již fixovaný objekt.

Tvorba kopie chromozomu

Fixování objektů a jim příbuzných objektů pracovalo výhradně s jednou instancí chromozomu, protože pozice fixovaných objektů a objektů jim příbuzných je přesně určena, a musí být tedy pro všechny návrhy naprosto stejná.

V tomto okamžiku je již vytvořený chromozom několikrát zkopírován, čímž je vytvořena množina chromozomů označovaná jako **generace** o velikosti dané na vstupu (označme ji **GenerationSize**). Tato generace je ovšem do jisté míry invalidní, neboť každý její jedinec je stejný a navíc není kompletně vytvořený. Nefixované objekty stále nemají určenou pozici.

Rozvrhování nefixovaných objektů

Pro každý chromozom se vytvoří jedna sekvence identifikačních čísel viditelných objektů a aplikuje se rozvrhovací strategie uvedená níže.

V případě, že je rozvrhovací strategie úspěšná, tedy se povede nalézt přípustnou pozici pro každý dosud nerozvržený objekt, je návrh dokončen a jemu příslušný jedinec je zařazen do aktuální generace. Je-li strategie neúspěšná, vygeneruje se pro daný chromozom jiná sekvence, na kterou se aplikuje rozvrhovací strategie. Pokud ještě není celá generace hotová a rozvrhování bylo přerušeno uživatelsky nastavitelným timeoutem nebo abortem, doplní se generace již vytvořenými jedinci na požadovanou velikost. V případě, že došlo k přerušení a dosud nebyl vygenerován žádný návrh, končí algoritmus neúspěšně.

Rozvrhovací strategie v úvodní generaci

Vstupem je sekvence identifikačních čísel objektů, která se prochází a hledá se vhodná pozice pro umístění objektu s příslušným id do místnosti. V okamžiku, kdy se podaří postavit objekt (označme ho **prvotní objekt**) na takovou pozici, že s ním nekoliduje žádný jiný již umístěný objekt, přichází na řadu rozestavení všech jeho příbuzných objektů. V rámci procesu rozestavování příbuzných objektů může docházet k dalším kolizím a proto bude potřeba pozici prvotního objektu ještě několikrát změnit. Algoritmus tedy musí nalézt takovou pozici prvotního objektu, aby tento objekt společně se všemi jeho příbuznými objekty nebyl v kolizi s nějakým již umístěným objektem.

Při hledání přípustné pozice se postupuje dle následujících strategií:

- **Strategie A (Strategie LayoutInLine)**

  Pokud má objekt specifikovány dvě souřadnice, posouvej objekt (označme ho jako prvotní objekt) ve směru vektoru třetí souřadnice a testuj ho společně s jeho příbuznými objekty na kolize. Dostane-li se do kolize prvotní objekt, je posunut v daném směru těsně za objekt, se kterým je v kolizi. Dostane-li se do kolize nějaký příbuzný objekt, je prvotní objekt posunut o 1. V případě, že se

nepovede objekt v daném směru umístit, nelze použít žádnou z rozvrhovacích strategií a pro danou vstupní sekvenci program nedokáže vytvořit návrh.

V případě, že má objekt specifikovanou právě jednu souřadnici, a to **x** (respektive **y**), je na něj uplatněna strategie LayoutInLine ve směru souřadnice **y** (respektive **x**). Končí-li strategie neúspěšně, přejde se na strategii A2 (LayoutUpInLine). Situace, kdy bude specifikována pouze souřadnice **z** se řeší pomocí strategie LayoutAround (popsána později) s tím, že objekt má pevně danou souřadnici **z**.

- **Strategie A2 (Strategie LayoutUpInLine)**

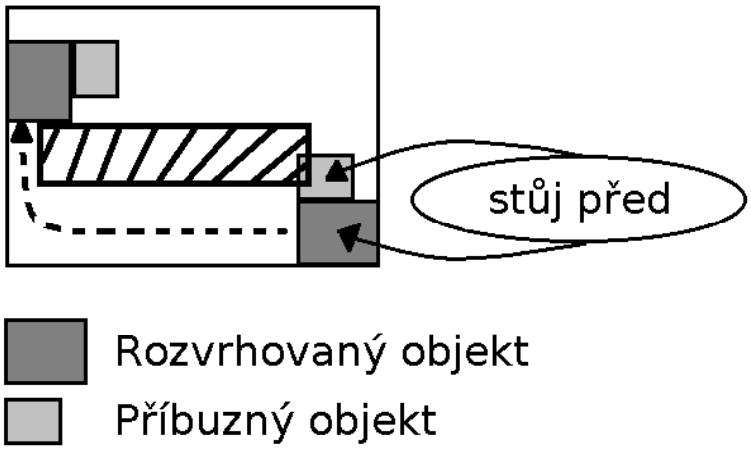
  Strategie se uplatňuje v okamžiku, kdy má objekt (označme ho jako prvotní objekt) specifikovanou souřadnici **x** nebo **y**, a souřadnice **z** není specifikována, tudíž má objekt volnost pohybu v rámci z-ové osy. Je-li specifikována souřadnice **x** (respektive **y**), je sestaven seznam všech dosud umístěných objektů ležících v pozici **x** (respektive **y**). Prvotní objekt se, dle postupu popsaného níže, postupně staví na náhodně vybrané objekty z vytvořeného seznamu. V případě, že prvotní objekt, ani žádný jeho příbuzný, nekoliduje s již umístěnými objekty, končí algoritmus s úspěšně nalezenou pozicí prvotního objektu. Nelze-li prvotní objekt postavit na žádný z objektů v seznamu, končí strategie neúspěšně.

  1. Uvažme objekty A a B. Objekt A chceme postavit na objekt B. Nechť má A fixovanou x-ovou souřadnici (pro fixovanou souřadnici y se problém řeší analogicky).
  2. Má-li A podstavu větší, než je podstava objektu B, nelze A postavit na B.
  3. Postav objekt A na pozici stejnou, jako je pozice objektu B, a posuň ho těsně nad B.
  4. Cyklicky opakuj následující body.
  5. Koliduje-li A s jiným objektem, posuň A ve směru **y** těsně za kolidující objekt. Koliduje-li některý z příbuzných objektu A, posuň A o 1.
  6. Ověř, zda A nepřesahuje přes objekt B. V případě, že A objekt B přesáhne, nelze A postavit na B.
  7. Nenastala-li kolize a A nepřesahuje přes B, je nalezena nová pozice A.

- **Strategie B (Strategie LayoutAround)**

  1. Náhodně vyber jeden z rohů místnosti a umísti do něj objekt.
  2. Po směru hodinových ručiček posouvej objekt podél stěn a testuj, zda na dané pozici objekt ani žádný z jeho příbuzných objektů není v kolizi s ostatními již umístěnými objekty.
  3. Pokud se dostaneš do výchozího místa, odkud jsi začal hledat potenciální pozici, **Strategie B** selhala, přejdi na **Strategii C**.

  Strategie B je zachycena na obrázku 4.7.



Obrázek 4.7: Strategie B

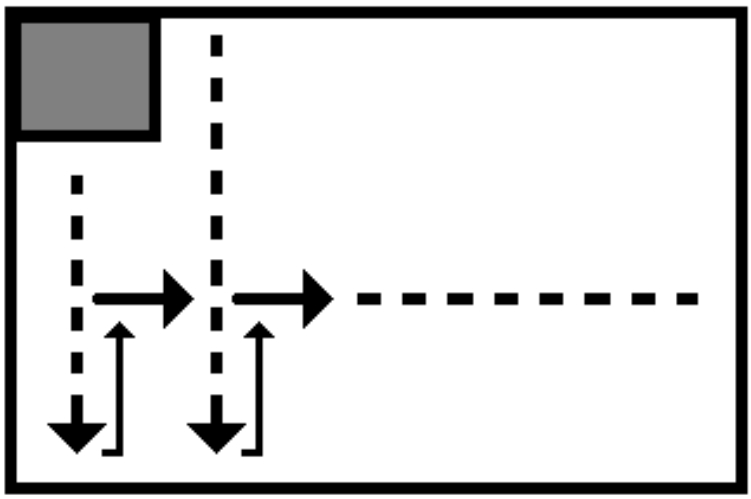
- **Strategie C**

  1. Vytvoř náhodnou sekvenci již umístěných objektů.
  2. Na každý z objektů v sekvenci zkus umístit rozvrhovaný objekt a zkontroluj, že žádný z jeho příbuzných objektů nekoliduje s jiným již rozvrženým objektem a zároveň se rozvrhovaný objekt nenachází ve vzduchu. Umísťování se provádí podobným způsobem jako je popsáno ve strategii A2.
  3. Pokud se nepodaří objekt postavit korektně na jiný objekt, **strategie C** selhala, přejdi na **strategii D**.

- **Strategie D**

  1. Postav objekt do jednoho rohu místnosti.

2. Posouvej objekt v místnosti po dráze naznačené na obrázku 4.8 a hledej přípustnou pozici rozvrhovaného objektu. To znamená takovou pozici, v níž objekt ani žádný z jeho příbuzných objektů nekoliduje s žádným jiným již rozvrženým objektem.
3. Pokud se uvedenou strategií nepodaří najít vhodnou pozici pro objekt, nelze vytvořit pomocí uvedených strategií návrh.



Obrázek 4.8: Strategie D

V tuto chvíli algoritmus končí buď úspěšně vygenerovanou kompletní generací, nebo neúspěchem. Kompletní pseudokód algoritmu je k nalezení níže.

4.4.4 Algoritmus dalších generací

Úkolem druhé části algoritmu je vygenerovat další generace na základě již existující generace předchozí.

Vstup

- **generation** - Prvotní generace vygenerovaná pomocí předchozího algoritmu.
- **mutation_prbl** - Pravděpodobnost mutace.
- **cross_prbl** - Pravděpodobnost křížení.

Základní tvar algoritmu dalších generací je založen na několika krocích, které jsou uvedeny níže.

1. Nad generací **generation** se provede selekce. Selekcí vznikne nová generace **selected_generation**.

Algorithm 1 Algoritmus programu Spaceout

1: Vytvoř nový chromozom $InitChromozome$ o velikosti $IndividuumSize$
2: **if** je zapnutá utilita $IntelligentLayout$ **then**
3: přidej do $InitChromozome$ inteligentní objekty
4: **end if**
5: Zafixuj geny, které mají jednoznačně určenou pozici
6: Zafixuj všechny příbuzné geny dosud fixovaných genů
7: Vytvoř novou generaci $Generation$ o velikosti $GenerationSize$
8: Do každého prvku generace nakopíruj $InitChromozome$
9: $Pos = 0$
10: **repeat**
11: Vytvoř kopii chromozomu jedince, který je na pozici pos v $Generation$ a přiřaď ho do $WorkChrom$
12: Vytvoř novou sekvenci $Sekvence$, která má velikost shodnou s počtem dosud neumístěných objektů v chromozomu $WorkChrom$
13: $WorkChrom$:= LayoutInLine($WorkChrom$, $Sekvence$)
14: $WorkChrom$:= LayoutAround($WorkChrom$, $Sekvence$)
15: **if** rozvrhování bylo úspěšné **then**
16: $Generation[pos] := WorkChrom$
17: $Pos++$
18: **else if** nastal timeout nebo abort **then**
19: break
20: **end if**
21: **until** ($Pos = Generation.size()$)
22: **if** $Pos = 0$ **then**
23: Není možné najít řešení; RETURN error
24: **else if** $Pos < Generation.size()$ **then**
25: Doplň generaci již vytvořenými návrhy
26: **end if**

2. Nad generací **selected_generation** se provede s pravděpodobností **mutation_prbl** mutace a vytvoří se **mutated_generation**.

3. Nad generací **mutated_generation** se provede s pravděpodobností **cross_prbl** křížení a vznikne **crossed_generation**.

4. **generation = crossed_generation**.

5. Všem jedincům v **generation** nastav stejnou hodnotu fitness funkce.

Selekce

Selekce se provede pomocí techniky ruletového kola, která je popsána níže.

1. Uvažme generaci velikosti 4, jejíž jedinci mají hodnoty fitness funkce po řadě 4, 3, 5, 8 jako v následující tabulce. **Id** označuje identifikační číslo jedince a **fitness** hodnotu jeho fitness funkce.

2. Uspořádáme identifikační čísla uvedených jedinců sestupně dle hodnot jejich fitness funkcí, tedy 8, 5, 4, 3.

3. Sečteme hodnoty fitness funkcí všech jedinců v generaci, tedy **sum = 20**.

4. V analogii s termínem **Ruletové kolo** musíme každému jedinci přiřadit určitou výseč ruletového kola, která odpovídá hodnotě jeho fitness funkce.

5. Pravděpodobnosti výběru jednotlivých jedinců jsou uvedeny v následující tabulce:

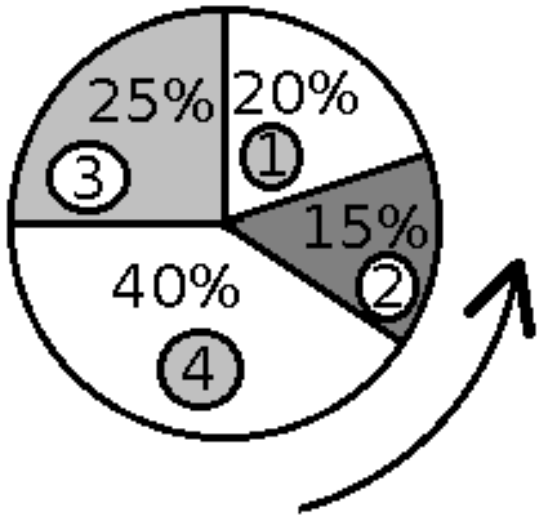
id	1	2	3	4	**sum**
fitness	4	3	5	8	20
probability	0.2	0.15	0.25	0.4	1
probability*	20%	15%	25%	40%	100%

   probability = fitness / sum
   probability* = probability * 100

6. Uvažme, že ruletové kolo nabývá hodnot 0 až 100. Výseče příslušející k jednotlivým jedincům určíme pomocí intervalů. Jednotlivé meze intervalů spočteme tak, že seřadíme jedince podle jejich pravděpodobností sestupně a každému jedinci přiřadíme součet všech pravděpodobností objektů ležících od něj v posloupnosti nalevo. Intervaly jsou tedy následující:

id	interval
4	0-39
3	40-64
1	65-84
2	85-99

7. Vygeneruje se náhodné číslo z intervalu 0-99. Pro toto číslo se určí odpovídající interval a následně je jedinec s **id** příslušejícím tomuto intervalu vybrán do další generace. Tomuto procesu odpovídá hození kuličky do ruletového kola, která se zastaví ve výseči příslušející danému jedinci, jak je naznačeno na obrázku 4.9.



Obrázek 4.9: Technika ruletového kola

Podrobnější popis mechanismu ruletového kola je možné najít v knize [4].

Mutace

Každý gen každého chromozomu se s pravděpodobností **mutation_prbl** vybere pro mutaci, která je provedena následujícím způsobem:

1. Je nalezena komponenta souvislosti, do které gen (označme ho T) přísluší v rozvrhovacím grafu.

2. Jestliže je některý z objektů v nalezené komponentě fixován, proces mutace je ukončen. V opačném případě se pokračuje dle následujících kroků.

3. Pro každý objekt A z komponenty je nalezen seznam všech objektů, které zasahují do sloupcové oblasti X nad objektem A nebo jsou příbuznými některého takového objektu. Důvodem je fakt, že by se po mutaci tyto objekty mohly vznášet ve vzduchu. Situace je naznačena na obrázku 4.10, kde do oblasti X zasahují objekty B,C,F a G.

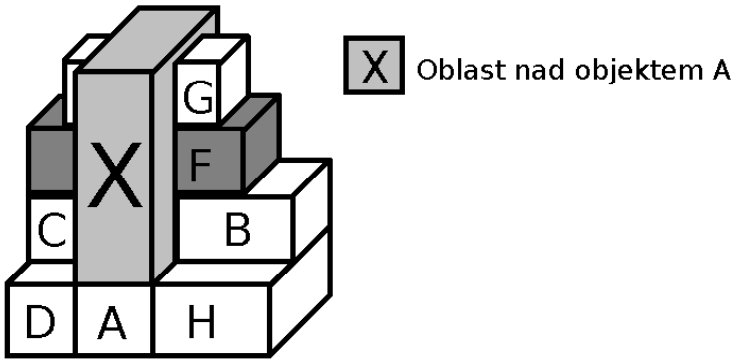
4. Všechny geny ležící v této komponentě, společně se všemi objekty nalezenými v předchozím kroku, jsou označeny jako nerozvržené.

5. Je nalezena nová náhodná pozice P vybraného objektu v rámci pokoje.

6. V případě, že objekt nebo některý z jeho příbuzných na nalezené pozici P koliduje s již rozvrženým objektem, je hledána jiná přijatelná pozice pomocí algoritmu pro generování prvotní generace. Tento algoritmus začíná hledat vhodnou pozici právě od pozice P.

7. V okamžiku, kdy je nalezena vhodná pozice pro komponentu, ve které leží objekt T, je rozvržen zbytek neumístěných objektů pomocí algoritmu prvotní generace.

8. Poté, co jsou nalezeny vyhovující pozice všech objektů, mutace končí.

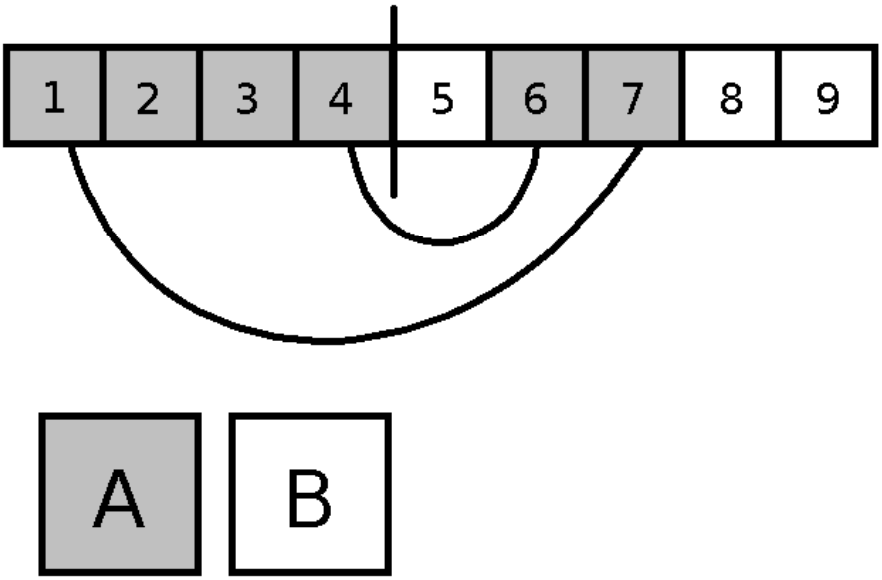


Obrázek 4.10: Objekty, které zasahují do sloupcové oblasti X nad objektem A

Křížení

Každý chromozom generace je s pravděpodobností **cross_prbl** zařazen mezi kandidáty na křížení. Vznikne-li lichá množina kandidátů na křížení, je poslední testovaný chromozom odebrán. Kandidáti na křížení jsou spárováni a nad každým párem je provedeno křížení následujícím způsobem:

1. Náhodně se zvolí počet křížených genů (označme ho **N**).



Obrázek 4.11: Určení množin křížených genů

2. Náhodně se vybere množina **N** křížených genů (označme ji **M**).

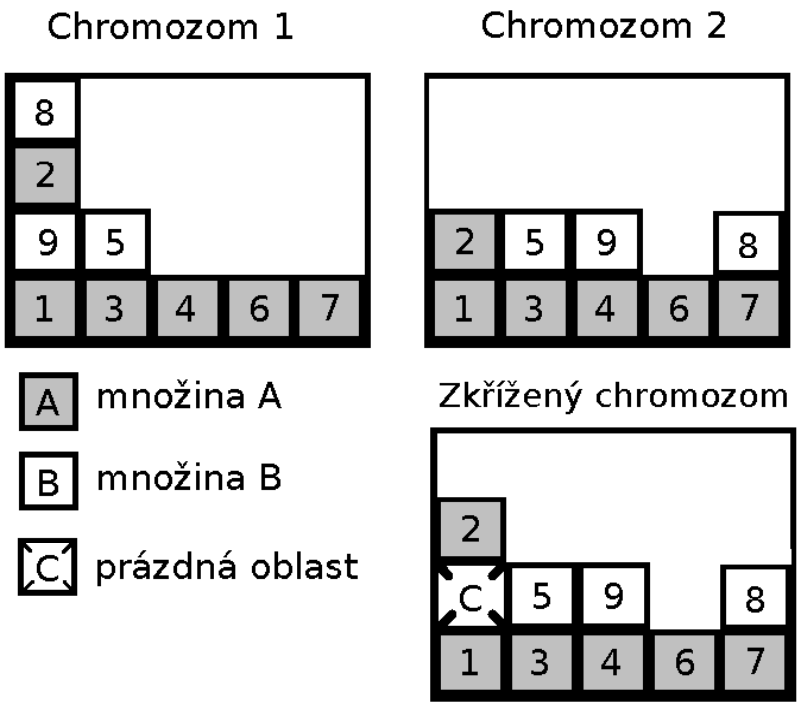
3. Pro každý gen nalézající se v množině **M** je určena množina genů, které jsou s ním ve stejné komponentě souvislosti. Provede se sjednocení těchto množin a množiny **M**, které je označeno jako **A**. Množina všech genů, které nejsou v množině A, je označena jako **B**. Rozdělení do množin je naznačeno na obrázku 4.11.

4. Je vytvořena množina objektů (označme ji S), které leží pod některým z objektů z množiny A (podobně jako na obrázku 4.10). Tato množina je rozšířena o všechny příbuzné objekty objektů z množiny S.

5. Je-li některý z objektů S obsažen v množině B, je z této množiny přesunut do množiny A. Důvodem je fakt, že by po křížení mohly vzniknout objekty vznášející se ve vzduchu, jak je naznačeno na obrázku 4.12.

6. Jsou vytvořeny dva nové chromozomy (označme je 1 a 2). Chromozom 1 vznikne z kopie prvního kandidáta na křížení tak, že všechny jeho geny z množiny B jsou nastaveny stejně jako geny druhého kandidáta na křížení. Dojde-li ke kolizi s některým z genů z množiny A chromozomu 1, jsou kolidující objekty rozvrhovány pomocí algoritmu generování prvotní generace s tím, že se vhodná pozice začíná hledat právě od místa, kde došlo ke kolizi. Chromozom 2 vznikne analogicky z kopie druhého kandidáta tak, že nastaví všechny geny množiny A, která je zmenšena o objekty ležící pod objekty z množiny B, stejně jako geny prvního kandidáta na křížení.

7. Na konci křížícího procesu vznikají dva nové chromozomy.



Obrázek 4.12: Problematická situace při neodebrání objektů při křížení. Pohled z boku.

Kapitola 5

Uživatelská dokumentace programu Spaceout

5.1 Instalace

5.1.1 Obecné pokyny a závislosti

Program je primárně navržen a testován v operačním systému Linux. Pro správný chod programu je potřeba mít nainstalované **OpenGL** a **Qt** verze **4.2.x** a vyšší.

Podrobnosti instalace **OpenGL** jsou dostupné na webu [5].

Pokyny k instalaci toolkitu Qt jsou dostupné na webu [6]. Při instalaci **Qt** by měl konfigurační skript automaticky detekovat, zda je již **OpenGL** nainstalováno a sám přidat **OpenGL** modul do knihovny **Qt**. Podrobnější informace jsou dostupné na webu [7].

5.1.2 Instalace programu

Kompilace a spuštění programu

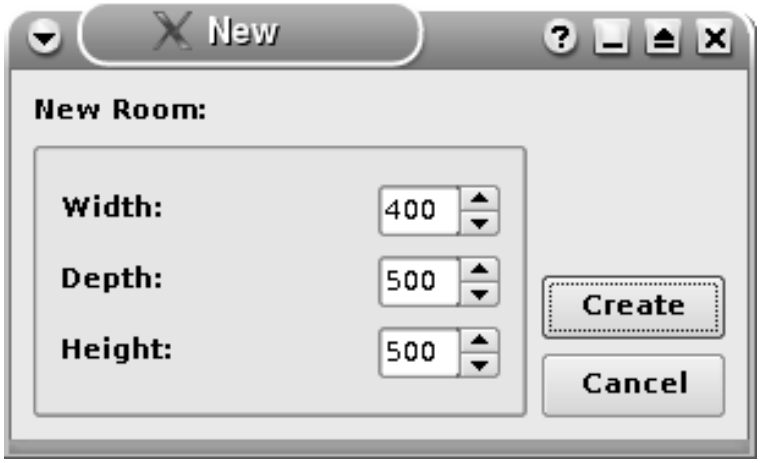
1. Na vámi zvolené místo rozbalte balíček **spaceout.rar**, který je k dispozici na přiloženém CD.

2. Zapněte terminál a přesuňte se do adresáře, kam jste rozbalili balíček **spaceout.rar**.

3. Zadejte příkaz `qmake -project` (případně `qmake-qt4 -project` jestliže máte nainstalováno více verzí Qt).

4. Zadejte příkaz `qmake` (případně `qmake-qt4`).

5. Do vytvořeného souboru **spaceout.pro** přidejte řádek **QT += opengl**.



Obrázek 5.1: Vytvoření pokoje

6. Do vytvořeného souboru **Makefile** přidejte do INCPATH cestu k nainstalované knihovně QtOpenGL. (například **-I/usr/include/qt4/QtOpenGL**).
7. Do vytvořeného souboru **Makefile** přidejte do LIBS **-lQtOpenGL**
8. Zadejte příkaz `make`.
9. Pokud vše proběhlo úspěšně, bude v aktuálním adresáři nový spustitelný soubor **spaceout**, který lze spustit příkazem `./spaceout`.

5.2 Popis uživatelského rozhraní

5.2.1 Pokoj

Vytvoření nového pokoje

1. V hlavní nabídce zvolte menu **File**.
2. Přejděte na položku **New**.
3. Zobrazí se dialog pro zadání rozměrů vámi požadovaného pokoje jako na obrázku **5.1**.
4. Zadejte požadované rozměry pokoje a stiskněte tlačítko **Create**.

Úprava rozměrů aktuálního pokoje

1. V hlavní nabídce zvolte menu **Room**.
2. Přejděte na položku **Set**.

3. Zobrazí se editační dialog podobný dialogu na obrázku **5.1**.
4. Upravte požadované rozměry pokoje a stiskněte tlačítko **OK**.

5.2.2 Objekty

Implementované objekty

- kuchyňský stůl,
- židle,
- skříň,
- komoda,
- pracovní stůl,
- knihovna,
- dveře,
- kvádr,
- kužel,
- prázdný objekt - určuje místo, kde se nesmí nic vyskytovat.

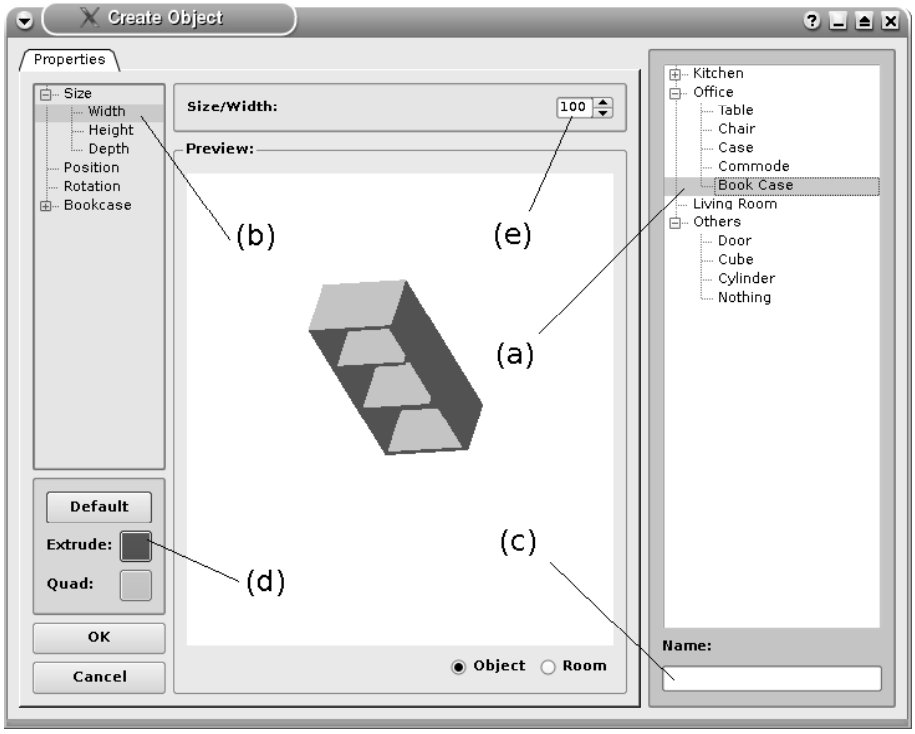
Otevření object dialogu

1. V hlavní nabídce zvolte menu **View**.
2. Přejděte na položku **Windows**.
3. Není-li zaškrtnuto políčko **Object dialog**, zaškrtněte ho.

Vytvoření nového objektu

1. V hlavní nabídce zvolte menu **Object**.
2. Přejděte na položku **Create**.
3. Zobrazí se dialog pro vytvoření nového objektu jako na obrázku **5.2**. [1]
4. V seznamu objektů **(a)** na obrázku 5.2 vyberte typ požadovaného objektu.
5. V text-boxu **(c)** na obrázku 5.2 zadejte jméno nově vytvářeného objektu.

[1] Dialog pro vytváření objektů lze též spustit dvojitým poklepáním na volný řádek v object dialogu.



Obrázek 5.2: Vytvoření nového objektu

6. V seznamu parametrů **(b)** na obrázku 5.2 vyberte vlastnost objektu, kterou chcete nastavit.

7. V list-boxu **(e)** na obrázku 5.2 upravte hodnotu vámi zvolené vlastnosti objektu.

8. V oblasti výběru barev **(d)** na obrázku 5.2 nastavte barvu vámi zvoleného objektu.

   Extrude označuje barvu vertikálních stěn objektu.

   Quad označuje barvu horizontálních stěn objektu.

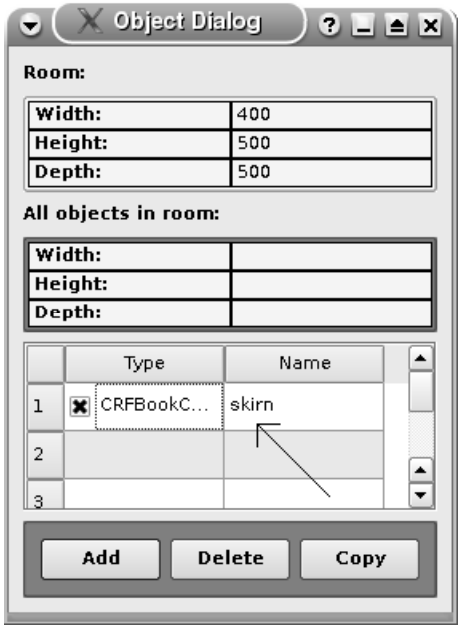
9. Stiskněte tlačítko **OK**.

10. Nový objekt byl vytvořen a o jeho existenci se můžete přesvědčit v dialogu na obrázku **5.3**.

Úprava objektu

1. Otevřete object dialog.

2. V object dialogu dvojklikem vyberte objekt, který chcete změnit.



Obrázek 5.3: Object dialog

3. Otevře se dialog podobný dialogu z obrázku **5.2**.
4. Objekt upravte postupem popsaným v předchozí sekci.
5. Stiskněte tlačítko **OK**.

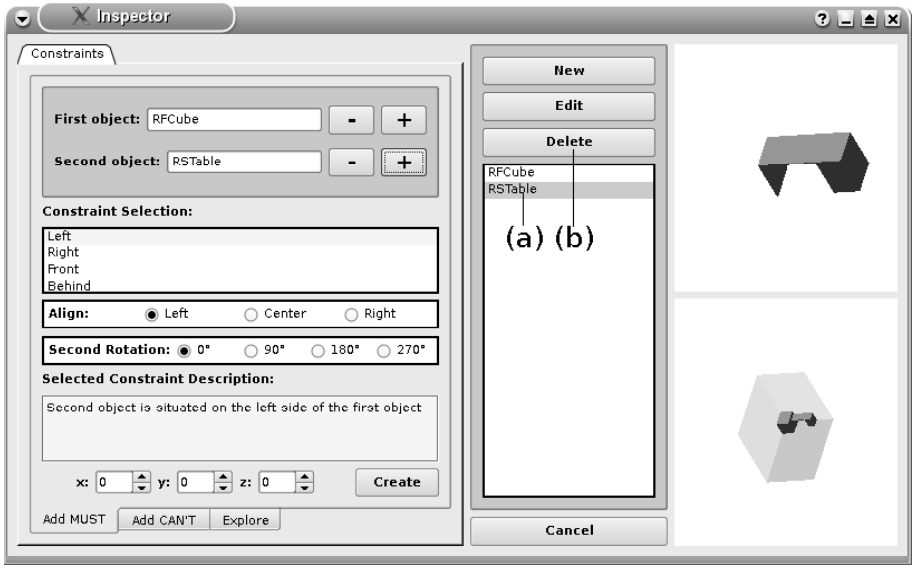
Vybrání objektu pro rozvrhování

1. V object dialogu je možné zakázat rozvrhování vybraných objektů.
2. Je-li v objekt dialogu zaškrtnut check box při daném objektu, bude objekt zařazen do rozvrhování, v opačném případě nebude objekt rozvrhován a tudíž se ani žádné vztahy příslušné k danému objektu při rozvrhování neuvažují.

Smazání objektu

1. V hlavní nabídce zvolte menu **Object**.
2. Přejděte na položku **Inspector**.
3. Zobrazí se **Inspector dialog** jako na obrázku **5.4**.
4. Klikněte na objekt, který chcete smazat, **(a)**.
5. Stiskněte tlačítko **Delete**, **(b)**.

Alternativně lze smazání objektu provést za použití tlačítka **Delete** v **Object dialogu** na obrázku **5.3**.



Obrázek 5.4: Smazání objektu

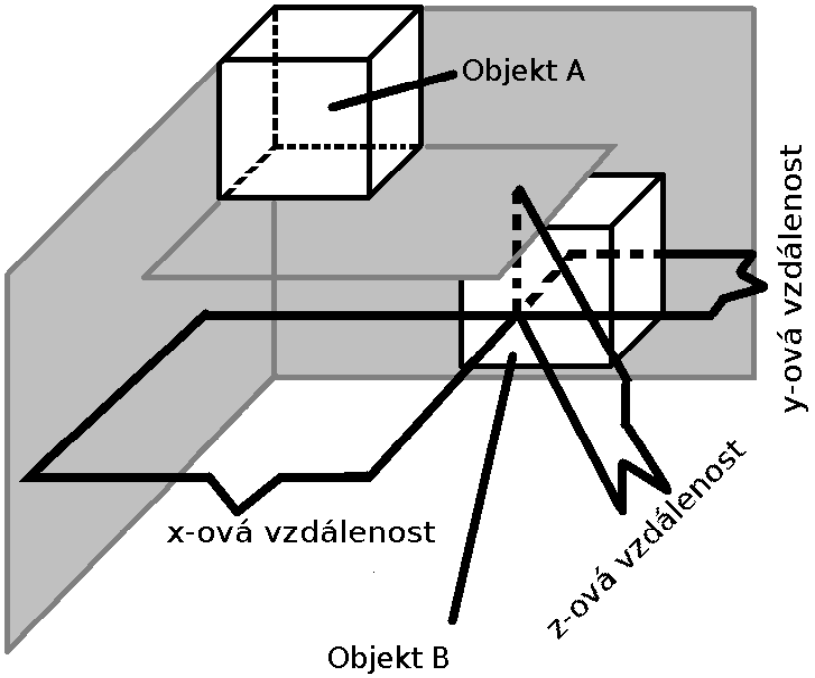
5.2.3 Omezující podmínky

Omezující podmínky v programu představují požadované vztahy mezi objekty. Uživatel si může určit pozici daného objektu vůči jinému objektu (podmínka typu MUST) nebo specifikovat prostor v okolí jistého objektu, kde se nesmí nalézat žádný jiný objekt (podmínka typu CAN'T).

Hodnoty, kterých může nabývat podmínka typu **MUST** a současně jejich popis jsou uvedeny níže. Nechť jsou tedy objekty **A** a **B** objekty, mezi kterými existuje podmínka typu **MUST** vzniklá z objektu **A**. Pak může podmínka **MUST** nabývat následujících hodnot:

- **LEFT** znamená, že objekt **B** musí stát **vlevo** od objektu **A**.
- **RIGHT** znamená, že objekt **B** musí stát **vpravo** od objektu **A**.
- **FRONT** znamená, že objekt **B** musí stát **před** objektem **A**.
- **BEHIND** znamená, že objekt **B** musí stát **za** objektem **A**.
- **ON** znamená, že objekt **B** musí stát na objektu **A**.
- **UNDER** znamená, že objekt **B** musí ležet **pod** objektem **A**.
- **SPECIFIC SIZE** znamená, že objekt **B** má od objektu **A** přesně definovanou polohu. Tato poloha je určena od pravého spodního zadního rohu objektu **A** při nulové rotaci objektu **A**. Situace je znázorněna na obrázku 5.5.

U každé podmínky typu **MUST** si může uživatel současně zvolit hodnotu **vztahové rotace** a hodnotu **vztahového zarovnání**. Bližší podrobnosti týkající se vztahové rotace a vztahového zarovnání se nalézají v kapitole **Algoritmus programu Spaceout**.



Obrázek 5.5: Podmínka SPECIFIC SIZE

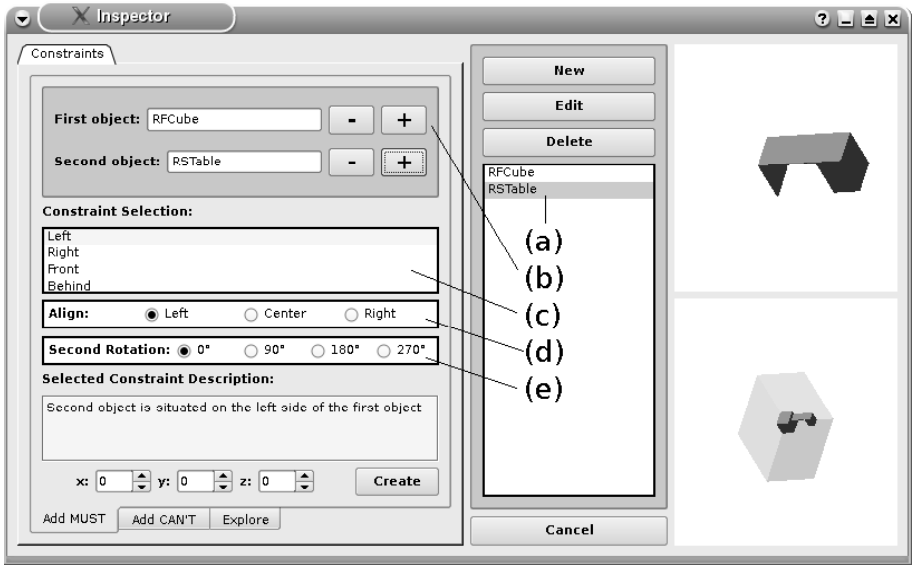
Hodnoty, kterých může nabývat podmínka typu **CAN'T** jsou uvedeny a popisány níže. S každou hodnotou musí být současně definována velikost oblasti (šířka, výška a hloubka), ve které se nesmí nic nalézat. Nechť je **A** objektem, na který je navázána podmínka typu **CAN'T**. Pak může podmínka typu **CAN'T** nabývat následujících hodnot:

- **NOTHING LEFT** značí, že v definované oblasti **nalevo** od objektu **A** nesmí stát žádný jiný objekt.
- **NOTHING RIGHT** značí, že v definované oblasti **napravo** od objektu **A** nesmí stát žádný jiný objekt.
- **NOTHING FRONT** značí, že v definované oblasti **před** objektem **A** nesmí stát žádný jiný objekt.
- **NOTHING BEHIND** znamená, že v definované oblasti **za** objektem **A** nesmí stát žádný jiný objekt.

- **NOTHING ON** znamená, že v definované oblasti **nad** objektem **A** nesmí stát žádný jiný objekt.

Vytvoření omezujících podmínek typu MUST

1. V hlavní nabídce programu zvolte menu **Object**.
2. Přejděte na položku **Inspector**.
3. Zobrazí se **Inspector dialog** jako na obrázku 5.6, ve kterém se přepněte do záložky **Constraint/Add MUST**.
4. Vyberte objekty, mezi kterými chcete vytvořit omezení **(a)**, a přidejte je pomocí **(b)**.
5. Zvolte typ omezující podmínky **(c)**.
6. Zvolte zarovnání **sekundárního** objektu vůči **primárnímu** objektu **(d)**.
7. Zvolte rotaci **sekundárního** objektu vůči **primárnímu** objektu **(e)**.
8. Omezující podmínku, kterou se chystáte vytvořit, si můžete prohlédnout v okně **Preview**, které se nalézá v právé dolní části **Inspector dialogu**.
9. Klikněte na tlačítko **Create**.



Obrázek 5.6: Vytvoření omezující podmínky

Vytvoření omezujících podmínek typu CAN'T

1. Otevřete **Inspector dialog** podle postupu uvedeného v předcházející sekci.
2. Přepněte se do složky **Constraint/Add CAN'T**.
3. Analogicky s vytvářením podmínek typu **MUST** vyberte a tlačítkem "+" přidejte objekt, kterému chcete vytvořit podmínku typu **CAN'T** a specifikujte typ podmínky,
4. Klikněte na tlačítko **Create**.

Prohlížení a mazání omezujících podmínek

1. Otevřete **Inspector dialog**.
2. Přepněte se do složky **Constraint/Explore**.
3. Vyberte objekt, jehož **Constrainty** chcete prohlížet.
4. V tabulce **Constraint for** se zobrazí všechny existující constrainty pro zvolený objekt.
5. V tabulce vyberte **Constraint**, jehož detaily chcete zobrazit.
6. Zvolený **Constraint** můžete případně smazat kliknutím na tlačítko **Delete** ve složce **Explore**.

5.2.4 Projekt

Uložení projektu

1. V hlavní nabídce zvolte menu **File**.
2. Přejděte na položku **Save**.
3. Zvolte umístění a název projektu.
4. Uložte projekt kliknutím na tlačítko **Save**.

Otevření existujícího projektu

1. V hlavní nabídce zvolte menu **File**.
2. Přejděte na položku **Open**.
3. Vyberte projekt s koncovkou 's', který chcete otevřít.
4. Klikněte na tlačítko **Open**.

5.2.5 Algoritmus

Nastavení algoritmu

1. V hlavní nabídce zvolte menu **Algorithm**.
2. Přejděte na položku **Options**.
3. Zobrazí se dialog pro nastavení algoritmu jako na obrázku **5.7**.
4. Nastavte požadované parametry:

   Individuums per generation - Počet jedinců v jedné generaci algoritmu.

   Probability of mutation - Pravděpodobnost mutace genetického algoritmu.

   Probability of cross over - Pravděpodobnost křížení v genetickém algoritmu.

   Intelligent layout ON - Zapne použití utility Intelligent Layout s tím, že staví užitečné objekty nad problematické objekty. (Její podrobnější popis je k dispozici v kapitole **Algoritmus programu Spaceout**).

   Intelligent layout FRONT - Zapne použití utility Intelligent Layout s tím, že staví užitečné objekty před problematické objekty.

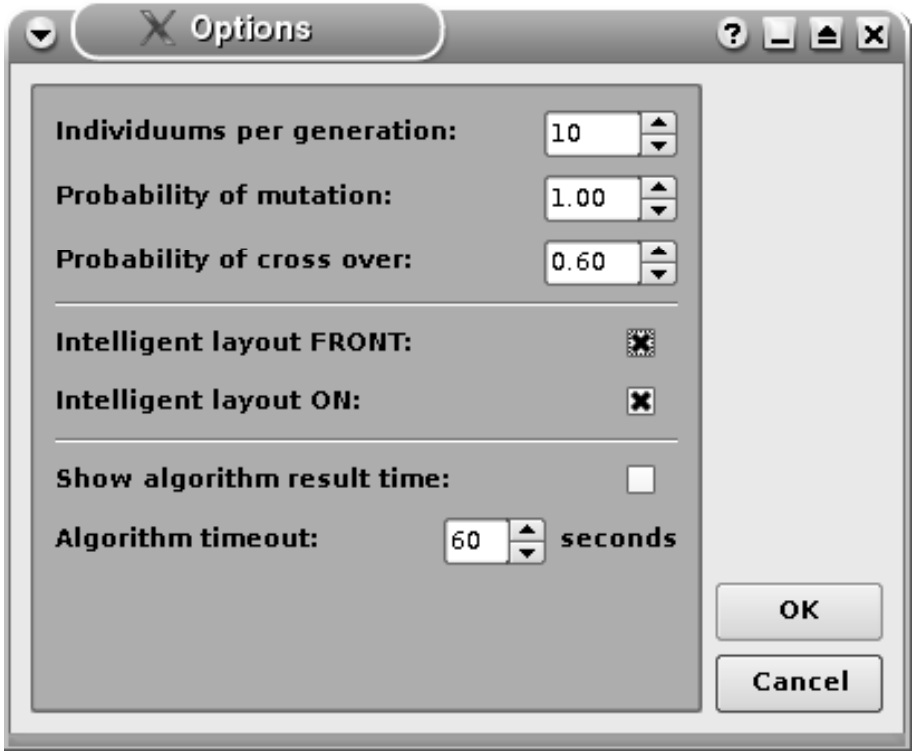
   Show algorithm result time - TIME result je utilita, která po skončení generování zobrazí celkový čas běhu algoritmu.

   Algorithm timeout - TIMEOUT označuje čas, po který má program možnost vytvořit návrh rozmístění. Pokud se programu nepovede ve zvoleném čase vytvořit návrh, bude automaticky rozvrhování ukončeno.
5. Potvrďte nastavení stiskem tlačítka **OK**.

Spuštění algoritmu a jeho ovládání

1. V hlavní nabídce zvolte menu **Algorithm**.
2. Přejděte na položku **Start Genetic**.
3. Spustí se algoritmus, jehož průběh lze sledovat v dialogovém okně, které se po spuštění otevře. Algoritmus je možné ukončit kliknutím na tlačítko **Abort** v tomto okně.
4. Ovládání je následující:

   Poté, co doběhne první část algoritmu, lze návrhy prohlížet pomocí šipek v tabulce **Element** a ohodnocovat pomocí tabulky **Evaluation**, které jsou umístěny v dolní části hlavního okna programu.



Obrázek 5.7: Nastavení algoritmu

Přechodu na specifikovaný prvek aktuální generace lze též docílit pomocí dialogu spuštěného z menu **Algorithm** a položky **Go To**.

Po stisku tlačítka **NEXT** v tabulce **Generation** se přejde do druhé fáze algoritmu, ve které se náhodně vygeneruje další množina návrhů na základě ohodnocení jednotlivých návrhů. Tlačítko **NEXT** se může použít opakovaně pro generování stále nových a nových návrhů.

5.2.6 Nastavení programu

1. V hlavní nabídce zvolte menu **Program**.
2. Přejděte na položku **Options**.
3. Zobrazí se okno, kde je možné změnit defaultní barvy nábytku a stěn pokoje.
4. Potvrďte nastavení stiskem tlačítka **OK**.

5.2.7 Scéna

Nastavení scény a ovládání scény

1. Scénu můžeme měnit primárně tažením myši. Při tažení myši a současném stisknutí **levého tlačítka** se scéna natáčí kolem **horizontálních** os. Při tažení myši a současném stisknutí **pravého tlačítka** se scéna natáčí kolem **vertikální** osy. Stejné akce můžeme docílit použitím ovládacích šipek, které se nacházejí v pravé dolní části hlavního okna programu.

2. Vlastnosti zobrazení je možné měnit v menu **Scene**:

   Zoom In respektive **Zoom Out** - Zajišťuje **přiblížení**, respektive **oddálení** scény. Stejné akce lze dosáhnout stisknutím tlačítka **"+"** respektive **"-"** v pravé dolní části okna programu.

   Show 3D, respektive **Show 2D** - Nastavuje 3D, respektive 2D pohled na scénu.

   Model/Fill model respektive **Model/Wired model** - Zobrazí model, kde jsou objekty vyplněny specifickou barvou, respektive zobrazí drátěný model objektů.

5.2.8 Výstup programu

Export vybraného návrhu do souboru

1. V hlavní nabídce zvolte menu **File**.

2. Přejděte na položku **Export**.

3. Zadejte jméno a umístění souboru, do kterého má být aktuální návrh exportován.

4. Klikněte na tlačítko **Save**.

Kapitola 6

Programátorská dokumentace programu Spaceout

V této kapitole popíši datové struktury, které program Spaceout využívá. Též se budu věnovat popisu celého programu z implementační stránky.

6.1 Implementační informace

Program je napsán v jazyce **C++** s použitím **OpenGL**. Pro vykreslování oken je použit toolkit **Qt 4.2.1**. Program byl odladěn v operačním systému **Linux** (Ubuntu 6.10 Edgy Eft). Díky multiplatformnímu toolkitu **Qt** může být ovšem kompilován a spuštěn i v operačním systému **Windows**. Nicméně pod operačním systémem Windows není funkčnost programu ověřena.

Číslo verze programu je rozloženo do 6 cifer. První dvě určují verzi **programu**, druhé dvě určují verzi **uživatelské dokumentace** a poslední dvě cifry označují verzi **programátorské dokumentace**.

6.2 Struktura programu

Program je rozdělen do pěti základních modulů.

- **Grafický** modul je tvořen rozhraním pro vykreslování objektů pomocí **OpenGL**.
- **Algoritmický** modul zajišťuje samotné rozvrhování objektů na základě předem daných podmínek a omezení.
- **Organizační** modul slouží pro správu a uchování dat.
- **Konfigurační** modul obsahuje defaultní konfigurace programu.

- **GUI modul** implementuje veškerá uživatelská rozhraní, která se v programu vyskytují.

6.2.1 Grafický modul

Grafický modul se stará výhradně o vykreslování objektů pomocí **OpenGL** na **vykreslovací widget**. Implementuje metody pro vykreslení všech objektů (stůl, židle,...), které jsou programem podporovány.

Vykreslovací widget

Základním prvkem, který slouží pro vykreslování objektů je **GLWidget**. Protože třída **GLWidget** je odvozená od **QGLWidget**, je nad tímto widgetem možná práce s **OpenGL**.

Deklaraci a definici třídy **GLWidget** nalezneme v adresáři **./GLWidget**.

6.2.2 Algoritmický modul

Algoritmický modul je implementován v souboru **./Algorithm/Algorithm.cpp**. Jeho úkolem je vygenerovat na základě omezení daný počet návrhů a předávat tyto návrhy **grafickému modulu**, který se postará o jejich vykreslení.
Podrobný rozbor algoritmu naleznete v kapitole **Algoritmus programu Spaceout**.

6.2.3 Organizační modul

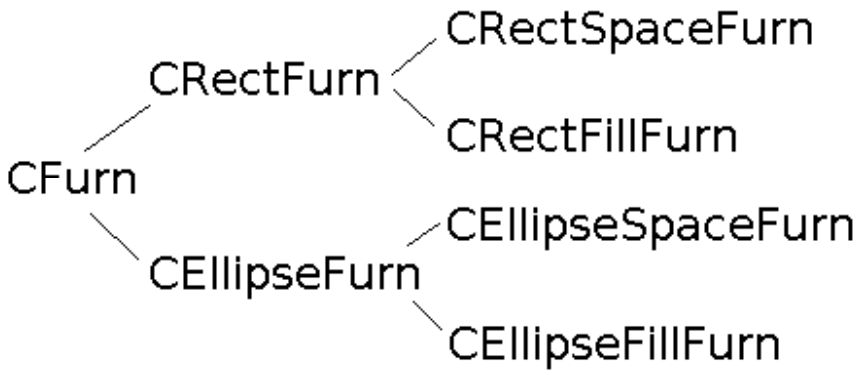
Centrálním prvkem **organizačního** modulu je instance třídy **CRoom**, jejíž implementace se nalézá v adresáři **(./CRoom)**. Tento objekt se stará o **načítání**, **uchování** a **správu** vstupních dat. Obsahuje informaci o rozměrech pokoje a o všech viditelných i neviditelných objektech pokoje.

Třída CFurn

Třída **CFurn** slouží k uchování veškerých parametrů objektů. Samotná třída **CFurn** je abstraktní třídou, od níž jsou dále odvozeny třídy **CRectFurn** a **CEllippseFurn**, které jsou též abstraktní. Od tříd **CRectFurn** a **CEllipseFurn** jsou zděděny třídy **CRectFillFurn**, **CRectSpaceFurn**, **CEllipseFillFurn** a **CEllipseSpaceFurn**, které rovněž vystupují jako abstraktní třídy. Hierarchie je naznačena na obrázku 6.1.

Od tříd v poslední úrovni této hierarchie jsou nakonec odvozeny cílové třídy jednotlivých kusů nábytku. Těmito třídami jsou například **CRFCube** nebo **CESTable**, kde druhý znak názvu třídy vždy označuje vizuální vlastnost (**R**ect - **R**ectangular - hranatý, **E**llipse - kulatý) a další znak označuje funkční vlastnost (**F**ill - plný, **S**pace - prostorný, což znamená, že pod objekt může být zastrčen jiný

objekt). Každý objekt navíc obsahuje metodu **draw**, která se postará o vykreslení na **GLWidget**.



Obrázek 6.1: Hierarchie odvozených tříd CFurn

Třída Constraint

Instance třídy **Constraint** vystupují jako omezující podmínky mezi jednotlivými objekty (objekty typu Constraint vytvářejí relaci mezi dvěma objekty). Každý objekt typu **Constraint** je specifikován **vlastníkem omezení**, **identifikací objektu**, který je vlastníkem omezen, a **typem constraintu**, který určuje pozici omezeného objektu vůči objektu vlastníka.

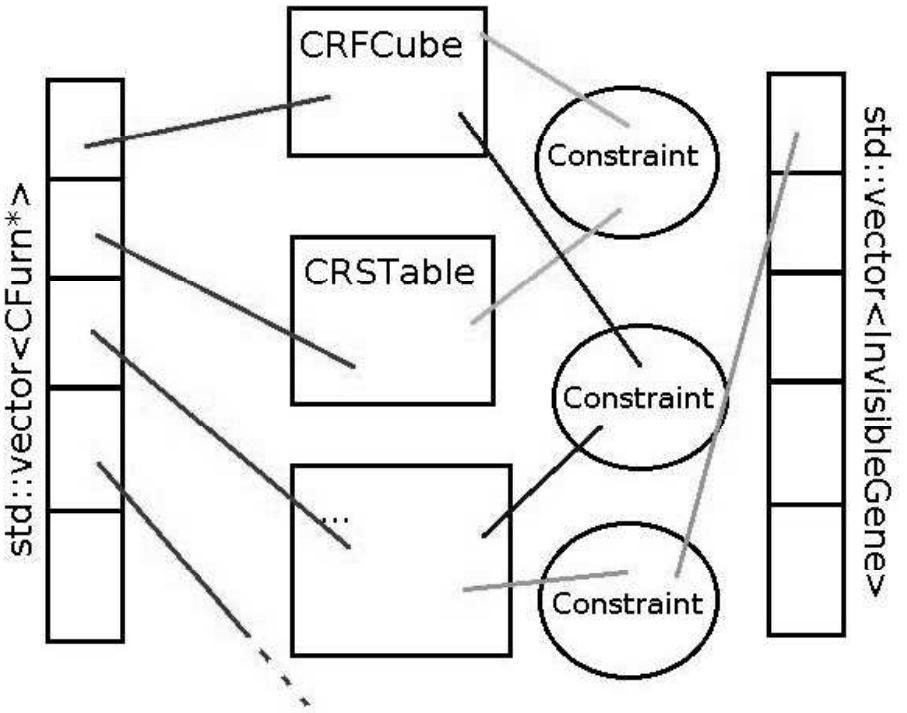
Třída InvisibleGene

Třída **InvisibleGene** je třídou neviditelných objektů. Každý pokoj (instance třídy **CRoom**) má kromě vektoru nábytku také vektor neviditelných genů (**InvisibleGene**), které určují místo, kde nesmí nic stát. **InvisibleGene** má dán svůj rozměr a pozici, která je buď absolutní vůči celému pokoji, nebo relativní vůči danému objektu. Mezi instancí **CFurn** a **InvisibleGene** může vzniknout určitá relace **Constraint**, která bude například určovat, že vedle daného objektu na určité straně nesmí stát žádný objekt (respektive musí stát **InvisibleGene**).

Třída Chromozome

Jelikož rozvrhovací algoritmus pracuje nad množinou chromozomů, musí existovat určitá interpretace objektu typu **Chromozome**, podle které lze sestavit přesný návrh pokoje.

Objekt **Chromozome** obsahuje **vektor genů**, jehož velikost je dána počtem objektů v pokoji. Dále obsahuje vektor **neviditelných genů** (InvisibleGene) a proměnnou nesoucí hodnotu **fitness** funkce. Každý gen obsahuje informaci o tom,



Obrázek 6.2: Vizuální představa organizace objektů a jejich constraintů

ke kterému objektu je vázán. Dále kóduje pozici a rotaci vázaného objektu, včetně informace o tom, zde je tento objekt fixní, či nikoliv. Chceme-li vykreslit nějaký chromozom, probíhá operace tak, že se hodnoty pozic a rotací objektů ve třídě **CRoom** nastaví podle vykreslovaného chromozomu. Tyto objekty se následně vykreslí. Postup vykreslování je zachycen na obrázku 6.3.

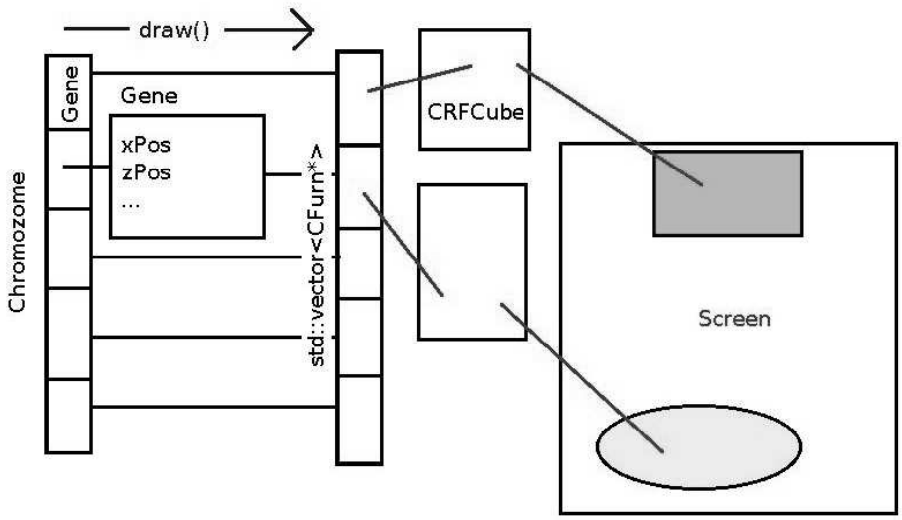
6.2.4 Konfigurační modul

Konfigurační modul slouží pro uchování defaultních parametrů programu. Implementace **konfiguračního modulu** se nachází v adresáři **./DefaultConst**.

Další třídou určenou pro konfiguraci je třída **Config**, která slouží pro načítání parametrů specifikovaných uživatelem. Její implementace se nachází v adresáři **./Config**. Program tedy primárně načítá parametry pomocí objektu **Config** a v případě, že zvolený parametr nenajde, použije příslušnou hodnotu nalezenou v **DefaultConst**.

6.2.5 GUI modul

GUI modul implementuje veškerá uživatelská rozhraní dialogů a hlavního okna. Zdrojové soubory všech dialogových oken se nalézají v adresáři **./dialog**.



Obrázek 6.3: Vykreslení chromozomu

Dialogová okna

Všechna dialogová okna kromě **object dialogu** jsou v programu implementována tak, že třídy reprezentující daný dialog obsahují nějakou statickou metodu, jejímž zavoláním se vytvoří a zobrazí daný dialog. Instance těchto tříd tedy nejsou vytvářeny předem.

Kapitola 7

Závěr

7.1 Problémy

V průběhu vývoje programu bohužel nastal podivný problém s exportem návrhů do obrázku. Ani po několikatýdenním pátrání se nepodařilo zjistit, proč se výstup OpenGL chová ve stejných verzích Qt naprosto odlišným způsobem. Důvodem je pravděpodobně bug v toolkitu Qt, který byl podle dostupných zdrojů částečně odstraněn v Qt verze 4.4. Správného exportu bylo v průběhu testování nakonec docíleno právě v Qt verze 4.4, nicméně i v této v současnosti nejnovější verzi se export chová občas velmi nestandardním způsobem, kdy vytvoří obrázek rozmazaný, nebo obrázek, který místo vytvořeného návrhu obsahuje část okna aplikace. Bohužel v této novější verzi Qt pro změnu dochází k problémům, kdy se nezobrazují barvy na tlačítkách určených k výběru barvy objektu nebo barvy pokoje.

7.2 Další vývoj

Program Spaceout ukazuje zajímavou možnost využití genetického algoritmu pro rozvrhování nábytku a domnívám se, že potenciál tohoto přístupu by bylo možné aplikovat i na složitější kusy nábytku. V současné verzi 3.1.2.4.2.1 ovšem pracuje jen s poměrně jednoduchými objekty, mezi kterými mohou vznikat podmínky vybrané pouze z úzkého spektra možných omezení. V budoucnu by bylo možné zahrnout do programu i přísnější omezující podmínky, s čímž jsou ovšem spojeny nemalé problémy popsané níže. Fitness funkce jednotlivých vygenerovaných návrhů by nemusela být určována výhradně uživatelem, ale mohla by být ovlivněna i ostatními elementy, jako je například dosah kabelů do zásuvky elektrického vedení nebo dostupnost přípojky vody a plynu. Měl by být zohledněn i nejčastější směr pohybu v místnosti, aby tento nebyl zvoleným rozmístěním objektů znesnadňován.

Všechna tato vylepšení s sebou ovšem přináší celou řadu problémů, z nichž nejzávažnější je časová náročnost výpočtu. Úvodní generace, nad kterou běží genetický algoritmus, musí být totiž vygenerována nějakým deterministickým algo-

ritmem. Tento algoritmus běží ve většině případů v naprosto nepřijatelném čase, protože všech možných rozestavení je nezanedbatelně velké množství, a pouze malá část z nich vyhovuje zadaným podmínkám. Proto také každá přísnější podmínka zanáší do výpočtu obrovskou časovou brzdu, protože významně zúží spektrum možných rozestavení. Pro genetický algoritmus je tedy přijatelnější podmínka, která nutí stát objekt v jisté definované oblasti v okolí jiného objektu, než podmínka, která určuje přesnou pozici objektu vůči jinému. Rozvrhování v prvotní generaci je proto nutné řešit velmi efektivním způsobem. Nalezení algoritmu pro vygenerování úvodní generace bych tedy označil za jednu z nejobtížnějších úloh celého programu. Rovněž způsob implementace genetických operátorů představuje v programu poměrně zásadní problém. Mutace nebo křížení totiž mohou řetězovou reakcí způsobit takové kolize, že bude nutné zbytek objektů rozvrhnout podobně složitým způsobem jako v algoritmu pro generování úvodní generace, což, jak jsem naznačil výše, může znamenat smrtící problém pro celý výpočet.

Dále by bylo velmi zajímavé propojit navržený algoritmus s nějakým jiným programem, který je naopak zaměřen na dokonalou grafiku. Tím by uživatel získal návrhy nejen funkčně užitečné, ale současně i návrhy zpracované graficky na profesionální úrovni.

Kapitola 8

Přílohy

8.1 Instalační CD

Přiložené instalační CD obsahuje mimo zdrojových souborů programu Spaceout verze 3.1.2.4.2.1 také elektronickou podobu této bakalářské práce.

Literatura

[1] Jun H. Jo, Johm S. Gero: *Space Layout Planning using an Evolutionary Approach*, Artificial Intelligence in Engineering 12, 1998.

[2] Tetsu Kuroda, Keiichi Yamagata, *A Trial of Genetic Algorithm to Furniture Arrangement in a Room*, Faculty of Integrated Arts and Sciences, Hiroshima University Higashi-Hiroshima 739-8521, Japan

[3] Toshihiko Ono, Tomoyuki Yokoi, *Optimal Cutting of Two-dimensional Free Patterns Using Genetic Algorithms*, Dept. of Computer Science and Eng., Fukuoka Institute of Technology 811-0295, Japan

[4] Vladimír Mařík, Olga Štěpánská, Jiří Lažanský a kolektiv.: *Umělá inteligence (3)*, ACADEMIA, 2001.

[5] OpenGL Home Page, http://www.opengl.org/wiki/index.php/

[6] Trolltech Home Page, Qt installation section, http://doc.trolltech.com/4.2/install-x11.htm

[7] Trolltech Home Page, QtOpenGL Module section, http://doc.trolltech.com/4.2/qtopengl.html

[8] SketchUp Home Page, http://sketchup.google.com

[9] Sweet Home 3D Home Page, http://sweethome3d.sourceforge.net

[10] Bo Concept Home Page, http://www.boconcept.cz

[11] Room Arranger Home Page, http://www.roomarranger.com