特別研究報告書

制約最適化アプローチによるWebページレイアウトデザイン

指導教員　　石田 亨

京都大学工学部情報学科

島田 寛基

平成27年1月30日

制約最適化アプローチによる Web ページレイアウトデザイン

島田 寛基

内容梗概

Web サイトデザインとは, クライアントの提示する要求仕様に基いてデザイナがクライアントの要望するコンテンツ（画像や文字など）の作成及び配置を行い, そのフィードバック・再デザインを通じてクライアントの真の要求に近づいていく作業である. この作業過程の中で特に Web サイトデザインにおけるワイヤフレーム（ページ内のどのような要素をどこに配置するかのみをまとめた骨組み）を作成する作業はデザインの全工程のうち一定の割合を占めていながら, デザイン原則や要求仕様に強く縛られているため最適化問題的な側面が強く, その後の工程と比較して人間特有の創造力を必要とするわけでもないため, 自動化されるべきであると考えられる.

しかし, Web デザインという問題の特性上, その問題の性質や自動化のためのモデル, そもそもデザインとは何なのかということが明らかにされてこなかったため自動化が困難であった. そこで, 本研究では無数の可能なレイアウトの組み合わせの中から, 制約に適合するものを絞り込むことでデザイナの作業負担を軽減するという目的の下, WEB デザインにおけるワイヤフレームの作成（レイアウト）の部分に焦点を当て, 次のふたつの課題に取り組んだ.

1. Web ページレイアウト問題のためのモデルの欠如
   Web ページレイアウト問題は他のレイアウト問題とは異なり明確な目的関数が定義できないほか, 最適化できる形で要求仕様を記述する方法が不明であったため, 問題をどのように扱うべきかというモデルがなかった.
2. 現行の自動化手法の問題
   現在 Web ページデザインには対話的遺伝的アルゴリズム（IGA）が使われる例が多いが, このアルゴリズムにはユーザの要望が直接的に反映されにくく, 人間がアルゴリズムに関与するため探索空間の広さのわりに試行回数が限られているという問題点がある.

Web ページレイアウトデザイン（ワイヤフレームの作成）とはクライアントの要求仕様に基づいてコンテンツの配置を行う作業であり, 本研究ではこれをレイアウト原則による制約とクライアントの要求仕様による制約の 2 つの制約を満たそうとする制約充足問題とみなした.

クライアントの要求を制約と捉えることはできるが, クライアントは当然満たすと思われるべき制約（レイアウトの構成要素であるボックスが整列されていること・ボックスが重なっていないことなど）を明示的に指定するわけではない．そこで, 当然満たすべきである制約・デザインのガイドライン上満たしたほうが良いとされると制約を予め記述しておき, その中でクライアントの要望する条件を満たすという形のモデルをとった．要求仕様を直接記述させることができればユーザの要望が反映されやすくなる．

制約を充足させる手法として, 本研究ではマルチエージェントシステムを用いた手法を提案した．つまり, 各レイアウト構成要素（ボックス）をエージェントとみなし, 各ボックスが自律的にレイアウト全体に課される制約とエージェントごとの制約をみたすように移動し, 配置を決定するというマルチエージェントシステムとして問題解決を行うこととした．マルチエージェントシステムは準最適解を高速で求めることに適している上, 最適化が終了した段階からの部分的修正もし易いため, プリント基板レイアウト問題や施設レイアウト問題などのレイアウト問題にも多く用いられてきた．本研究では, プリント基板レイアウト問題に使われた組織学習指向型分類子システム（Organizational-learning oriented Classifier System: OCS）を制約充足問題を解決できるよう拡張した．

本研究では, 提唱したモデル・システムの正当性を検証するため, 提案手法を用いて実際に Web ページのワイヤフレームを作成するソフトウェアを実装した．簡単なレイアウト問題における収束状況をランダムサーチと比較する実験と, より複雑なレイアウトを作成できるかを検証する実験を行った．

本研究の貢献は以下の 2 点である．

1. Web ページレイアウト問題のモデル提案
   前述のとおり, 問題の性質や自動化のためのモデルなどに関しての議論は盛んに行われておらず, 未だ不明瞭な部分も多かった．本研究では, WEB ページレイアウト問題を制約充足問題として解くためのモデルを提唱した．
2. OCS を用いたレイアウト制約充足問題解法の提案
   本研究の手法がベースとした OCS はエージェントが協調することで目的関数を最大化・最小化するシステムであるが, このシステムを各エージェントが満たすべき制約とレイアウト全体が満たすべき制約の両方を満たす制約充足問題を解決できるものに拡張する方法を示した．

A Constraint Optimization Approach to Web Page Layout Design

Hiroki Shimada

Abstract

Web design is a process where designers create and arrange the web content (text and images) based on client requirement and progressively redesign it by taking account the user feedback to approach client's true requirements. The wireframe (a blueprint that designates where to put contents in the page) design is a part of this process that is subject to design guideline and user requirement, and thus it is more like optimization problem. Moreover, it accounts for certain proportion of the whole design process. Therefore, it is reasonable to automate the process of layout design by modeling it as an instance of optimization problem to decrease the human effort.

However, realizing the automated layout design is a challenging task due to complexity of design problem. In this study, to reduce the workload of web designers by narrowing possible layout options, we addressed the following two problems regarding the layout design focusing on the process making wireframe of web page.

1. Absence of the model for Web page layout design

   The optimization model and formulations of web layout design is yet to be proposed. Unlike the other type of layout problems, it is difficult to create a clear objective function for layout design because there is no known approach can describe the user requirement in the form of optimization.

2. Problem in Existing Automation Method

   Interactive genetic algorithm has been widely used in web design. However, it is problematic: 1) it is difficult to reflect the user requirement in the result of design directly, and 2) the number of trail is limited in spite of big search space due to human interposition.

We, first, regard the web layout design as a constraint satisfaction problem (CSP) to satisfy two constraints: 1) a constraint specified by design guideline and 2) a constraint specified by client request. Generally, the clients do not designate constraints which should be satisfied naturally (such as alignment of

boxes and absence of overlapping). Among these two constraints, one that is specified by guideline is given in advance, and the system takes account the second constraint iteratively during the solving process. It is easy to reflect the user request if any requirement is described directly.

The multi-agent system has been used in various layout design problems such as printed circuit board layout and facility layout problem due to its high performance in finding the sub-optimal solution in short time. Moreover, it is easy to make partial change after optimization. To solve the CSP, we extended the Organization-learning Oriented Classifier System (OCS) to solve CSP, which is used in printed circuit board design. Proposed method is based on multi-agent system: each agent corresponds to an element of layout (box), and tries to satisfy both its own constraint and a constraint that is imposed on the layout. The agent also decides its position.

In this study, we also designed a software as an implementation of our optimization-based proposal for web layout design. Using this software, we evaluated the validity of the method by comparing its convergence process with random search, and we conducted an experiment to generate complex layout using the software.

This study makes following two contributions:

1. Proposal for the model for Web page layout problem
   As mentioned above, a clear picture of the Web page layout design problem has not been revealed, and there are still a lot of fuzzy parts. This study proposed the model for the problem to solve it as CSP.
2. Proposal for the method for layout CSP using OCS
   OCS is the system to maximize/minimize objective function by letting agents cooperate. This study made it possible for OCS to solve CSP to find a solution which satisfies both constraints that each agent should independently satisfy and ones that whole layout should satisfy.

制約最適化アプローチによる Web ページレイアウトデザイン

目次

第1章　はじめに

近年, コンピュータ技術の発達に伴い人間の多くの作業がコンピュータにより自動化されてきた．特に, ものづくり産業において人間は設計のみを記述し, 実際に手を動かす（製作）作業はコンピュータやロボットに行わせるという自動化の動きが各所で見られるようになった．産業ロボットによる製造作業の自動化や 3D プリンタなどもその一例である．しかし, 人工知能技術が発達した今でも, WEB デザインをはじめとする人間の知的生産活動についての自動化は他の分野に比べ未だ十分な議論がなされていない．本研究では WEB レイアウトデザインのためのモデルと最適化手法の提案を通じて, 上に述べたような問題のの新たな切り口を提供したい.

WEB サイトを製作する工程は, クライアントからの要求仕様の聞き取り, ラフスケッチ, ラフスケッチのワイヤフレーム化, ワイヤフレームのモックアップ化（ワイヤフレームに実際のコンテンツを配置し, 肉付けしていく作業）, モックアップの HTML コーディング, バックエンドのコーディングに分けられるが, これらの作業は単純作業と, 人間の独特な判断力や創造力を要する単純作業で無い部分に分けることができる．例えば, 画像やアイコンなどのコンテンツを作成しワイヤフレームに肉付けしていく工程は自動化は非常に困難であるが, モックアップの HTML コーディングや, ワイヤフレームの作成はその前の工程の終了後ただちにすべきことが決まる工程であり, コンピュータによる置き換えが可能であると考えられる．今回は, その中でもワイヤフレームの作成に焦点を当て, 自動化のモデルと手法の提案を行う．この工程はデザインの全工程のうち一定の時間を占める部分であり, 仕様から直接ワイヤフレームの候補を絞ることができればデザイナの手間と時間を削減することができる．単純作業が自動化されれば, デザイナはより重要な工程に注力することができるため, デザイン工程内の単純作業の自動化には意義があると考えられる．

この分野では主に対話型遺伝的アルゴリズム（Interactive Genetic Algorithm: IGA）[3] が多く用いられてきた [2]．IGA では, ランダムで生成された複数の個体（レイアウト）をユーザに提示し, 最も要求に近いものをユーザに選択させ, 選択された個体どうしの交配, 突然変異によって次の世代を形成する．IGA はこのような手順でユーザの要求を満たすデザインが生まれるまで世代を更新していくことで求めるレイアウトを生成しようとするアルゴリズムである．しか

し, IGA で WEB レイアウト問題を解こうとすることには以下のような問題が伴う.

1. 探索空間が広いわりに試行回数が限られている
   WEB レイアウト問題では, レイアウト構成要素（ボックス）が複数存在し, 各レイアウト構成要素が自由な横幅・縦幅・位置をとるため組み合わせ数が莫大になる. 従って, ランダムな初期配置からスタートする場合, 相当数の試行回数（世代数）が必要になるはずであるが, IGA の場合毎回の世代生成のたびに人間の判断を必要とするため, 現実的に試行回数が限られている.
2. ユーザの要望が反映されにくい
   デザインというものはクライアントの要求仕様を満たすことを目的とした作業であるが, IGA の場合デザインに対する要求を直接記述できないため, そのユーザの要望が反映されにくいという欠点がある. また, ユーザが取れるアクションは提示されたデザインの選択のみなので, 要求に適合したデザインが現れない限り要求を反映させることができない.

そこで, 上記の問題に対応するため, 本研究ではボックスのレイアウト問題を制約充足問題としてとらえることのできるモデルを提唱する. 最適化プロセスそのものからはユーザの介入を外すことで大量の試行を行うことを可能にし, 制約を直接ユーザに記述させることで要求を結果に反映しやすくした.

制約充足のためのアルゴリズムにはマルチエージェントシステムを採用した. つまり, 各レイアウト構成要素（ボックス）をエージェントとみなし, 各エージェントが自律的にレイアウト全体の制約とエージェントごとの制約をみたすように移動し, 配置を決定するというマルチエージェントシステムとして問題解決を行うこととした. 本研究では, 組織学習指向型分類子システム（Organizational-learning oriented Classifier System: OCS）[1] をベースにして制約充足問題に対応できるよう改良したシステムを提唱する. マルチエージェントシステムを採用した理由は以下である.

1. クイック計算
   一般にレイアウト問題は各レイアウト構成要素が自由な横幅・縦幅・位置をとるため組み合わせ数が莫大になる. 分枝限定法やスケジューリング理論などのような従来手法では莫大な計算時間がかかる一方, マルチエージェントシステムでは各エージェント単位で制約を満たす場合, 各エージェン

トは自分のこと（自分の移動可能な範囲）のみを考慮するため, 最適解は保証されないが準最適解を非常に早く計算できる [4].

2. ロバスト性

   遺伝的アルゴリズム（GA）のような手法の場合, 完成したレイアウトを変更すると全配置を計算し直す必要があるが, マルチエージェントシステムの場合, 修正の際は各エージェントの視点に立てば, 修正箇所に該当するエージェントのみ修正すればよく, 最初からやり直す必要は無い [4]．これは各ページごとに共通部と可変部が存在し, 部分的な変更を行うことの多い WEB レイアウトデザインでは非常に重要となる．

レイアウト問題を解く上で問題となる探索空間の広さは, マルチエージェントシステム特有のクイック計算で対応することとした． 実際に, レイアウト問題にマルチエージェントシステムが使われている例は多い [1, 7].

以降, 第 2 章ではデザイン問題に利用されたアルゴリズム・レイアウト問題や制約充足問題に使われるマルチエージェントシステムなどの関連研究の紹介を行い, 第 3 章ではデザインの目的やデザインそのものについての考察を経てモデルの提案を行い, 第 4 章では OCS をベースとした最適化手法の詳細, 第 5 章では提案手法を用いたソフトウェアの実装と実行結果およびその評価を記述する．第 6 章では本研究の応用例と今後の課題を考察する．

第 2 章　関連研究

2.1　Web デザインに関するアルゴリズム

WEB サイトデザインについて論じた研究の例として Oliver らの研究 [2] が挙げられる．Oliver らはサイトのテキストやカラー, レイアウトを定めるのは時間がかかる上, デザインに詳しくない人にとっては難しい作業であることを問題として提起している．テンプレートのようなモデルを作るということも解決策として考えられるが, サイトのパターンは何千何万とあるため, ユーザにパーソナライズされたサイトを作ることはできない．Oliver らはユーザの趣向をどのように取り入れるかということを自動生成上の最大の問題とみなし, 対話型遺伝的アルゴリズム（Interactive Genetic Algorithm: IGA）の利用を提案した．IGA は以下のステップから構成される．

1. N 個の初期個体（サイト）をランダム又はすでに存在するレイアウト・ス

タイルから生成

2. 生成された個体を表示
3. 終了するかどうかを選択, 終了した場合はアルゴリズムを終了する
4. 趣向に合う個体を選択
5. 選択された個体を残し, 残りの個体を, 選択された個体を交配あるいは突然変異させた新規個体と置き換える
6. 2 のステップに戻る

この研究ではテキストやカラーなどのスタイルと, レイアウト（ボックスの配置）を別々に最適化している．レイアウトに関しては, 全体を横 3 ×縦 n のセルに分割し, 各セルの色・セル内のオブジェクトの縦横位置・隣のセルとの結合数などをパラメータとして与えている．

前章で述べた通り, IGA で WEB レイアウト問題を解こうとすることには以下のような問題が伴う．

1. 探索空間が広いわりに試行回数が限られている
   WEB レイアウト問題では, レイアウト構成要素（ボックス）が複数存在し, 各レイアウト構成要素が自由な横幅・縦幅・位置をとるため組み合わせ数が莫大になる．従って, ランダムな初期配置からスタートする場合, 相当数の試行回数（世代数）が必要になるはずであるが, IGA の場合毎回の世代生成のたびに人間の判断を必要とするため, 現実的に試行回数が限られている．
2. ユーザの要望が反映されにくい
   デザインというものはクライアント（ユーザ）の要求仕様を満たすことを目的とした作業であるが, IGA の場合デザインに対する要求を直接記述できないため, ユーザの要望が反映されにくいという欠点がある．また, ユーザが取れるアクションは提示されたデザインの選択のみなので, 要求に適合したデザインが現れない限り要求を反映させることができない．

また, IGA に由来する上の問題の以外にも, セルの数を横 3 つに限定することや, セルのサイズがパラメータに含まれないことなどにより自由度が大きく損なわれ, 表現できるパターンの数が少なくってしまうという問題点のほか, Oliver らも指摘している通り要素間の依存関係が加味されないという問題点が存在する．

2.2 マルチエージェントを用いたレイアウト問題

マルチエージェントシステムは元来シミュレーションなどに使われることが多かったが, 近年レイアウト問題の解決に利用される例も多く見られるようになった．第一章で述べたようにレイアウト問題に対しメリットが多く, エージェントの動きに関しては一定の自由度を実現しながらも計算量が爆発しないため, IGA で発生した問題を解決できる可能性があると考えた．以下では, レイアウト問題にマルチエージェントシステムを利用する意義を示すため, そのような例をいくつか紹介する．

2.2.1 施設レイアウト問題

施設レイアウト問題（Facility Layout Problem: FLP）は, 工場施設間の素材輸送コストを最小化するか, 施設のペアが生ずる定性的な価値を表した隣接要件を最大化するという目的を持つ最適化問題である．レイアウト m 個の施設を n 個 $(m \leq n)$ の場所に配置する問題は NP 問題に分類され, 施設の数に対して計算量が指数的に増加するという性質を持つ．そのため, 準最適解を比較的高速で出すというヒューリスティックな手法が主流となってきた．

Tarkesh らの研究では FLP を解く方法としてマルチエージェントシステムを提案している [7]．マルチエージェントシステムは多様なソースから来る情報の処理に基いて意思決定がなされる複雑な問題の処理に適していると述べており, 複雑な問題を考えやすいことおよび実装のしやすさをマルチエージェントシステムの利点としている．システム内では各エージェントは施設に対応していて, それぞれ所持金・感情などの特徴量を持っていて, 各自の効用関数を形作っている．各エージェントは学習の過程でマネージャーエージェントからの指令により所持金を全体のコストが最小になるように調整する．空いている土地が全エージェントに開示され, エージェントは自らの効用を最大化するべく値段を提示し, 価格最大のエージェントが土地を手にすることができる．以上の流れでレイアウトが行われる．

Tarkesh らはエージェントの意思決定に感情（Emotion）というパラメータを導入している．現在得られた知識のもと最適な戦略を選び続けるエージェントを合理的エージェントと呼ぶのに対し, 自分の効用関数と性格（パラメータの個体差）のみにおいて最適な選択をするエージェントを合理的感情エージェントと呼び, 感情を導入したほうがデザイナの不完全な設計に対応できることが

わかっている．つまり, 同じ状況におかれているエージェントでも性格が違えば違った選択をし, その方が良い結果が生まれるのである．

他文献で議論されたヒューリスティックス・メタヒューリスティクスより良い結果を出した上, リーズナブルな時間で解けたことを報告している．また, 提案された手法は分散的な制御の求められる他の難しい問題にも適用可能であると述べている．

2.2.2 CAD プリント基板レイアウト問題

Takadama らは, CAD (Computer Aided Design) におけるプリント基板設計において, シミュレーテッドアニーリングや遺伝的アルゴリズムよりも個々のエージェントが各々の判断基準を変更しながら行動する組織学習指向型分類子システム（Organizational-learning oriented Classifier System: OCS）の方が優れた結果を出すことを示した [1]．Takadama らは回路上の部品が重なってはならないという制約のもとで部品の総配線長が最小化するというプリント基板レイアウト問題に OCS を適用し, 大域的な統制なしに専門家よりも短い配線長を見出す組織構造を構築することに成功した．OCS は, 各エージェントが各々の局所評価関数に従って行動を決定し, 更に行動に関する知識を交換しながら組織全体のパフォーマンスを向上させる組織構造を構築する．OCS の具体的なアルゴリズムは 4.1 節に述べた。

第 3 章　提案モデル

3.1　デザインの定義

デザイン問題をモデル化するにあたり, 本研究においてデザインという言葉を定義しておく必要がある．本来, デザインの対象は Web ページにとどまらず, 都市や建築, 衣服やソフトウェアなど多岐にわたる．本研究では, デザインを「与えられた制約の中で目的を最大限達成するための概念の定式化及び実装（の反復）」と定義する．概念の定式化（以下設計と呼ぶことがある）とは, デザインされようとしている物に関し, それに先立って作成されるもの [8] を定式化することで, 実装とは実際のメディアを使って実体を作ることである．なお, 実装を伴わない概念の定式化のみの作業もデザインに含めることとする．制約と目的は単一とは限らず, 複数個とることができる．建築を例にして制約と目的の例および意味を以下に示した．

- 制約
  土地の面積・使える材料・周辺の環境（隣に高い木がある, など）のようなデザインの可能性を狭める条件
- 目的
  機能性（風通しや採光）・美しさ・耐久年数などの達成することが望ましいとされる性質

制約・目的はデザインごとに異なる．さらに, 項目は同じでも優先度が異なる場合もある．通常デザインを依頼するクライアントとデザイナ間で行われるヒアリング作業によって目的や制約・その優先順位が明確化される．本研究ではそれらをまとめて仕様（要求仕様）と呼ぶ．しかし, 実際のデザインの工程ではヒアリング作業によって明確化された仕様はクライアントが本当に求めている真の仕様とは異なる場合が多く, 実装した結果をクライアントが見て仕様を再構成し, 再設計および再実装を行うことが一般的である．そのためデザインは設計及び実装の反復と定義することができるが, 単なる設計及び実装という意味でデザインという言葉を用いることもある．

本研究では以上の定義をふまえて, Web サイトデザインを「クライアントの提示する要求仕様に基いてデザイナがクライアントの要望するコンテンツ（画像や文字など）の作成及び配置を行い, そのフィードバック・再デザインを通じてクライアントの真の要求に近づいていく作業」として解釈した．本文中のデザインはクライアントの存在を前提としているが, クライアントをデザイナ自身とみなすことでクライアントの存在を前提としないデザインにも適用可能である．なお, Web ページレイアウト問題とは, Web サイトデザインの中でコンテンツを配置する問題のことであり, 本文では特に断りの無い限りワイヤフレームの配置を意味することとする．

3.2 既存手法の問題点の改善

2.1 節では, 従来 Web ページデザインに用いられてきた IGA には, 1．探索空間が広いわりに試行回数が限られている 2．ユーザの要望が反映されにくい という 2 つの問題があったことを述べた．

試行回数が限られているという問題は, 最適化プロセスの中にユーザが介入するため試行のたびにユーザによる選択が求められることが原因であった．本研究では, 最適化プロセスそのものからはユーザの介入を外し, 制約最適化アプ

ローチをとることで大量の試行を行うことを可能にした．ユーザとシステムとの対話性は最適化結果から得られるフィードバックと条件の再設定という形で実現した．詳しくは3.5節を参照されたい．また, 大きな探索空間において準最適な解を高速で求めることに適したマルチエージェントシステムを用いることで探索空間の広さに対応した．提案手法のマルチエージェントシステムでは, 各エージェントはボックス（レイアウトの構成要素と成る, コンテンツの入る領域）に対応付けた．

ユーザの要望が反映されにくいという問題は, ユーザは直接的にデザインの要求を記述することができず, 個体を選択することしか要望をシステムに伝える手段が無いことが原因であった．そこで, 本研究ではWebページレイアウト問題を次節に示すような制約充足問題としてとらえ, ユーザに制約を直接記述させることで仕様をより反映しやすくした．また, 初期レイアウトを大幅に変えたくない場合は固定するボックスを直接指定して残りのボックスだけで最適化を行うこともできる．

3.3 制約条件

Webページレイアウトを行う上で, クライアントからの要望はデザイン結果の自由度を制限し, 成果物のあり方をより明確にする点で制約と捉えることができる．だが, クライアントは当然満たすと思われるべき制約を明示的に指定するわけではない．そこで, 当然満たすべきである制約・デザインのガイドライン上満たしたほうが良いとされると制約を満たすべき制約として導入し, Webページレイアウト問題を次の2つの制約を充足する制約充足問題としてとらえた．

- レイアウト原則（ガイドライン）による制約
  ボックスを一直線に揃えるという事や, 同種のボックスの間隔とサイズは統一されるべきであるという事などのような, それに従うことでサイトのユーザビリティが向上するとされているルールや, ボックスどうしが重なってはいけないという事などのようなどんなサイトも満たすべきであると思われる制約.
- クライアントの要求仕様による制約
  クライアントの要望するコンテンツを, クライアントの要望する条件（重要度に基づくコンテンツの大きさの指定, どのコンテンツをどのグループに属させるか, など）に沿うように配置するという制約.

3.1 節に述べた通り、クライアントに採用されるデザインは制約条件に適合している必要があるため、制約条件に適合しないレイアウトを排除することで可能な選択肢を狭めることが可能である。従って、可能な選択肢を絞り込むためデザイン問題を制約充足問題としてとらえることは妥当である。

また, 提案モデル・手法では制約を以下の 2 種類に分類し, Web ページレイアウトデザインの制約を表現することとした.

- レイアウト制約
  ひとつの最適化（レイアウト）ごとに持っていて, どのボックスも他のいずれかのボックスと整列していなければいけないという条件や, ボックス間の最低間隔距離などエージェント全体やエージェント間の属性を束縛する制約. 特定の 2 つのエージェント間の制約もレイアウト制約に分類される.
- エージェント制約
  エージェント（ボックス）ごとに持っていて, 自分のサイズ（横幅と縦幅）の最低値や最大値, 滞在可能な領域など他のエージェントの振る舞いに関係の無い属性を束縛する制約. 必ずしも持っている必要はない.

以上の 4 項目の関係は図 1 のように表すことができる. クライアントもガイドラインもレイアウト制約・エージェント制約の両方の制約を指定することはあるが, ガイドラインからエージェント制約が指定されることは少ない. なぜなら, デザインのガイドラインは一般的に満たすべきルールであるため, 特定のボックスに関する規定は少ないからである. なお, どの制約も逸脱を許さない

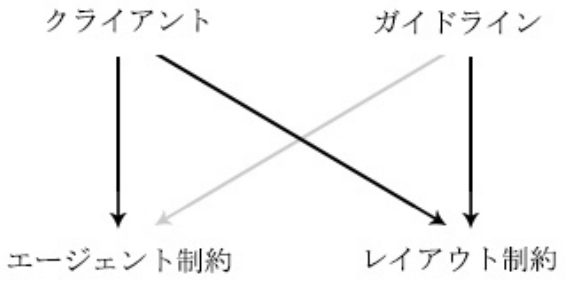

図 1: 制約の関係図

絶対制約（ハードな制約）と逸脱を許す考慮制約（ソフトな制約）の両方の属性をとれるようにした.

3.4 制約充足問題としての定式化

Web ページレイアウトデザイン問題を制約最適化アプローチで解くにあたり, 本節に示す定式化を行った. はじめに, 問題の構成要素を以下のように定義した.

(1) エージェント (ボックス)

$\boldsymbol{a} = (x,\, y,\, width,\, height,\ \boldsymbol{L})$

(2) レイアウト

$\boldsymbol{L} = \{\boldsymbol{a_1}, \cdots, \boldsymbol{a_n}\}$

(3) 制約条件

$C_L(\boldsymbol{L}) = C_{L1}(\boldsymbol{L}) \wedge \cdots \wedge C_{Lm_L}(\boldsymbol{L})$

$C_{a_1}(\boldsymbol{a_1}) = C_{11}(\boldsymbol{a_1}) \wedge \cdots \wedge C_{1m_1}(\boldsymbol{a_1})$

$\vdots$

$C_{a_n}(\boldsymbol{a_n}) = C_{n1}(\boldsymbol{a_n}) \wedge \cdots \wedge C_{nm_n}(\boldsymbol{a_n})$

(4) 最適化関数

$F_L(\boldsymbol{L}) = f_{L1}(\boldsymbol{L}) + \cdots + f_{Lm_L}(\boldsymbol{L})$

$F_{a_1}(\boldsymbol{a_1}) = f_{11}(\boldsymbol{a_1}) + \cdots + f_{1m_1}(\boldsymbol{a_1})$

$\vdots$

$F_{a_n}(\boldsymbol{a_n}) = f_{n1}(\boldsymbol{a_n}) + \cdots + f_{nm_n}(\boldsymbol{a_n})$

ページレイアウトの際の各コンテンツは, 位置（x 座標と y 座標）と横幅と縦幅と持つ領域であるボックスの中に配置されるものとする. このボックスをエージェントとした. レイアウトはエージェントの集合である. エージェントはレイアウト $\boldsymbol{L}$ を内部に持つことができ, これによってレイアウトは階層構造をとることができる. 制約条件に関しては, 前節で述べた通りレイアウト制約 $C_L(\boldsymbol{L})$ とエージェント制約 $C_a(\boldsymbol{a})$ が存在する. それぞれの制約条件は複数の条件要素に分割され, 全体の真偽値は全条件要素の論理和とした. 各条件要素は $True, False$ のいずれかの値をとる. 最適化関数は, 制約条件の各要素に関してひとつ定義され, 制約が満たされた時, またその時のみ 0 となる. 従って最適化ではその最小化を目的とする. 各関数要素の単純和をひとつの最適化関数の値とした.

本研究におけるレイアウト充足問題の目的は, 以下に定義する解レイアウト $\boldsymbol{L_s}$ を求めることである.

$$\boldsymbol{L_s} = \{\boldsymbol{a_1}, \cdots, \boldsymbol{a_n}\}$$
$$s.t.\ \ C_L(\boldsymbol{L_s}) = True \wedge C_{a_1}(\boldsymbol{a_1}) = True \wedge \cdots \wedge C_{a_n}(\boldsymbol{a_n}) = True$$

なお, このとき 4.2.2 節に示した性質により, 最適化関数は

$$F_L(\boldsymbol{L}) = 0 \wedge F_{a_1}(\boldsymbol{a_1}) = 0 \wedge \cdots \wedge F_{a_n}(\boldsymbol{a_n}) = 0$$

となっている.

3.5 提案モデル

本研究は, Web デザインの一つの工程であるワイヤフレームの作成を対象としている. 可能なレイアウトの組み合わせの中から, 制約に適合するものを絞り込むことでデザイナのワイヤフレーム作成の負担を軽減するのが目的である. ワイヤフレームの作成は, デザイン原則や要求仕様に強く縛られているため制約充足問題的な側面が強く, また, 熟練したデザイナでも考え付かない組み合わせや配置が存在することがあるため, 最適化アルゴリズムを用いて可能な解を示すことには意義がある. また, 提案モデルは前節に述べた目的の達成（目的関数の最大化）には踏み込まないことに注意されたい.

提案モデルの概要図を図 2 に示した. まず, ユーザであるデザイナはユーザイ

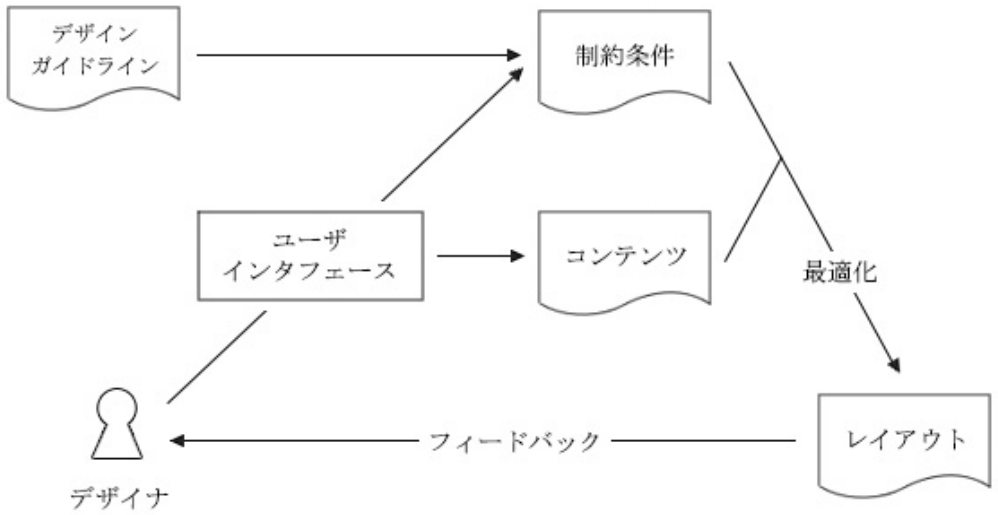

図 2: 提案モデル図

ンタフェースを介してボックス（コンテンツ）の個数・階層構造・初期設定（座標・サイズ）を指定する. また, それと同時に制約条件もユーザインタフェースを通じて指定する. 制約条件は, デザインのガイドラインに基いて予め作成さ

れたものも同時に取り入れられる．(3.3 節参照）これに関してはユーザは関与しない．最適化関数に関しては, 制約条件とペアになっているため制約が指定された段階で自動的に設定される．次に, 与えられたコンテンツがこれら 2 つの制約条件を満たすようにレイアウト最適化が行われる．最適化の結果制約を満たすレイアウトが得られたら, それをユーザに対し表示, ユーザはその結果を見て自らの設定した制約条件・コンテンツを再設定し, 再度最適化を行う．その際ユーザは動かしたくないボックスを固定することもできる．直接的な対話性はないが, 出力（最適化結果）を見て入力（制約条件とコンテンツ）を調整するという対話的なサイクルを通じてユーザはだんだんと自分の欲している結果に近づいていくことができる．

本モデルにおけるサイクルは, クライアントがデザイナにデザインを依頼するという実際の工程を模したもので, クライアントはデザイナが生じた出力（デザイン結果）をみて要求仕様に変更を加える・あるいは新たな要求を加えるということを行う．このサイクルを何度か繰り返すことでクライアントは自分の真のニーズに見合ったデザインを得る．

第 4 章　最適化手法

4.1　組織学習指向型分類子システム

本節では, 本研究に用いた手法のベースとした組織学習指向型分類子システム（Organizational-learning oriented Classifier System: OCS）[1, 5] の仕組みについて簡単に説明する．OCS は複数の学習エージェントからなり, 各エージェントは適切なルールを学習し, エージェント間で分業をすることによって個体レベルでは解けない問題の解決を目的とする．各エージェントは以下の 4 種の学習メカニズムによって学習を行う [5].

- 強化学習メカニズム
  個体の持つルールの強度を変化させることによってルールの適用の順序を変更させ, 個体に一連の適切な行動を獲得させる．
- ルール生成メカニズム
  個体が自分の持つルールで対処できない環境に遭遇したとき, その環境にあったルールを生成させる．ルールが生成されるとき, ルールの条件（IF）部はそのときの状況同じように設定され, アクション（THEN）部はラン

ダムに決定する．新しく生成されたルールの強度は初期値にセットされる．なお, エージェントの保持するルール数が上限数に達しているときは強度の小さいルールが削除され, 新たなルールに置き換わる．

- ルール交換メカニズム
  一定間隔ごとに個体間で有効なルールを交換させることによって, 個体レベルでは獲得できないルールを獲得させる．交換相手をランダムに選択後, 2 つのエージェント間で強度の大きいルールを授受し, 強度の小さいルールをそのルールと置き換える．また, 交換するルールの強度は初期値にセットする．
- 組織知識メカニズム
  与えられた問題をエージェント全体で効果的に解いた時（解が今までで一番良かった時）の個体知識（ルールの集合）の組み合わせを個体間で共有し, 次回の問題解決時に初期ルールとして利用する．

OCS では, 上記の 4 種類の学習メカニズムを用いて, 以下のアルゴリズムを実行する [5].

1. 組織知識がすでに蓄えられていれば, それを初期ルールとして設定する．
2. ルール生成及び交換メカニズムによって問題を解く．
3. 解が収束後, エージェントのルールを適用．強化学習により各エージェントの行動を評価し, 良い分業作業は組織知識として記憶する．
4. 同じ問題をある設定回数に至るまで 2 に戻って, サイクルを繰り返す．

つまり, 一回の試行では目的関数を改善する, あるいは平均よりも良い目的関数値を出すルールに報酬を与え強化し, ルールを実行することで目的関数を改善していくということを解が収束するまで繰り返すということになる．

4.2 提案手法

4.2.1 エージェントの設計

WEB レイアウトをマルチエージェントシステムで設計するにあたり, WEB レイアウトの構成要素すなわちボックス（レイアウト上の長方形の領域）をエージェントとしてみなした．各エージェントはルール集合と制約条件を有している．ルールは前節で説明した OCS[5] と同様, 強度と条件（IF）部とアクション（THEN）部を持っており, 各エージェントはルールを上限数に至るまで複数個持つことができる．エージェントのもつ制約条件とは, 3.3 節で述べた自分の横

幅・縦幅・x 座標・y 座標の最小値や最大値の条件などのような各エージェントごとに個別に適用される条件のことである．

4.2.2 制約条件の設計

3.3 節で説明した通り, システムはレイアウト全体が満たすべき条件とエージェントが個別に満たすべき条件のふたつを持っている．そのどちらの条件についても, その充足に寄与するルールには高い報酬を与える必要がある．しかし, 「適用することによって制約条件を満たしたらそのルールに報酬を与える」という強化条件は上手く機能しない．なぜなら, 一度のルール適用で条件の評価結果が変わることの方が稀であるからである．例えば, あるエージェントが「横幅が 100 以下でなければならない」という制約を持っていた場合, 「横幅が 100 以上の時に横幅を 10 減少させる」というルールには高い報酬が与えられるべきであるが, 例えばこのルール適用によって横幅が 150 のときに 10 減少したとしても評価結果は変わらないのでこのルールには報酬が与えられない．

実際の最適化において上ようなケースがほとんどであるため, 提案手法では次のような方法をとった．すなわち, 一つの制約条件に対して一つの関数を導入し, 制約を条件 $C(\boldsymbol{s})$ と関数 $f(\boldsymbol{s})$ のペアとした．その条件 $C(\boldsymbol{s})$ と関数 $f(\boldsymbol{s})$ は以下の条件を満たすものとする．なお, $\boldsymbol{s}$ は状況を表し, $\boldsymbol{s} \in \{\boldsymbol{L}, \boldsymbol{a_1}, \cdots, \boldsymbol{a_n}\}$ である．

$$\begin{cases} C(\boldsymbol{s}) = True \Leftrightarrow f(\boldsymbol{s}) = 0 \\ f(\boldsymbol{s}) \geq 0 \end{cases}$$

つまり, 制約条件 $C(\boldsymbol{s})$ を満たしている時, また満たしている時のみ関数 $f(\boldsymbol{s})$ は 0 となる．従って, $f(\boldsymbol{s})$ が小さければ小さいほど条件の充足に近いということなる．これにより, 制約充足問題を最適化問題として解くことができる．

例えば, 先の「横幅が 100 以下でなければならない」というエージェント制約条件は自らの横幅を width として,

$$C(\boldsymbol{a}) = \begin{cases} True & (width \leq 100) \\ False & (otherwise) \end{cases}$$

と表せるが, この制約関数は

$$f(\boldsymbol{a}) = \begin{cases} width - 100 & (width - 100 > 0) \\ 0 & (otherwise) \end{cases}$$

となり, これは上の条件を満たすことがわかる．

4.2.3 ルールの設計

ルールは OCS[1, 5] と同様, 条件（IF 部）とアクション（THEN 部）, 価値（強度）から成る．エージェントは複数のルールを上限数まで持つことができ, 強度の高いルールから優先して実行する．強度は 0 から 1 までの値をとる．

条件はエージェントの置かれている状況（他のボックスと重なっている, 他のボックスと整列している, 等間隔である, 密着している など）を細かく表現する必要がある．条件はルール生成時の状況に適合するものが設定される．提案手法では対象ボックスとレイアウト中の他のボックスとの関係によって決まるいくつかの条件を独立に候補として記述しておき, そのうち生成時に成立している条件を集め条件の部分集合をつくり, これをそのルールの条件として設定する．条件の候補は以下である．

- 自分と重なっているボックスがあるか
- 一定範囲（重心間距離）内にボックスが存在するか
- 横幅/縦幅が一定範囲にあるか

これに加え, 各エージェントのもつエージェント制約条件も条件の候補に加えることとした．そうすることで制約充足に関係するルールが生成されやすくなるようにした．

アクションはルール生成時にアクション候補からランダムに選択される．なお, アクションの自由度を下げることによって生成されるレイアウトの組み合わせ数を大幅に下げることができる．例えば, ボックスどうしが常に整列していることの多い Web デザインという問題においては「縦/横に 1 ピクセル移動する」というような原始的な動きだけをアクションとして設定するよりも,「最近接のボックスと整列する」というアクションを設定したほうが制約を短いステップで充足しやすい．このように, 問題のドメインに応じてアクションを変化させることで, 解候補数を減らし, 解の探索回数を大幅に削減することができる．このような理由から, 提案手法では以下のアクション候補を採用した．

- 待機（停止）
- 上・右・下・左に 10 ピクセル移動する
- 横幅・縦幅を 10 ピクセル増やす/減らす
- 最近接ボックスに整列
- 最も整列している/最近接の ボックスとの距離を指定値にする
- 最も整列しているボックスと横幅/縦幅を揃える

4.2.4 強化学習

4.2.2 節で示した制約関数を利用して, ルールの強化を行った．強化には制約条件の真偽値は用いず, 関数値のみを用いた．すなわち, 制約関数を減少させるルールに高い報酬を与えるため, ルール適用前のレイアウト制約関数と自分（ルールを保持しているエージェント）のエージェント制約関数の値をそれぞれ記憶しておき, ルール適用後のそれらと比較を行う．今回はレイアウト制約関数とエージェント制約関数のどちらかがルール適用前よりも改善しているか, それ以上改善のしようが無い状態（両関数共に 0）の場合, そのルールに報酬を与えることとした.

強化学習アルゴリズムには Profit Sharing[5] を用い, 報酬関数は常に固定値 r とした.

4.2.5 最適化サイクル

提案手法の最適化サイクルは OCS 同様ルール生成・ルール交換・強化学習・組織学習の 4 メカニズムから成る．そのアルゴリズムのフローチャートを図 3 に示した．OCS の終了条件が一定回数繰り返すことだったのに対し, 提案手法ではすべての絶対制約を満たすことなっている．従って, ループ終了時には絶対制約が満たされていることが保証される．繰り返しを行う限り, 制約関数を減少させるルールに高い報酬が与えられ, そのルールが実行され, 制約充足に近づいていく．考慮制約については, 終了条件には含まれないので満足されずに終了されるケースはあるが, 制約関数が減少するように強化学習は行われるので充足に近い状態に向かって変化していく.

OCS[1, 5] においてはルール交換はすべてのエージェント間で行われるが, 提案手法ではルール交換は同じ条件を持つエージェント間のみで行うこととした. つまり, 違う条件を持っていればその充足のため持つべきルールも変わるため, あるエージェント内で高い強度を持つルールが他のエージェントでも高い強度をもつとは限らないのである．また, 強度の弱いルールが交換・伝搬されるのを防ぐため強度が指定した基準を超えていないルールを交換しないものとした.

提案手法では強化学習メカニズムの後にルールを除去するフェーズを設けた. ルール除去フェーズでは, 強化学習によって負の報酬が与えられ強度が下がり, 一定値以下になったもの（0 に近いもの）を除去する．ルール除去フェーズがないと, ルール生成メカニズムで強度の低いルールが現在の状況に適合してしまってルールが新しく生成されない場合が生じる．ルール生成時にエージェントの

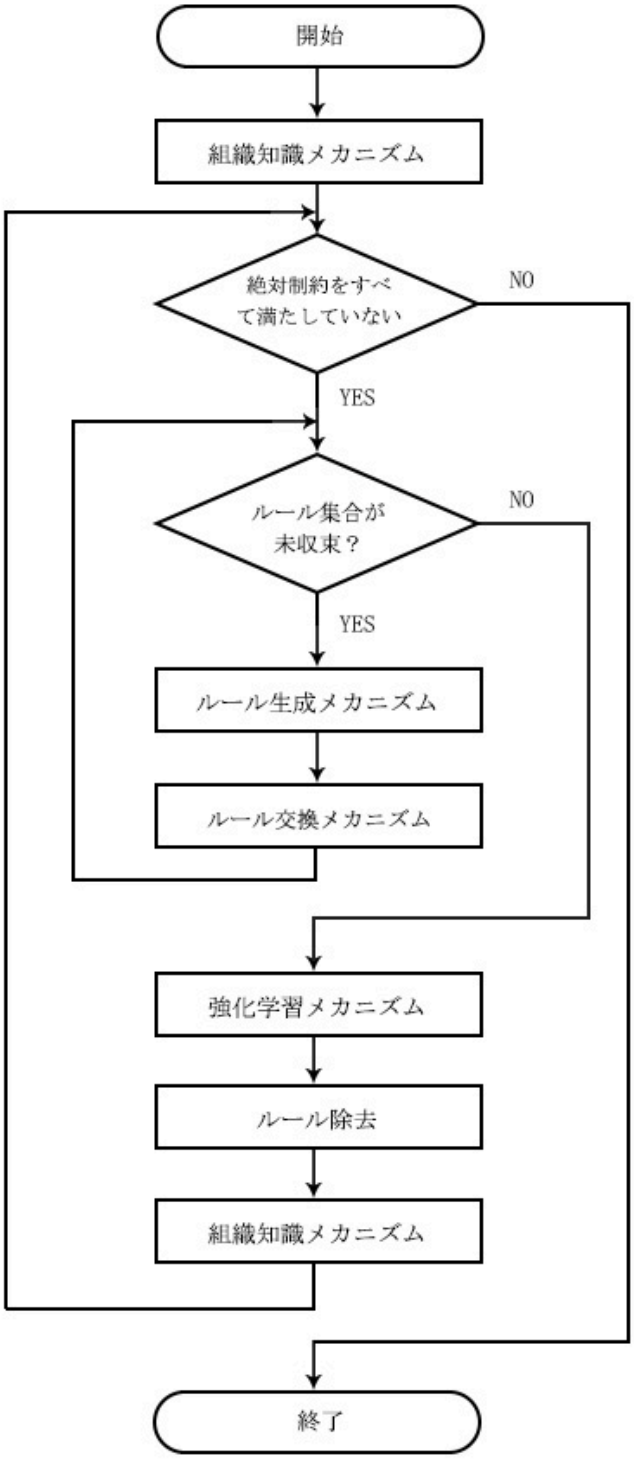

図 3: 提案手法フローチャート

保持するルールが一定数以上になると強度の低いものから削除されるが, 積極的に除去しなければ先に述べた現象が生じるのでルール除去フェーズを導入した．強化学習フェーズにおいて, 強度が最も高いルールが適用ルールとして選択されるが, 選択されたルールの強度が著しく低い場合は適用を避け, 強化学習フェーズを脱する．その後, そのルールが除去され, 新たなルールが生成される．

第5章　実装と評価

5.1　ソフトウェア仕様

5.1.1　ユースケース

本研究で作成したソフトウェアは, クライアント（デザイナ本人でも良い）から要望を受けてWebページを作成するデザイナを対象ユーザとしている．冒頭でも述べた通り, デザインの工程のひとつであるワイヤフレームの作成の補助を目的としている．本ソフトウェアのユースケース図を図4に示した．デザイ

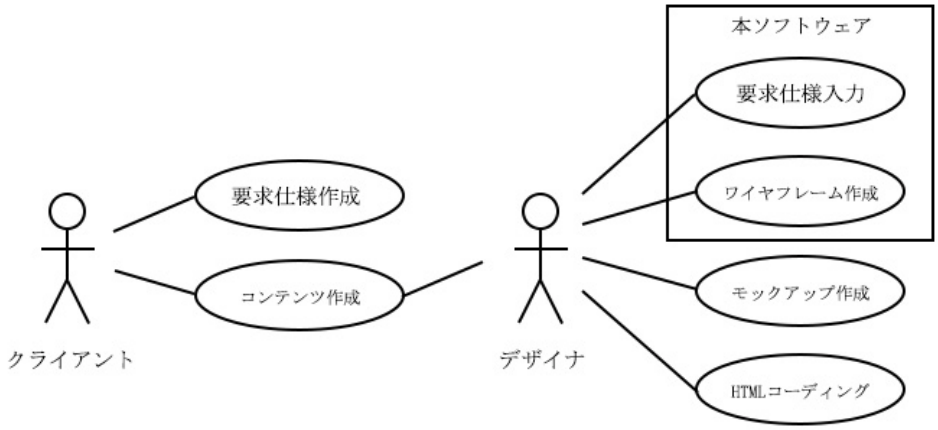

図4: ユースケース図

ナはクライアントの持っている要求仕様をヒアリングし, それをシステムに解釈可能な形に変換し, ユーザ・インタフェースを通じて入力を行う．最適化を何度か行い, 入力を修正し, ワイヤフレームを得る．その後, デザイナは生成されたワイヤフレームを利用してWebページのモックアップを作成することができる．なお, ページ上の画像やテキストは, デザイナかクライアントが予め作成しておくことを想定している．

5.1.2 外部仕様

本ソフトウェアでは, 3.4 節で示した通りレイアウト（ボックスの集合）はその中にレイアウトをとることで階層構造をとることができる．はじめにユーザは図 5 に示したユーザインタフェースを用いてレイアウトの階層構造を指定する．階層構造を指定する際に, レイアウト内のボックスに X・Y・横幅・縦幅の

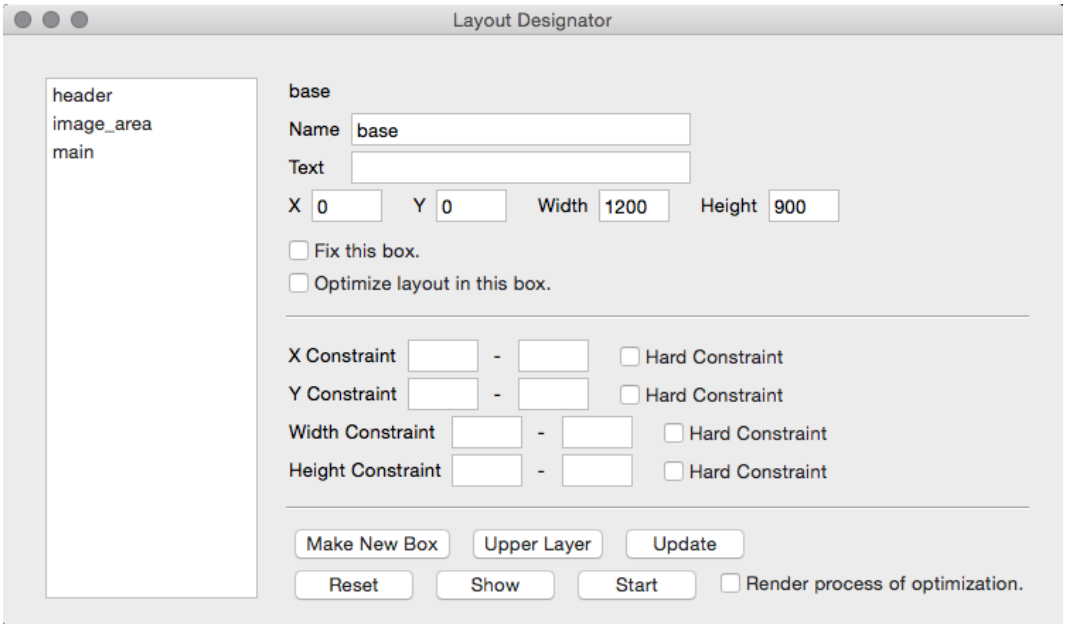

図 5: インターフェース

初期値・名前・ボックス内のテキストと, エージェント制約（ここでは X・Y・横幅・縦幅の領域条件）を指定することができる．その他, 各レイアウトに対して最適化を行うかどうか, 最適化時に途中経過を描画するかどうかなどの設定のほか, 設定段階でのレイアウト状況の表示を行うことができる．

インターフェース上 Start ボタンを押すと, 別ウィンドウにおいて最適化が始まり, 終了すれば結果が表示される．ユーザは何回か最適化を行い, その結果を見て制約を変更していく．その過程で配置を変えたくないボックスが出現した場合, ユーザはそのボックスの位置を固定して最適化を行うことができる．これは 1 章で述べたマルチエージェントシステムの利点ロバスト性 [4] による．望ましい（ユーザが使用に値すると判断した）レイアウトが出力されれば, ユーザはそれをワイヤフレームの候補として考慮に入れる．満足いく候補が得られれば, ユーザはアプリケーションを終了する．

5.2 動作確認と評価

5.2.1 収束状況の比較

提案手法および提案モデルの有効性を検証するため, 実装したソフトウェアを用いて実験を行った．はじめに, 本節では収束の状況を解説するための実験を行う．簡単のためボックスは 2 個とし, 以下の制約を与えた.

まず, レイアウト制約を以下のように設定した. ただし, $OverlappedArea(\boldsymbol{a_i}, \boldsymbol{a_j})$ はふたつのエージェント $\boldsymbol{a_i}, \boldsymbol{a_j}$ の重なり面積を返す関数, $Aligned(\boldsymbol{a_i}, \boldsymbol{a_j})$ はふたつのエージェント $\boldsymbol{a_i}, \boldsymbol{a_j}$ が整列している（x または y が一致）とき $True$ を返す関数, x_i は $\boldsymbol{a_i}$ の x 座標, x_{ia} は $\boldsymbol{a_i}$ に最も整列している（x 又は y が全エージェントの中で最も $\boldsymbol{a_i}$ に近い）エージェントの x 座標を表す.

$$C_L(\boldsymbol{L}) = C_{L1}(\boldsymbol{L}) \wedge C_{L2}(\boldsymbol{L})$$
$$F_L(\boldsymbol{L}) = f_{L1}(\boldsymbol{L}) + f_{L2}(\boldsymbol{L})$$

$$C_{L1}(\boldsymbol{L}) = \begin{cases} True & (\forall \boldsymbol{a_i}, \boldsymbol{a_j} \in \boldsymbol{A},\ OverlappedArea(\boldsymbol{a_i}, \boldsymbol{a_j}) = 0) \\ False & (otherwise) \end{cases}$$

$$C_{L2}(\boldsymbol{L}) = \begin{cases} True & (\forall \boldsymbol{a_i} \in \boldsymbol{A}, \exists \boldsymbol{a_j} \in \boldsymbol{A},\ Aligned(\boldsymbol{a_i}, \boldsymbol{a_j}) = True) \\ False & (otherwise) \end{cases}$$

$$f_{L1}(\boldsymbol{L}) = \sum_{i=1}^{n} \sum_{j=i}^{n} \sqrt{OverlappedArea(\boldsymbol{a_i}, \boldsymbol{a_j})}$$

$$f_{L2}(\boldsymbol{L}) = \sum_{i=1}^{n} \min(|x_i - x_{ia}|, |y_i - y_{ia}|)$$

つまり, $C_{L1}(\boldsymbol{L})$ はエージェントどうしが重ならない条件, $C_{L2}(\boldsymbol{L})$ は各エージェントが他のどれかのエージェントと整列している条件を表す．次にエージェント制約であるが, $\boldsymbol{a_1}$ に関しては以下のような制約条件と制約関数を与えた.

$$C_{a_1}(\boldsymbol{a_1}) = C_{11}(\boldsymbol{a_1}) \wedge C_{12}(\boldsymbol{a_1})$$
$$F_{a_1}(\boldsymbol{a_1}) = f_{11}(\boldsymbol{a_1}) + f_{12}(\boldsymbol{a_1})$$

$$C_{11}(\boldsymbol{a_1}) = \begin{cases} True & (340 < width < 360) \\ False & (otherwise) \end{cases}$$

$$C_{12}(\boldsymbol{a_1}) = \begin{cases} True & (230 < height < 260) \\ False & (otherwise) \end{cases}$$

$$f_{11}(\boldsymbol{a_1}) = \begin{cases} width - 340 & (width - 340 > 0) \\ 320 - width & (width - 320 < 0) \\ 0 & (otherwise) \end{cases}$$

$$f_{12}(\boldsymbol{a_1}) = \begin{cases} width - 260 & (width - 260 > 0) \\ 230 - width & (width - 230 < 0) \\ 0 & (otherwise) \end{cases}$$

つまり, $\boldsymbol{a_1}$ に対しては横幅と縦幅に制約を与えた．同様にして, $\boldsymbol{a_2}$ に対しては $210 < width < 230, 110 < height < 130$ の制約を与えた.

また, OCS の設定はエージェントの最大ルール保持数が 20, 交換対象のルール強度は 0.9 以上, 報酬関数は固定値 0.5 とし, ルール除去およびスキップの対象強度の値は 0.01 未満とした．アクションおよび条件は 4.2.3 節に示したものを利用した.

以上の設定で最適化を行った結果, 図 6 のグラフに表される収束結果を得た．グラフは 5 回の最適化のうち代表的なものを選んだものである．また, 5 回の最適化の各試行回数および収束までの時間を表 1 に示した． 次に, 比較のためランダムサーチによって解を探索した場合の収束状況のグラフを図 7 に示した．5 回中 5 回とも試行回数が 10000 回を超え, 収束はしなかった．比較のため図 6 のグラフと同じ 66 回までの状況をグラフに表した.

表 1: 試行回数

回数	1	2	3	4	5	平均値
試行回数	113	99	58	66	94	86
収束までの時間 (s)	0.54	0.47	0.28	0.32	0.45	0.41

5.2.2 収束結果の考察

OCS ではルール生成の際アクションがランダムに生成されるため, 制約関数値のジャンプ・振動が見られる.（たとえば, 「最近接のボックスに縦幅を合わせる」アクションが適用されると縦幅が制約で指定された領域から外れ, 制約関数の値が跳ね上がる.）しかし, そのようなルールには負の報酬が与えられ除去されるため, 振動は繰り返し起こらない．今回は報酬値を 0.5 と高めに設定したため, 制約関数値を上昇させる悪いルールは一度の適用で強度が 0 に設定さ

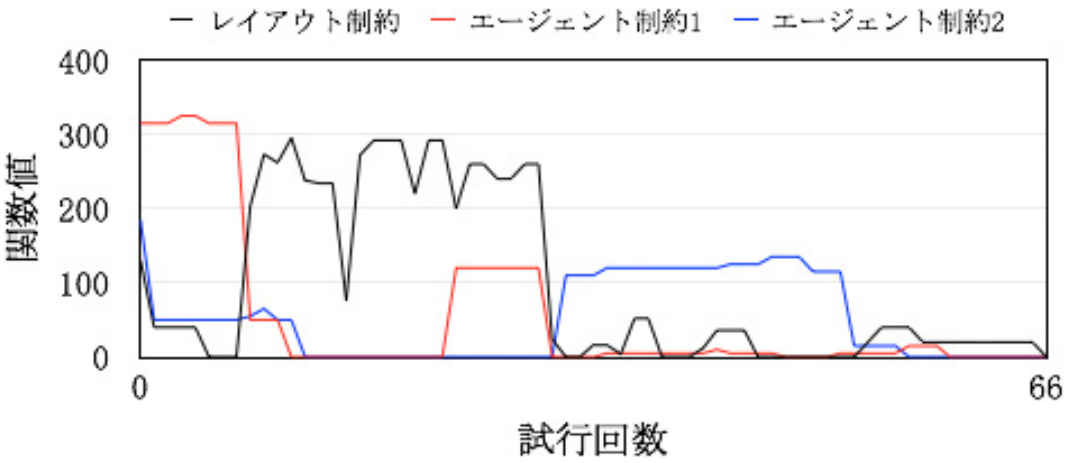

図 6: 提案手法の収束過程

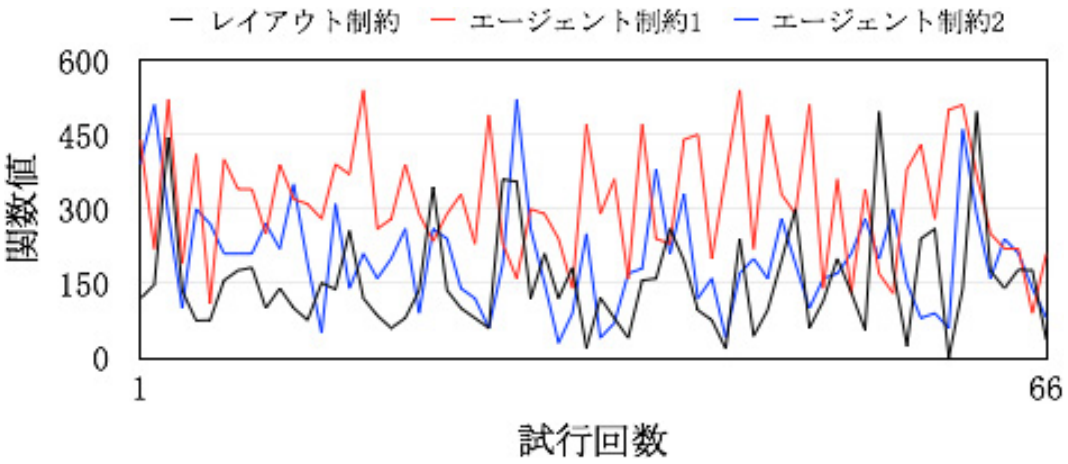

図 7: ランダムサーチの実行過程

れ, 即座に除去される．その効果によりルールの回転が早くなり, 良いルールの探索が早く行われた．それにより関数値がジャンプしても比較的早く回復することができた．また, 良いルールは強化が行われ残り続けるので, 関数値が一度 0 になればしばらく 0 の値を保っていることがグラフからも読み取れる．

ランダムサーチでは x 座標, y 座標, 横幅, 縦幅を 10 ピクセル単位でランダムに変化させ探索を行った．領域は 750×270 なので, 可能な位置は $76 \times 28 = 2128$ 通り, そのそれぞれに対して横幅と縦幅がそれぞれ平均して 38 通り, 14 通り取れるため, サイズの組み合わせ数は 532 通りである．従って, ひとつのボックスがとれる可能な組み合わせ数は $2128 \times 532 = 1132096$ 通り, エージェントがふたつの場合は $1132096^2 = 1281641353216$ 通りの組み合わせが考えられる．これをランダムで行おうとすれば当然図 7 のグラフのように関数値は大きく振動し, 10000 回程度では収束しない．ランダムサーチの挙動と比較した場合, OCS では 100 回程度で収束していることから OCS が本問題に対し有効だということが確認できる．

5.2.3 レイアウト結果の例

収束結果の解説のため前節ではボックスを 2 個としたが, 本節では提案手法がより複雑なレイアウト作成に対応できることを示すため 2 カラムの Web サイトのワイヤフレームを作成する実験を行った．ボックスの階層構造を活かし, 外側のボックスから順に最適化を行っている．ページ内計 5 箇所のレイアウトで最適化を行った.

ほとんどのボックスに前節と同様の縦幅と横幅の制約を課した．また, ガイドラインから定められる制約として一部のボックスに中央寄せの制約（左右の余白の差を 0 にする）, 親ボックス上端からの余白のサイズの制約を課した．レイアウト制約は前節と同様の $C_L(\boldsymbol{L})$ に加え, レイアウト内のボックスの横幅・縦幅を統一するという制約, 左右の余白を一定値以下にするという制約を課した．レイアウトは階層ごとに存在するため, レイアウト制約は複数存在する．OCS の設定およびアクション・条件は前節と同じである．

初期位置を図 8 のように設定した．階層構造をわかりやすくするため重なりを解いたが、結果は初期位置に依存しないためボックスが重なっていても良い。

図 8 の初期状態から始め, 収束に至ったレイアウト結果の例を図 9 に示した．ボックス内の文字はボックス名であり, 最適化そのものには影響を与えない．

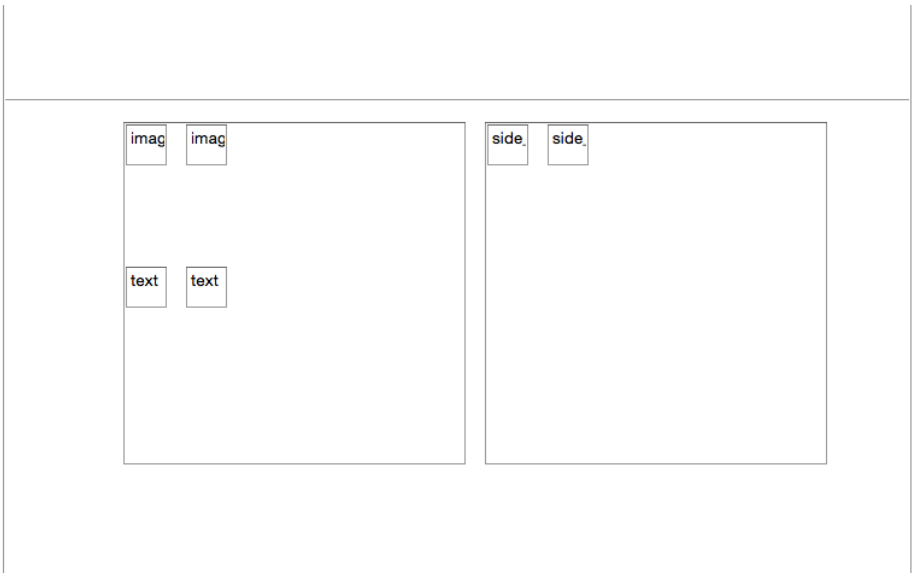

図 8: 最適化の初期レイアウト

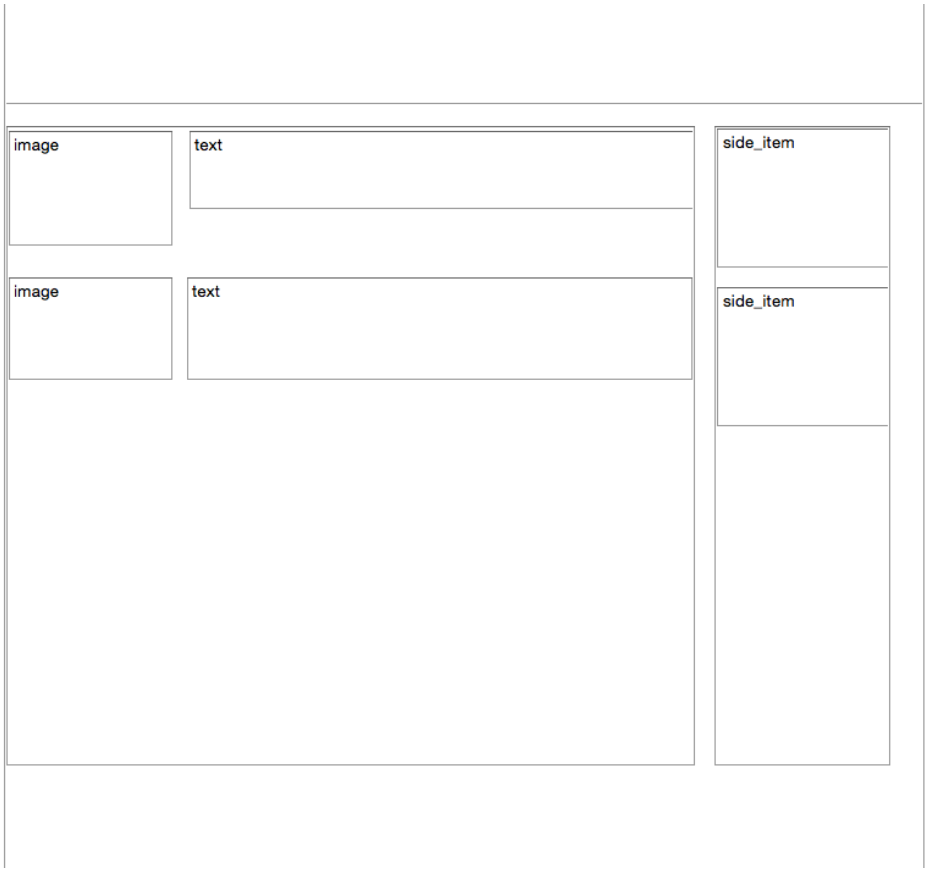

図 9: 最適化結果

5.2.4　レイアウト結果の考察

前節の実験では, 構成要素の横幅と縦幅とレイアウトの原則から導かれるいくつか制約を課しただけで一般的な 2 カラム Web サイトに近い構造を生成した. 実際のデザインではコンテンツの縦幅や横幅はすでに決まっている事が多いので, ユーザはいくつかのレイアウト原則を指定するだけでよい.

何回か最適化を行うとサイドバー（右側の縦長のボックス）とメイン部（左側のボックス）が反転したパターンや, メイン部の image ボックスと text ボックスが反転したパターンなどが得られたが, 大まかな構造に変化はなかった. すなわち, レイアウト候補が約数通りにまで絞られるため, デザイナの作業負担を大幅に削減できると考えられる.

第 6 章　おわりに

本稿の冒頭で, Web サイトデザイン一般, 特に最適化問題的な側面が強いレイアウト問題をコンピュータにより自動化することには意義があることを述べた. しかし, それをコンピュータに行わせるためには次の 2 つの課題があった.

1. Web ページレイアウト問題のためのモデルの欠如
   Web ページレイアウト問題は他のレイアウト問題とは異なり明確な目的関数が定義できないほか, 最適化できる形で要求仕様を記述する方法が不明であったため, 問題をどのように扱うべきかというモデルがなかった.
2. 現行の最適化・自動化手法の問題
   現在 Web ページデザインには対話的遺伝的アルゴリズム（IGA）が使われる例が多いが, このアルゴリズムにはユーザの要望が直接的に反映されにくく, 人間がアルゴリズムに関与するため探索空間の広さのわりに試行回数が限られているという問題点があった.

そこで, 本研究では Web デザインにおけるワイヤフレーム作成（レイアウト問題）に焦点を当て, ユーザの嗜好や目的の成果物の性質を制約という形で記述させ, 最適化問題として解く制約最適化アプローチを提案した. 本研究の貢献は以下の 2 点である.

1. Web ページレイアウト問題のモデル提唱
   デザインガイドラインとユーザの両方からレイアウトとエージェントに制約を課し, 結果に基いて制約を変更していくことで目的のレイアウト候補

を得るという, Web ページレイアウト問題を制約充足問題として解くためのモデルを提唱した.

2. OCS を用いたレイアウト制約充足問題解法の提唱
   制約条件が満たされたときに 0 になる制約関数を導入し, それに基いて報酬を決定するという手法で, OCS を制約充足問題に対応できるよう拡張した．最適化アプローチのため試行回数を増やす事ができ, 制約を直接記述させることでユーザの要望が反映されやすい．また, 提案手法を用いて少ない試行回数で意味のあるレイアウト結果が得られることを示した.

デザインの全工程を自動化する上での今後の課題としては, デザインにおける目的関数の最大化のためのモデルと手法の提案が残されている．3.1 節ではデザインは制約下で目的を最大限達成するための設計および実装であると述べたが, 今回は制約充足の部分にフォーカスし, 本研究の最適化は立ち入らなかった. その最適化を可能にするためにはレイアウトの美しさや機能性の定量化が必要であり, 新たな問題が浮かび上がってくる．また, コンテンツ生成やモックアップ作成などのワイヤフレーム作成以外の作業に対する手法の提案も今後の課題である.

本研究の応用領域としては, 施設レイアウト問題のような Web ページレイアウト以外のレイアウト制約充足問題が考えられる．本研究の主旨とは逸れるのでそのドメインにおける既存の手法との詳細な比較はしていないが, アクションと条件を入れ替えることで様々な問題に対応することが可能である.

謝辞

本研究を行うにあたり, 熱心なご指導, ご助言を賜りました石田亨教授に厚く御礼申し上げます．また, 日頃より多くのご助言をいただきました松原繁夫 准教授に感謝いたします．そして, 時間を惜しまず多大なご協力をいただきました後藤真介様をはじめ, 日頃から多くのご助言とご協力をいただきました石田研究室の皆様に心より感謝いたします.

参考文献

[1] 高玉 圭樹, 中須賀 真一, 寺野 隆雄, 組織学習エージェントによるプリント基板問題への接近, 電子情報通信学会論文誌, Vol.J81-D-I, No.5, pp.514-522 (May, 1998).

[2] A.Oliver, N.Monmarche , G.Venturini, INTERACTIVE DESIGN OF WEB SITES WITH A GENETIC ALGORITHM, IADIS International Conference WWW/Internet 2002, pp.355-362 (2002).

[3] Juan C. Quiroz, Sushil J. Louis, Anil Shankar, Sergiu M. Dascalu Interactive Genetic Algorithms for User Interface Design, IEEE Congress on Evolutionary Computation, pp.1366-1373 (2007).

[4] 高玉 圭樹, マルチエージェントに基づくカーゴレイアウトシステム, 人工知能学会誌, Vol.21, No.1, pp.39-44 (Jan, 2001).

[5] 高玉 圭樹, マルチエージェント学習, コロナ社 (2004).

[6] Jiming Liu, Han Jing, Y.Y. Tang, Multi-agent oriented constraint satisfaction, Artificial Intelligence 136, pp.101-144 (2002).

[7] Hamed Tarkesh, Arezoo Atighehchian, Ali S. Nookabadi, Facility layout design using virtual multi-agent system, J Intell Manuf, pp.347-357 (May 2008).

[8] Frederick P. Brooks, Jr. / 松田昇一, 小沼千絵 訳 デザインのためのデザイン, ピアソン桐原, p.4 (2010).

付録

A.1　内部仕様概要

実装は Python を用いて行った．ソフトウェアは次節にコードを付した 7 つのモジュールから成る．クラス図を図 A.1 に示した．

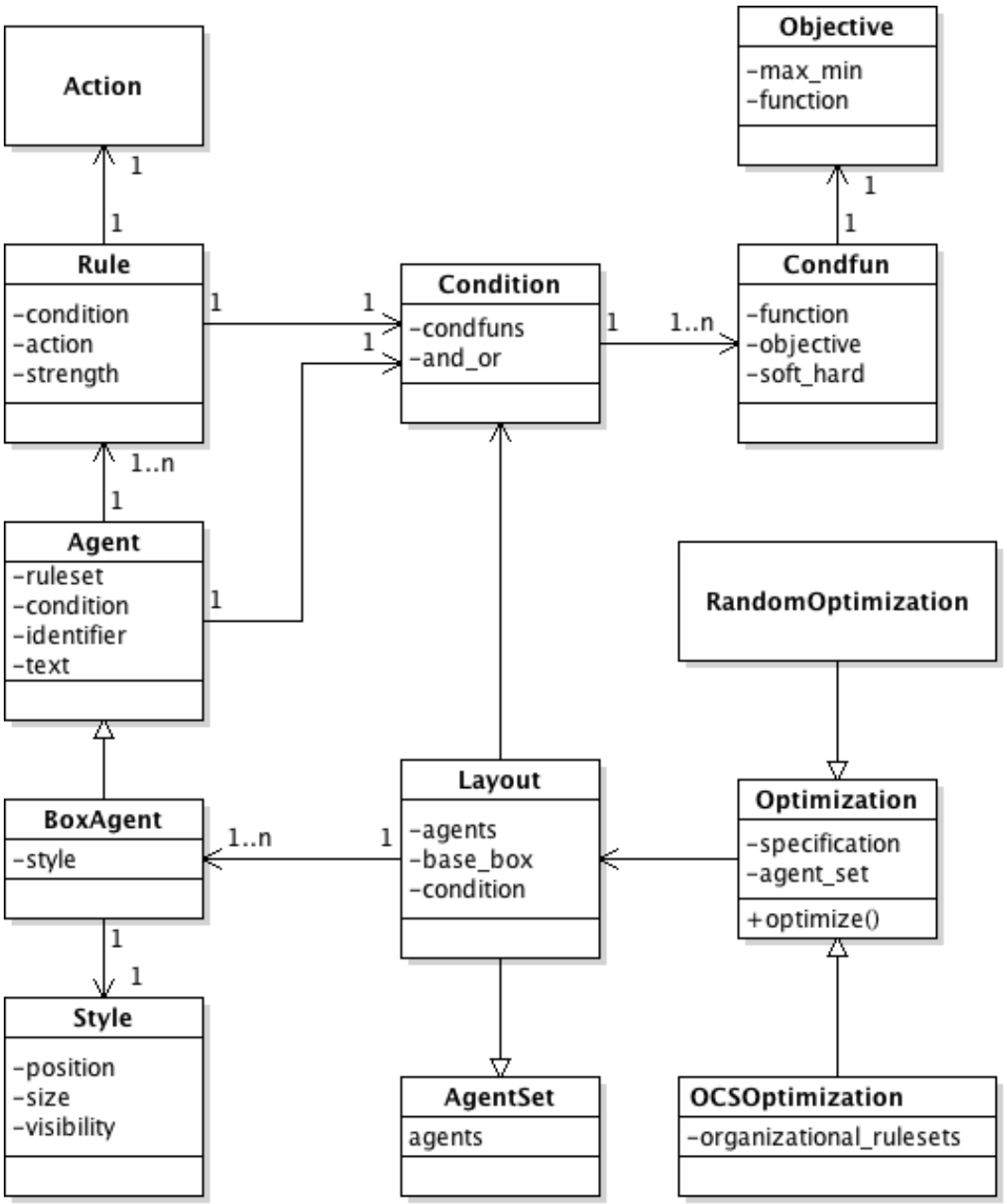

図 A.1: クラス図

各クラスの機能および概要は以下のとおりである．

- Agent クラス
  エージェントを定義するクラスで, 今回の最適化で使われたボックスは Agent クラスのサブクラス BoxAgent クラスによって定義される．
- Agent Set クラス
  エージェントの集合を定義するクラスで, レイアウトは Agent Set クラスのサブクラス Layout クラスによって定義される．
- Optimization クラス
  最適化を行うクラスで, インスタンスメソッド optimize() で最適化を開始する．インスタンス変数に最適化中のレイアウトをもつ．ランダムサーチおよび OCS による最適化はこのクラスのサブクラスとして定義される．
- Rule クラス
  エージェントのもつルールを定義する．
- Condition クラス
  ルールの IF 部に設定されるほか, レイアウトやエージェントの制約条件としても利用される．ひとつの Condition は複数の条件 Condfun からなり, それを組み合わせて評価が行われる．
- Condfun クラス
  条件 Condfun は制約関数と制約条件のペアからなる．各 Condfun が考慮制約か絶対制約かというオプションをインスタンス変数として有する．
- Objective クラス
  Condfun がもつ制約関数を定義する．
- Action クラス ··· ルールの THEN 部に設定されるアクションを, このクラスのクラスメソッドとして定義する．

A.2 ソースコード

A.2.1 main.py

```
#-*- coding: utf-8 -*-
# imports - - - - - - -
import wx
import time
from agent import *
from specification import *
from optimization import *
from layout import *

WINDOW_SIZE = [1200,900]
MAIN_WINDOW_SIZE = (750,440)
MAIN_PADDING = (30,30)

g_main_frame = None
g_widgets = {}

class OptimizationFrame(wx.Frame):

    def __init__(self, parent=None, id=-1, title=None, *args, **
        kwargs):
        wx.Frame.__init__(self, parent, id, title,*args, **kwargs)

        self.SetBackgroundColour("#ffffff")
        self.SetSize(WINDOW_SIZE)

        self.base_panel = wx.Panel(self, wx.ID_ANY)

        self.timer = wx.Timer(self)
        self.Bind(wx.EVT_TIMER, self.on_timer)
        self.timer.Start(50)

    # the method which runs at regular interval
    def on_timer(self, event):
        pass
```

```
34          #self.move()
35
36      def move(self):
37          present_x, present_y = self.child_panel1.GetPosition()
38
39          self.child_panel1.SetPosition((present_x + 1, present_y + 1)
                )
40          self.child_panel1.Update()
41
42  def widget_factory(widget_class,parent,widget_id,*args,**kwargs):
43      widget = widget_class(parent,widget_id,*args,**kwargs)
44      g_widgets[widget_id] = widget
45      return widget
46
47  def get_widget_by_id(widget_id):
48      return g_widgets.get(widget_id, None)
49
50  def get_agent_list(box):
51      if box.inner_layout:
52          return [agent.identifier for agent in box.inner_layout.
                agents]
53      else:
54          return []
55
56  def get_map(box):
57      if box.parent_layout:
58          return get_map(box.parent_layout.base_box) + "␣>␣" + box.
                identifier
59      else:
60          return box.identifier
61
62
63  def get_right_position(basic_object, margin, y_adjustment=0):
64      basic_x, basic_y = basic_object.Position
65      basic_width, basic_height = basic_object.Size
66      return [basic_x + basic_width + margin, basic_y + y_adjustment]
67
```

```
68 def get_bottom_position(basic_object, margin, x_adjustment=0):
69     basic_x, basic_y = basic_object.Position
70     basic_width, basic_height = basic_object.Size
71     return [basic_x + x_adjustment, basic_y + basic_height + margin]
72 
73 class MainFrame(wx.Frame):
74 
75     def __init__(self, current_box, parent=None, id=-1, title=None,
           *args, **kwargs):
76         wx.Frame.__init__(self, parent, id, title, *args, **kwargs)
77 
78         self.base_panel = wx.Panel(self, wx.ID_ANY)
79         self.set_current_box(current_box)
80         self.set_base_box(current_box)
81         self.refresh(self.current_box)
82 
83     def set_current_box(self, current_box):
84         self.current_box = current_box
85 
86     def set_main_panel(self, main_panel):
87         self.main_panel = main_panel
88 
89     def set_base_box(self, base_box):
90         self.base_box = base_box
91 
92     def get_base_box(self):
93         return self.base_box
94 
95     def render_box_options(self, target_box):
96         ADJUST= 2
97         main_panel = self.main_panel
98 
99         label_map = widget_factory(wx.StaticText, main_panel, 0,
              get_map(target_box), pos=get_right_position(
              get_widget_by_id(15), 20))
100 
```

```
101         label_name = widget_factory(wx.StaticText, main_panel, 6, "
                Name", pos=get_bottom_position(label_map, 10))
102         text_name = widget_factory(wx.TextCtrl, main_panel, 1,
                target_box.identifier, pos=get_right_position(label_name
                , 5, -ADJUST), size=(240, 23))
103
104         label_text = widget_factory(wx.StaticText, main_panel, 40, "
                Text", pos=get_bottom_position(label_name, 10))
105         text_text = widget_factory(wx.TextCtrl, main_panel, 41,
                target_box.text, pos=get_right_position(label_text, 16,
                -ADJUST), size=(240, 23))
106
107         STYLE_TEXT_SIZE = (50,23)
108         label_x = widget_factory(wx.StaticText, main_panel, 7, "X",
                pos=get_bottom_position(label_text, 10))
109         text_x = widget_factory(wx.TextCtrl, main_panel, 2, str(
                target_box.get_x()) , pos=get_right_position(label_x, 5,
                 -ADJUST), size=STYLE_TEXT_SIZE)
110         text_x.SetMaxLength(4)
111
112         label_y = widget_factory(wx.StaticText, main_panel, 8, "Y",
                pos=get_right_position(text_x, 20, ADJUST))
113         text_y = widget_factory(wx.TextCtrl, main_panel, 3, str(
                target_box.get_y()), pos=get_right_position(label_y, 5,
                -ADJUST), size=STYLE_TEXT_SIZE)
114         text_y.SetMaxLength(4)
115
116         label_width = widget_factory(wx.StaticText, main_panel, 9, "
                Width", pos=get_right_position(text_y, 20, ADJUST))
117         text_width = widget_factory(wx.TextCtrl, main_panel, 4, str(
                target_box.get_width()), pos=get_right_position(
                label_width, 5, -ADJUST), size=STYLE_TEXT_SIZE)
118         text_width.SetMaxLength(4)
119
120         label_height = widget_factory(wx.StaticText, main_panel, 10,
                 "Height", pos=get_right_position(text_width, 20, ADJUST
                ))
```

```
121         text_height = widget_factory(wx.TextCtrl, main_panel, 5, str
                (target_box.get_height()), pos=get_right_position(
                label_height, 5, -ADJUST), size=STYLE_TEXT_SIZE)
122         text_height.SetMaxLength(4)
123
124         check_fixedness = widget_factory( wx.CheckBox, main_panel,
                44, "Fix this box.", pos=get_bottom_position(label_x,
                15))
125         check_fixedness.SetValue(target_box.fixedness)
126
127         if target_box.inner_layout:
128             check_optimization = widget_factory( wx.CheckBox,
                    main_panel, 15, "Optimize layout in this box.", pos=
                    get_bottom_position(check_fixedness, 5))
129             check_optimization.SetValue(target_box.inner_layout.
                    optimization_needed)
130
131     def render_conditions(self, target_box):
132         ADJUST= 2
133         main_panel = self.main_panel
134
135         def get_func_dict_value(box, name, key):
136             function = None
137             for condfun in box.condition.condfuns:
138                 if condfun.function.__name__ == name:
139                     function = condfun.function
140                     break
141
142             if function:
143                 return function.func_dict[key]
144             else:
145                 return ""
146
147         STYLE_TEXT_SIZE = (50,23)
148         label_x_from = widget_factory(wx.StaticText, main_panel, 18,
                "X Constraint", pos=get_bottom_position(
                get_widget_by_id(16), 17))
```

```
149         text_x_from = widget_factory(wx.TextCtrl, main_panel, 19,
                str(get_func_dict_value(target_box, '_x_constraint', '
                lower')) , pos=get_right_position(label_x_from, 5, -
                ADJUST), size=STYLE_TEXT_SIZE)
150         label_x_to = widget_factory(wx.StaticText, main_panel, 20, "
                ␣-␣", pos=get_right_position(text_x_from, 5, ADJUST))
151         text_x_to = widget_factory(wx.TextCtrl, main_panel, 21, str(
                get_func_dict_value(target_box, '_x_constraint', 'upper'
                )) , pos=get_right_position(label_x_to, 5, -ADJUST),
                size=STYLE_TEXT_SIZE)
152         check_x_constraint = widget_factory(wx.CheckBox, main_panel,
                 34, "Hard␣Constraint", pos=get_right_position(text_x_to
                , 20, 3))
153         check_x_constraint.SetValue(bool(get_func_dict_value(
                target_box, '_x_constraint', 'soft_hard')))
154         text_x_from.SetMaxLength(4)
155         text_x_to.SetMaxLength(4)
156
157         label_y_from = widget_factory(wx.StaticText, main_panel, 22,
                 "Y␣Constraint", pos=get_bottom_position(label_x_from,
                10))
158         text_y_from = widget_factory(wx.TextCtrl, main_panel, 23,
                str(get_func_dict_value(target_box, '_y_constraint', '
                lower')) , pos=get_right_position(label_y_from, 5, -
                ADJUST), size=STYLE_TEXT_SIZE)
159         label_y_to = widget_factory(wx.StaticText, main_panel, 24, "
                ␣-␣", pos=get_right_position(text_y_from, 5, ADJUST))
160         text_y_to = widget_factory(wx.TextCtrl, main_panel, 25, str(
                get_func_dict_value(target_box, '_y_constraint', 'upper'
                )) , pos=get_right_position(label_y_to, 5, -ADJUST),
                size=STYLE_TEXT_SIZE)
161         check_y_constraint = widget_factory(wx.CheckBox, main_panel,
                 35, "Hard␣Constraint", pos=get_right_position(text_y_to
                , 20, 3))
162         check_y_constraint.SetValue(bool(get_func_dict_value(
                target_box, '_y_constraint', 'soft_hard')))
163         text_y_from.SetMaxLength(4)
```

```
164         text_y_to.SetMaxLength(4)
165
166         label_width_from = widget_factory(wx.StaticText, main_panel,
                26, "Width Constraint", pos=get_bottom_position(
                label_y_from, 10))
167         text_width_from = widget_factory(wx.TextCtrl, main_panel,
                27, str(get_func_dict_value(target_box, '
                _width_constraint', 'lower')) , pos=get_right_position(
                label_width_from, 9, -ADJUST), size=STYLE_TEXT_SIZE)
168         label_width_to = widget_factory(wx.StaticText, main_panel,
                28, " - ", pos=get_right_position(text_width_from, 5,
                ADJUST))
169         text_width_to = widget_factory(wx.TextCtrl, main_panel, 29,
                str(get_func_dict_value(target_box, '_width_constraint',
                'upper')) , pos=get_right_position(label_width_to, 5, -
                ADJUST), size=STYLE_TEXT_SIZE)
170         check_width_constraint = widget_factory(wx.CheckBox,
                main_panel, 36, "Hard Constraint", pos=
                get_right_position(text_width_to, 20, 3))
171         check_width_constraint.SetValue(bool(get_func_dict_value(
                target_box, '_width_constraint', 'soft_hard')))
172         text_width_from.SetMaxLength(4)
173         text_width_to.SetMaxLength(4)
174
175         label_height_from = widget_factory(wx.StaticText, main_panel
                , 30, "Height Constraint", pos=get_bottom_position(
                label_width_from, 10))
176         text_height_from = widget_factory(wx.TextCtrl, main_panel,
                31, str(get_func_dict_value(target_box, '
                _height_constraint', 'lower')) , pos=get_right_position(
                label_height_from, 5, -ADJUST), size=STYLE_TEXT_SIZE)
177         label_height_to = widget_factory(wx.StaticText, main_panel,
                32, " - ", pos=get_right_position(text_height_from, 5,
                ADJUST))
178         text_height_to = widget_factory(wx.TextCtrl, main_panel, 33,
                 str(get_func_dict_value(target_box, '_height_constraint
                ', 'upper')) , pos=get_right_position(label_height_to,
```

```
            5, -ADJUST), size=STYLE_TEXT_SIZE)
179         check_height_constraint = widget_factory(wx.CheckBox,
                main_panel, 37, "Hard Constraint", pos=
                get_right_position(text_height_to, 20, 3))
180         check_height_constraint.SetValue(bool(get_func_dict_value(
                target_box, '_height_constraint', 'soft_hard')))
181         text_height_from.SetMaxLength(4)
182         text_height_to.SetMaxLength(4)
183
184
185     def render_buttons(self):
186         main_panel = self.main_panel
187
188         make_button = widget_factory(wx.Button, main_panel, 11, "
                Make New Box", pos=get_bottom_position(get_widget_by_id
                (17), 10))
189         make_button.Bind(wx.EVT_BUTTON, click_make_button)
190
191         upper_button = widget_factory(wx.Button, main_panel, 12, "
                Upper Layer", pos=get_right_position(make_button, 10,
                -4))
192         upper_button.Bind(wx.EVT_BUTTON, click_upper_button)
193
194         update_button = widget_factory(wx.Button, main_panel, 13, "
                Update", pos=get_right_position(upper_button, 10, -4))
195         update_button.Bind(wx.EVT_BUTTON, click_update_button)
196
197         reset_button = widget_factory(wx.Button, main_panel, 42, "
                Reset", pos=get_bottom_position(make_button, 5, -6))
198         reset_button.Bind(wx.EVT_BUTTON, click_reset_button)
199
200         show_button = widget_factory(wx.Button, main_panel, 38, "
                Show", pos=get_right_position(reset_button, 10, -4))
201         show_button.Bind(wx.EVT_BUTTON, click_show_button)
202
203         start_button = widget_factory(wx.Button, main_panel, 14, "
                Start", pos=get_right_position(show_button, 10, -4))
```

```
204         start_button.Bind(wx.EVT_BUTTON, click_start_button)
205
206     def refresh(self, target_box):
207
208         self.set_main_panel(wx.Panel(self.base_panel, wx.ID_ANY, pos
                =MAIN_PADDING, size=(MAIN_WINDOW_SIZE[0],
                MAIN_WINDOW_SIZE[1])))
209         main_panel = self.main_panel
210
211         check_render_value_before = False
212         if get_widget_by_id(43): check_render_value_before =
                get_widget_by_id(43).GetValue()
213
214         # reset all widgets on panel
215         for child in g_widgets.values():
216             child.Destroy()
217
218         list_box = widget_factory(wx.ListBox, main_panel, 15,
                choices=get_agent_list(target_box), style=wx.LB_SINGLE,
                pos=(2,2), size=(150, 368))
219         list_box.Bind(wx.EVT_LISTBOX, select_list_box)
220
221         self.render_box_options(target_box)
222
223         line1 = widget_factory(wx.StaticLine, main_panel, 16, pos=
                get_bottom_position(get_widget_by_id(7), 70), size
                =(525,2))
224
225         self.render_conditions(target_box)
226
227         line1 = widget_factory(wx.StaticLine, main_panel, 17, pos=
                get_bottom_position(get_widget_by_id(7), 205), size
                =(525,2))
228
229         self.render_buttons()
230
```

```
231         check_render = widget_factory( wx.CheckBox, main_panel, 43,
                "Render process of optimization.", pos=
                get_right_position(get_widget_by_id(14), 15))
232         check_render.SetValue(check_render_value_before)
233 
234         self.Show()
235         wx.Yield()
236 
237 def reset():
238     global g_main_frame
239 
240     main_app = wx.App()
241     #base_layout = Layout.get_softplanner_layout()
242     base_layout = Layout.get_sample_layout()
243     base_box =  base_layout.base_box
244     #base_box =  BoxAgent(Style([0,0], WINDOW_SIZE, 0), "base")
245     #Layout([], base_box)
246 
247     if g_main_frame: g_main_frame.Destroy()
248     g_main_frame = MainFrame(base_box, None, -1, u'Layout Designator
            ', pos=(100,100),size=MAIN_WINDOW_SIZE)
249 
250     main_app.MainLoop()
251 
252 def main():
253     reset()
254 
255 
256 def select_list_box(event):
257     update()
258     current_box = g_main_frame.current_box
259 
260     object = event.GetEventObject()
261     nth = object.GetSelection()
262     selected_box = current_box.inner_layout.agents[nth]
263 
264     g_main_frame.set_current_box(selected_box)
```

```
265     g_main_frame.refresh(selected_box)
266 
267 def click_reset_button(event):
268     reset()
269 
270 def click_make_button(event):
271     update()
272     new_box = BoxAgent(Style([0,0], [0,0], 1), "new box")
273     Layout([], new_box) # create inner layout of new_box
274 
275     g_main_frame.current_box.inner_layout.add_box(new_box)
276 
277     g_main_frame.set_current_box(new_box)
278     g_main_frame.refresh(new_box)
279 
280 def click_upper_button(event):
281     update()
282     parent_layout = g_main_frame.current_box.parent_layout
283     if parent_layout:
284         g_main_frame.set_current_box(parent_layout.base_box)
285         g_main_frame.refresh(parent_layout.base_box)
286 
287 
288 def click_update_button(event):
289     update()
290 
291 def myint(value):
292     if value == "" or value == "None":
293         return None
294     else:
295         return int(value)
296 
297 def update():
298     current_box = g_main_frame.current_box
299     current_box.set_identifier(get_widget_by_id(1).GetValue())
300     current_box.set_text(get_widget_by_id(41).GetValue())
301     current_box.set_width(int(get_widget_by_id(4).GetValue()))
```

```
302     current_box.set_height(int(get_widget_by_id(5).GetValue()))
303     current_box.set_x(int(get_widget_by_id(2).GetValue()))
304     current_box.set_y(int(get_widget_by_id(3).GetValue()))
305     current_box.set_fixedness(get_widget_by_id(44).GetValue())
306
307     current_box.condition.remove_condfun_by_name("_x_constraint")
308     if get_widget_by_id(19).GetValue() or get_widget_by_id(21).
            GetValue():
309         new_condfun = BoxCondFun.x_constraint(myint(get_widget_by_id
                (19).GetValue()), myint(get_widget_by_id(21).GetValue())
                , int(get_widget_by_id(34).GetValue()))
310         current_box.condition.add_condfun(new_condfun)
311
312     current_box.condition.remove_condfun_by_name("_y_constraint")
313     if get_widget_by_id(23).GetValue() or get_widget_by_id(25).
            GetValue():
314         new_condfun = BoxCondFun.y_constraint(myint(get_widget_by_id
                (23).GetValue()), myint(get_widget_by_id(25).GetValue())
                , int(get_widget_by_id(35).GetValue()))
315         current_box.condition.add_condfun(new_condfun)
316
317     current_box.condition.remove_condfun_by_name("_width_constraint"
            )
318     if get_widget_by_id(27).GetValue() or get_widget_by_id(29).
            GetValue():
319         new_condfun = BoxCondFun.width_constraint(myint(
                get_widget_by_id(27).GetValue()), myint(get_widget_by_id
                (29).GetValue()), int(get_widget_by_id(36).GetValue()))
320         current_box.condition.add_condfun(new_condfun)
321
322     current_box.condition.remove_condfun_by_name("_height_constraint
            ")
323     if get_widget_by_id(31).GetValue() or get_widget_by_id(33).
            GetValue():
324         new_condfun = BoxCondFun.height_constraint(myint(
                get_widget_by_id(31).GetValue()), myint(get_widget_by_id
                (33).GetValue()), int(get_widget_by_id(37).GetValue()))
```

```
325         current_box.condition.add_condfun(new_condfun)
326
327     if current_box.inner_layout:
328         current_box.inner_layout.set_optimization_needed(
                get_widget_by_id(15).GetValue())
329
330     g_main_frame.refresh(current_box)
331
332 def click_show_button(event):
333
334     optimization_app = wx.App()
335     optimization_frame = OptimizationFrame(None, -1, u'optimization'
            , pos=(400,100))
336
337     base_layout = g_main_frame.get_base_box().inner_layout
338     base_layout.render(optimization_frame.base_panel)
339
340     optimization_frame.Show()
341     optimization_app.MainLoop()
342
343 def click_start_button(event):
344
345     def start_optimization():
346         def render_closure():
347             def _render_closure():
348                 base_layout.render(optimization_frame.base_panel)
349                 optimization_frame.Show()
350                 wx.Yield()
351             return _render_closure
352
353         def pass_closure():
354             def _pass_closure():
355                 pass
356             return _pass_closure
357
358         def optimize_layout_inside(layout):
359             if layout:
```

```
360                 if layout.optimization_needed == True:
361                     optimization = OCSOptimization(layout)
362                     #optimization = RandomOptimization(layout)
363 
364                     if get_widget_by_id(43).GetValue():
365                         optimization.set_render_function(
                                render_closure())
366                     else:
367                         optimization.set_render_function(
                                pass_closure())
368 
369                     optimization.optimize()
370 
371                 for agent in layout.agents:
372                     optimize_layout_inside(agent.inner_layout)
373         base_layout = g_main_frame.get_base_box().inner_layout
374         optimize_layout_inside(base_layout)
375         base_layout.render(optimization_frame.base_panel)
376 
377 
378     update()
379 
380     optimization_app = wx.App()
381     optimization_frame = OptimizationFrame(None, -1, u'optimization'
            , pos=(400,100))
382 
383     start_optimization()
384 
385     print "finish"
386 
387     optimization_frame.Show()
388 
389     optimization_app.MainLoop()
390 
391 if __name__ == '__main__':
392     main()
```

A.2.2 agent.py

```
#-*- coding: utf-8 -*-

import math

class Agent():

    max_rules = 20

    def __init__(self, condition, identifier=""):
        self.set_ruleset([])
        self.condition = condition
        self.set_identifier(identifier)

    def set_ruleset(self, ruleset):
        self.ruleset = ruleset

    def set_identifier(self, identifier):
        self.identifier = identifier

    def get_copy(self):
        agent = Agent(condition, identifier)
        agent.set_ruleset(self.get_copy_of_ruleset)
        return agent

    def get_copy_of_ruleset(self):
        return self.ruleset[:]

    def get_sorted_ruleset(self):
        """ Return sorted ruleset according to its strength."""

        pairs = [(rule.strength, rule) for rule in self.ruleset]
        pairs.sort()
        pairs.reverse()
        sorted_ruleset = [pair[1] for pair in pairs]

        return sorted_ruleset
```

```
37
38      def get_number_of_rules(self):
39          return len(self.ruleset)
40
41      def add_rule(self, rule):
42          if Agent.max_rules <= self.get_number_of_rules():
43              self.replace_weakest_rule(rule)
44          else:
45              self.ruleset.append(rule)
46
47      def add_rule_with_random_action(self, agent_set):
48          pass
49
50      def delete_weak_rules(self):
51          border_strength = 0.01
52
53          for rule in self.ruleset:
54              if rule.strength < border_strength:
55                  self.delete_rule(rule)
56
57      def delete_rule(self, rule):
58          self.ruleset.remove(rule)
59
60      def find_matching_rule_and_apply(self, agent_set):
61          pass
62
63      # let this agent take given action
64      def execute_action(self, action):
65          action(self)
66
67      def get_index_of_weakest_rule(self):
68          weakest_value = 1000
69
70          index_weakest = None
71          for i,rule in enumerate(self.ruleset):
72              if rule.strength < weakest_value:
73                  weakest_value = rule.strength
```

```
74                  index_weakest = i
75
76          return index_weakest
77
78      def replace_weakest_rule(self, rule):
79          """Change weakest rule of agent into given rule. (
                DESTRUCTIVE)"""
80
81          i = self.get_index_of_weakest_rule()
82          new_rule = rule.get_copy()
83          #new_rule.set_strength(Rule.initial_strength)
84
85          if i != None:
86              self.ruleset[i] = new_rule
87
88          # if this agent has no rule, newly add the rule
89          else:
90              self.add_rule(new_rule)
91
92      def rule_select(self):
93          def _greedy(agent):
94              sorted_ruleset = agent.get_sorted_ruleset()
95              return sorted_ruleset[0]
96
97          return _greedy(self)
98
99      @classmethod
100     def exchange_rule_randomly(cls, agents):
101         """Exchange two rules between agents having same condition. (
                DESTRUCTIVE)"""
102
103         BORDER = 3
104
105         i1 = random.randint(0,len(agents)-1)
106         agent1 = agents[i1]
107
108         # make array of agents which has same rule with agent1
```

```
109         agents_with_same_condition = [agent for agent in agents if
                agent.condition == agent1.condition and agent != agent1]
110
111         if len(agents_with_same_condition) != 0:
112             i2 = random.randint(0,len(agents_with_same_condition)-1)
113             agent2 = agents_with_same_condition[i2]
114             if agent1.get_number_of_rules() != 0:
115                 sending_rule1 = agent1.get_sorted_ruleset()[0].
                        get_copy()
116                 #sending_rule1.set_strength(Rule.initial_strength)
117                 agent2.add_rule(sending_rule1)
118
119             if agent2.get_number_of_rules() != 0:
120                 sending_rule2 = agent2.get_sorted_ruleset()[0].
                        get_copy()
121                 #sending_rule2.set_strength(Rule.initial_strength)
122                 agent1.add_rule(sending_rule2)
123
124 def length_of_vector(vector):
125     return math.sqrt(vector[0] ** 2 + vector[1] ** 2)
126
127
128 class Style():
129     def __init__(self, position=[0,0], size=[10,10], visibility=1):
130         self.position = position
131         self.size = size
132         self.set_visibility(visibility)
133
134     def set_visibility(self, visibility):
135         self.visibility = visibility
136
137     def get_copy(self):
138         return Style(self.position[:], self.size[:], self.visibility
                )
139
140
141 class BoxAgent(Agent):
```

```
142     def __init__(self, style, identifier="", condition=None, text=""
            ):
143
144         # to avoid import error, avoid to use initial value
145         if condition == None: condition = Condition()
146
147         Agent.__init__(self, condition, identifier)
148         self.set_style(style)
149         self.set_parent_layout(None)
150         self.set_text(text)
151         self.set_inner_layout(None)
152
153     def get_copy(self):
154         box = BoxAgent(self.style.get_copy(), self.identifier, self.
                condition.get_copy(), self.text)
155         box.set_parent_layout(self.parent_layout)
156         box.set_inner_layout(self.inner_layout)
157         return box
158
159     def get_position(self):
160         return self.style.position
161
162     def get_x(self):
163         return self.get_position()[0]
164
165     def get_y(self):
166         return self.get_position()[1]
167
168     def get_size(self):
169         return self.style.size
170
171     def get_width(self):
172         return self.get_size()[0]
173
174     def get_height(self):
175         return self.get_size()[1]
176
```

```
177     def get_visibility(self):
178         return self.style.visibility
179 
180     def get_inner_layout(self):
181         return self.inner_layout
182 
183     def get_limit_position(self):
184         if self.parent_layout:
185             base_box = self.parent_layout.base_box
186             return base_box.get_size()
187         else:
188             return WINDOW_SIZE
189 
190     def set_x(self, value):
191         left_limit = 0
192         right_limit = self.get_limit_position()[0]
193 
194         if value < left_limit:
195             x_after_movement = left_limit + 2
196         elif right_limit < value + self.get_width():
197             x_after_movement = right_limit - self.get_width() - 2
198         else:
199             x_after_movement = value
200 
201         self.style.position[0] = x_after_movement
202 
203     def set_y(self, value):
204         top_limit = 0
205         bottom_limit = self.get_limit_position()[1]
206 
207         if value < top_limit:
208             y_after_movement = top_limit + 2
209         elif bottom_limit < value + self.get_height():
210             y_after_movement = bottom_limit - self.get_height() - 2
211         else:
212             y_after_movement = value
213 
```

```
214         self.style.position[1] = y_after_movement
215 
216     def set_width(self, value):
217         right_limit = self.get_limit_position()[0]
218 
219         if value < 0: value = 0
220 
221         # in case of overflow
222         if self.get_x() + value > right_limit:
223 
224             if value < right_limit:
225                 self.set_x(right_limit - value)
226             else:
227                 self.set_x(0)
228                 value = right_limit
229 
230         self.style.size[0] = value
231 
232 
233     def set_height(self, value):
234         bottom_limit = self.get_limit_position()[1]
235 
236         if value < 0: value = 0
237 
238         # in case of overflow
239         if self.get_y() + value > bottom_limit:
240 
241             if value < bottom_limit:
242                 self.set_y(bottom_limit - value)
243             else:
244                 self.set_y(0)
245                 value = bottom_limit
246 
247         self.style.size[1] = value
248 
249 
250     def set_parent_layout(self, parent_layout):
```

```
251         self.parent_layout = parent_layout
252 
253     def set_inner_layout(self, inner_layout):
254         self.inner_layout = inner_layout
255 
256     def set_style(self, style):
257         self.style = style
258 
259     def set_text(self, text):
260         self.text = text
261 
262     def make_visible(self):
263         self.style.set_visibility(1)
264 
265     def make_invisible(self):
266         self.style.set_visibility(0)
267 
268     def render(self, parent_panel):
269 
270         if self.get_visibility() == 1:
271             border = wx.SIMPLE_BORDER
272         else:
273             border = wx.NO_BORDER
274 
275         panel = wx.Panel(parent_panel, wx.ID_ANY, pos=self.
               get_position(), size=self.get_size(), style=border)
276         panel.SetBackgroundColour("#ffffff")
277 
278         if self.text:
279             text = wx.StaticText(panel, wx.ID_ANY, self.text, (0,0),
                    size=self.get_size())
280             text.SetBackgroundColour('#ffffff')
281 
282         # if this box has boxes(layout) in itself, render them
283         if self.inner_layout:
284             self.inner_layout.render(panel)
285 
```

```
286     def add_rule_with_random_action(self, layout):
287         """Add rule which has present condition and random action.
                """
288 
289         situation = Situation(agent_set=layout, agent=self)
290         condition = Condition.make_condition(situation)
291 
292         if condition.condfuns != []:
293             new_rule = BoxRule.generate_rule_with_random_action(
                    condition, layout)
294             self.add_rule(new_rule)
295             return True
296 
297         # if no condition can be represents current situation, add
                no rule
298         else:
299             return False
300 
301     def get_matching_rule(self, layout):
302         """Find (the strongest) rule which matches current condition
                (layout and box) and return it. return None if no rule
                is found."""
303 
304         self.ruleset = self.get_sorted_ruleset()
305         matching_rule = None
306         current_situation = Situation(agent_set=layout, agent=self)
307 
308         for rule in self.ruleset:
309             if rule.condition.evaluate(current_situation, 1):
310                 matching_rule = rule
311                 break
312 
313         return matching_rule
314 
315     def get_right_position(self, margin):
316         present_x, present_y = self.get_position()
317         return [present_x + self.get_width() + margin, present_y]
```

```
318
319     def get_bottom_position(self, margin):
320         present_x, present_y = self.get_position()
321         return [present_x, present_y + self.get_height() + margin]
322
323     def get_center_x(self):
324         parent_box = self.parent_layout.base_box
325         return (parent_box.get_width() - self.get_width()) / 2
326
327     def get_center_y(self):
328         parent_box = self.parent_layout.base_box
329         return (parent_box.get_width() - self.get_height()) / 2
330
331     def add_x(self, amount):
332         x_after_movement = (self.get_x() + amount)
333         self.set_x(x_after_movement)
334
335     def add_y(self, amount):
336         y_after_movement = (self.get_y() + amount)
337         self.set_y(y_after_movement)
338
339     def add_vector(self, vector):
340         self.add_x(vector[0])
341         self.add_y(vector[1])
342
343     def add_width(self, amount):
344         self.set_width(self.get_width() + amount)
345
346     def add_height(self, amount):
347         self.set_height(self.get_height() + amount)
348
349     def align_left(self, target_box):
350         self.set_x(target_box.get_x())
351
352     def align_top(self, target_box):
353         self.set_y(target_box.get_y())
354
```

```
355     def unify_width_and_align(self, target_box):
356         self.align_left(target_box)
357         self.set_width(target_box.get_width())
358 
359     def unify_height_and_align(self, target_box):
360         self.align_top(target_box)
361         self.set_height(target_box.get_height())
362 
363     def make_vertical_space(self, box, amount):
364         """Place itself next to given box making given amount of
                vertical space."""
365 
366         if box.get_y() < self.get_y():
367             self.set_y(box.get_y() + box.get_height() + amount)
368         else:
369             self.set_y(box.get_y() - self.get_height() - amount)
370 
371     def make_horizontal_space(self, box, amount):
372         """Place itself next to given box making given amount of
                horizontal space."""
373 
374         if box.get_x() < self.get_x():
375             self.set_x(box.get_x() + box.get_width() + amount)
376         else:
377             self.set_x(box.get_x() - self.get_width()- amount)
378 
379     def get_the_best_box_from_pairs(self, pairs):
380         """Return the box which has the highest pair value in the
                list of pairs."""
381 
382         pairs.sort()
383 
384         if len(pairs) == 1 :
385             best_box = pairs[0][1]
386         else:
387             # take 1st box as nearest box because 0th box is box
                    itself
```

```
388             best_box = pairs[1][1]
389
390         return best_box
391
392     def get_nearest_box(self, layout):
393         pairs = [(BoxAgent.get_gravity_distance(self, box), box)
                for box in layout.agents]
394
395         return self.get_the_best_box_from_pairs(pairs)
396
397     def get_most_aligned_box(self, layout):
398
399         pairs = []
400         for box in layout.agents:
401             alignemnt_distance = self.get_alignment_distance(self,
                    box)
402
403             # in case smaller_difference is same, add gravity
                    distance to make difference
404             scaled_gravity_distance = BoxAgent.get_gravity_distance(
                    self, box) / 200
405             pairs.append((alignemnt_distance +
                    scaled_gravity_distance, box))
406
407         return self.get_the_best_box_from_pairs(pairs)
408
409     def get_position_difference(self,box,i):
410         return abs(self.get_position()[i] - box.get_position()[i])
411
412     def get_gravity_difference(self, target_box):
413         """Get difference(signed distance) of gravity."""
414
415         self_gravity = self.get_gravity_position()
416         target_gravity = target_box.get_gravity_position()
417
418         gravity_difference_x = self_gravity[0] - target_gravity[0]
419         gravity_difference_y = self_gravity[1] - target_gravity[1]
```

```
420
421         return [gravity_difference_x, gravity_difference_y]
422
423     def change_distance(self, target_box, amount):
424         """Stay away from target, or approach to target."""
425
426         gravity_difference = self.get_gravity_difference(target_box)
427         gravity_distance = length_of_vector(gravity_difference)
428
429         if gravity_distance != 0:
430             vector_to_move = [amount * (gravity_difference[0] /
                    gravity_distance), amount * (gravity_difference[1] /
                     gravity_distance)]
431             self.add_vector(vector_to_move)
432
433     def stay_away(self, target_box, amount=None):
434
435         # set default amount
436         if amount == None:
437             gravity_difference = self.get_gravity_difference(
                    target_box)
438             gravity_distance = length_of_vector(gravity_difference)
439
440             basic_distance_vector = BoxAgent.
                    get_basic_distance_vector(self, target_box)
441             basic_distance = length_of_vector(basic_distance_vector)
442
443             if gravity_distance != 0:
444                 # the closer two boxes are, the further they move
445                 amount =  basic_distance ** 2 / gravity_distance
446             else:
447                 amount = 0
448
449         self.change_distance(target_box, amount)
450
451     def approach(self, target_box, amount=None):
452         gravity_difference = self.get_gravity_difference(target_box)
```

```
453         gravity_distance = length_of_vector(gravity_difference)
454 
455         #set default amount
456         if amount == None:
457             amount = gravity_distance / 2
458 
459         # do not approach if boxes are already close enough
460         if gravity_distance > amount:
461             self.change_distance(target_box, -amount)
462 
463     def get_gravity_position(self):
464         """Get the position of center of gravity."""
465         gravity_x = self.get_x() + self.get_width() / 2
466         gravity_y = self.get_y() + self.get_height() / 2
467 
468         return [gravity_x, gravity_y]
469 
470     def off_the_edge_or_not(self):
471         off_right_or_bottom = (self.get_x() + self.get_width()) >
               WINDOW_SIZE[0] or (self.get_y() + self.get_height()) >
               WINDOW_SIZE[1]
472         off_left_or_top = self.get_x() < 0 or self.get_y() < 0
473 
474         return off_right_or_bottom or off_left_or_top
475 
476     @classmethod
477     def overlap_or_not(cls, box1, box2):
478         return cls.get_overlaped_area(box1, box2) > 0
479 
480     @classmethod
481     def aligned_or_not(cls, box1, box2):
482         return box1.get_x() == box2.get_x() or box1.get_y() == box2.
               get_y()
483 
484     @classmethod
485     def get_basic_distance_vector(cls, box1, box2):
486         basic_distance_x = (box1.get_width() + box2.get_width()) /2
```

```
487         basic_distance_y = (box1.get_height() + box2.get_height())
                /2
488         return [basic_distance_x, basic_distance_y]
489
490     @classmethod
491     def get_alignment_distance(cls, box1, box2):
492         return min(box1.get_position_difference(box2,0), box1.
                get_position_difference(box2,1))
493
494     @classmethod
495     def get_gravity_distance(cls, box1, box2):
496         return length_of_vector(box1.get_gravity_difference(box2))
497
498     @classmethod
499     def get_overlaped_area(cls, box1, box2):
500         overlaped_width = min(box1.get_x() + box1.get_width(), box2.
                get_x() + box2.get_width()) + 2 - (max(box1.get_x(),
                box2.get_x()))
501         overlaped_height = min(box1.get_y() + box1.get_height(),
                box2.get_y() + box2.get_height()) + 2 - (max(box1.get_y
                (), box2.get_y()))
502
503         if overlaped_width > 0 and overlaped_height > 0:
504             return overlaped_width * overlaped_height
505         else:
506             return 0
507
508
509 # imports - - - - - - -
510 import wx
511 import random
512 from main import WINDOW_SIZE
513 from rule import *
514 from condition import *
```

A.2.3 optimization.py

```
1 #-*- coding: utf-8 -*-
```

```
2
3  # imports - - - - - - -
4  from layout import *
5  import random
6  import time
7
8  def update_something(bool_condition, process):
9      """A high order function which updates given value and return
           True if updated."""
10
11     if bool_condition:
12         process()
13         return True
14     else:
15         return False
16
17 class Optimization():
18
19     def __init__(self, agent_set):
20         self.agent_set = agent_set#agent_set.get_copy()
21         self.reset_record()
22
23     def optimize(self):
24         pass
25
26     def set_render_function(self, render_function):
27         self.render_function = render_function
28
29     def render(self):
30         self.render_function()
31
32     def reset_record(self):
33         self.set_best_value(100000)
34         self.set_worst_value(0)
35
36     def get_whole_constraints_satisfied_or_not(self):
37         constraints = self.agent_set.condition
```

```
38          situation = Situation(agent_set=self.agent_set.get_copy())
39          return constraints.evaluate(situation)
40
41      def get_agent_constraints_satisfied_or_not(self):
42          situation = Situation(agent_set=self.agent_set.get_copy())
43          return situation.agent_set.evaluate_agent_constraint()
44
45      def get_best_value_now_or_not(self):
46          return abs(self.best_value - self.get_objective_value()) <
                OCSOptimization.minimum_difference
47
48      def get_objective_function(self):
49          return self.agent_set.condition.
                get_sum_of_constraint_objective
50
51      def get_objective_value(self):
52          objective_funtion = self.get_objective_function()
53          situation = Situation(agent_set=self.agent_set)
54          return objective_funtion(situation)
55
56      def get_rulesets(self):
57          return self.agent_set.get_rulesets()
58
59      def set_best_value(self, value):
60          self.best_value = value
61
62      def set_worst_value(self, value):
63          self.worst_value = value
64
65      def update_best_value(self, new_value):
66          """Update␣best␣objective␣value␣if␣gained␣new␣value␣is␣better
                (smaller)␣than␣the␣best␣value.␣If␣updated,␣return␣True.␣
                """
67          update_something((new_value < self.best_value), (lambda :
                self.set_best_value(new_value)))
68
69      def update_worst_value(self, new_value):
```

```
70          """Update worst objective value if gained new value is worse
                (bigger) than the worst value. If updated, return True.
                """
71          update_something((self.worst_value < new_value), (lambda :
                self.set_worst_value(new_value)))
72
73      def clear_output_file(self, filename):
74          file = open(filename, 'w')
75          file.write("")
76          file.close
77
78      def output_obvective_values(self, filename):
79
80          constraints = self.agent_set.condition
81          situation = Situation(agent_set=self.agent_set.get_copy())
82
83          output_line = ""
84          output_line += str(constraints.
                get_sum_of_constraint_objective(situation)) + ","
85
86          agents = self.agent_set.agents
87          for i,agent in enumerate(agents):
88              situation_with_agent = Situation(agent_set=self.
                    agent_set.get_copy(), agent=agent.get_copy())
89              output_line += str(agent.condition.
                    get_sum_of_constraint_objective(situation_with_agent
                    )) + ","
90
91          output_line += "\n"
92
93          file = open(filename, 'a')
94          file.write(output_line)
95          file.close()
96
97
98  class RandomOptimization(Optimization):
99
```

```
100     max_trial = 10000
101 
102     def __init__(self, agent_set=AgentSet()):
103         Optimization.__init__(self, agent_set)
104 
105     def optimize(self):
106 
107         base_box = self.agent_set.base_box
108         grid_size = 10
109         max_x_grid =  base_box.get_width() / grid_size
110         max_y_grid =  base_box.get_height() / grid_size
111 
112         self.clear_output_file("output_data.csv")
113 
114         number_of_trial = 0
115         total_cycle_duration = 0
116         while True:
117             start = time.time()
118             constraints_satisfied = self.
                    get_whole_constraints_satisfied_or_not() and self.
                    get_agent_constraints_satisfied_or_not()
119             reach_max_trial = self.max_trial < number_of_trial
120             if constraints_satisfied or reach_max_trial: break
121 
122             self.render()
123 
124             for agent in self.agent_set.agents:
125                 if agent.fixedness == True: continue
126 
127                 agent.set_x(random.randint(0, max_x_grid) *
                        grid_size)
128                 agent.set_y(random.randint(0, max_y_grid) *
                        grid_size)
129 
130                 max_width_grid =  (base_box.get_width() - agent.
                        get_x()) / grid_size
```

```
131                 max_height_grid =  (base_box.get_height() - agent.
                        get_y()) / grid_size
132                 agent.set_width(random.randint(0, max_width_grid) *
                        grid_size)
133                 agent.set_height(random.randint(0, max_height_grid)
                        * grid_size)
134
135             self.output_obvective_values("output_data.csv")
136
137             number_of_trial += 1
138             total_cycle_duration += time.time() - start
139
140         average_cycle_duration = total_cycle_duration /
                number_of_trial
141         print average_cycle_duration
142
143 class OCSOptimization(Optimization):
144     minimum_iteration = 1
145     max_cycle_of_learning = 1
146     minimum_difference = 20
147
148     def __init__(self, agent_set=AgentSet()):
149         Optimization.__init__(self, agent_set)
150         self.organizational_rulesets = []
151
152     def get_half_value(self):
153         return (self.worst_value + self.best_value) / 2
154
155     def set_organizational_rulesets(self, rulesets):
156         self.organizational_rulesets = rulesets
157
158     def optimize(self):
159         # use organizational rulesets for default rulesets
160         self.agent_set.set_rulesets(self.organizational_rulesets)
161         print self.organizational_rulesets
162
163         constraints = self.agent_set.condition
```

```
164         situation = Situation(agent_set=self.agent_set.get_copy())
165 
166         self.clear_output_file("output_data.csv")
167 
168         # Termination condition for whole optimization: satisfaction
                of all hard constraints, best_value
169         number_of_trial = 0
170         total_cycle_duration = 0
171         while True:
172             start = time.time()
173             constraints_satisfied = self.
                    get_whole_constraints_satisfied_or_not() and self.
                    get_agent_constraints_satisfied_or_not()
174             iteration_finished = number_of_trial > OCSOptimization.
                    minimum_iteration
175 
176             if constraints_satisfied and iteration_finished: break
177 
178             #self.display_break_condition()
179 
180             self.one_optimization_cycle()
181             number_of_trial += 1
182             total_cycle_duration += time.time() - start
183 
184         average_cycle_duration = total_cycle_duration /
                number_of_trial
185         print "Average Cycle Duration: " + str(
                average_cycle_duration)
186         print "Number of Trials: " + str(number_of_trial)
187 
188         self.output_obvective_values("output_data.csv")
189 
190     def one_optimization_cycle(self):
191 
192         # repeat while ruleset is not converged(while new rule is
                generated)
193         rule_not_found_or_not = True
```

```
194         while rule_not_found_or_not == True:
195             rule_not_found_or_not, rule_generated_or_not = self.
                    agent_set.generate_rules()
196             Agent.exchange_rule_randomly(self.agent_set.agents)
197 
198         #self.display_status()
199 
200         self.output_obvective_values("output_data.csv")
201 
202         # learn and adjust a strength of each rule
203         self.reinforcement_learning()
204 
205         self.render()
206 
207         current_objective_value = self.get_objective_value()
208         self.update_worst_value(current_objective_value)
209         self.update_organizational_rulesets(current_objective_value)
210 
211         self.agent_set.delete_weak_rules()
212 
213         #self.display_status()
214 
215     def update_organizational_rulesets(self, new_value):
216         """update␣objective␣value␣if␣it␣is␣the␣best,␣and␣record␣the␣
                rulesets"""
217 
218         new_rulesets = self.agent_set.get_rulesets()
219         #update_something(self.update_best_value(new_value), (lambda
                : self.set_organizational_rulesets(new_rulesets))) とに
                かく毎週記録してれば終了時のルールはゲットできる#
220         self.set_organizational_rulesets(new_rulesets)
221 
222     def reinforcement_learning(self):
223         for agent in self.agent_set.agents:
224             # do not move fixed box
225             if agent.fixedness == True: continue
226 
```

```
227             self.learn_strength_with_profit_sharing(agent)
228
229     def learn_strength_with_profit_sharing(self, agent):
230
231         border_enough = 0 # condition evaluation gets loose when
                border gets higher
232         agent_set = self.agent_set
233
234         for i in range(OCSOptimization.max_cycle_of_learning):
235
236             episode = 1
237             applied_pairs = []
238
239             current_value = self.get_objective_value()
240
241             constraints = self.agent_set.condition
242             situation = Situation(agent_set=agent_set.get_copy(),
                    agent=agent)
243
244             while True:
245                 agent_condition_satisfied = agent.condition.
                        get_sum_of_constraint_objective(situation) <=
                        border_enough
246                 whole_condition_satisfied = constraints.
                        get_sum_of_constraint_objective(situation) <=
                        border_enough
247                 selected_rule = agent.rule_select()
248
249                 if agent_condition_satisfied and
                        whole_condition_satisfied: break
250                 if selected_rule.strength < 0.001: break # to avoid
                        infinite loop, skip too weak rule
251
252                 # record objective value, constraint satisfaction
                        before execution
253                 previous_situation = situation.get_copy()
254                 agent.execute_action(selected_rule.action)
```

```
255                 situation = Situation(agent_set=agent_set.get_copy()
                        , agent=agent.get_copy())
256
257                 # rememebr applied rule with episode(time)
258                 pair_of_episode_and_rule = {'episode':episode, 'rule
                        ':selected_rule}
259                 applied_pairs.append(pair_of_episode_and_rule)
260
261                 episode += 1
262                 #self.display_status()
263
264                 self.reward_process(situation, previous_situation,
                        applied_pairs)
265
266     def reward_process(self, present_situation, previous_situation,
            applied_pairs):
267
268         fixed_value = (lambda x: 0.5)
269         decreasing_function = (lambda x: (0.1)**x)
270
271         if self.positive_reward_or_not(present_situation,
                previous_situation):
272             reward_function = (lambda x: fixed_value(x))
273         else:
274             reward_function = (lambda x: -1 * fixed_value(x))
275
276         self.give_rewards(applied_pairs, reward_function)
277
278     def positive_reward_or_not(self, present_situation,
            previous_situation):
279         """return True if positive reward is to be given, otherwise
                return False"""
280
281         whole_constraints = self.agent_set.condition
282         agent_constraints = previous_situation.agent.condition
283
```

```
284         previous_value = self.get_objective_function()(
                previous_situation)
285         previous_whole_constraints_objective = whole_constraints.
                get_sum_of_constraint_objective(previous_situation)
286         previous_agent_constraints_objective = agent_constraints.
                get_sum_of_constraint_objective(previous_situation)
287 
288         current_value = self.get_objective_value()
289         half_value = self.get_half_value()
290         whole_constraints_objective = whole_constraints.
                get_sum_of_constraint_objective(present_situation)
291         agent_constraints_objective = agent_constraints.
                get_sum_of_constraint_objective(present_situation)
292 
293         compare_with_half = (half_value > current_value)
294         compare_with_before = (previous_value > current_value)
295 
296         difference_whole = previous_whole_constraints_objective -
                whole_constraints_objective
297         difference_agent = previous_agent_constraints_objective -
                agent_constraints_objective
298 
299         # if objective is already the best, give positive reward
300         best_now =  whole_constraints_objective == 0 and
                agent_constraints_objective == 0
301 
302         improve_both_objective = difference_whole + difference_agent
                > 0
303         improve_either_objective = difference_whole > 0 or
                difference_agent > 0
304 
305         return improve_either_objective or best_now
306 
307     def give_rewards(self, pairs, reward_function):
308         """give␣rewards␣to␣serial␣rules␣at␣one␣time"""
309 
310         for pair in pairs:
```

```
311             rule = pair['rule']
312             episode = pair['episode']
313             rule.reinforce(episode, reward_function)
314 
315     def display_break_condition(self):
316 
317         constraints = self.agent_set.condition
318         situation = Situation(agent_set=self.agent_set.get_copy())
319 
320         print "-------------------------------------------"
321         print "Whole Constraints: " + str(self.
                get_whole_constraints_satisfied_or_not())
322         print "   Whole Objective:   " + str(constraints.
                get_sum_of_constraint_objective(situation))
323         print "Agent Constraints: " + str(self.
                get_agent_constraints_satisfied_or_not())
324         agents = self.agent_set.agents
325 
326         for i,agent in enumerate(agents):
327             situation_with_agent = Situation(agent_set=self.
                    agent_set.get_copy(), agent=agent.get_copy())
328             print "   Agent Number " + str(i) + ": " + str(agent.
                    condition.evaluate(situation_with_agent))
329             print "       Objective:   " + str(agent.condition.
                    get_sum_of_constraint_objective(situation_with_agent
                    ))
330 
331         print "Best Value now:" + str(self.get_best_value_now_or_not
                ())
332         print "Best Value: " + str(self.best_value)
333         print "Current Value: " + str(self.get_objective_value())
334 
335     def display_status(self):
336 
337         print "==========================================="
338         print "Best Value so far: " + str(self.best_value)
339         print "Worst Value so far: " + str(self.worst_value)
```

```
340         print "Current Value: " + str(self.get_objective_value())
341         print ""
342         agents = self.agent_set.agents
343 
344         for i,agent in enumerate(agents):
345             print "Agent Number " + str(i)
346             ruleset = agent.get_sorted_ruleset()
347 
348             for rule_i, rule in enumerate(ruleset):
349                 print "  Rule Number " + str(rule_i) + ": " + str(
                        rule.strength)
350                 print [condfun.function for condfun in rule.
                        condition.condfuns]
351                 print rule.action
352 
353             print ""
354 
355 # imports - - - - - - -
356 from condition import *
```

A.2.4 layout.py

```
1  #-*- coding: utf-8 -*-
2  
3  
4  class AgentSet():
5      def __init__(self, agents=[], condition=None):
6          self.agents = agents
7          self.set_condition(condition)
8  
9      def get_copy(self):
10         return AgentSet(self.agents[:])
11 
12     def add_agent(self, agent):
13         self.agents.append(agent)
14 
15     def remove_agent(self, agent):
16         self.agents.remove(agent)
```

```
17
18      def update(self, agents):
19          self.agents = agents
20
21      def get_number_of_agents(self):
22          return len(self.agents)
23
24      def set_condition(self, condition):
25          self.condition = condition
26
27      def set_rulesets(self, rulesets):
28          if rulesets != []:
29              for i, agent in enumerate(self.agents):
30                  agent.set_ruleset(rulesets[i])
31
32      def get_rulesets(self):
33          return [agent.ruleset for agent in self.agents]
34
35      # generate new rules
36      def generate_rules(self):
37          rule_generated_or_not = False
38          rule_not_found_or_not = False
39
40          for agent in (self).agents:
41
42              # if no rule is matched, make new rule
43              if agent.get_matching_rule(self) == None:
44
45                  rule_not_found_or_not = True
46
47                  # generate new rule with condition of present
                        situation
48                  if agent.add_rule_with_random_action(self):
49                      rule_generated_or_not = True
50
51          return [rule_not_found_or_not, rule_generated_or_not]
52
```

```
53      def delete_weak_rules(self):
54          rule_deleted_or_not = False
55
56          for agent in (self).agents:
57              if agent.delete_weak_rules():
58                  rule_deleted_or_not = True
59
60          return rule_deleted_or_not
61
62
63  class Layout(AgentSet):
64      def __init__(self, agents=[], base_box=None, condition=None):
65
66          # to avoid import error, avoid to use initial value
67          if base_box == None: base_box = BoxAgent(Style([0,0], main.
                WINDOW_SIZE, 0))
68          if condition == None: condition = Condition()
69
70          AgentSet.__init__(self, agents, condition)
71
72          self.set_base_box(base_box)
73          self.optimization_needed = False
74
75          for agent in agents:
76              agent.set_parent_layout(self)
77
78      def set_base_box(self, base_box):
79          self.base_box = base_box
80          base_box.set_inner_layout(self)
81
82      def set_optimization_needed(self, value):
83          self.optimization_needed = value
84
85      def add_box(self, box):
86          self.agents.append(box)
87          box.set_parent_layout(self)
88
```

```
89      def get_minimum_inclusion_size(self):
90          """Get size of minimum rectangle which includes all box in
                layout"""
91
92          x_list = [box.get_x() for box in self.agents]
93          end_x_list = [box.get_x() + box.get_width() for box in self.
                agents]
94          y_list = [box.get_y() for box in self.agents]
95          end_y_list = [box.get_y() + box.get_height() for box in self
                .agents]
96
97          inclusion_width = reduce(max, end_x_list) - reduce(min,
                x_list)
98          inclusion_height = reduce(max, end_y_list) - reduce(min,
                y_list)
99
100         return [inclusion_width, inclusion_height]
101
102
103     def get_sum_of_all_constraint_objective(self):
104         sum_of_all_constraint_objective = 0
105         sum_of_all_constraint_objective += self.condition.
                get_sum_of_constraint_objective(Situation(agent_set=self
                ))
106
107         for agent in self.agents:
108             situation = Situation(agent_set=self, agent=agent)
109             sum_of_all_constraint_objective += agent.condition.
                    get_sum_of_constraint_objective(situation)
110
111         return sum_of_all_constraint_objective
112
113     def get_agent_with_identifier(self, identifier):
114
115         result = None
116         for agent in self.agents:
117
```

```
118             if agent.identifier == identifier:
119                 result = agent
120                 break
121 
122             if agent.inner_layout:
123                 result = agent.inner_layout.
                        get_agent_with_identifier(identifier)
124                 if result != None: break
125 
126         return result
127 
128     def get_copy_of_agents(self):
129         return [agent.get_copy() for agent in self.agents]
130 
131     def get_copy(self):
132         return Layout(self.get_copy_of_agents(), self.base_box, self
            .condition)
133 
134     def render(self, parent_panel):
135         for agent in self.agents:
136             agent.render(parent_panel)
137 
138     def evaluate_agent_constraint(self):
139         bool_list = []
140         for agent in self.agents:
141             situation = Situation(agent=agent)
142             bool_list.append(agent.condition.evaluate(situation))
143 
144         if len(bool_list) == 0:
145             return True
146         else:
147             return reduce((lambda b1, b2: b1 and b2), bool_list)
148 
149     @classmethod
150     def get_sample_layout(cls):
151 
152         MAIN_WIDTH = 750
```

```
153         margin = 20
154         MAX_WIDTH = main.WINDOW_SIZE[0]
155         MAX_HEIGHT = main.WINDOW_SIZE[1]
156 
157         # base_layout
158 
159         base_box = BoxAgent(Style([0,0], main.WINDOW_SIZE, 0), "base
                ")
160         base_inner_box = BoxAgent(Style([0,0], [MAIN_WIDTH,
                MAX_HEIGHT], 1), "base_inner")
161         base_inner_box.set_x(base_inner_box.get_center_x())
162 
163         base_layout = Layout([base_inner_box], base_box)
164 
165         # base_inner_layout
166 
167         header_style = Style([0,0], [MAIN_WIDTH, 80], 1)
168         header_box = BoxAgent(header_style, "header")
169 
170         main_style = Style(header_box.get_bottom_position(margin), [
                MAIN_WIDTH, 600], 0)
171         main_box = BoxAgent(main_style, "main")
172 
173         base_inner_layout_condition = Condition([BoxCondFun.
                no_overlap(), BoxCondFun.all_aligned(), BoxCondFun.
                width_unification(1)], 1)
174         base_inner_layout = Layout([header_box, main_box],
                base_inner_box, base_inner_layout_condition)
175         base_inner_layout.set_optimization_needed(False)
176 
177         # main_layout
178 
179         content_style = Style([100,2], [280, 280], 1)
180         content_condition = Condition([BoxCondFun.width_constraint
                (550,590), BoxCondFun.height_constraint(500), BoxCondFun
                .y_constraint(2, 40)] , 1)
```

```
181         content_box = BoxAgent(content_style, "content",
                content_condition)
182
183         side_style = Style([400,2], [280, 280], 1)
184         side_condition = Condition([BoxCondFun.width_constraint
                (140,160), BoxCondFun.height_constraint(500), BoxCondFun
                .y_constraint(2, 40)] , 1)
185         side_box = BoxAgent(side_style, "side", side_condition)
186
187         main_layout_condition = Condition([BoxCondFun.no_overlap(),
                BoxCondFun.all_aligned()], 1)
188         main_layout = Layout([side_box, content_box], main_box,
                main_layout_condition)
189         main_layout.set_optimization_needed(True)
190
191         # side_layout
192
193         side_item_style1 = Style([2,2], [30, 30], 1)
194         side_item_style2 = Style([52,2], [30, 30], 1)
195         side_item_condition = Condition([BoxCondFun.width_constraint
                (120,140), BoxCondFun.height_constraint(80, 160, 0),
                BoxCondFun.centering_constraint(0)] , 1)
196         side_item_boxes = []
197         side_item_boxes.append(BoxAgent(side_item_style1, "
                side_item1", side_item_condition, "side_item" ))
198         side_item_boxes.append(BoxAgent(side_item_style2, "
                side_item2", side_item_condition, "side_item"))
199
200         side_layout_condition = Condition([BoxCondFun.no_overlap(),
                BoxCondFun.all_aligned(), BoxCondFun.width_unification
                (1), BoxCondFun.height_unification(1)], 1)
201         side_layout = Layout(side_item_boxes, side_box,
                side_layout_condition)
202         side_layout.set_optimization_needed(True)
203
204         # content_layout
205
```

```
206         content_item_style1 = Style([0,0], [240, 240], 0)
207         content_item_style2 = Style([50,0], [240, 240], 0)
208         content_item_condition = Condition([BoxCondFun.
                height_constraint(80, 160), BoxCondFun.
                horizontal_margin_constraint(50), BoxCondFun.
                centering_constraint(1), BoxCondFun.y_constraint(2, 300)
                 ] , 1)
209         content_item_box1 = BoxAgent(content_item_style1, "
                content_item1", content_item_condition)
210         content_item_box2 = BoxAgent(content_item_style2, "
                content_item2", content_item_condition)
211
212         content_layout_condition = Condition([BoxCondFun.no_overlap
                (), BoxCondFun.all_aligned(), BoxCondFun.
                width_unification(0)], 1)
213         content_layout = Layout([content_item_box1,
                content_item_box2], content_box,
                content_layout_condition)
214         content_layout.set_optimization_needed(True)
215
216         # content_item_layout
217
218         content_image_style = Style([2,2], [30, 30], 1)
219         content_image_condition = Condition([BoxCondFun.
                width_constraint(120,140, 1), BoxCondFun.
                vertical_margin_constraint(50)] , 1)
220         content_image_box = BoxAgent(content_image_style, "
                content_image", content_image_condition, "image")
221
222         content_text_style = Style([2, 120],[30, 30], 1)
223         content_text_condition = Condition([BoxCondFun.
                height_constraint(50, 80, 1)] , 1)
224         content_text_box = BoxAgent(content_text_style, "
                content_text", content_text_condition, "text")
225
226         content_item_layout_condition = Condition([BoxCondFun.
                no_overlap(), BoxCondFun.all_aligned(), BoxCondFun.
```

```
            horizontal_layout_margin(40,1)], 1)
227         content_item_layout1 = Layout([content_image_box.get_copy(),
                content_text_box.get_copy()], content_item_box1,
            content_item_layout_condition)
228         content_item_layout2 = Layout([content_image_box.get_copy(),
                content_text_box.get_copy()], content_item_box2,
            content_item_layout_condition)
229         content_item_layout1.set_optimization_needed(True)
230         content_item_layout2.set_optimization_needed(True)
231
232         return base_layout
233
234 # imports - - - - - - -
235 from condition import *
236 from specification import *
237 import main
238 from agent import *
```

A.2.5 specificaiton.py

```
1  #-*- coding: utf-8 -*-
2
3  import math
4
5  class Objective():
6      def __init__(self, max_min=1, function=(lambda x: 0)):
7          self.max_min = max_min
8          self.function = function
9
10     @staticmethod
11     def sum_of_overlapped_area(situation):
12
13         boxes = situation.agent_set.agents
14
15         if len(boxes) > 1:
16             overlapped_area_list = [[int(math.sqrt(BoxAgent.
                   get_overlaped_area(boxes[i], boxes[j]))) for j in
                   range(i, len(boxes)) if i != j] for i in range(len(
```

```
                boxes))]
        else:
            overlapped_area_list = [[0]]

        # return sum of distances
        return reduce((lambda x,y: x+y), reduce((lambda x,y: x+y),
            overlapped_area_list))

    @staticmethod
    def sum_of_alignment_distance(situation):

        boxes = situation.agent_set.agents

        if len(boxes) > 1:
            alignment_distance_list = [BoxAgent.
                get_alignment_distance(box, box.get_most_aligned_box
                (situation.agent_set)) for box in boxes]
        else:
            alignment_distance_list = [0]

        # return sum of distances
        return reduce(lambda x,y: x+y, alignment_distance_list)

    @staticmethod
    def sum_of_distance_between_gravities(situation):

        boxes = situation.agent_set.agents
        distance_list = [[BoxAgent.get_gravity_distance(boxes[i],
            boxes[j]) for j in range(i, len(boxes))] for i in range(
            len(boxes))]

        # return sum of distances
        return reduce((lambda x,y: x+y), reduce((lambda x,y: x+y),
            distance_list))

    @staticmethod
    def penalize_overlap(situation):
```

```
47          boxes = situation.agent_set.agents
48          penalty = 0
49
50          for i, box1 in enumerate(boxes):
51              for j in range(i+1,len(boxes) - 1):
52                  if BoxAgent.overlap_or_not(boxes[i], boxes[j]):
53                      penalty += 1
54
55          # return sum of distances
56          return penalty * 1000
57
58      @staticmethod
59      def width_difference(situation):
60
61          boxes = situation.agent_set.agents
62          scale = 10
63
64          distance_list = [[abs(boxes[i].get_width() - boxes[j].
                get_width()) * scale for j in range(i, len(boxes))] for
                i in range(len(boxes))]
65
66          # return sum of distances
67          return reduce((lambda x,y: x+y), reduce((lambda x,y: x+y),
                distance_list))
68
69      @staticmethod
70      def height_difference(situation):
71
72          boxes = situation.agent_set.agents
73          scale = 10
74
75          distance_list = [[abs(boxes[i].get_height() - boxes[j].
                get_height()) * scale for j in range(i, len(boxes))] for
                 i in range(len(boxes))]
76
77          # return sum of distances
```

```
78          return reduce((lambda x,y: x+y), reduce((lambda x,y: x+y),
                distance_list))
79
80
81
82
83  class OverlappedAreaObjective(Objective):
84      def __init__(self):
85
86          objective_function = (lambda situation: Objective.
                sum_of_overlapped_area(situation) )
87
88          Objective.__init__(self, 1, objective_function)
89
90
91  class DistanceObjective(Objective):
92      def __init__(self):
93
94          objective_function = (lambda layout: Objective.
                sum_of_distance_between_gravities(layout) + Objective.
                penalize_overlap(layout))
95
96          Objective.__init__(self, 1, objective_function)
97
98  # imports - - - - - - -
99  from layout import *
100 from agent import *
101 from rule import *
102 from condition import *
103 import math
104
105 from compiler.node import *
```

A.2.6 rule.py

```
1   #-*- coding: utf-8 -*-
2
3   # imports - - - - - - -
```

```
4  from condition import *
5  import random
6
7  class Action():
8      pass
9
10 class BoxAction(Action):
11     @classmethod
12     def stay(cls):
13         def _stay(box):
14             pass
15
16         return _stay
17
18     @classmethod
19     def move_horizontally(cls, amount_to_move):
20         def _move_horizontally(box):
21             box.add_x(amount_to_move)
22
23         return _move_horizontally
24
25     @classmethod
26     def move_horizontally_at_random(cls, max_amount):
27         def _move_horizontally_at_random(box):
28             amount_to_move = random.randint(-max_amount, max_amount)
29             box.add_x(amount_to_move)
30
31         return _move_horizontally_at_random
32
33     @classmethod
34     def move_vertically(cls, amount_to_move):
35         def _move_vertically(box):
36             box.add_y(amount_to_move)
37
38         return _move_vertically
39
40     @classmethod
```

```
41      def move_vertically_at_random(cls, max_amount):
42          def _move_vertically_at_random(box):
43              amount_to_move = random.randint(-max_amount, max_amount)
44              box.add_y(amount_to_move)
45
46          return _move_vertically_at_random
47
48      @classmethod
49      def change_width(cls, amount):
50          def _change_width(box):
51              box.add_width(amount)
52
53          return _change_width
54
55      @classmethod
56      def change_width_at_random(cls, max_amount):
57          def _change_width_at_random(box):
58              amount_to_change = random.randint(-max_amount,
                    max_amount)
59              box.add_width(amount_to_change)
60
61          return _change_width_at_random
62
63      @classmethod
64      def change_height(cls, amount):
65          def _change_height(box):
66              box.add_height(amount)
67
68          return _change_height
69
70      @classmethod
71      def change_height_at_random(cls, max_amount):
72          def _change_height_at_random(box):
73              amount_to_change = random.randint(-max_amount,
                    max_amount)
74              box.add_height(amount_to_change)
75
```

```
76          return _change_height_at_random
77
78
79      @classmethod
80      def unify_width_to_nearest_box(cls, layout):
81
82          def _unify_width_to_nearest_box(box):
83              nearest_box = box.get_nearest_box(layout)
84              box.set_width(nearest_box.get_width())
85
86          return _unify_width_to_nearest_box
87
88      @classmethod
89      def unify_height_to_nearest_box(cls, layout):
90
91          def _unify_height_to_nearest_box(box):
92              nearest_box = box.get_nearest_box(layout)
93              box.set_height(nearest_box.get_height())
94
95          return _unify_height_to_nearest_box
96
97      @classmethod
98      def unify_size_to_most_aligned_box(cls, layout):
99
100         def _unify_size_to_most_aligned_box(box):
101             most_aligned_box = box.get_most_aligned_box(layout)
102             x_difference = box.get_position_difference(
                    most_aligned_box, 0)
103             y_difference = box.get_position_difference(
                    most_aligned_box, 1)
104
105             if x_difference < y_difference:
106                 box.unify_width_and_align(most_aligned_box)
107             else:
108                 box.unify_height_and_align(most_aligned_box)
109
110         return _unify_size_to_most_aligned_box
```

```
111
112     @classmethod
113     def align_to_nearest_box(cls, layout):
114
115         def _align_to_nearest_box(box):
116             nearest_box = box.get_nearest_box(layout)
117
118             difference_x = abs(box.get_x() - nearest_box.get_x())
119             difference_y = abs(box.get_y() - nearest_box.get_y())
120
121             if difference_x < difference_y:
122                 box.align_left(nearest_box)
123             else:
124                 box.align_top(nearest_box)
125
126         return _align_to_nearest_box
127
128     @classmethod
129     def align_top_to_nearest_box(cls, layout):
130
131         def _align_top_to_nearest_box(box):
132             nearest_box = box.get_nearest_box(layout)
133             box.align_top(nearest_box)
134
135         return _align_top_to_nearest_box
136
137     @classmethod
138     def align_left_to_nearest_box(cls, layout):
139
140         def _align_left_to_nearest_box(box):
141             nearest_box = box.get_nearest_box(layout)
142             box.align_left(nearest_box)
143
144         return _align_left_to_nearest_box
145
146     @classmethod
147     def stay_away_to_nearest_box(cls, layout):
```

```
148
149         def _stay_away_to_nearest_box(box):
150             nearest_box = box.get_nearest_box(layout)
151             box.stay_away(nearest_box)
152
153         return _stay_away_to_nearest_box
154
155     @classmethod
156     def approach_to_nearest_box(cls, layout):
157
158         def _approach_to_nearest_box(box):
159             nearest_box = box.get_nearest_box(layout)
160             box.approach(nearest_box)
161
162         return _approach_to_nearest_box
163
164     @classmethod
165     def space_nearest_box(cls, amount, layout,compare=(lambda x1, x2
        : x1 < x2)):
166         """Set space with the nearest box designated value"""
167
168         def _space_nearest_box(box):
169             nearest_box = box.get_nearest_box(layout)
170             x_difference = box.get_position_difference(nearest_box,
                0)
171             y_difference = box.get_position_difference(nearest_box,
                1)
172
173             if compare(x_difference, y_difference):
174                 box.make_vertical_space(nearest_box, amount)
175             else:
176                 box.make_horizontal_space(nearest_box, amount)
177
178         return _space_nearest_box
179
180     @classmethod
```

```
    def space_most_aligned_box(cls, amount, layout, compare=(lambda
        x1, x2: x1 < x2)):
        """Set space with the most aligned box designated value"""

        def _space_most_aligned_box(box):
            most_aligned_box = box.get_most_aligned_box(layout)
            x_difference = box.get_position_difference(
                most_aligned_box, 0)
            y_difference = box.get_position_difference(
                most_aligned_box, 1)

            if compare(x_difference,  y_difference):
                box.make_vertical_space(most_aligned_box, amount)
            else:
                box.make_horizontal_space(most_aligned_box, amount)

        return _space_most_aligned_box

class Rule():
    initial_strength = 0.5

    def __init__(self, condition=Condition(), action=BoxAction.stay
        (), strength=initial_strength):
        self.condition = condition
        self.action = action
        self.strength = strength

    def set_strength(self, strength):

        if 1.0 <= strength:
            strength = 1.0
        elif strength <= 0:
            strength = 0

        self.strength = strength

    def add_strength(self, amount):
```

```
214         self.set_strength(self.strength +  amount)
215
216     def get_copy(self):
217         return Rule(self.condition.get_copy(), self.action, self.
                strength)
218
219     def reinforce(self, episode, reward_function):
220         """Increase(decrease) strength."""
221
222         self.set_strength(self.strength + reward_function(episode))
223
224     @classmethod
225     #take layout as argument because some actions need layout for
            argument
226     def generate_rule_with_random_action(cls, condition, agent_set):
227         pass
228
229 class BoxRule(Rule):
230     @classmethod
231     # take elayout as argument because some actions need layout for
            argument
232     def generate_rule_with_random_action(cls, condition, layout):
233         action_candidates = [BoxAction.stay(),
234                              BoxAction.move_horizontally(10),
235                              BoxAction.move_vertically(10),
236                              BoxAction.move_horizontally(-10),
237                              BoxAction.move_vertically(-10),
238                              BoxAction.change_width(10),
239                              BoxAction.change_width(-10),
240                              BoxAction.change_height(10),
241                              BoxAction.change_height(-10),
242                              BoxAction.align_to_nearest_box(layout),
243                              BoxAction.space_nearest_box(20, layout,
                                     (lambda x1, x2: x1 < x2)),
244                              BoxAction.space_nearest_box(20, layout,
                                     (lambda x1, x2: x1 < x2)),
```

```
                                BoxAction.space_nearest_box(20, layout,
                                    (lambda x1, x2: x1 > x2)),
                                BoxAction.space_most_aligned_box(20,
                                   layout, (lambda x1, x2: x1 < x2)),
                                BoxAction.space_most_aligned_box(20,
                                   layout, (lambda x1, x2: x1 < x2)),
                                BoxAction.space_most_aligned_box(20,
                                   layout, (lambda x1, x2: x1 > x2)),
                                BoxAction.
                                   unify_size_to_most_aligned_box(
                                   layout)]

        action = random.choice(action_candidates)
        return Rule(condition, action)


class SampleRule():
    def __init__(self):
        self.condition = Condition([Condition.nearby_object(60)])
        self.action = BoxAction.stay()
        self.strength = Rule.initial_strength


# imports - - - - - - -
```

A.2.7 condition.py

```
#-*- coding: utf-8 -*-


class Situation():
    def __init__(self, **kwargs):
        self.agent_set = kwargs.get('agent_set', None)
        self.agent = kwargs.get('agent', None)

    def get_copy(self):
        return Situation(agent_set=self.agent_set.get_copy(), agent=
            self.agent.get_copy())
```

```
11
12 def bool_for_layout_and_box(agent_set, agent, bool_function):
13     """Return True if relation between given box and any box in
           given layout satisfies the given bool function"""
14
15     agents = agent_set.agents[:]
16
17     #ignore exception
18     try:
19         agents.remove(agent)
20     except:
21         pass
22
23     result = False
24     for i in range(len(agents)):
25         if bool_function(agent, agents[i]) == True:
26             result = True
27             break
28
29     return result
30
31 class Condition():
32     # we call function which reperesents one condition as "condfun"
33     def __init__(self, condfuns=[], and_or=1):
34         self.condfuns = condfuns
35         self.and_or = and_or
36
37     def evaluate(self, situation, latitude=0):
38         """Evaluate the condition with given situation and return
               True or False."""
39
40         if self.condfuns != []:
41             results = [condfun.evaluate_function(situation) for
                   condfun in self.condfuns]
42             number_of_true = reduce(lambda a,b: a + b, results) #
                   Bool value is calculatable (True = 1, False = 0)
43             number_of_false = len(results) - number_of_true
```

```
44
45              # case of "and"
46              if self.and_or == 1:
47                  return number_of_false <= latitude
48              else:
49                  return number_of_true >= 1
50          else:
51              return True
52
53      def get_copy(self):
54          return Condition(self.condfuns[:], self.and_or)
55
56      def get_size(self):
57          return len(self.condfuns)
58
59      def get_sum_of_constraint_objective(self, situation):
60          objective_value_list = [condfun.get_objective_value(
                situation) for condfun in self.condfuns]
61
62          return sum(objective_value_list)
63
64      def add_condfun(self, condfun):
65          self.condfuns.append(condfun)
66
67      def remove_condfun_by_name(self, name):
68          for i, condfun in enumerate(self.condfuns):
69              if condfun.function.__name__ == name:
70                  self.remove_condfun(i)
71                  break
72
73      def remove_random_condfun(self):
74          index = random.randint(0, len(self.condfuns) - 1)
75          self.remove_condfun(index)
76
77      def remove_condfun(self, index):
78          del(self.condfuns[index])
79
```

```
80      @classmethod
81      def make_condition(cls, situation):
82          """Make_Condition_instance_which_represents_given_situation(
                layout_and_box)"""
83
84          condfun_candidates = [BoxCondFun.in_the_edge(),
85                                BoxCondFun.having_overlapped_box()]
86
87          partition_unit_size = 25
88          for i in range(15):
89              condfun_candidates.append(BoxCondFun.width_constraint(i
                    * partition_unit_size, (i+1) * partition_unit_size -
                     1))
90              condfun_candidates.append(BoxCondFun.height_constraint(i
                     * partition_unit_size, (i+1) * partition_unit_size
                    - 1))
91
92          distance_unit_size = 100
93          for i in range(1,5):
94              condfun_candidates.append(BoxCondFun.
                    having_box_in_given_distance(i * distance_unit_size)
                    )
95              condfun_candidates.append(BoxCondFun.
                    keeping_given_distance_from_box(i *
                    distance_unit_size))
96
97          # add agent condition to candidate
98          if situation.agent:
99              condfun_candidates.extend(situation.agent.condition.
                    condfuns)
100
101         # extract which matches given state(layout and box)
102         matched_condfuns = [condfun for condfun in
                condfun_candidates if condfun.evaluate_function(
                situation) == True]
103
104         and_or = 1 # represents "and"
```

```
105         condition = Condition(matched_condfuns, and_or)
106 
107         # remove if condition has a lot of condfuns (loosening
                condition)
108         condufns_max = 3
109         if condufns_max < condition.get_size():
110             condition.remove_random_condfun()
111 
112         return condition
113 
114 class BoxCondition(Condition):
115     pass
116 
117 class CondFun():
118     def __init__(self, function, objective=None, soft_hard=1):
119 
120         # to avoid import error, avoid to use initial value
121         if objective == None: objective = Objective()
122 
123         self.set_function(function)
124         self.set_objective(objective)
125         self.set_soft_hard(soft_hard)
126 
127     def set_function(self, function):
128         self.function = function
129 
130     def set_objective(self, objective):
131         self.objective = objective
132 
133     def set_soft_hard(self, soft_hard):
134         self.soft_hard = soft_hard
135 
136     def get_objective_value(self, situation):
137         return self.objective.function(situation)
138 
139     def evaluate_function(self, situation):
140         if self.soft_hard == 1:
```

```
141             return self.function(situation)
142         else:
143             return True
144 
145 class BoxCondFun(CondFun):
146     def __init__(self, function, objective=(lambda x: 0)):
147         CondFun.__init__(self, function, objective)
148 
149     # # # CONDFUNS # # #
150 
151     @classmethod
152     def width_constraint(cls, lower_limit=None, upper_limit=None,
        soft_hard=1):
153 
154         if lower_limit == None: lower_limit = 0
155         if upper_limit == None: upper_limit = 9999
156 
157         def _width_constraint(situation):
158             box = situation.agent
159             return lower_limit <= box.get_width() and box.get_width
                () <= upper_limit
160 
161         def _width_constraint_objective(situation):
162             box = situation.agent
163             if box.get_width() < lower_limit:
164                 return abs(lower_limit - box.get_width())
165             elif upper_limit < box.get_width():
166                 return abs(box.get_width() - upper_limit)
167             else:
168                 return 0
169 
170         _width_constraint.func_dict['lower'] = lower_limit
171         _width_constraint.func_dict['upper'] = upper_limit
172         _width_constraint.func_dict['soft_hard'] = soft_hard
173 
174         return CondFun(_width_constraint, Objective(1,
            _width_constraint_objective), soft_hard)
```

```
175
176     @classmethod
177     def height_constraint(cls, lower_limit=None, upper_limit=None,
            soft_hard=1):
178
179         if lower_limit == None: lower_limit = 0
180         if upper_limit == None: upper_limit = 9999
181
182         def _height_constraint(situation):
183             box = situation.agent
184             return lower_limit <= box.get_height() and box.
                    get_height() <= upper_limit
185
186         def _height_constraint_objective(situation):
187             box = situation.agent
188             if box.get_height() < lower_limit:
189                 return abs(lower_limit - box.get_height())
190             elif upper_limit < box.get_height():
191                 return abs(box.get_height() - upper_limit)
192             else:
193                 return 0
194
195         _height_constraint.func_dict['lower'] = lower_limit
196         _height_constraint.func_dict['upper'] = upper_limit
197         _height_constraint.func_dict['soft_hard'] = soft_hard
198
199         return CondFun(_height_constraint, Objective(1,
                _height_constraint_objective), soft_hard)
200
201     @classmethod
202     def x_constraint(cls, lower_limit=None, upper_limit=None,
            soft_hard=1):
203
204         if lower_limit == None: lower_limit = 0
205         if upper_limit == None: upper_limit = 9999
206
207         def _x_constraint(situation):
```

```
208             box = situation.agent
209             return lower_limit <= box.get_x() and box.get_x() <=
                    upper_limit
210
211         def _x_constraint_objective(situation):
212             box = situation.agent
213             if box.get_x() < lower_limit:
214                 return abs(lower_limit - box.get_x())
215             elif upper_limit < box.get_x():
216                 return abs(box.get_x() - upper_limit)
217             else:
218                 return 0
219
220         _x_constraint.func_dict['lower'] = lower_limit
221         _x_constraint.func_dict['upper'] = upper_limit
222         _x_constraint.func_dict['soft_hard'] = soft_hard
223
224         return CondFun(_x_constraint, Objective(1,
                _x_constraint_objective), soft_hard)
225
226     @classmethod
227     def y_constraint(cls, lower_limit=None, upper_limit=None,
            soft_hard=1):
228
229         if lower_limit == None: lower_limit = 0
230         if upper_limit == None: upper_limit = 9999
231
232         def _y_constraint(situation):
233             box = situation.agent
234             return lower_limit <= box.get_y() and box.get_y() <=
                    upper_limit
235
236         def _y_constraint_objective(situation):
237             box = situation.agent
238             if box.get_y() < lower_limit:
239                 return abs(lower_limit - box.get_y())
240             elif upper_limit < box.get_y():
```

```
241                 return abs(box.get_y() - upper_limit)
242             else:
243                 return 0
244 
245         _y_constraint.func_dict['lower'] = lower_limit
246         _y_constraint.func_dict['upper'] = upper_limit
247         _y_constraint.func_dict['soft_hard'] = soft_hard
248 
249         return CondFun(_y_constraint, Objective(1,
                _y_constraint_objective), soft_hard)
250 
251     @classmethod
252     def x_end_constraint(cls, lower_limit, upper_limit=9999,
            soft_hard=1):
253         def _x_end_constraint(situation):
254             box = situation.agent
255             return lower_limit <= box.get_x() + box.get_width() and
                    box.get_x() + box.get_width() <= upper_limit
256 
257         def _x_end_constraint_objective(situation):
258             box = situation.agent
259             if box.get_x() + box.get_width() < lower_limit:
260                 return abs(lower_limit - (box.get_x() + box.
                        get_width()))
261             elif upper_limit < box.get_x() + box.get_width():
262                 return abs((box.get_x() + box.get_width()) -
                        upper_limit)
263             else:
264                 return 0
265 
266         return CondFun(_x_end_constraint, Objective(1,
                _x_end_constraint_objective), soft_hard)
267 
268     @classmethod
269     def y_end_constraint(cls, lower_limit, upper_limit=9999,
            soft_hard=1):
270         def _y_end_constraint(situation):
```

```
271             box = situation.agent
272             return lower_limit <= box.get_y() + box.get_height() and
                    box.get_y() + box.get_height() <= upper_limit
273
274         def _y_end_constraint_objective(situation):
275             box = situation.agent
276             if box.get_y() + box.get_height() < lower_limit:
277                 return abs(lower_limit - (box.get_y() + box.
                        get_height()))
278             elif upper_limit < box.get_y():
279                 return abs((box.get_y() + box.get_height()) -
                        upper_limit)
280             else:
281                 return 0
282
283         return CondFun(_y_end_constraint, Objective(1,
                _y_end_constraint_objective), soft_hard)
284
285     @classmethod
286     def in_the_edge(cls):
287         def _in_the_edge(situation):
288             layout = situation.agent_set
289             boxes = layout.agents
290
291             result = True
292             for box in boxes:
293                 if box.off_the_edge_or_not() == True:
294                     result = False
295                     break
296
297             return result
298
299         return CondFun(_in_the_edge)
300
301     @classmethod
302     def no_overlap(cls):
303         def _no_overlap(situation):
```

```
304             layout = situation.agent_set
305
306             boxes = layout.agents
307             result = True
308             for i in range(len(boxes) - 1):
309                 for j in range(i+1, len(boxes)):
310                     if BoxAgent.overlap_or_not(boxes[i], boxes[j])
                            == True:
311                         result = False
312                         break
313
314             return result
315
316         return CondFun(_no_overlap, Objective(1, Objective.
                sum_of_overlapped_area))
317
318     @classmethod
319     def all_aligned(cls):
320         # return False if box which does not align to any box exists
                , otherwise True
321         def _all_aligned(situation):
322             layout = situation.agent_set
323             boxes = layout.agents
324
325             if len(boxes) != 1:
326                 list_of_number_of_aligned_box = []
327                 for i in range(len(boxes)):
328
329                     number_of_aligned_box = 0
330                     for j in range(len(boxes)):
331                         if BoxAgent.aligned_or_not(boxes[i], boxes[j
                                ]) == True:
332                             number_of_aligned_box += 1
333
334                     # minus 1 because every box aligned to the box
                            itself
```

```
335                     list_of_number_of_aligned_box.append(
                            number_of_aligned_box - 1)
336
337                 list_of_having_aligned_box_or_not = map((lambda x: x
                        >= 1), list_of_number_of_aligned_box)
338
339                 # return True only if all the element is True
340                 return reduce((lambda b1,b2: b1 and b2),
                        list_of_having_aligned_box_or_not)
341             else:
342                 return True
343
344         return CondFun(_all_aligned, Objective(1, Objective.
                sum_of_alignment_distance))
345
346     @classmethod
347     def width_unification(cls, soft_hard=1):
348         def _width_unification(situation):
349             layout = situation.agent_set
350             boxes = layout.agents
351
352             if len(boxes) == 1:
353                 return True
354             else:
355                 width_equalities = [box.get_width() == boxes[0].
                        get_width() for box in boxes]
356                 return reduce((lambda b1,b2: b1 and b2),
                        width_equalities)
357
358         return CondFun(_width_unification, Objective(1, Objective.
                width_difference), soft_hard)
359
360     @classmethod
361     def height_unification(cls, soft_hard=1):
362         def _height_unification(situation):
363             layout = situation.agent_set
364             boxes = layout.agents
```

```
365
366             if len(boxes) == 1:
367                 return True
368             else:
369                 height_equalities = [box.get_height() == boxes[0].
                        get_height() for box in boxes]
370                 return reduce((lambda b1,b2: b1 and b2),
                        height_equalities)
371
372         return CondFun(_height_unification, Objective(1, Objective.
                height_difference), soft_hard)
373
374     @classmethod
375     def having_box_in_given_distance(cls, distance):
376         def _having_box_in_given_distance(situation):
377             layout = situation.agent_set
378             box = situation.agent
379
380             bool_function = (lambda box1, box2: BoxAgent.
                    get_gravity_distance(box1, box2) < distance)
381
382             return bool_for_layout_and_box(layout, box,
                    bool_function)
383
384         return CondFun(_having_box_in_given_distance)
385
386     @classmethod
387     def keeping_given_distance_from_box(cls, distance):
388         condfun = BoxCondFun.having_box_in_given_distance(distance)
389         return CondFun((lambda situation: not condfun.
                evaluate_function(situation)))
390
391     @classmethod
392     def having_overlapped_box(cls):
393         def _having_overlapped_box(situation):
394             layout = situation.agent_set
395             box = situation.agent
```

```
396
397             bool_function = BoxAgent.overlap_or_not
398
399             return bool_for_layout_and_box(layout, box,
                    bool_function)
400
401         return CondFun(_having_overlapped_box)
402
403     @classmethod
404     def minimum_member(cls, number):
405         def _minimum_member(situation):
406             agent_set = situation.agent_set
407
408             if agent_set.get_number_of_boxes() >= number:
409                 return True
410             else:
411                 return False
412
413         return CondFun(_minimum_member)
414
415 # imports - - - - - - -
416 from layout import *
417 from rule import *
418 from agent import *
419 from specification import *
420 import random
```