ソフトウェア工学

E1237 森 篤史

1　文章化

- このプログラムはある大きさのイラストロジックの答えを割り出すプログラムである。
- イラストロジックを解き、その解を画面に出力する。
- イラストロジックには解を書き込む解エリアと、ヒントを与える出題エリアがある。
- 解エリアは長方形をしていて、$n \times m$ 個の回答マスによって構成されている。
- 解エリアのそれぞれの出題マスには、塗るマス（黒マス）か塗らないマス（白マス）かの 2 値が入力さ左部に $y \times m$ のマスによって構成されている。
- 上部の出題エリアはそれぞれその列の、左部の出題エリアにはそれぞれその行に塗るマスのヒントとなる数字が書かれている。
- 1 個の数字は連続する黒マスの数を表している。
- 数字が複数ある場合、それぞれが連続で黒マスの数を表し、間には必ず白マスが最低 1 つ入る。

2　名刺の抽出

2.1　仕様書から抽出された名刺

イラストロジック、画面、解エリア、長方形、回答マス、出題マス、出題エリア、黒マス、白マス、数字、列、行

2.2　クラス候補

イラストロジック、解エリア、出題エリア、数字、回答マス、出題マス、列、行

3　関連の洗い出し

図 1 に関連を反映したクラス図を示す。

4　操作の洗い出し

図 2 に簡単な操作を書き込んだクラス図を示す。

5　属性の洗い出し

図 3 に属性を書き込んだクラス図を示す。

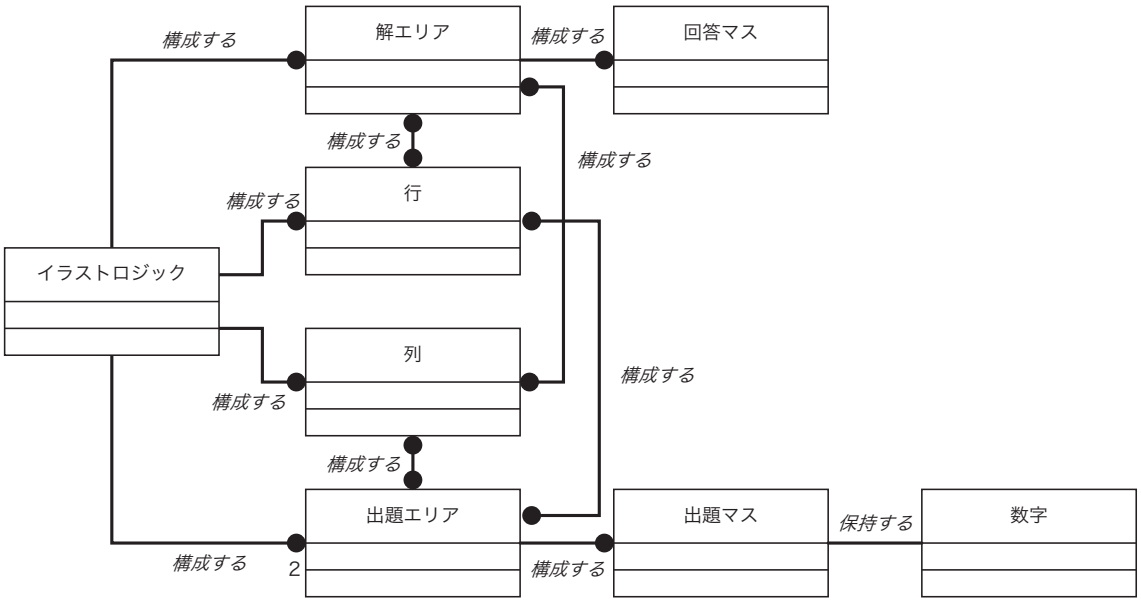

図 1: 関連を反映させたクラス図

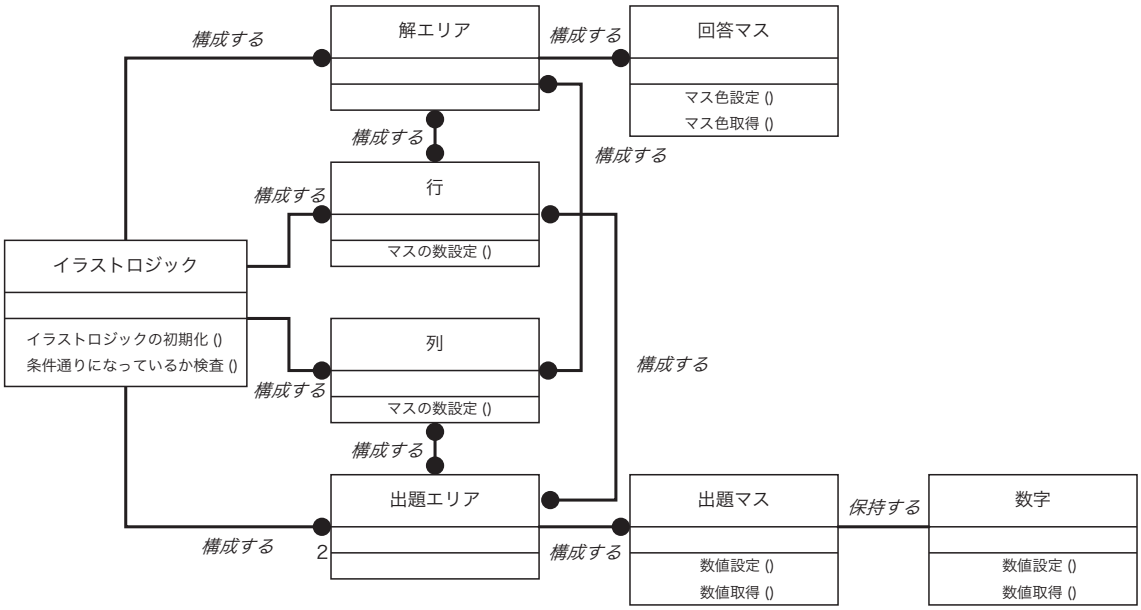

図 2: 簡単な操作を書き込んだクラス図

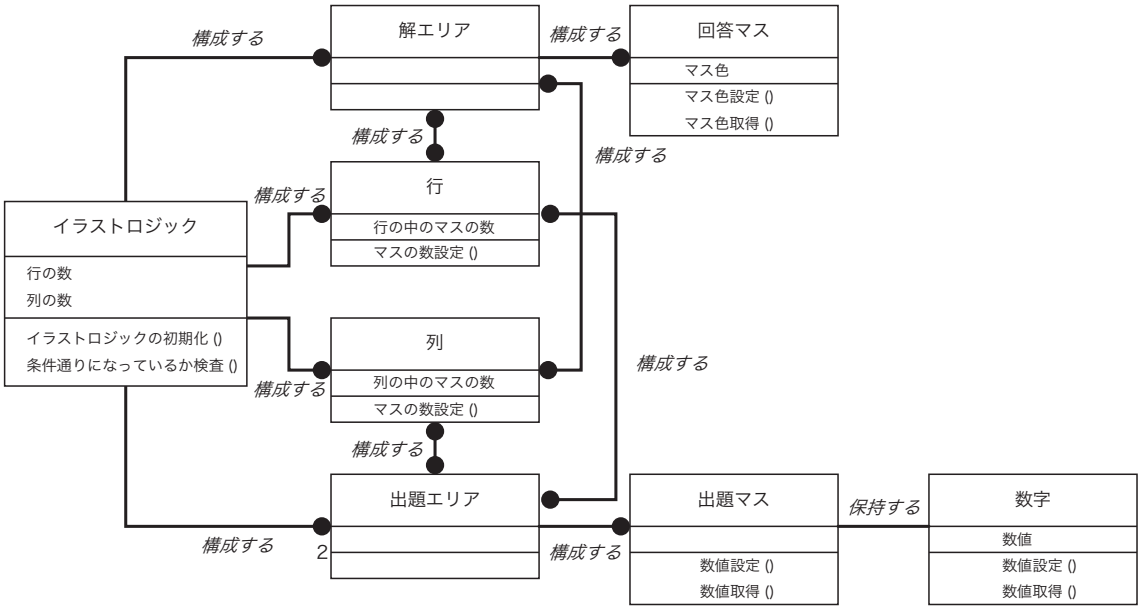

図 3: 属性を書き込んだクラス図

図 4: 属性を書き込んだクラス図

6　動的分析

6.1　状態遷移図

図 4 にアプリケーション全体の状態遷移図を示す。

6.2　シーケンス図

全体の流れとしては以下のようになる。

1. 問題を入力する。
2. 左部出題エリアより 1 行取得する。
3. それをもとに、解エリアのその行で答えが確定するマスに回答を書き込む。
4. 2〜3 を全行で繰り返す。
5. 上部出題エリアより 1 列取得する。
6. それをもとに、解エリアのその列で答えが確定するマスに回答を書き込む。
7. 5〜6 を全列で繰り返す。
8. 全てのマスが確定しているか確認する。確定していれば終了。していなければ 2 に戻る。

図 5 に回答検索のシーケンス図を示す。

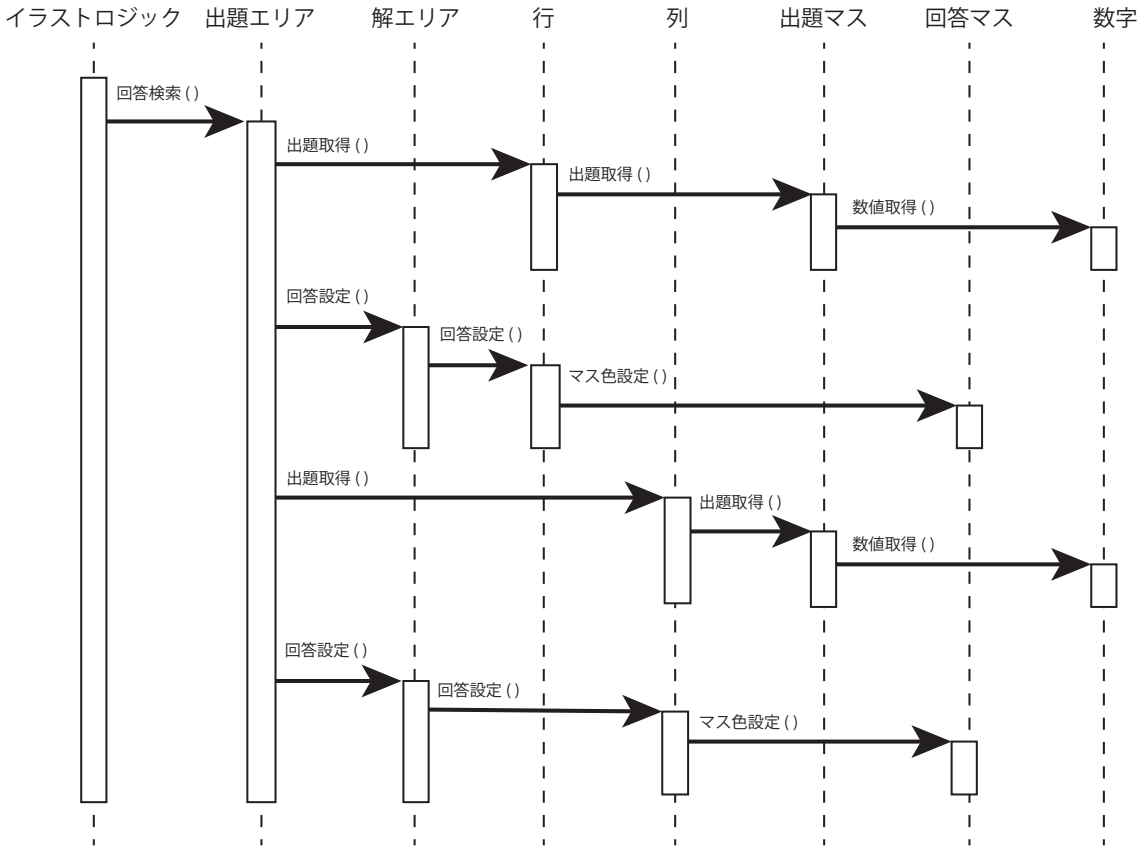

図 5: 回答検索のシーケンス図

6.3　クラス図の変更

図 5 のシーケンス図に合わせたクラス図を 6 に示す。

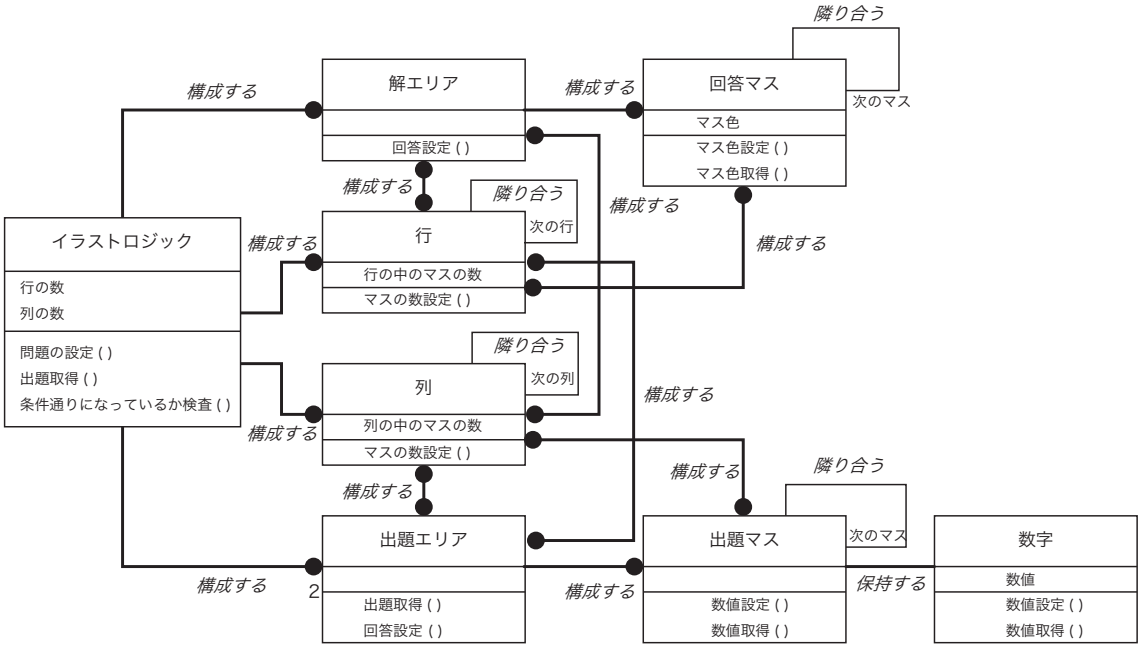

図 6: 属性を書き込んだクラス図

7　クラス図の洗礼

図 7 にクラスの吸収を行ったクラス図を示す。

8　アルゴリズム

出題取得を行った後、回答設定を行うまでのアルゴリズムを以下に記述する。また、解エリアには黒マス、白マスの他に未確定マスがあるとする。

1. 解エリアから一行（一列）を取りし、出題エリアからも同じ行（列）を取り出す。
2. 一行の解エリアで未確定マスがそれぞれ取りうる値の全パターンを用意する。
3. それらパターンを、出題エリアの条件に合うもののみに絞り込む。
4. 絞り込まれたパターン全てで共通の値を保つ場合、その値を確定し黒マスか白マスか解エリアに書き込む。

以上の作業を行で全てやった後、列で全てやり、また行に戻って繰り返す。

9　実装

python を使用し、実装を行った。

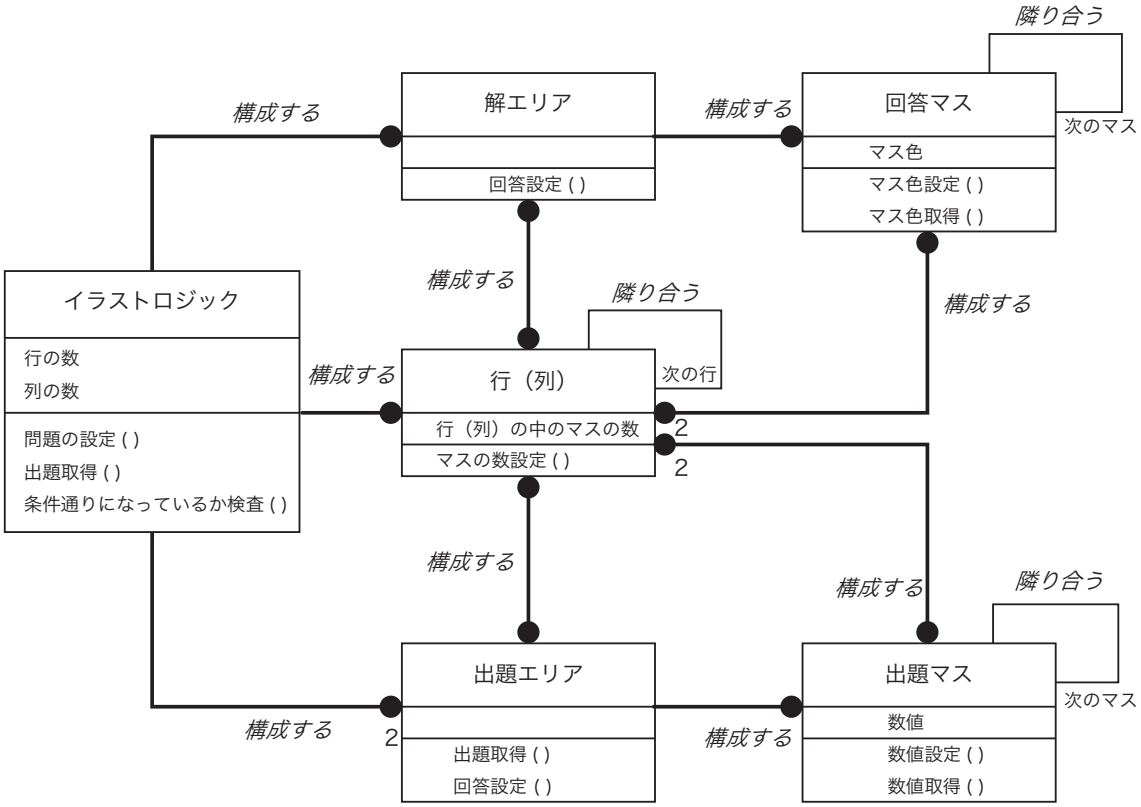

図 7: クラスの吸収を行ったクラス図

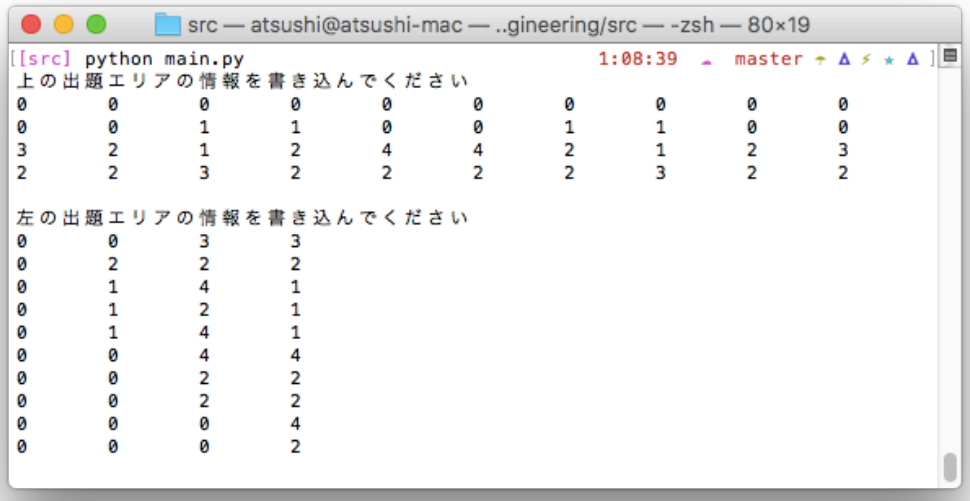

図 8: データの入力方法

9.1 仕様書

python main.py と実行することでプログラムが開始される。まず、図 8 のようにデータを入力する。すると、図 9 のようにイラストロジックを自動で解き始める。

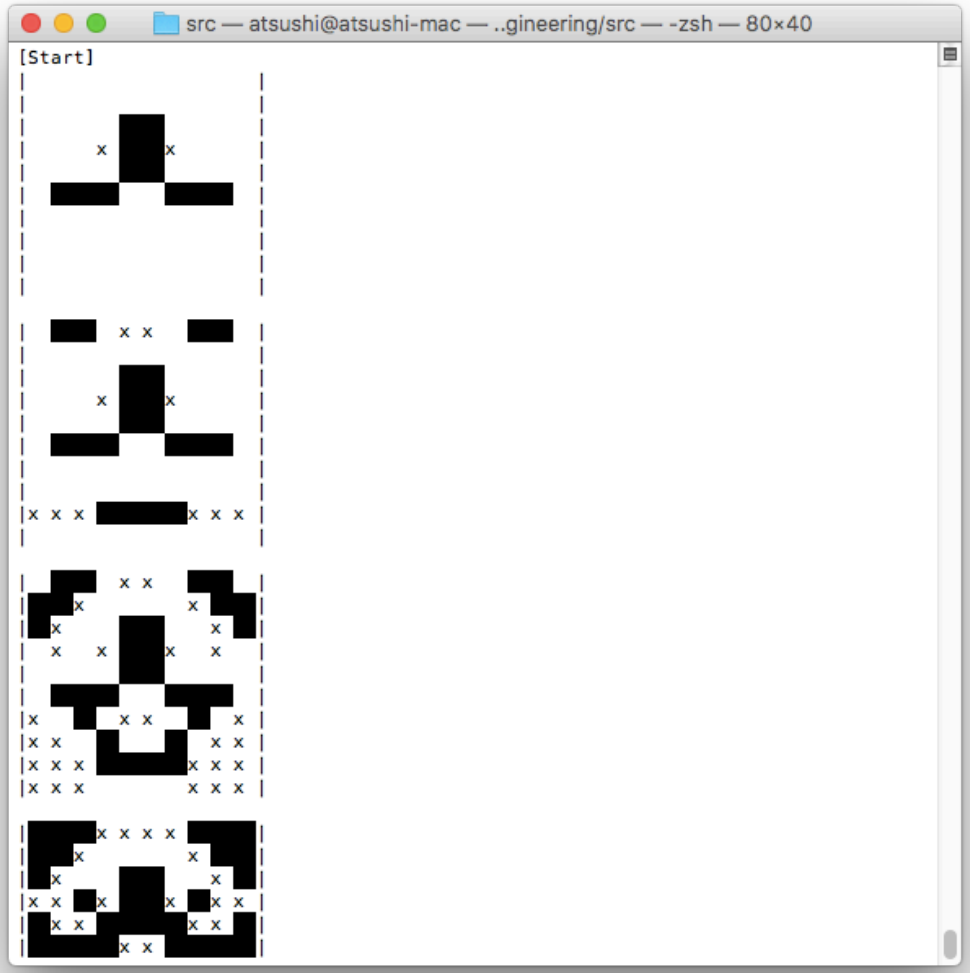

図 9: イラストロジックの回答計算中

最終的に、図 10 のように解けたイラストロジックが表示される。

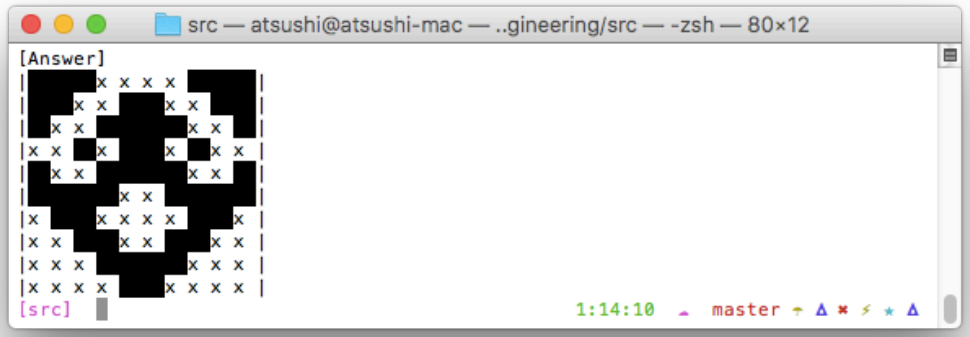

図 10: イラストロジックの解

9.2 ソースコード

以下にソースコードを記述する。

[main.py]

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

import illustration
import sys

if __name__ == '__main__':
    issueUp = []
    issueLeft = []

    print '上の出題エリアの情報を書き込んでください'
    while True:
        input = sys.stdin.readline()
        if input == '\n':
            break
        input = input.rstrip()
        issueUp.append(map(lambda data: int(data), input.split()))

    print '左の出題エリアの情報を書き込んでください'
    while True:
        input = sys.stdin.readline()
        if input == '\n':
            break
        input = input.rstrip()
        issueLeft.append(map(lambda data: int(data), input.split()))

    illustration = illustration.Illustration(issueUp, issueLeft)
    print '[Start]'
    illustration.calc()
    print '[Answer]'
    illustration.show()
```

[illustration.py]

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

import various

class Illustration:
    def __init__(self, issueUp, issueLeft):
        self.__issue = various.Issue(issueUp, issueLeft)
        self.__m = len(issueUp[0])
        self.__n = len(issueLeft)
        self.__answer = various.Answer(self.__m, self.__n)

    def calc(self):
        i = 0
        lineM = various.Line(self.__m)
        lineN = various.Line(self.__n)
        while True:
            upFlag = 0
            leftFlag = 0
            for m in range(self.__m):
                issueLine = self.__issue.get(m, 0)
                answerLine = self.__answer.get(m, 0)
                upFlag += lineM.calc(issueLine, answerLine)

            for n in range(self.__n):
                issueLine = self.__issue.get(n, 1)
                answerLine = self.__answer.get(n, 1)
                leftFlag += lineN.calc(issueLine, answerLine)

            self.show()
            print ''

            if upFlag == 0 and leftFlag == 0:
                break

    def show(self):
        self.__answer.show()
```

[various.py]

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

import sys
import numpy as np
import copy

class IssueCell:
    def __init__(self, data):
        self.__data = data

    def get(self):
        return self.__data

class Issue:
    def __init__(self, issueUp, issueLeft):
        self.__up = np.array(map(lambda line:map(lambda data:IssueCell(data), line), issueUp))
        self.__left = np.array(map(lambda line:map(lambda data:IssueCell(data), line), issueLeft))

    def get(self, target, flag):
        if flag == 0:
            return self.__up[:, target]
        else:
           return self.__left[target, :]

class AnswerCell:
    def __init__(self):
        self.__label = ['  ', 'x ', '\x1b[40m  \x1b[49m']
        self.__data = 0

    def get(self):
        return self.__data

    def getString(self):
        return self.__label[self.get()]

    def set(self, data):
        self.__data = data

class Answer:
    def __init__(self, m, n):
        npdata = np.empty((n, m))
        self.__data = np.array(map(lambda line:map(lambda cell:AnswerCell(), line), npdata))

    def show(self):
        for line in self.__data:
            print '|',
            for item in line:
                sys.stdout.write(item.getString())
            print '|'

    def get(self, target, flag):
        if flag == 0:
            return self.__data[:, target]
        else:
            return self.__data[target, :]

class Line:
    def __init__(self, num):
        npdata = np.empty(num)
        answer = np.array(map(lambda cell:AnswerCell(), npdata))
        self.__candidateInit = self.__candidate(answer, True)

    def calc(self, issueLine, answerLine):
        sum = 0
        candidate = self.__candidate(answerLine)
        match = filter(lambda line:self.__match(line, issueLine), candidate)
        confirm = self.__confirm(match)
        for i in range(len(answerLine)):
            if answerLine[i].get() != confirm[i].get():
                answerLine[i] = confirm[i]
                sum += 1

        return sum

    def __candidate(self, answerLine, force=False):
```

```
        candidateArray = []
        sum = 0
        for cell in answerLine:
            sum += cell.get()
        if sum == 0 and not force:
            return self.__candidateInit

        for cell in answerLine:
            if cell.get() == 0:
                cell.set(1)
                duplicate = map(lambda cell:copy.deepcopy(cell), answerLine)
                candidateArray += self.__candidate(duplicate)
                cell.set(2)
                duplicate = map(lambda cell:copy.deepcopy(cell), answerLine)
                candidateArray += self.__candidate(duplicate)
                cell.set(0)
                return candidateArray

        return [answerLine]

    def __match(self, line, issueLine):
        ans = [0]
        for cell in line:
            if cell.get() == 2:
                ans[-1] += 1
            elif cell.get() == 1 and ans[-1] != 0:
                ans.append(0)

        ans = filter(lambda data: data != 0, ans)
        issue = map(lambda cell: cell.get(), issueLine)
        issue = filter(lambda data: data != 0, issue)
        return ans == issue

    def __confirm(self, match):
        temp = copy.deepcopy(match[0])
        for line in match:
            for i in range(len(line)):
                if temp[i].get() != line[i].get():
                    temp[i].set(0)
        return temp
```

9.3 クラス図の訂正

図 11 に実際に実装を行ったクラス図を示す。

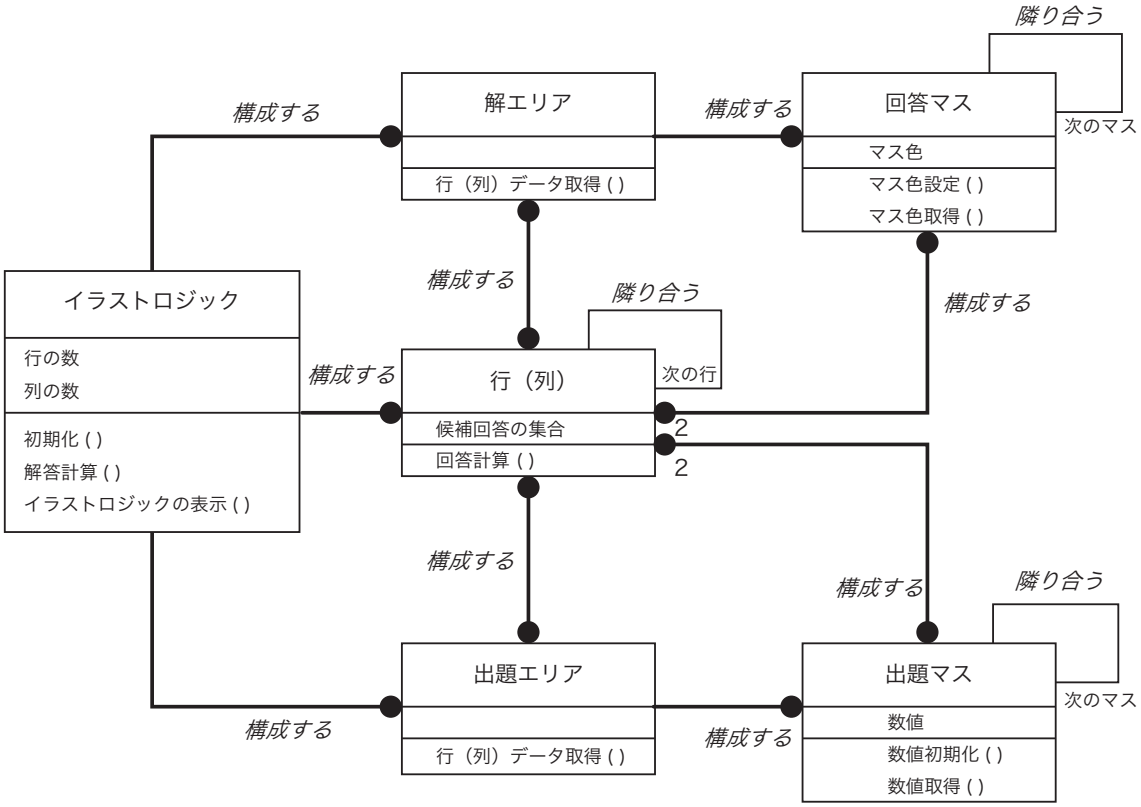

図 11: 実装を行ったクラス図