**АННОТАЦИЯ**

В данном документе приведена пояснительная записка к программе "Kakuro\_Game\_Solver.jar" ("Игра Какуро на основе алгоритма симплексной дифференциальной эволюции"), предназначенной для решение компьютером или пользователем головоломок какуро, сконструированных пользователем в редакторе программы или сгенерированных программой, и сторонних, загружаемых с помощью редактора или текстового файла.

В документе, в разделе "Введение" указано наименование программы, краткое наименование программы и документы, на основании которых ведется разработка.

В разделе "Назначение и область применения" указано функциональное назначение программы, эксплуатационное назначение программы и краткая характеристика области применения программы.

В разделе "Технические характеристики" содержатся следующие подразделы: постановка задачи на разработку программы, описание применяемых математических методов и алгоритмов, алгоритмы функционирования программы и ее интерфейс, организация входных и выходных данных, описание и обоснование выбора состава технических и программных средств.

В разделе "Ожидаемые технико-экономические показатели" указана предполагаемая потребность и полезность разработки, преимущества разработки по сравнению с отечественными и зарубежными образцами или аналогами.

Настоящий документ разработан в соответствии с требованиями:

1) ГОСТ 19.101-77 Виды программ и программных документов [1];

2) ГОСТ 19.102-77 Стадии разработки [2];

3) ГОСТ 19.103-77 Обозначения программ и программных документов [3];

4) ГОСТ 19.104-78 Основные надписи [4];

5) ГОСТ 19.105-78 Общие требования к программным документам [5];

6) ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом [6];

7) ГОСТ 19.404-79 Пояснительная записка. Требования к содержанию и оформлению [7].

Изменения к данному Техническому заданию оформляются согласно ГОСТ 19.603-78 [8], ГОСТ 19.604-78 [9].

СОДЕРЖАНИЕ

1. Введение 5

1.1. Наименование Программы 5

1.2. Документ, на основе которого ведется разработка 5

2. Назначение и область применения 6

2.1. Назначение Программы 6

2.2. Краткая характеристика области применения 6

3. Технические характеристики 7

3.1. Постановка задачи на разработку программы 7

3.2. Применяемые математические методы 8

3.3. Алгоритмы функционирования программы и ее интерфейс 18

3.4. Организация входных и выходных данных 25

3.5. Выбор технических и программных средств 26

4. Ожидаемые технико-экономические показатели 27

5. Источники, использованные при разработке 28

Приложение 1. Таблицы с описанием классов и интерфейсов 30

Приложение 2. Таблицы с описанием методов 32

Лист регистрации изменений 42

**1. ВВЕДЕНИЕ**

**1.1. Наименование Программы**

Название программы: "Игра какуро на основе алгоритма симплексной дифференциальной эволюции" ("Kakuro Game based on Simplex Differential Evolution Algorithm") (далее Программа "Игра какуро", исполняемый файл Kakuro\_Game\_Solver.jar). Изложено описание игры какуро, применяемых алгоритмов и математических методов решения и генерации головоломок какуро, метода оценки сложности игры, выбора технических средств, интерфейса программы, организации входных и выходных данных.

**1.2. Документ, на основе которого ведется разработка**

Приказ № \_\_\_\_\_\_\_\_\_\_\_\_\_ от \_\_\_\_\_\_\_\_\_\_\_ «Об утверждении тем и руководителей курсовых работ студентов 3 курса Факультета компьютерных наук, образовательная программа «Программная инженерия».

**Организация, утвердившая этот документ**: Национальный исследовательский университет «Высшая школа экономики», Факультет компьютерных наук, образовательная программа «Программная инженерия».

**Наименование темы разработки**: "Игра какуро на основе алгоритма симплексной дифференциальной эволюции" ("Kakuro Game based on Simplex Differential Evolution Algorithm").

**2. НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ**

**2.1. Назначение Программы**

Назначение разработки – решение компьютером или пользователем головоломок какуро, сконструированных пользователем в редакторе программы, сгенерированных программой и сторонних, загружаемых с помощью редактора или текстового файла. Головоломка *какуро*, известная также как головоломка пересекающихся сумм, является математическим аналогом кроссворда. Цель игры – заполнить клетки кроссворда цифрами от 1 до 9 так, чтобы сумма цифр в каждом горизонтальном и вертикальном блоке ("слове") была равна числу, указанному для каждого блока. При этом цифры в пределах одного блока не должны повторяться. Краткое наименование программы: "Игра какуро".

**2.2. Краткая характеристика области применения**

Представляется, что Программа "Игра какуро" будет полезна для широкого круга любителей головоломок какуро, а также инженерам и исследователям, связанным с решением задач дискретной оптимизации, кодирования и создания ключей, поскольку нетрудно понять, что задача решения головоломки какуро представляет собой сложную задачу дискретной оптимизации с ограничениями, на которой может быть проверена эффективность того или иного подхода, что и используется в научных работах [10 - 13].

**3. ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ**

**3.1. Постановка задачи на разработку программы**

Целью проекта "Игра Какуро на основе алгоритма симплексной дифференциальной эволюции" является разработка и создание компьютерной программы с удобным интерфейсом для решения, генерации и редактирования головоломок какуро на основе известного алгоритма симплекcной дифференциальной эволюции, адаптируемого для решения задачи дискретной оптимизации. Представляется, что указанный генетический алгоритм обеспечит надежность и достаточно высокую скорость решения головоломок какуро. Программа должна обеспечить решение компьютером или пользователем головоломок какуро, как сконструированных пользователем в редакторе программы или сгенерированных программой, так и сторонних, загружаемых с помощью редактора или текстового файла.

сумма (ключ) горизонтального блока

|  |  |  |
| --- | --- | --- |
| ***а*** | сумма (ключ)  вертикального  блока | ***б*** |

Рис. 1. Пример головоломки какуро с размером поля 9×9 клеток: до решения (***а***), после решения (***б***).

Головоломка *какуро*, известная также как головоломка пересекающихся сумм, является математическим аналогом кроссворда. Прямоугольное поле головоломки поделено горизонтальными и вертикальными линиями на клетки - квадраты, часть которых имеют диагональную линию, соединяющую верхний левый угол с нижним правым углом. Клетка с диагональю, обычно выделенная цветом, может быть пустой, содержать цифру над и/или под диагональю, а указанные цифры называются *ключами* или *суммами* (рис. 1). Последовательность белых клеток, идущая подряд по горизонтали или вертикали и ограниченная в начале и конце клетками с диагональной линией или границей прямоугольного поля, называется *блоком* (рис. 1). Цель игры – заполнить все белые клетки цифрами от 1 до 9 так, чтобы сумма цифр в блоке была равна числу, указанному над диагональю клетки, расположенной слева от горизонтального блока, и под диагональю клетки, расположенной сверху от вертикального блока. При этом цифры в пределах одного блока не должны повторяться (рис. 1 ***б***). Нетрудно понять, что задача решения головоломки какуро представляет собой задачу дискретной оптимизации с ограничениями, относящуюся к NP-полной задаче дискретной оптимизации [10, 11], на основе которой может разрабатываться и проверяться эффективность того или иного нового алгоритма, чему посвящен ряд научных статей [10 - 12] и диссертационная работа [13]. Однако компьютерные программы по решению головоломок какуро в этих работах не приводятся.

В настоящее время не существует достаточно универсальных подходов к решению задач дискретной оптимизации. В то же время разрабатываются новые алгоритмы, высоко эффективные для поиска экстремумов в задачах оптимизации с непрерывными переменными. Одним из таких новых методов оптимизации является новый *алгоритм симплексной дифференциальной эволюции* [14], относящийся к генетическим методам, образующих класс алгоритмов поисковой оптимизации, основанных на математическом моделировании биологических механизмов и процессов в живой природе с помощью принципов популяционной генетики [15]. Генетические методы используются как для решения задач дискретной оптимизации, так и для поиска глобального экстремума функций с непрерывными переменными. Алгоритм симплексной дифференциальной эволюции, разработанный для решения задач оптимизации с непрерывными переменными [14, 16, 17], может оказаться эффективным для дискретной оптимизации. Его эффективность может быть изучена при примере решения популярных в настоящее время головоломок какуро.

**3.2. Применяемые математические методы и алгоритмы программы**

Достижение поставленной цели потребовало выполнения двух основных частей проекта: 1) реализация подхода по математическому описанию задачи решения головоломки какуро как задачи дискретной оптимизации с ограничениями и 2) реализация в программе алгоритма симплексной дифференциальной эволюционной оптимизации и его адаптация на задачи дискретной оптимизации для решения головоломок какуро.

***3.2.1. Задача решения головоломки какуро как задача дискретной оптимизации***

***с ограничениями***

Пусть *hK* - ширина прямоугольного поля *K* головоломки какуро, а *vK* - высота поля. Белые клетки поля (рис. 2) в количестве *m* являются неизвестными переменными *x1, x2, ..., xm*, принимающими значения в интервале от *xmin* = 1 до *xmax* = 9. Белые клетки составляют *W* = *H* + *V* блоков цифр ("слов" из цифр, например, на рис. 1 *W =* 30), *H* из которых являются горизонтальными блоками (на рис. 1 *H* = 16), и *V* из которых являются вертикальными блоками (на рис. 1 *V* = 14). Сумма цифр каждого блока *wk* (*k* = 1, 2, ..., *W*) должна быть равна указанной перед блоком величине *tk* – ключу (сумме блока), например, на рис. 1 для первого горизонтального блока: 13 = 7 + 6, для первого вертикального блока: 16 = 1 + 8 + 3 + 4. Как результат решения головоломки, для каждого блока *wk* должно выполняться равенство:

и в каждом блоке *wk* не может быть одинаковых цифр:

, ∈ , *i ≠ j* (2)

Например, на рис. 3 для первого горизонтального блока: *x1* + *x2* = 13 и *x1* ≠ *x2*.

|  |
| --- |
| *x6*  *x2*  *x4*  *x3*  *...*  *...*  *x44*  *x7*  *x5*  *x1* |

Рис. 2. Белые клетки поля головоломки какуро являются неизвестными переменными *x1, x2, ..., xm*, здесь *m* = 44.

1.87

-0.30

log*K*BI2, exp

Таким образом, задача решения головоломки какуро состоит в том, чтобы соотношения (1) и (2) были выполнены для всех *W* блоков цифр. Система *W* линейных уравнений (1) с *m* неизвестными целыми числами *x1, x2, ..., xm* с ограничениями (2) сведена к поиску минимума нелинейной функции от дискретных переменных для того, чтобы применить алгоритм симплексной дифференциальной эволюционной оптимизации к решению цифрового кроссворда какуро:

Здесь - символ Кронекера – индикатор равенства элементов *xi* и *xj*, т.е. эта функция равна 1, если *xi* = *xj*, и 0 в противном случае. Правая двойная сумма в (3) является штрафной функцией на возможные равенства цифр в блоке. Параметр > 0 является весовым коэффициентом на равенства *xi* = *xj*, он был оптимизирован, чтобы обеспечить наибольшую скорость решения задачи какуро. Оптимальное значение выбрано равным двум при переборе его значений от 1 до 10. Оптимизация параметра выполнялась по оценке скорости решения 15 различных головоломок какуро с число неизвестных *m* от 20 до 116. Для правильно решенной головоломки какуро минимум функции (3) должен быть равен нулю.

Готовая форма какуро (рис. 1а) может быть использована для генерации новой головоломки, заменяя в форме неизвестные переменные *x1, x2, ..., xm*, и ключи . Для этого достаточно сгенерировать в каждом блоке новые цифры без повторов, а их суммы составят ключи, которые надо вписать в соответствующие клетки с диагоналями для последующего решения головоломки. В этом случае задача генерации какуро сводится к поиску минимума ( = 0) значительно более простой функции :

а необходимые суммы затем вычисляются и подставляются в таблицу какуро:

где *k* = 1, 2, ..., *W.* Очевидно, что сгенерированная головоломка всегда имеет решение, однако, оно может быть не единственным.

***3.2.2. Алгоритм разложения числа на слагаемые, исключая равенство слагаемых***

Для задания начального приближения решения задачи какуро методом симплексной дифференциальной эволюционной оптимизации и для проверки корректности головоломки какуро на основе алгоритма генерации разбиений числа [19] был разработан алгоритм представления числа wSum в виде суммы из wLen чисел из множества от 1 до 9, исключая равные числа в сумме. Алгоритм реализован в виде рекурсии:

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// decomposition of wSum in the 1, 2, ..., 9 summands

// where no digit is duplicated

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

public static void wDecomp(int wSum, int wLen, int k, int i) {

if ( wSum < 0 ) return;

// wSum = 0 indicates that array term[] includes some decomposition of n

if ( wSum == 0 ) {

if (i == wLen) {

for (int j = 0; j < i; j++) {

wSet[iset][j] = word[j];

}

iset += 1;

}

} else {

if (wSum >= k) {

// i-th summand

word[i] = k;

if (k >= 1) wDecomp(wSum - k, wLen, k - 1, i + 1);

}

if ( k > 1) {

wDecomp(wSum, wLen, k - 1, i);

}

}

} // wDecomp

Время выполнения алгоритма ~*O*(wSum). Пример вызова метода:

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// get all decompositions of wSum in wLen summands

// having no duplicates

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

public static void genClues(int wSum, int wLen) {

if (wLen == 1) {

iset = 1;

wSet[0][0] = wSum;

return;

}

iset = 0;

int max = 9;

int k;

int i = 0;

if (wSum > max) {

k = max;

} else {

k = wSum - 1;

}

wDecomp(wSum, wLen, k, i);

} // genClues

***3.2.3. Алгоритм симплексной дифференциальной эволюции решения головоломки какуро***

В разделе 3.2.1 было показано, что задача решения головоломки какуро может быть сведена к поиску минимума нелинейной функции (3) от дискретных переменных. Для решения задачи используется эффективный алгоритм симплексной дифференциальной эволюционной оптимизации (Simplex Differential Evolution = SDE) [14, 16, 17] поиска минимума нелинейных функций многих переменных. Потребовалась адаптация алгоритма SDE для решения задачи дискретной оптимизации. Об изменениях алгоритма будет указано ниже по мере изложения шагов работы алгоритма.

Итак, имеется дискретная функция (3) от некоторого числа независимых переменных ***X*** = { *x1, x2, ..., xm* }. Необходимо найти ее минимум (*ymin* = 0) на отрезке переменных

*xj, min* ≤ *xj* ≤ *xj, max*, *j* = 1, 2, …, *m*.

В случае задачи какуро *xj, min* = 1 и *xj, max* = 9, ∀*j*. Поиск минимума функции выполняется методом ***симплексной дифференциальной эволюционной оптимизации*** (SDE). Так как алгоритм является стохастическим, основанном на генерации случайных начальных приближений решения, не всякое начальное приближение может привести к глобально минимуму *ymin* = 0. Поэтому для решения задачи какуро алгоритм SDE выполняется несколько раз с целью достижения глобального минимума. Заданное максимальное число вызовов алгоритма SDE называется число цивилизаций *Cmax*. По умолчанию для решения головоломок какуро *Cmax* = 24.

*Начало цикла* CCIVIL. Счетчик цикла *Ccurrent* изменяется от 1 до *Cmax*.

1) С использованием случайных чисел генерируют четное число *n* векторов ***X*** начальных приближений к решению задачи, и вычисляются значения функции *y* для каждого вектора:

*y1*, *x11*, *x12*, ..., *x1m*

*y2*, *x21*, *x22*, ..., *x2m*

*...*

*yi*, *xi1*, *xi2*, ..., *xim*

*...*

*yn*, *xn1*, *xn2*, ..., *xnm,*

где

*xij* = *xj, min* + *rnd*(*xj, max*)

и *i* = 1, 2, …, *n*, *j* = 1, 2, …, *m*. Функция *rnd* генерирует случайное целое число на полуотрезке [0, *xj, max*). Значение *n* оптимизировано для задачи какуро в процессе оценки скорости решения различных кроссвордов какуро с размером поля от 5×5 до 14×14. В итоге по умолчанию выбрано значение *n* равное 16.*m*. В зависимости от скорости решения задачи какуро (3) величину *n* рекомендуется выбирать на отрезке [4.*m*, 60.*m*]. В алгоритме SDE *n* векторов ***X*** называется популяцией.

2) Затем выполняется предварительная оптимизация сгенерированных значений переменных ***X***, чтобы их исправленные значения обеспечивали более низкие величины функции *y*, чем до оптимизации. В исходном алгоритме [14] эта предварительная оптимизация выполняется *n* раз и всякий раз для одной точки ***Xs*** среди случайно выбранного набора векторов ***Xi*** (i = 1, 2, ..., *m*+1), - симплекса в *m-*мерном пространстве, - методом однократного цикла нелинейного симплекс-метода Нелдера и Мида [18]. Однако в случае задачи какуро алгоритм Нелдера и Мида оказался совершенно не эффективным вследствие дискретности искомых переменных.

В курсовой работе был разработан следующий ***алгоритм предварительной оптимизации*** сгенерированных значений переменных ***X***.

*Начало цикла* CI, который выполняется для всех горизонтальных блоков какуро, если число *Ccurrent* четное, иначе он выполняется для всех вертикальных блоков. Для ключевой суммы блока *k*, состоящего из *L* клеток, генерируется *d* ее возможных разложений *D*1L, *D*2L, ..., *D*dL на сумму *L* цифр из множества {1, 2, ..., 9}, исключая одинаковые цифр в любом разложении. Для этого используется алгоритм из пункта 3.2.2.

*Начало цикла* CII. Порядок цифр в каждом разложении *DjL* случайным образом перемешивается. Имея *n* векторов ***X***, последовательно в каждом векторе ***X*** на значения слагаемых очередного разложения *DjL* заменяются те компоненты вектора, которые еще не заменены и соответствуют блоку *k*. Если замена выполнена для всех *D*1L, *D*2L, ..., *D*dL, но множество *n* не пройдено, перейти на начало цикла CII, иначе выйти из цикла.

*Конец цикла* CII.

*Конец цикла* CI.

Среди оптимизированных *n* векторов ***X*** выбирают их "элитную" половину nPM = *n*/2, обеспечивающую самые низкие значения функции *y*. Кроме того, если выполнена очередная итерация цикла CCIVIL, где получен вектор ***Xmin***, обеспечивающий *ymin*(***Xmin***) ≠ 0, то в элитном наборе векторов заменить вектор ***Xmax*** на вектор ***Xmin***, если соответствующая величина *ymax*(***Xmax***) > *ymin*.

Далее выполняется поиск минимума функции *y*, используя и изменяя nPM векторов ***X***. Число nPM называется размером популяции G (generation), а конкретный вектор ***Xi***, соответствующий набору его компонент *xi1*, *xi2*, ..., *xim*, называется членом популяции PM (population member). Определяется индивид популяции ***X****min*, для которого функция имеет минимум *ymin*. В соответствии со значением *m* (число компонент вектора ***X***) значение nPM по умолчанию равно 8.*m*. Это значение оптимизировано в процессе оценки скорости решения различных кроссвордов какуро с размером поля от 5×5 до 14×14. В зависимости от скорости приближения функции *y* к минимуму величина nPM рекомендована к выбору на отрезке [2.*m*, 30.*m*].

3) *Выполняется цикл* CA, счетчик которого Gsum изменяется от 1 до заданного максимального числа поколений Gmax. В зависимости от вида функции (3) Gmax варьируется от нескольких сотен до нескольких тысяч. При решении задачи какуро по умолчанию используется число Gmax = 10200, оптимизированное на ряде указанных выше задач какуро. Здесь и далее параметры алгоритма SDE [14] несколько отличаются от приведенных в работе [14]. В курсовой работе оптимизация параметров алгоритма SDE для решения головоломок какуро выполнялась по оценке скорости решения 15 различных головоломок с число неизвестных *m* от 20 до 116.

В цикл CA вложен цикл CB, счетчик которого *i* изменяется от 1 до nPM в соответствии с размером популяции.

4) *Выполнение цикла* CB. Используются два алгоритма AM1 и AM2 модификации (мутации) текущего вектора ***Xi***. Алгоритма AM1 – составная часть SDE, а алгоритм AM2 был разработан для увеличения эффективности мутаций при решении задачи какуро. Случайно-закономерным образом выбирается и выполняется один из алгоритмов AM1 и AM2. В цикле CB подсчитываются числа m1 и m2 успешных вызовов каждого из алгоритмов AM1 и AM2, приводящих к уменьшению функции *y*. Случайный вызов алгоритма AMi происходит пропорционально отношению ratio = mi/(m1 + m2), т.е. чаще вызывается более успешный алгоритм. Если radio достигает определенного порога (например, меньше 0.1 или больше 0.9), то величины m1 и m2 изменяются на начальные: m1 = m2 и ratio = 0.5 и процесс их изменения продолжается, т.е. алгоритмам вновь даются равные шансы на их исполнение.

Метод создания нового вектора ***X\*i***: Алгоритмы AM1 и AM2.

5.1) *Алгоритм AM1*.

Из множества G случайным образом выбирают три вектора ***Xi1***, ***Xi2*** и ***Xi3***, несовпадающие с текущим вектором ***Xi***, компоненты которого изменяют в соответствии с биологическими процессами мутации и кроссинговера (перекрёста) хромосом (векторов ***X***), получая новый вектор ***X\*i***.

*Цикл* CC. Случайное число *p* < *m* выбранных по закону случая компонент вектора ***Xi*** заменяют новыми значениями ***z*** (кроссинговер). Замена производится в цикле CC до тех пор, пока генерируемое случайное число из полуотрезка [0, 1) меньше кроссинговерной константы CR [14] и пока *p* < *m*. Величины CR и *p* определяют долю компонент вектора ***Xi***, которая будет изменена в этом цикле CC. Здесь "мутированную" компоненту *zk* вектора (хромосомы) ***Xi*** вычисляют по формуле:

*zk* = round(*xi1 k* + P.(*xi2 k* - *xi3 k*)), *k* = 1, 2,..., *p.*

При этом, если *zk* выходит за границы отрезка [*xk, min , xk, max*], то при *zk* < *xk, min* вместо *zk* используют

*z\*k* = *xk, min* + round(*rand*.(*xk, min* – *zk*)),

а при *zk* > *xk, max* используют

*z\*k* = *xk, max* + round(*rand*.(*xk, max* – *zk*)).

Функция round означает округлить до целого числа, а *rand* – случайное действительное число на полуотрезке [0, 1).

*Конец цикла* CC.

Для задачи какуро рекомендуемая величина CR = 0.9, или она может быть выбрана на отрезке [0, 1]. Масштабный множитель P выбирают на отрезке [0.1, 1.0] в зависимости от вида кроссворда какуро. Рекомендуемое значение P = 0.4 для полей какуро менее 12×12. Для больших полей рекомендуется P = 0.2. В курсовой работе величины CR и P оптимизированы по скорости решения 15 головоломок какуро с размером поля от 5×5 до 14×14 и числом неизвестных *m* от 20 до 116.

5.2) *Алгоритм AM2*.

Если *i* – четное число, то далее алгоритм выполняется для горизонтальных блоков, иначе – для вертикальных. Для вектора ***Xi*** определить, для каких блоков равенство (1) не выполнено. Для ключевой суммы каждого такого блока*k*, состоящего из *L* клеток, генерируется *d* ее возможных разложений *D*1L, *D*2L, ..., *D*dL на сумму *L* цифр из множества {1, 2, ..., 9}, исключая одинаковые цифры в любом разложении. В каждом разложении порядок цифр перемешивается. Случайным образом выбирается одно *D*jL из разложений, и компоненты вектора ***Xi***, соответствующие блоку *k*, заменяются на слагаемые этого разложения.

6) Полученный вектор ***X\*i*** затем используется для вычисления значения функции *y\*i*. Если значение функции *y\*i* меньше или равно величине функции *yi* при значении вектора ***Xi***, то вектор (индивид популяции) ***Xi*** заменяется на порожденный индивид ***X\*i***. И если *y\*i* меньше минимального текущего значения функции *ymin*(***Xmin***), то оно заменяется на *y\*i*, и соответствующий вектор ***Xmin*** заменяется на ***X\*i***. Если теперь *ymin* меньше заданного пользователем значения минимума функции ***ε***, то происходит выход из цикла CB. В случае решения задачи какуро выход из цикла соответствует значению *ymin* = 0.

*Конец цикла* CB. В цикле CB сформировано новое поколение ***X*** популяции G.

Если *ymin* меньше заданного пользователем значения минимума функции ***ε***, то происходит выход из цикла CA.

*Конец цикла* CA.

Если *ymin* меньше заданного пользователем значения минимума функции ***ε***, то происходит выход из цикла CCIVIL.

*Конец цикла* CCIVIL.

7) Происходит вывод значений ***Xmin***, *ymin*, числа цивилизаций Сsum (числа итераций цикла CCIVIL) и числа поколений Gsum (числа итераций цикла CA для последней цивилизации). В случаях, если *ymin* оказывается больше заданного пользователем значения минимума функции ***ε***, то наряду с информацией о *ymin*, ***Xmin***, Сsum, Gsum выводится сообщение, что установленный пользователем минимум функции *y* не достигнут. Он может быть достигнут в последующей оптимизации функции при изменении числа Сsum или Gmax и при подборе констант P и CR.

***3.2.4. Алгоритм генерации головоломок какуро***

Для генерации головоломок какуро с использованием готовых форм (см., например, рис. 1а), изменяя лишь суммы блоков, также используется алгоритм симплексной дифференциальной эволюционной оптимизации. В этой задаче размер поля, число заполняемых блоков и белых клеток, расположение блоков и пересечение блоков не меняется, т.е. полностью сохраняется структура пазла, изменяются лишь ключи (суммы) блоков и состав цифр блоков. Поставленная задача генерации головоломки, сводящаяся к поиску минимума (*yg* = 0) функции (4), значительно проще предыдущей задачи решения головоломки, не рассматривая очень сложную задачу поиска единственности решения генерируемой головоломки. Поэтому используется упрощенная схема SDE алгоритма. Приведем эти изменения в алгоритме по сравнению с разделом 3.2.3. Пункт (2) раздела 3.2.3 предварительной оптимизации сгенерированных значений переменных ***X*** не используется. В пункте (5.2) вместо алгоритма *AM2*. применяется следующий более простой алгоритм *AM3*.

Алгоритм *AM3*.

Как и в алгоритме *AM1* (см. пункт 5.1 раздела 3.2.3) из множества G случайным образом выбираются три вектора ***Xi1***, ***Xi2*** и ***Xi3***, несовпадающие с текущим вектором ***Xi***. В векторе ***Xi*** заменяется часть компонент. Эти компоненты соответствуют трем случайным образом выбранным не совпадающим между собой горизонтальным блокам какуро. Величины компонент *xij* этих трех блоков заменяются соответствующими компонентами трех векторов ***Xi1***, ***Xi2*** и ***Xi3***.

***3.2.5. Оценка сложности алгоритма решения головоломки какуро***

Сложность головоломки зависит от числа белых клеток *m* (рис. 1). Оценим сложность алгоритма в зависимости от числа клеток *m* поля какуро, которые необходимо заполнить правильными значениями. Алгоритмическая сложность центральной части алгоритма SDE, начиная с пункта (4) (раздел 3.2.3), невысокая ~*O*(*m2*). Однако центральная часть алгоритма выполняется несколько тысяч раз (параметр Gmax в пункте (3) раздела 3.2.3), чтобы найти глобальный минимум функции (3) среди множества локальных. Величина параметра Gmax, как показывает опыт решения головоломок какуро (см. пункт (2) раздела 3.2.3.), должна быть сопоставима с величиной *m2,* т.е. в итоге сложность алгоритма SDE по решению задачи какуро составляет ~*O*(*m4*).

***3.2.6. Оценка сложности алгоритма генерации головоломки какуро***

В случае генерации головоломки какуро под готовую форму, не требуется поиска глобального минимума, как в случае решения головоломки (см. выше раздел 3.2.5). Вследствие этого выполняется, в основном, только центральная часть алгоритма SDE, начиная с пункта (4) (раздел 3.2.3), алгоритмическая сложность которого ~*O*(*m2*) и соответствует сложности алгоритма генерации головоломки какуро.

***3.2.7. Оценка сложности решения головоломки какуро***

Нижней границей сложности решения головоломки какуро можно считать число белых клеток *m*, которые необходимо правильно заполнить цифрами от 1 до 9 (рис. 1). Верхней границей сложности решения можно считать число 9*m*. Эта оценка соответствует тому, что в каждую белую клетку какуро можно поставить любое число от 1 до 9. Т.е. здесь не учитываем ни возможные одинаковые цифры в каждом блоке, ни неравенства сумм чисел в блоке ключу. Более реалистичной является оценка сложности, учитывающая число разложений *dk* ключа (суммы блока) на сумму из *Lk* неповторяющихся цифр для каждого блока *k* с учетом перестановок цифр:

Эта формула показывает, что сложность головоломки очень быстро возрастает с увеличением ее размера, большой долей длинных блоков и ключей с большим числом разложений *dk* (например, разложение числа 25 на 5 слагаемых имеет 12 вариантов), что соответствует действительности. Головоломка оказывается значительно сложнее, если каждая белая клетка является составной частью как горизонтального, так и вертикального блока ("головоломка полных пересечений"). Головоломки с присутствием заметной доли клеток, не вовлеченных в пересечения, как правило, решаются программой значительно быстрее по сравнению с аналогами - "головоломками полных пересечений" (рис. 3).

|  |  |  |
| --- | --- | --- |
| *а* | *б* | *в* |

Рис. 3. Головоломка какуро с размером поля 7×7: простая (*а*), средняя (*б*) и сложная (*в*) для решения согласно www.kakuro.cc.

**3.3. Алгоритмы функционирования программы и ее интерфейс**

Программа имеет удобный графический интерфейс (рис. 4), который позволяет:

1) загрузить головоломку какуро из текстового файла,

2) сохранить текущее (незаконченное) решение головоломки,

3) сохранить решение головоломки,

4) вставить (рис. 1б) или удалить (рис. 1а) из таблицы имеющееся решение головоломки.

5) сконструировать пользователю новую головоломку какуро из графических элементов: белых клеток блоков (), клеток сумм (), задавая их ключи (суммы), и стен (), первоначально задав размер поля какуро,

6) сгенерировать новую головоломку, используя любую готовую форму какуро (например, на рис. 3), где будут заменены ключи (суммы) и значения цифр блоков для решения,

7) получить помощь для представления ключа в виде суммы заданного количества несовпадающих между собой чисел из множества от 1 до 9 (рис. 5),

8) включить программный решатель головоломок какуро для поиска решения в фоновом режиме (в это время пользователь может сам решать головоломку); если решение получено, включается зеленый сигнал в нижнем левом углу основного окна программы (рис. 4),

9) получить информацию о программе,

10) получить помощь по пользованию программой.

На всех стадиях выполняется контроль данных, столкновений процессов и завершения расчетов. В случае ошибок в головоломке какуро загрузка файла или расчеты завершаются сообщением об ошибке.

Структура классов Программы приведена в Приложении 1, а описание методов приведено в Приложении 2.

Рассмотрим последовательно алгоритмы выполняемых действий с указанием имен задействованных методов.

|  |
| --- |
|  |

Рис. 4. Главное окно графического интерфейса программы "Игра какуро".

|  |
| --- |
|  |

Рис. 5. Окно помощи о разложении числа на сумму чисел. Пример вариантов представления числа 25 в виде суммы пяти несовпадающих между собой чисел из множества от 1 до 9.

***3.3.1. Чтение исходных данных из текстового файла***

Входные текстовые данные содержат текстовую информационную строку и матрицу чисел и знаков слэш "/"; строка матрицы соответствует одной строке поля кроссворда какуро; строка содержат целые числа, символы "/", пробелы или знаки табуляции, чтобы описать ввод цифр блоков, пустых клеток и ключей (сумм блоков). Выполняется контроль, что размер поля не менее 3×3, что загружаются целые числа и что число колонок в какуро матрице постоянно. Происходит одна из проверок, может ли головоломка решена: если сумму блока нельзя представить в виде суммы чисел данного числа клеток блока, выдается сообщение, что решение невозможно. Для этого используется метод Kakuro.decompositionsVerify. О форматах входного и выходного файлов изложено в разделе 3.4.

Задействованы методы: Kakuro.jMenuItemLoadActionPerformed главной формы frameKakuro для открытия текстового файла, где метод InputOutputData.getKakuroTaskSize определяет размер матрицы входных данных, и инициализируются размеры массивов, описывающих головоломку какуро, а метод InputOutputData.loadKakuroTask считывает эти величины.

***3.3.2. Графический интерфейс для решения головоломки какуро***

Когда массивы исходных данных инициализированы и загружены, на главной форме какуро создается графическое изображение таблицы какуро и под ней - линейка цифр от 1 до 9 (рис. 4), чтобы пользователь мог решать головоломку. Если загружено также правильное решение, включается зеленый сигнал в нижнем левом углу основного окна программы (рис. 4), а пункты меню Edit → Insert solution и Edit → Clear the puzzle формы frameKakuro позволяют вставить в таблицу (рис. 1б) или удалить из таблицы (рис. 1а) имеющееся решение головоломки. Клетки головоломки являются графическими изображениями, которые рисуются на панели при помощи метода setElements класса KakuroSolver. Размер таблицы какуро масштабируется в соответствии с размером основного окна Программы (рис. 4). Линейка цифр под таблицей какуро позволяет заполнять ее блоки цифрами решения, перетаскивая с помощью мышки цифру с линейки в соответствующую клетку таблицы: нажать левую кнопку мышки при указателе на цифре, перетащить цифру в белую клетку, отпустить кнопку. Если блок заполнен правильно, т.е. выполнено равенство (1), ключ блока станет зеленым, иначе красным (рис. 6). Если в блоке есть одинаковые цифры, т.е. не выполнено условие (2), они будут окрашены в красный цвет (рис. 6).

Используются метод Kakuro.loadedSolutionVerify и методы классов KakuroSolver и KakuroPuzzle.

|  |
| --- |
|  |

Рис. 6. Примеры контроля Программой правильного и неправильно заполнения блоков цифрами решения.

***3.3.3. Конструирование пользователем новой головоломки какуро***

Пункты меню Tools → Designer (рис. 4) главной формы frameKakuro позволяет пользователю сконструировать новую головоломку какуро из графических элементов: белых клеток блоков (), клеток сумм (), задавая их ключи (суммы), и стен (), первоначально выбрав размер поля какуро или загрузив (см. раздел 3.3.1) текстовый файл готовой головоломки (рис. 7). Головоломка не обязательно должна быть прямоугольной, важно, чтобы ее поле вписывалось в указанные размеры (ширина × высота) прямоугольника. В случае загрузки готовой формы в ней можно поменять все, кроме размера поля.

В режиме конструирования (рис. 8) отсутствует линейка с цифрами для заполнения, изображенная на рис. 6. Щелчок правой кнопкой мышки циклически меняет тип выбранной ячейки (,  или ). Щелчок левой кнопкой мышки по ячейке  под диагональю или над ней вызывает мигание соответствующей цифры и позволяет редактировать ключевую сумму: в режиме мигания цифры, ввести однозначное или двузначное число от 1 до 45 и нажать нецифровую клавишу. Величина числа программой контролируется в пределах от 1 до 45. Нули любой ячейки вида  означают, что в готовом какуро пазле на этом месте будет стенка . Сконструированный пазл какуро можно сохранить, используя меню File → Save running kakuro, а затем загрузить для решения пользователем. Пункты меню Edit → Insert solution и Edit → Clear the puzzle формы frameKakuro позволяют вставить (рис. 1б) или удалить (рис. 1а) из таблицы имеющееся решение головоломки.

|  |
| --- |
|  |

Рис. 7. Выбор способа конструирования головоломки какуро: начать с пустого поля задаваемого размера или с редактирования загружаемой головоломки какуро.

Используются методы InputOutputData.getKakuroTaskSize, InputOutputData.loadKakuroTask, Kakuro.kakuroVariables, Kakuro.parameterSDE, Kakuro.loadedSolutionVerify, Kakuro.entriesClear, а также методы классов KakuroDesigner, KakuroEditor и KakuroPuzzle.

|  |
| --- |
|  |

Рис. 8. Режим Программы для конструирования новой головоломки какуро.

***3.3.4. Генерирование новой головоломки какуро***

Пункты меню Tools → Generator (рис. 4) главной формы frameKakuro генерирует новую головоломку, используя предварительно загруженный готовый пазл какуро (например, на рис. 3), в котором будут заменены лишь ключи и значения цифр блоков для последующего решения. Т.е. в задаче генерирования программой новой головоломкиразмер поля, число заполняемых блоков и белых клеток, расположение блоков и пересечение блоков не меняется, т.е. полностью сохраняется структура пазла. При этом появляется зеленый сигнал (кружок) в нижнем левом углу основного окна программы (рис. 4), что решение получено. Программа способна генерировать головоломки какуро с общим числом белых клеток не менее ста без доказательства единственности полученного решения. Если пазл какуро сгенерирован, выдается сообщение "Kakuro puzzle was generated" и включается зеленый сигнал в нижнем левом углу основного окна программы (рис. 4), иначе выдается сообщение о неудаче генерации. Сгенерированный пазл какуро можно сохранить, используя меню File → Save running kakuro или File → Save solved kakuro. Пункты меню Edit → Insert solution и Edit → Clear the puzzle формы frameKakuro позволяют вставить (рис. 1б) или удалить (рис. 1а) из таблицы какуро имеющееся решение головоломки.

Используется метод Kakuro.jMenuItemGenerActionPerformed, Kakuro. keepElementsSolution, Kakuro.calcClues, Kakuro.entriesClear, AlgorithmsDE.kakuroGeneratorDE, AlgorithmsDE.kakuroGeneratorTask и методы классов KakuroSolver и KakuroPuzzle.

***3.3.5. Помощник для представления числа в виде суммы слагаемых***

Пункты меню Tools → Helper (рис. 4) главной формы frameKakuro открывает окно помощи о разложении числа на сумму чисел из множества от 1 до 9 без их повторов в сумме (рис. 5). Пользователь вводит ключевую сумму и число слагаемых (клеток блока) на форме (рис. 5), при нажатии кнопки Launch возможные разложения выводятся в текстовой области формы Helper.

Используется метод Kakuro.jMenuItemHelperActionPerformed и методы класса Helper.

***3.3.6. Программный решатель головоломок какуро***

Программный решатель головоломок какуро служит для поиска программой решения загруженной головоломки. Процесс решения может выполняться в фоновом режиме так, что в это время пользователь может сам решать головоломку. Если решение получено, включается зеленый сигнал в нижнем левом углу основного окна программы (рис. 4). Пункты меню Tools → Solver (рис. 4) главной формы frameKakuro открывает окно программного решателя (рис. 9).

|  |
| --- |
|  |

Рис. 9. Окно программного решателя головоломок какуро.

Слева на форме указаны пять параметров используемого алгоритма симплексной дифференциальной эволюции. В курсовой работе эти оптимизированы по скорости решения 15 головоломок какуро с размером поля от 5×5 до 14×14 и числом неизвестных *m* от 20 до 116. Эти параметры автоматически предлагаются по умолчанию при вводе какуро пазла из файла. Они могут быть изменены в некоторых пределах. Таким образом, из текстовых полей (рис. 9) считываются параметры алгоритма симплексной дифференциальной эволюции и определяются их допустимые границы:

1) число цивилизаций (запусков алгоритма SDE), 1 ≤ Cmax ≤ 1000, по умолчанию Cmax = 24,

2) число итераций (поколений), 102 ≤ Gmax ≤ 106, по умолчанию Gmax = 10200,

3) число векторов независимых переменных (членов популяции), 2.m ≤ nPM ≤ 30.m, где m – число заполняемых клеток пазла какуро, по умолчанию nPM = 8.m,

4) масштабный множитель P, 0.4 ≤ P ≤ 1.0, по умолчанию P = 0.4 или P = 0.2,

5) кроссинговерная константа CR, 0.0 ≤ CR ≤ 1.0, по умолчанию CR = 0.9.

Решатель запускается кнопкой START, после чего надпись на которой меняется на STOP, чтобы можно было отказаться от вычислений. Если решение головоломки получено, оно будет выведено в текстовой области формы вместе с выдачей задействованных параметров алгоритма. Включится зеленый сигнал в нижнем левом углу основного окна программы (рис. 4), пункты меню Edit → Insert solution и Edit → Clear the puzzle формы frameKakuro позволяют вставить (рис. 1б) или удалить (рис. 1а) из таблицы полученное решение головоломки.

Если решение не достигнуто для заданных параметров алгоритма, то полученное приближение также будет выведено в текстовой области формы вместе с выдачей задействованных параметров алгоритма. Будет предложено использовать другие параметры алгоритма симплексной дифференциальной эволюции. Включится красный сигнал в нижнем левом углу основного окна программы.

Задействованы методы класса Solver и методы Kakuro.jMenuItemSolverActionPerformed, AlgorithmsDE.quickSort, AlgorithmsDE.generatePopulation, AlgorithmsDE.kakuroSolverDE, AlgorithmsDE.kakuroSolverTask, AlgorithmsDE.wDecomp, AlgorithmsDE.genClues, AlgorithmsDE.shuffleWSet, AlgorithmsDE.sumVerify.

**3.4. Организация входных и выходных данных**

Файлы входных и выходных данных совпадают по структуре хранимой информации о головоломке какуро. Входные и выходные текстовые данные содержат текстовую информационную строку и матрицу чисел и знаков слэш "/"; строка матрицы соответствует одной строке поля кроссворда какуро; строка содержат целые числа, символы "/", пробелы или знаки табуляции. Выполняется контроль, что размер поля не менее 3×3, что загружаются целые числа и что число колонок в какуро матрице постоянно. Пример входного файла для головоломки какуро на рис. 3а:

Easy Kakuro 7x7, URL: www.kakuro.cc

/ / / 10/ / 16/ /

/ / 17/17 2 8 7 /

/ /9 8 1 30/17 9 8

/ /20 9 3 8 17/ /

/ / 17/21 4 9 8 /

/17 8 9 /16 7 9 /

/ /15 8 1 6 / /

Здесь знак "/" без примыкающих (нет пробела между знаком и цифрой) цифр – стена –пустая клетка (); 10/ - пример вертикального ключа  (см. первую строку какуро на рис. 3а); /20 – пример горизонтального ключа  (см. четвертую строку на рис. 3а); 17/17 - пример вертикального и горизонтального ключей стоящих рядом  (см. вторую строку какуро на рис. 3а). Цифра от 0 до 9 без примыкающего к ней знака "/" – число белой клетки какуро – это элемент решения (число ≠ 0) или пустая клетка (число = 0).

Для сохранения игры используется метод InputOutputData.saveKakuroPuzzle.

**3.5. Выбор технических и программных средств**

Программа была разработана и проверена на функционирование под управлением операционных системам WINDOWS 7/8 на платформах x86 и x86-64. Исходным языком данной разработки является язык Java. Среды разработки - NetBeans IDE 8.0.2 и IntelliJ IDEA 14.0.1. Для работы Программы требуется программное обеспечение Java Runtime Environment версии не ниже 7.0 под Windows.

**4. ОЖИДАЕМЫЕ ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ**

Программа "Игра Какуро на основе алгоритма симплексной дифференциальной эволюции" будет полезна для широкого круга любителей головоломок какуро. Она полезна для развития математического мышления, особенно, для школьников при решении простых головоломок какуро. Но и любителей сложных какуро пазлов также порадует. Она может заинтересовать инженеров и научных сотрудников, связанных с решением задач дискретной оптимизации, кодирования и создания ключей, поскольку задача решения головоломки какуро представляет собой сложную задачу дискретной оптимизации с ограничениями, на которой может быть проверена эффективность того или иного подхода, что и используется в научных работах [10 - 13]. В интернет сети появились интерактивные программы для решения головоломок какуро (см., например, [kakuro.ru](http://www.gazetakrot.ru/online/numeric/kakuro.php)). Они предназначены для решения кроссвордов какуро пользователем, программы не способны решать головоломки какуро, подготовленные пользователем, а также их конструировать. В научных работах [10 - 13], включая диссертацию [13], изложены некоторые детали разрабатываемых эффективных алгоритмов по решению головоломок какуро. Однако компьютерные программы по решению головоломок какуро в этих работах не приводятся. Разработанная программа "Игра Какуро на основе алгоритма симплексной дифференциальной эволюции" не привязана к Интернету, позволяет решать головоломки как компьютером, так и пользователем, который может конструировать новые кроссворды какуро в редакторе программы или генерировать с помощью программы, он может загружать головоломки какуро с web-сайтов и печатных изданий с помощью графического редактора программы или текстового файла.

**5. ИСТОЧНИКИ, ИСПОЛЬЗОВАННЫЕ ПРИ РАЗРАБОТКЕ**

1. ГОСТ 19.101-77 Виды программ и программных документов. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

2. ГОСТ 19.102-77 Стадии разработки. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

3. ГОСТ 19.103-77 Обозначения программ и программных документов. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

4. ГОСТ 19.104-78 Основные надписи. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

5. ГОСТ 19.105-78 Общие требования к программным документам. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

6. ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом.// Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

7. ГОСТ 19.404-79 Пояснительная записка. Требования к содержанию и оформлению. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

8. ГОСТ 19.603-78 Общие правила внесения изменений. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

9. ГОСТ 19.604-78 Правила внесения изменений в программные документы, выполненные печатным способом. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

10. Ruepp O. The Computational Complexity of the Kakuro Puzzle, Revisited / O. Ruepp, M. Holzer. - in: Fun with Algorithms. 5th International Conference, FUN 2010 / P. Boldi, L. Gargano Eds. - Ischia, Italy, June 2010. Springer: Berlin, 2010. – p. 319-330.

11. da Fonseca J. B. A novel linear MILP model to solve kakuro puzzles / J. B. da Fonseca. - in: Proceedings of the 10th Portuguese Conference on Automatic Control; 16-18 July 2012, Funchal Madeira Island, Portugal, 2012. – p. 185-190.

12. Cazenave T. Monte-Carlo Kakuro. / T. Cazenave. – in: Advances in Computer Games 12th International Conference, ACG 2009, Pamplona, Spain, May 11-13, 2009 / van den Herik J. H., Spronck P. (Eds.). – Springer: Berlin, 2010. - 233 p.

13. Davies R. P. An Investigation into the Solution to, and Evaluation of, Kakuro Puzzles. Thesis for the degree of Master of Philosophy. / R. P. Davies. – Trefforest: Division of Mathematics and Statistics, University of Glamorgan, 2009. – 172 p.

14. Ali M., Pant M., Abraham A. A simplex differential evolution algorithm: development and applications/ M. Ali, M. Pant, A. Abraham. - Transactions of the Institute of Measurement and Control, 2011, V. 34, no. 6, pp. 691–704.

15. Батищев Д.И. Применение генетических алгоритмов к решению задач дискретной оптимизации: учебно-методический материал / Д. И. Батищев, Е. А. Неймарк, Н. В. Старостин. - Нижний Новгород: Нижегородский государственный университет им. Н.И. Лобачевского, 2007. - 85 с.

16. Mohamed A. W., Sabry H. Z., Khorshid M. An alternative differential evolution algorithm for global optimization/ A. W. Mohamed, H. Z. Sabry, M. Khorshid. - J. Advanced Research, 2012, V. 3, pp. 149–165.

17. Wang Y., Chen X., Gui W., Yang C., Caccetta L., Xu H. A Hybrid Multiobjective Differential Evolution Algorithm and Its Application to the Optimization of Grinding and Classification/ Y. Wang, X. Chen, W. Gui, C. Yang, L. Caccetta, H. Xu. - J. Applied Mathematics, 2013, V. 2013, pp. 1–15.

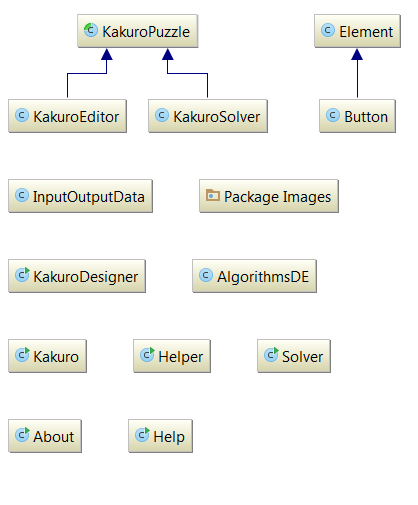
18. Nelder J. A., Mead R. A simplex method for function minimization / J. A. Nelder, R. Mead. - Computer Journal, 1965, v. 7, no. 4, pp. 308–313.

19. Иванов Б. Н. Дискретная математика. Алгоритмы и программы: Учеб. пособие /Б. Н. Иванов. — М.: Лаборатория Базовых Знаний, 2003. — 288 с.

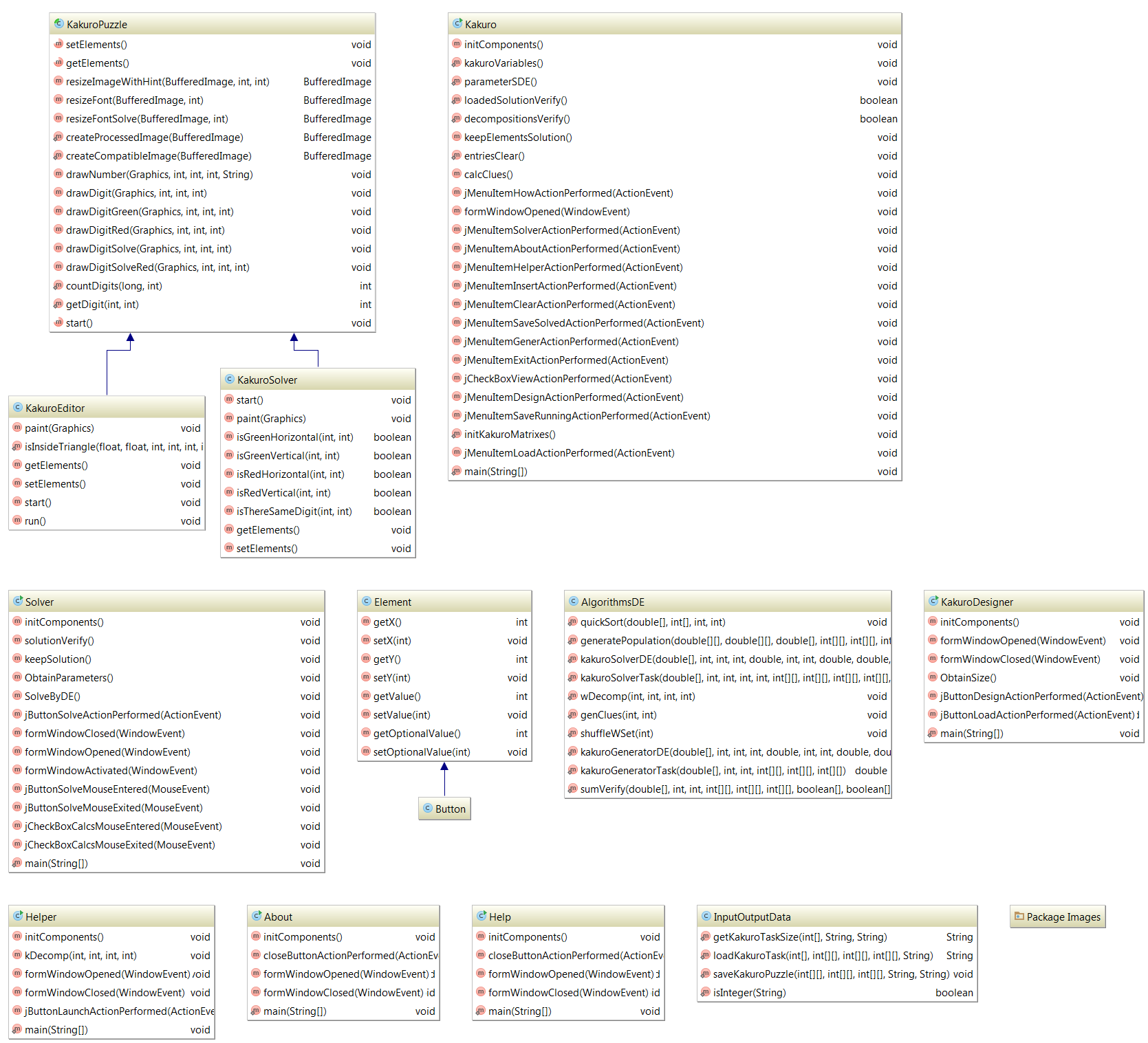
**ПРИЛОЖЕНИЕ 1**

**Таблицы с описанием классов и интерфейсов**

**Таблица 1**. Диаграмма классов пакета kakuro



**Таблица 2**. Структура классов пакета kakuro



**ПРИЛОЖЕНИЕ 2**

**Таблицы с описанием методов**

**Таблица 3.** Описание методов класса KakuroPuzze

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| имя | модификаторы доступа | тип | аргументы | назначение |
| setElements | public | void | - | Присвоить массиву клеток таблицы какуро значения массивов решенного какуро |
| getElements | public | void | - | Присвоить значениям массивов решенного кауро значения клеток таблицы какуро |
| resizeImageWithHint | private | BufferedImage | BufferedImage originalImage, int type, int del | Изменить размер изображения пропорционально заданной константе |
| resizeFont | protected | BufferedImage | BufferedImage originalImage, int type | Изменить размер изображения символа для служебных клеток какуро |
| resizeFontSolve | protected | BufferedImage | BufferedImage originalImage, int type | Изменить размер изображения символа для решения какуро |
| createProcessedImage | private | static BufferedImage | BufferedImage image | Обработать изображение, применить сглаживание |
| createCompatibleImage | private | Static BufferedImage | BufferedImage image | Сделать изображение прозрачным, совместимым с другими изображениями |
| drawNumber | protected | void | Graphics g, int number, int x, int y, String color | Нарисовать число |
| drawDigit | protected | void | Graphics g, int digit, int x, int y | Нарисовать черную цифру |
| drawDigitGreen | protected | void | Graphics g, int digit, int x, int y | Нарисовать зеленую цифру |
| drawDigitRed | protected | void | Graphics g, int digit, int x, int y | Нарисовать красную цифру |
| drawDigitSolve | protected | void | Graphics g, int digit, int x, int y | Нарисовать черную цифру для решения какуро |
| drawDigitSolveRed | protected | void | Graphics g, int digit, int x, int y | Нарисовать красную цифру для решения какуро |
| countDigits | public | static int | long n, int r | Посчитать количество цифр в числе |
| getDigit | public | static int | int number, int position | Получить цифру в числе по индексу |
| start | public | void | - | Запустить тред, визуализирующий головоломку какуро |

**Таблица 4.** Описание методов класса KakuroSolver

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| имя | модификаторы доступа | тип | аргументы | назначение |
| start | public | void | - | Запустить тред, визуализирующий головоломку какуро для решения |
| paint | public | void | Graphics g | Визуализация какуро для решения |
| isGreenHorizontal | private | boolean | int m, int k | Проверка на правильность решения горизонтальной суммы |
| isGreenVertical | private | boolean | int m, int k | Проверка на правильность решения вертикальной суммы |
| isRedHorizontal | private | boolean | int m, int k | Проверка на неправильность решения горизонтальной суммы |
| isRedVertical | private | boolean | int m, int k | Проверка на неправильность решения вертикальной суммы |
| isThereSameDigit | private | boolean | int m, int k | Проверка на дублирование той же цифры по горизонтали или по вертикали |
| getElements | public | void | - | Присвоить значениям массивов решенного кауро значения клеток таблицы какуро для решения |
| setElements | public | void | - | Присвоить массиву клеток таблицы какуро для решения значения массивов решенного какуро |

**Таблица 5.** Описание методов класса KakuroEditor

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| имя | модификаторы доступа | тип | аргументы | назначение |
| start | public | void | - | Запустить тред, визуализирующий головоломку какуро для редактирования |
| run | public | void | - | Визуализация мигающих цифр головоломки какуро для редактирования |
| paint | public | void | Graphics g | Визуализация какуро для редактирования |
| isInsideTriangle | public | static boolean | float x, float y, int x1, int y1, int x2, int y2, int x3, int 3 | Проверка на принадлежности точки треугольнику |
| getElements | public | void | - | Присвоить значениям массивов решенного кауро значения клеток таблицы какуро для редактирования |
| setElements | public | void | - | Присвоить массиву клеток таблицы какуро для редактирования значения массивов решенного какуро |

**Таблица 6.** Описание методов класса InputOutputData

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| имя | модификаторы доступа | тип | аргументы | назначение |
| getKakuroTaskSize | public | static String | int[] kTaskSize, String filePathName, String msg | Открыть текстовый файл с головоломкой какуро, определить размеры головоломки |
| isInteger | public | static boolean | String str | Проверить является ли строка числом |
| loadKakuroTask | public | static String | int[] kTaskSize, int[][] H, int[][] V, int[][] solvK, String filePathName | Открыть текстовый файл, загрузить головоломку какуро |
| saveKakuroPuzzle | public | static void | int[][] H, int[][] V, int[][] solvK, String kakuroInfo, String filePathName | Сохранить решение головоломки какуро в текстовый файл |

**Таблица 7.** Описание методов класса Help

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| имя | модификаторы доступа | тип | аргументы | назначение |
| closeButtonActionPerformed | private | void | ActionEvent evt | Закрыть форму Help |
| formWindowClosed | private | void | WindowEvent evt | Внести пометку о закрытие формы Help |
| formWindowOpened | private | void | WindowEvent evt | Загрузка и демонстрация файла помощи |
| initComponents | private | void | - | Инициализация компонентов формы Help |

**Таблица 8.** Описание методов класса About

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| имя | модификаторы доступа | тип | аргументы | назначение |
| closeButtonActionPerformed | private | void | ActionEvent evt | Закрыть форму About |
| formWindowClosed | private | void | WindowEvent evt | Внести пометку о закрытие формы About |
| formWindowOpened | private | void | WindowEvent evt | Загрузка и демонстрация файла о программе |
| initComponents | private | void | - | Инициализация компонентов формы About |

**Таблица 9.** Описание методов класса Helper

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| имя | модификаторы доступа | тип | аргументы | назначение |
| formWindowClosed | private | void | WindowEvent evt | Внести пометку о закрытие формы Helper |
| formWindowOpened | private | void | WindowEvent evt | Загрузить иконку kakuro.png |
| initComponents | private | void | - | Инициализация компонентов формы Helper |
| jButtonLaunchActionPerformed | private | void | ActionEvent evt | Проверить правильность входных данных числа для разложения |
| kDecomp | public | void | int wSum, int wLen, int k, int i | Разложить число на сумму из неповторяющихся цифр от 1 до 9 |

**Таблица 10.** Описание методов класса KakuroDesigner

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| имя | модификаторы доступа | тип | аргументы | назначение |
| initComponents | private | void | - | Инициализация компонентов формы KakuroDesigner |
| formWindowClosed | private | void | WindowEvent evt | Внести пометку о закрытии формы |
| formWindowOpened | private | void | WindowEvent evt | Загрузить иконку kakuro.png |
| jButtonDesignActionPerformed | private | void | ActionEvent evt | Разработать новую головоломку какуро |
| jButtonLoadActionPerformed | private | void | ActionEvent evt | Загрузить имеющуюся головоломку для создания из нее новой головоломки какуро |
| ObtainSize | public | void | - | Проверка корректности входных данных необходимых для создания новой головоломки какуро |

**Таблица 11.** Описание методов класса Element

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| имя | модификаторы доступа | тип | аргументы | назначение |
| getOptionalValue | public | int | - | Получить опциональное значение клетки какаро |
| getValue | public | int | - | Получить значение клетки какуро |
| getX | public | int | - | Получить x координату клетки какуро |
| getY | public | int | - | Получить y координату клетки какуро |
| setOptionalValue | public | void | int optionalValue | Задать опциональное значение клетки какаруо |
| setValue | public | void | int value | Задать значение клетки какуро |
| setX | public | void | int x | Задать x координату клетки какуро |
| setY | public | void | int y | Задать y координату клетки какуро |

**Таблица 12.** Описание методов класса Solver

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| имя | модификаторы доступа | тип | аргументы | назначение |
| formWindowActivated | private | void | WindowEvent evt | Обновить все параметры алгоритма, задать значения по умолчанию |
| formWindowClosed | private | void | WindowEvent evt | Внеcти отметку о закрытии формы |
| formWindowOpened | private | void | WindowEvent evt | Загрузить иконку kakuro.png |
| initComponents | private | void | - | Инициализация компонентов формы Solver |
| jButtonSolveActionPerformed | private | void | ActionEvent evt | Начать расчет для решения головоломки какуро |
| jButtonSolveMouseEntered | private | void | MouseEvent evt | Изменить стиль курсора на «по умолчанию» |
| jButtonSolveMouseExited | private | void | MouseEvent evt | Изменить стиль курсора на «в режиме ожидания», если алгоритм запущен на выполнение |
| jCheckBoxCalcMouseEntered | private | void | MouseEvent evt | Изменить стиль курсора на «по умолчанию» |
| jCheckBoxCalcMouseExited | private | void | MouseEvent evt | Изменить стиль курсора на «в режиме ожидания», если алгоритм запущен на выполнение |
| keepSolution | public | void | - | Записать решения расчетов алгоритма в матрицы для решенного какуро |
| ObtainParameters | public | void | - | Проверка правильности параметров для запуска алгоритма |
| solutionVerify | public | void | - | Проверка правильности решения какуро |
| SolveByDe | public | void | - | Начать выполнять алгоритм решения какуро |

**Таблица 13.** Описание методов класса AlgorithmsDE

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| имя | модификаторы доступа | тип | аргументы | назначение |
| genClues | public | static void | int wSum, int wLen | Сгенерировать все разложения числа на неповторяющиеся цифры для заданного количества цифр |
| generatePopulation | public | static void | double[][] xG0, double[][] xG1, double[] fG, int[][] H, int[][] V, int[][] xN, int[][] hE, int[][] vE, int nPM. int nx, int hK, int vK, int nh, int nv, int nn | Сгенерировать популяцию |
| kakuroGeneratorDE | public | static String[] | double[] xbest, int nx, int nh, int nv, double epsilon, int maxG, int nPM, double P, double CR, int[][] H, int[][] xN, int[][] hE, int[][] vE, int nn | Алгоритм генерации какуро |
| kakuroGeneratorTask | public | static double | double[] x, int nh, int nv, int[][] xN. int[][] hE. int[][] vE | Инициализация задачи какуро для генерации |
| kakuroSolverDE | public | static String[] | double[] xbest, int nx, int nh, int nv, double epsilon, int maxG, int nPM, double P, double CR, int hK, int vK, int[][] H, int[][] V, int[][] xN, int[][] hE, int[][] vE, int nn | Алгоритм решения какуро |
| kakuroSolverTask | public | static double | double[] x, int hK, int vK, int nh, int nv, int[][] H, int[][] V, int[][] xN, int[][] hE, int[][] vE | Инициализация задачи какуро для решения |
| quickSort | public | static void | double[] a, int[] b, int in, int ik | Алгоритм быстрой сортировки массива |
| shuffleWSet | public | static void | int wLen | Перемешать все имеющиеся в данный момент разложения числа на неповторяющиеся цифры |
| sumVerify | public | static void | double[] x, int hK, int[][] H, int[][] V, int[][] xN, boolean[] hOk, boolean[] vOk | Проверить все суммы какуро по вертикали и по горизонтали |
| wDecomp | public | static void | int wSum, int wLen, int k, int i | Разложить число на неповторяющиеся цифры |

**Таблица 14.** Описание методов класса Kakuro

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| имя | модификаторы доступа | тип | аргументы | назначение |
| calcClues | public | void | - | Посчитать соответствующие суммы по горизонтали и по вертикале |
| decompositionsVerify | public | static boolean | - | Проверить на возможность разложения исходных сумм на цифры |
| entriesClear | public | static void | - | Очистить текущий какуро пазл |
| formWindowOpened | private | void | WindowEvent evt | Загрузить иконку kakuro.png |
| initComponents | private | void | - | Инициализация компонентов формы Kakuro |
| initKakuroMatrixes | public | static void | - | Инициализировать какуро матрицы для решения какуро |
| jCheckBoxViewActionPerformed | private | void | ActionEvent evt | Сменить вид визуализации какуро |
| jMenuItemAboutActionPerformed | private | void | ActionEvent evt | Открыть форму About |
| jMenuItemClearActionPerformed | private | void | ActionEvent evt | Очистить текущий пазл какуро для решения |
| jMenuItemDesignActionPerformed | private | void | ActionEvent evt | Разработать новый пазл |
| jMenuItemExitActionPerformed | private | void | ActionEvent evt | Выйти из программы |
| jMenuItemGenerActionPerformed | private | void | ActionEvent evt | Сгенерировать новый пазл |
| jMenuItemHelperActionPerformed | private | void | ActionEvent evt | Запустить помощник для решения головоломки |
| jMenuItemHowActionPerformed | private | void | ActionEvent evt | Открыть файл помощи |
| jMenuItemInsertActionPerformed | private | void | ActionEvent evt | Показать решения текущего какуро |
| jMenuItemLoadActionPerformed | private | void | ActionEvent evt | Загрузить какуро из файла |
| jMenuItemSaveRunningActionPerformed | private | void | ActionEvent evt | Сохранить текущее какуро в файл |
| jMenuItemSaveSolvedActionPerformed | private | void | ActionEvent evt | Сохранить решенное какуро в файл |
| jMenuItemSolverActionPerformed | private | void | ActionEvent evt | Открыть решатель какуро |
| kakuroVariables | public | static void | - | Заполнить значения переменных для решения загруженного какуро |
| keepElementsSolution | public | void | - | Сохранить решение какуро в таблицу какуро для визуализации |
| loadedSolutionVerify | public | boolean | - | Проверить наличие уже имеющегося решения для загруженного какуро |
| parametersSDE | public | static void |  | Инициализация параметров симплексной дифференциальной эволюции |

**ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Изм. | Номера листов (страниц) | | | | Всего листов (страниц) в документе | № документа | Входящий № сопроводительного документа и дата | Подпись | Дата |
| измененных | замененных | новых | аннулированных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |