Санкт-Петербургский государственный университет
информационных технологий, механики и оптики
Факультет информационных технологий и программирования
Кафедра компьютерных технологий

Ф. Н. Царев

Разработка метода совместного применения генетического программирования и конечных автоматов

Бакалаврская работа

Научный руководитель – докт. техн. наук, профессор А.А. Шалыто А

Санкт-Петербург

2007

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Генетические алгоритмы являются одним из современных и быстро развивающихся направлений в искусственном интеллекте. С их помощью могут быть найдены решения многих задач в различных областях. Примерами таких задач являются: задачи синтеза расписаний, задачи планирования работ и распределения ресурсов, задачи маршрутизации транспортных средств, синтез топологии сетей различного назначения, построение *искусственных нейронных сетей*, деревьев принятия решений, автоматическое построение программ, проектирование *мультиагентных систем*, *конечных автоматов* и клеточных автоматов. К сожалению, в нашей стране этой области уделяется мало внимания. Одна из *задач* работы – хотя бы частично восполнить этот пробел.

Генетическое программирование – разновидность генетических алгоритмов, в которой вместо низкоуровневого представления объектов (битовые строки) используется высокоуровневое представление: деревья разбора программ, диаграммы переходов конечных автоматов и

т. д. С помощью генетического программирования наиболее эффективно решаются задачи автоматического построения программ, конечных автоматов, клеточных автоматов.

Автоматное программирование – парадигма программирования, при использовании которой программу предлагается строить в виде совокупности *поставщиков событий*, *системы взаимодействующих конечных автоматов* и *объектов управления*.

Поставщик событий характеризуется множеством *событий*, которые он может генерировать.

Объект управления характеризуется множеством *вычислительных состояний*, а также двумя наборами функций: множеством *предикатов*,

отображающих вычислительное состояние в логическое значение (истина или ложь), и множеством *действий*, позволяющих изменять вычислительное состояние.

Управляющий автомат определяется конечным множеством *управляющих состояний*, *функцией переходов* и *функцией действий*.

Управляющие конечные автоматы часто характеризуются сложным поведением, как например, в задачах об «Умном муравье» и «Летающие тарелки», рассматриваемых в настоящей работе. В таком случае их эвристическое проектирование представляет собой весьма трудоемкую задачу. Эта задача становится еще более сложной в случае проектирования системы взаимодействующих автоматов или в случае наличия в системе нескольких взаимодействующих агентов, каждый из которых управляется отдельным автоматом. Возникает естественное желание – автоматизировать процесс проектирования автоматов, поручив основную работу компьютеру.

В настоящей работе в качестве метода автоматизированного построения автоматов выбрано *генетическое программирование*. В начале работы излагаются общие концепции генетических алгоритмов, генетического программирования и автоматного программирования, приводится перечень задач, в которых успешно применяются указанные методы. Далее формулируются и изучаются две задачи – задача об «Умном муравье» и «Летающие тарелки». При этом основная цель исследований – установить применимость генетического программирования к построению автоматов рассматриваемого класса, и предложить подход к решению некоторых проблем, возникающих при автоматизированном построении автоматов.

Рассматриваемые в работе задачи выбраны не случайно. Задача об «Умном муравье» достаточно хорошо изучена. Сравнение результатов

применения разработанного в настоящей работе алгоритма генетического программирования позволяет установить его эффективность – с его помощью построен решающий эту задачу автомат, содержащий количество состояний, меньшее, чем известные ранее автоматы. Задача-игра «Летающие тарелки» предлагалась на заочном туре Всесибирской олимпиады по информатике 2005 года. Она представляет собой пример задачи, в которой необходимо построить автоматы, управляющие агентами. Кроме этого, существуют построенные вручную решения этой задачи, с которыми можно сравнить построенное автоматически.

В заключение работы анализируются ее результаты, и приводится ряд открытых вопросов и направлений для дальнейших исследований в этой области.

ГЛАВА 1. ОБЩИЕ КОНЦЕПЦИИ ГЕНЕТИЧЕСКОГО И АВТОМАТНОГО ПРОГРАММИРОВАНИЯ

1.1. Генетические алгоритмы

Генетический алгоритм (*genetic algorithm*) [1–4] представляет собой стохастический метод оптимизации. Основная идея генетических алгоритмов состоит в использовании принципа *естественного отбора*, который составляет основы теории эволюции живых организмов, предложенной Чарльзом Дарвином.

Основы генетических алгоритмов были заложены Дж. Холландом (J. Holland) – в 1975 году он опубликовал книгу «Адаптация в естественных и искусственных системах». Описанный в ней подход был в дальнейшем развит автором этой книги и его учениками в Мичиганском университете.

Кратко сформулировать принцип естественного отбора можно следующим образом: наименее приспособленные особи умирают раньше, а наиболее приспособленные выживают и дают потомство. Потомство выживших особей оказывается в среднем более приспособленным, но среди них опять выделяются более приспособленные особи, и т. д.

В генетических алгоритмах используются те же принципы. В качестве особей выступают элементы пространства возможных решений некоторой задачи (маршруты коммивояжера, диаграммы переходов автомата и т. п.). Задан набор генетических операций, с помощью которых из существующих особей формируются новые. Кроме этого определена так называемая *функция приспособленности* (*fitness-function*), которая показывает, насколько «хорошим» решением задачи является особь.

Процесс работы генетического алгоритма состоит в генерации поколений особей до тех пор, пока не будет выполнено некоторое условие останова (например, достигнуто целевое значение функции приспособленности или сгенерировано заданное число поколений). На рис. 1 приведена общая схема работы генетического алгоритма.

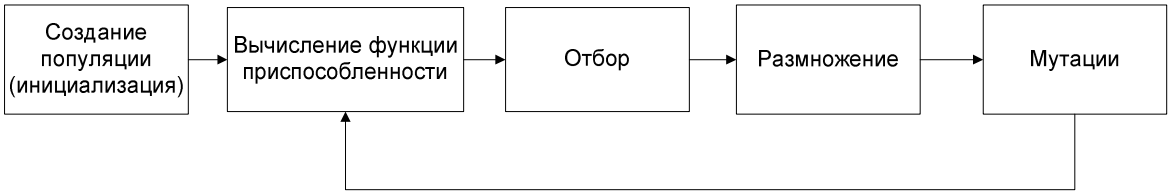

Рис. 1. Общая схема работы генетического алгоритма

По сравнению с традиционными методами оптимизации генетические алгоритмы имеют ряд преимуществ:

- генетические алгоритмы легко модифицируются для параллельных вычислений;
- они хорошо подходят для оптимизации недифференцируемых функций.

Основными недостатками генетических алгоритмов являются:

- высокая трудоемкость, ограничивающая их область применения;
- сложность оценки степени пригодности конкретных генетических операций для конкретной задачи.

1.1.1. Традиционный генетический алгоритм

В этом разделе приведено краткое описание *традиционного генетического алгоритма* (*conventional genetic algorithm*) [4] с одноточечной рекомбинацией и мутацией. В этом алгоритме каждая особь представляет собой битовую строку длины l, а размер популяции равен n.

Шаг 1. Генерации начальной популяции $\{x_i\}_{i=1}^{n}$. Для этого, например, можно сгенерировать n битовых строк, в которых каждый из l битов равен *0* или *1* с одинаковой вероятностью.

Шаг 2. Вычислить функцию приспособленности $f(x)$ для каждой особи из текущей популяции. Этот шаг обычно является наиболее трудоемким по времени во всем алгоритме. Отметим, что при вычислении функции приспособленности, как правило, необходимо осуществить декодирование некоторого объекта из двоичной строки.

Шаг 3. Переход к следующему поколению популяции. При создании нового поколения могут использоваться различные стратегии. В качестве примера приведем так называемый *алгоритм рулетки* (*roulette wheel selector*) [2].

Создание нового поколения в этом случае разбивается на n итераций, на каждой из которых в популяцию добавляется одна особь. Для этого случайно с вероятностями, пропорциональными их функция приспособленности, из текущего поколения популяции выбираются две родительские особи x и y.

После этого с некоторой наперед заданной вероятностью происходит рекомбинация (кроссовер, скрещивание) родительских особей. В результате образуются их «потомки» – z_1 и z_2. Происходит это следующим образом: случайно (с равномерным распределением на множестве $\{1, 2, \ldots, l\}$) выбирается число k. Хромосомы «потомков» z_1 и z_2 теперь строятся следующим образом: для z_1 – первые k символов совпадают с первыми k символами x, последние $(l-k)$ – с последними $(l-k)$ символами y, для z_2 – наоборот. Например, если $l=7$, $x=1001010$, $y=0001001$, $k=3$, то $z_1=1001001$, $z_2=0001010$. «Забудем» теперь о «родителях» x и y, обозначив $x=z_1$, $y=z_2$.

После этого равновероятно выбирается одна из особей x либо y – выбранную особь обозначим как z.

С некоторой (как правило, порядка *0.01*) вероятностью производится мутация особи z – в ней случайно выбирается один бит и изменяется.

Получившаяся в результате особь z добавляется в следующее поколение.

Шаг 4. Повторять шаги 2 и 3 до тех пор, пока не будет выполнено условие останова (максимальная приспособленность в популяции достигла целевого значения, прекратился рост максимальной приспособленности, число поколений достигло некоторого предела и т.д.).

Отметим, что кроме описанных выше одноточечного кроссовера, алгоритма рулетки и мутации изменением случайного бита, существуют и другие методы – обзор таких методов приведен в работе [5].

1.1.2. Математический аппарат традиционного генетического алгоритма

Вопрос о том, почему традиционный генетический алгоритм работает – происходит в вероятностном смысле постепенная оптимизация функции приспособленности обсуждается в работах [3, 6–9]. В работах [3,9] приводится, по сути неформальное, объяснение функционирования генетических алгоритмов, в работе [6] – формализованное, но при условии достаточно сильных ограничений на функцию приспособленности. Математическая модель традиционного генетического алгоритма описана в работе [7, 8].

1.2. Генетическое программирование

Генетическое программирование (*genetic programming*), предложенное *J.R. Koza* в 1992 году [9], – это применение генетических алгоритмов для автоматизированного построения программ.

Основным отличием генетического программирования от традиционных генетических алгоритмов является способ кодирования особей. Если в генетическом алгоритме особи кодируются с помощью битовых строк, то в генетическом программировании используется более высокоуровневое представление: используются деревья разбора, тексты программ на языках программирования с несложной структурой и т. д.

Такой подход позволяет определить генетические операции скрещивания и мутации, которые лучше подходят для решаемой задачи. Например, если каждая особь представляет собой программу, то возможны так называемые *операции, изменяющие архитектуру*, (*architecture-altering operations*) – например, добавление подпрограммы [10].

1.3. Автоматное программирование

Автоматное программирование [11–16] – парадигма программирования, предложенная А.А. Шалыто в 1991 году. При использовании этой парадигмы программы проектируются так же, как выполняется автоматизация технологических процессов – выделяются *поставщики событий*, *объекты управления* и *система управления*, которая представляет собой *систему взаимодействующих конечных автоматов* (рис. 2).

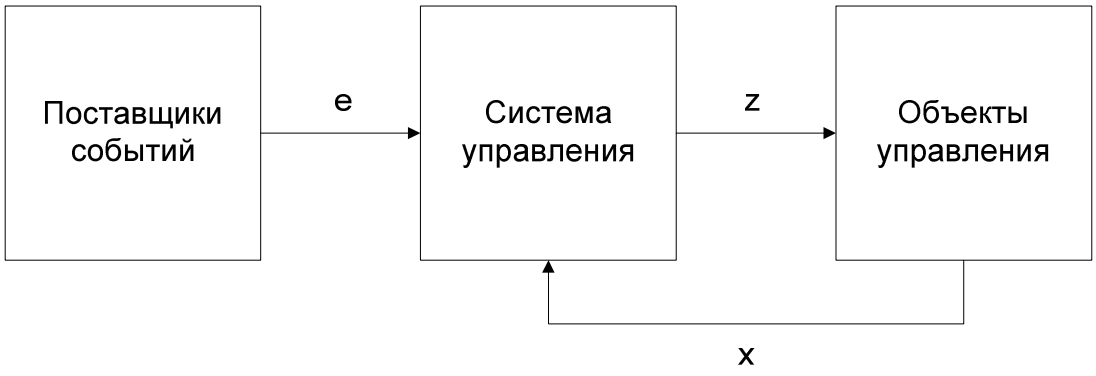

Рис. 2. Схема программы в автоматном программировании

Поставщик событий характеризуется множеством *событий* e, которые он может генерировать. Объект управления характеризуется множеством *вычислительных состояний*, а также двумя наборами функций: множеством *предикатов – входных переменных* x, отображающих вычислительное состояние в логическое значение (истина или ложь), и множеством *действии (выходных воздействий* z*)*, позволяющих изменять вычислительное состояние. При этом отметим, что в общем случае предикаты могут формироваться не только объектами управления. Управляющий автомат определяется конечным множеством *управляющих состояний*, *функцией переходов* и *функцией действий*.

Если говорить более формально, задано множество событий $E = \{e_i\}_{i=1}^{n}$, вырабатываемых поставщиком событий, множество предикатов $X = \{x_i\}_{i=1}^{m}$ и множество действий $Z = \{z_i\}_{i=1}^{k}$, которые связаны с объектом управления. Управляющий автомат характеризуется конечным множеством состояний S, начальным состоянием s_0, функцией перехода $\varphi: S \times E \times 2^X \to S$ и функцией действий z: $S \times E \times 2^X \to 2^Z$. Таким образом, выбор перехода зависит от текущего состояния автомата, поступившего события и значений предикатов, а при переходе в новое состояние производятся некоторые действия.

Автоматное программирование успешно применяется при создании программного обеспечения реактивных систем, таких как, например, некоторые мультиагентные системы [13–16].

Для поддержки автоматного программирования существует инструментальное средство *UniMod* [18, 19]. Это средство позволяет строить и редактировать схемы связей и диаграммы состояний, обеспечивать проверку формальной корректности этих диаграмм, проводить отладку диаграмм в графическом режиме и т. д.

После построения диаграмм и автоматической проверки их корректности, по ним строится их *XML*-описание. Далее вручную пишутся следующие фрагменты программы на языке *Java*: для поставщиков событий – их объявления, инициализация и преобразование системных событий в автоматные, а для объектов управления – методы, реализующие входные переменные и выходные воздействия.

Инструментальное средство *UniMod* применялось автором при решении задачи «Летающие тарелки» без использования генетического программирования [16, 17].

1.4. Несколько задач, в которых генетические алгоритмы применяются для построения автоматов

Приведем список таких задач, известных автору:

- итерированная дилемма заключенного (*iterated prisoner's dilemma*) [4, 20];
- задача классификации плотности для клеточных автоматов (*density classification task for cellular automata*) [21, 22];
- задача синхронизации для клеточных автоматов (*synchronization task for cellular automata*) [23];
- задача упорядочивания для клеточных автоматов (*ordering task for cellular automata*) [24];
- задача о «флибах» [25, 26];
- задача об «Умном муравье» (*artificial ant problem*) [3, 27, 28] – одна из задач, рассматриваемых в настоящей работе.

Выводы по главе 1

1. Автоматное программирование представляет собой парадигму программирования, которую целесообразно использовать при создании программных систем некоторых типов.
2. Существует ряд задач, в которых управляющие автоматы удается построить автоматически – с помощью генетических алгоритмов.

ГЛАВА 2. ЗАДАЧА ОБ «УМНОМ МУРАВЬЕ»

2.1. Постановка задачи

Приведем описание задачи об «Умном муравье» [28]. Игра происходит на поверхности тора размером 32 на 32 клетки (рис. 3). В некоторых клетках (обозначены на рис. 3 черным цветом) находится еда. Она расположена вдоль некоторой ломаной, но не во всех ее клетках. Клетки ломаной, в которых нет еды, обозначены серым цветом. Белые клетки не содержат еду и не принадлежат ломаной. Всего на поле 89 клеток с едой. Отметим, что рассматриваемое игровое поле называется *John-Muir Trail* [28].

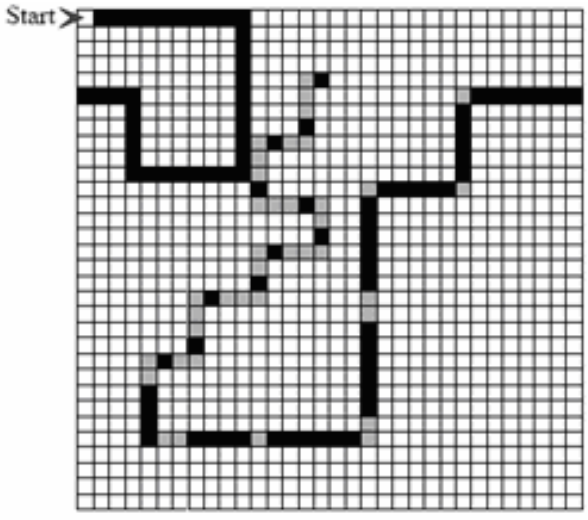

Рис. 3. Игровое поле

В клетке, помеченной меткой «Start», в начале игры находится муравей. Он занимает одну клетку и смотрит в одном из четырех направлений (север, юг, запад, восток). В начале игры муравей смотрит на восток.

Муравей умеет определять находится ли непосредственно перед ним еда. За один игровой ход муравей может совершить одно из четырех действий:

- сделать шаг вперед, съедая еду, если она там находится;

- повернуть налево;
- повернуть направо;
- ничего не делать.

Съеденная муравьем еда не восполняется, муравей жив на протяжении всей игры, еда не является необходимым ресурсом для его жизни. Ломаная не случайна, а строго фиксирована. Муравей может ходить по любым клеткам поля.

Игра длится 200 ходов, на каждом из которых муравей совершает одно из четырех действий. По истечении 200 ходов подсчитывается количество еды, съеденной муравьем. Это значение и есть результат игры.

Цель игры – создать муравья, который за 200 ходов съест как можно больше еды (желательно, все 89 единиц).

Один из способов описания поведения муравья – конечный автомат с действиями на переходах (автомат Мили), у которого есть одна входная переменная логического типа (находится ли еда перед муравьем), а множество выходных воздействий состоит из четырех упомянутых выше действий.

Например, эвристически построенный автомат с пятью состояниям из работы [28], граф переходов которого изображен на рис. 4, описывает поведение муравья, который съедает 81 единицу еды за 200 ходов, а всю еду – за 314 ходов.

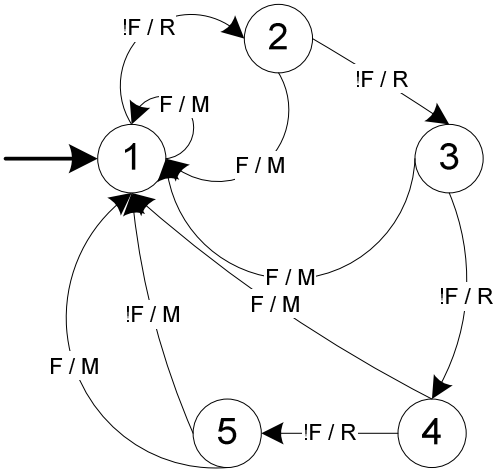

Рис. 4. Эвристически построенный автомат с пятью состояниями

Поясним используемые на рис. 4 обозначения. Пометки на переходах имеют формат `условие/действие`.

Условия обозначаются следующим образом:

- `F` – перед муравьем есть еда;
- `!F` – перед муравьем нет еды.

Действия обозначаются следующим образом:

- `M` – «Сделать шаг вперед»;
- `L` – «Повернуть налево»;
- `R` – «Повернуть направо»;
- `N` – «Ничего не делать».

2.2. Известные решения задачи об «Умном муравье»

Для решения задачи об «Умном муравье» в работах [3, 28, 27] применялись различные генетические алгоритмы. Построенные в указанных работах автоматы содержат от восьми до тринадцати состояний. В настоящей работе с помощью генетического программирования построен автомат, содержащий **семь** состояний.

В работе [3] описан генетический алгоритм, позволяющий построить автомат из восьми состояний, позволяющий муравью съесть всю еду. В этом алгоритме автоматы кодировались битовыми строками, и при этом использовалась перенумерация состояний (*SFS = **S**tandardizing transitions to the **F**uture or next **S**tates*, *MTF = **M**ove **T**o **F**ront*), позволяющая проще находить изоморфные автоматы и, таким образом, ускорить сходимость алгоритма. Это позволило сгенерировать автомат, содержащий восемь состояний.

В работе [27] для построения автоматов также используется генетический алгоритм с кодированием особей в виде битовых строк. Он представляет собой улучшенную версию алгоритма из работы [28]. Улучшение состоит в использование так называемого *замораживания* (*freezing*) состояний и переходов. Замораживание состоит в том, что некоторые состояния и переходы не могут быть изменены при мутации. При этом множество замороженных состояний и переходов также меняется в процессе работы генетического алгоритма. С помощью этого алгоритма были построены автоматы из 11 состояний, в то время как в работе [27] автомат содержит 13 состояний.

2.3. Предлагаемый метод решения задачи об «Умном муравье»

В настоящей работе для решения задачи об «Умном муравье» предлагается использовать генетическое программирование. Ниже приведено описание разработанного алгоритма генетического программирования.

Алгоритм генетического программирования состоит из пяти частей:

- создание начального поколения;
- мутация;

- скрещивание (кроссовер);
- отбор особей для формирования следующего поколения;
- вычисление функции приспособленности (фитнес-функции).

Каждая **особь** представляет собой некоторый автомат, описывающий поведение муравья. Хромосома особи состоит из номера начального состояния и описаний состояний. Описание состояния содержит описания двух переходов, соответствующих тому, что перед муравьем либо есть еда, либо ее нет. Описание перехода состоит из номера состояния, в которое он ведет, и действия, выполняемого при выборе этого перехода.

Хромосома *представляется не в виде битовой строки, как в генетических алгоритмах, а в виде объекта в языке программирования Java*. Этот объект имеет описанную структуру:

```
public class Automaton {
        public Transition[][] transitions;
        public int initialState;
        public int stateCount;
}
```

Создание начального поколения. Начальное поколение состоит из фиксированного числа случайно сгенерированных автоматов. Все автоматы в поколении имеют одинаковое наперед заданное число состояний.

Мутация. При мутации случайно выбирается один из четырех равновероятных вариантов:

- изменение начального состояния – в этом случае новое начальное состояние выбирается случайно и равновероятно;

- изменение действия на переходе – случайно и равновероятно выбирается переход, и действие на нем изменяется на случайное. При этом все возможные действия равновероятны;
- изменение состояния, в которое ведет переход, – случайно и равновероятно выбирается переход. После этого состояние, в которое ведет переход, заменяется на случайно выбранное состояние;
- изменение условия на переходе – случайно и равновероятно выбирается состояние. После этого переходы из этого состояния, соответствующие условиям «Перед муравьем есть еда» и «Перед муравьем нет еды», меняются местами.

Скрещивание. Оператор скрещивания получает на вход две особи и выдает также две особи. Процесс скрещивания происходит следующим образом. Обозначим родительские особи *P1* и *P2*, а потомков – *S1* и *S2*.

Обозначим начальное состояние автомата *A* как *A.is*. Тогда для потомков *S1* и *S2* будет верно одно из двух: либо $S1.is = P1.is$ и $S2.is = P2.is$, либо $S1.is = P2.is$ и $S2.is = P1.is$, причем оба варианта равновероятны.

Опишем, как «устроены» переходы автоматов-потомков *S1* и *S2* – может быть реализован один из двух равновероятных вариантов.

Первый вариант скрещивания. Обозначим переход из состояния номер *i* в автомате *P1* по значению входной переменной «Перед муравьем есть еда» как *P1(i, 1)*, а по значению «Перед муравьем нет еды» как *P1(i, 0)*. Аналогичный смысл придадим обозначениям *P2(i, 0)* и *P2(i, 1)*. Тогда для переходов из состояния с номером *i* в автоматах-потомках *S1* и *S2* будет справедливо одно из четырех соотношений:

- либо $S1(i, 0) = P1(i, 0)$, $S1(i, 1) = P2(i, 1)$ и $S2(i, 0) = P2(i, 0)$, $S2(i, 1) = P1(i, 1)$;
- либо $S1(i, 0) = P2(i, 0)$, $S1(i, 1) = P1(i, 1)$ и $S2(i, 0) = P1(i, 0)$, $S2(i, 1) = P2(i, 1)$;
- либо $S1(i, 0) = P1(i, 0)$, $S1(i, 1) = P1(i, 1)$ и $S2(i, 0) = P2(i, 0)$, $S2(i, 1) = P2(i, 1)$;
- либо $S1(i, 0) = P2(i, 0)$, $S1(i, 1) = P2(i, 1)$ и $S2(i, 0) = P1(i, 0)$, $S2(i, 1) = P1(i, 1)$.

Все четыре варианта равновероятны. Возможные варианты переходов при первом варианте скрещивания графически изображены на рис. 5.

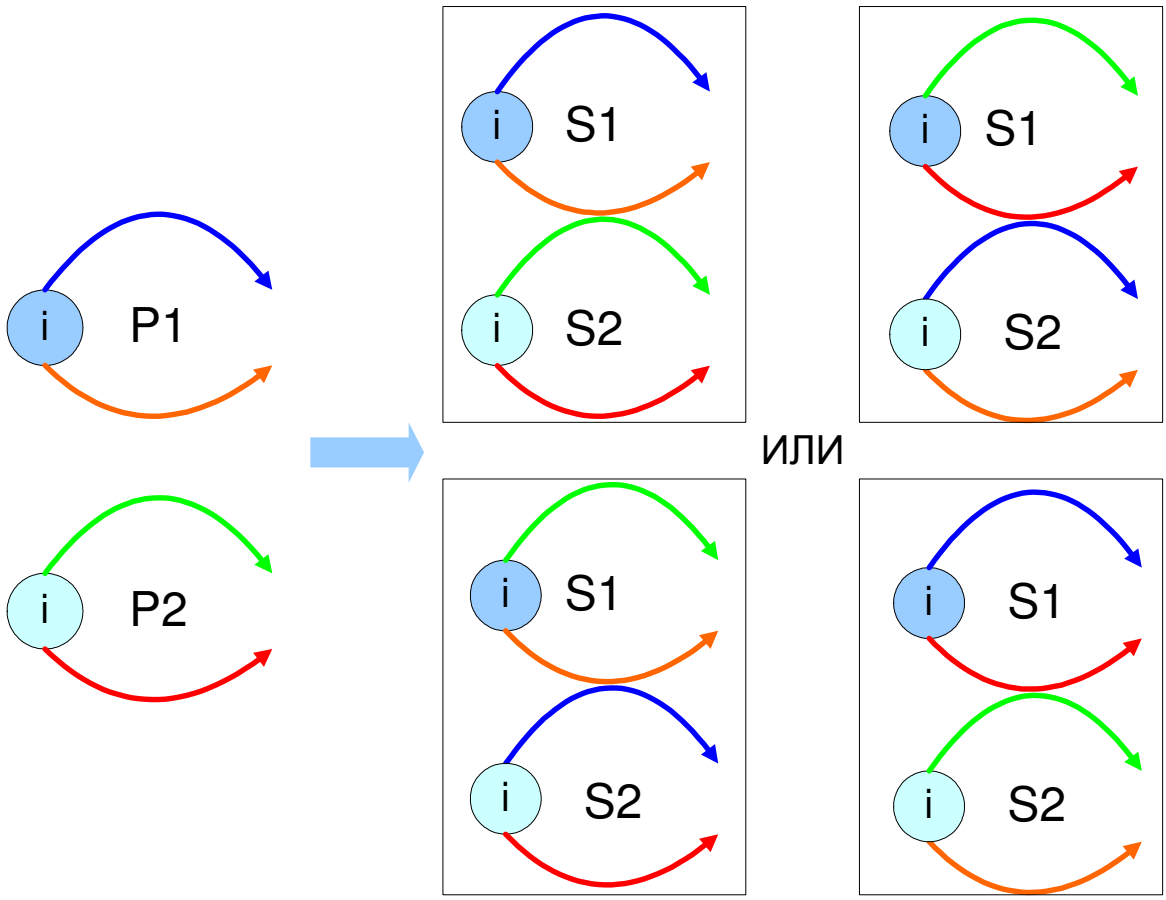

Рис. 5. Первый вариант скрещивания

Второй вариант скрещивания. В автоматах *P1* и *P2* найдем переходы, которые они выполняют в течение первых сорока ходов по игровому полю. Обозначим множество таких переходов автоматов *P1* и *P2* как *TF(P1)* и *TF(P2)* соответственно. Множество переходов некоторого автомата *A* обозначим *T(A)*. Возможны два равновероятных варианта:

- $T(S1) = TF(P1) \cup (T(P2) \setminus TF(P2))$ и $T(S2) = TF(P2) \cup (T(P1) \setminus TF(P1))$;
- $T(S2) = TF(P1) \cup (T(P2) \setminus TF(P2))$ и $T(S1) = TF(P2) \cup (T(P1) \setminus TF(P1))$.

Возможные варианты переходов при втором варианте скрещивания показаны на рис. 6.

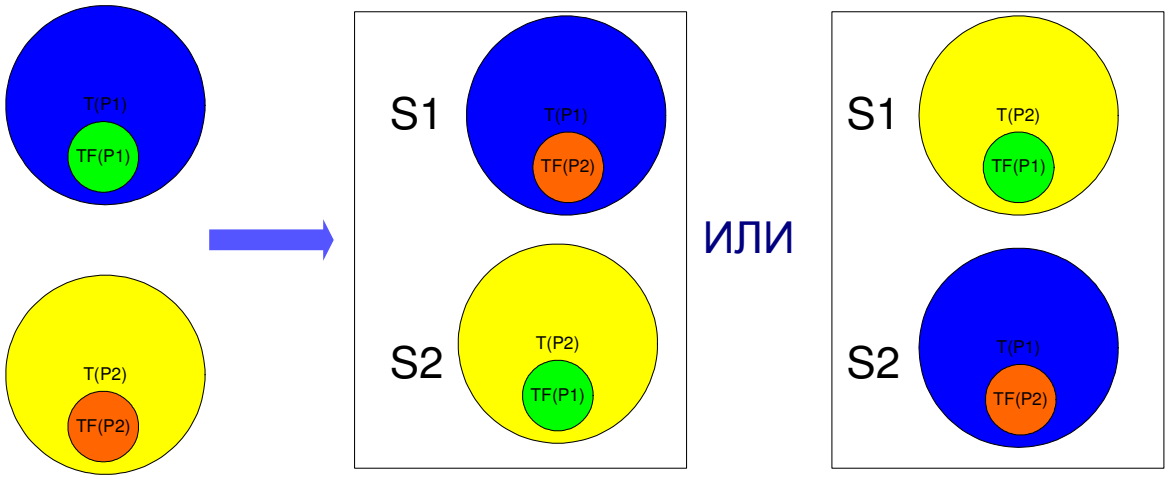

Рис. 6. Второй вариант скрещивания

Формирование следующего поколения. В качестве основной стратегии формирования следующего поколения используется элитизм [7]. При обработке текущего поколения отбрасываются все особи, кроме нескольких наиболее приспособленных. Доля выживающих особей постоянна для каждого поколения и является одним из параметров алгоритма.

Эти особи переходят в следующее поколение. После этого оно дополняется до требуемого размера следующим образом: пока оно не заполнено выбираются две особи из текущего поколения, и они с

некоторой вероятностью скрещиваются или мутируют. Обе особи, полученные в результате мутации или скрещивания, добавляются в новое поколение.

Кроме этого, если на протяжении достаточно большого числа поколений не происходит увеличения приспособленности, то применяются «малая» и «большая» мутации поколения. При «малой» мутации поколения ко всем особям, кроме 10% лучших, применяется оператор мутации. При «большой» мутации каждая особь либо мутирует, либо заменяется на случайно сгенерированную.

Число поколений до «малой» и «большой» мутации постоянно во время работы алгоритма, но может быть различным для разных его запусков.

Вычисление функции приспособленности. Функция приспособленности особи (автомата) равна $F + \frac{200 - T}{200}$, где F – количество еды, которое съедает за 200 ходов муравей, поведение которого задается этим автоматом, а T – номер хода, на котором муравей съедает последнюю единицу еды. Она вычисляется моделированием и запоминается. Таким образом, для каждой особи функция приспособленности вычисляется один раз.

Настраиваемые параметры алгоритма. Следующие параметры алгоритма генетического программирования могут быть изменены:

- размер поколения;
- доля особей, переходящих в следующее поколение;
- число состояний;
- вероятность мутации;
- время до «малой» мутации поколения;

- время до «большой» мутации поколения.

Программная реализация описанного алгоритма генетического программирования приведена в приложении 1.

2.4. Построение автомата, содержащего семь состояний

Описанный генетический алгоритм позволил построить автомат, решающий задачу об «Умном муравье» и содержащий меньше состояний, чем другие известные автоматы для этой задачи.

Отметим, что действие «Ничего не делать» бесполезно и, в некотором смысле, подобно ε-переходам в конечных недетерминированных автоматах, используемых для распознания регулярных языков. Поэтому оно может быть устранено из автоматов алгоритмом, подобным алгоритму ε-замыкания [29]. Таким образом, можно считать, что в автоматах используются только три действия (идти вперед, повернуть направо, повернуть налево).

С помощью описанного алгоритма генетического программирования был построен автомат из **семи** состояний (рис. 7), решающий задачу об «Умном муравье» за 193 хода. Этот автомат был построен за 130000 поколений, при этом было проанализировано 160 миллионов автоматов. Процесс построения занял порядка четырех часов на компьютере с процессором *Intel Celeron M 1.5 GHz*.

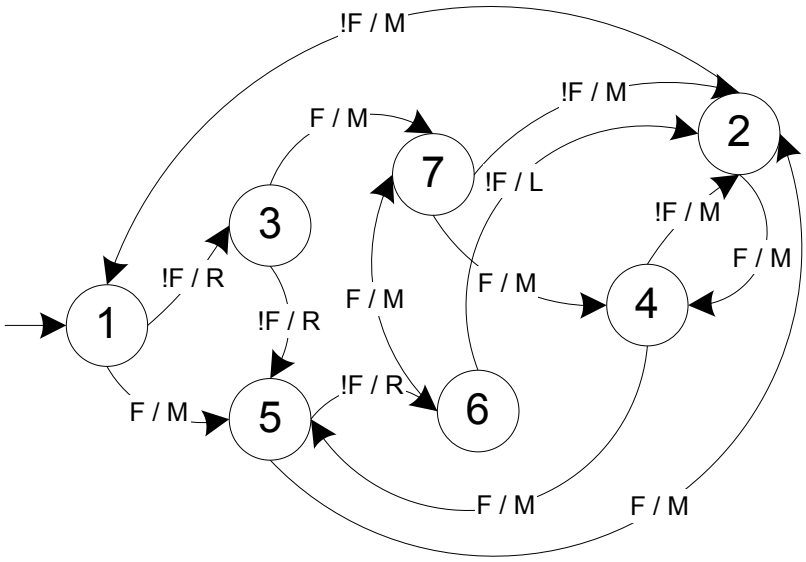

Рис. 7. Автомат из семи состояний, построенный алгоритмом генетического программирования

Отметим, что при повторном запуске алгоритма после анализа 230 миллионов автоматов, содержащих семь состояний, также был найден автомат, решающий задачу об «Умном муравье». Построить автомат, содержащий менее семи состояний, не удалось. Наилучший автомат из пяти состояний, который удалось построить, позволяет муравью съесть 83 единицы еды, из шести состояний – 85 единиц еды.

Также с помощью разработанного алгоритма было построено семейство автоматов, решающих задачу об «Умном муравье» и содержащих восемь состояний. В табл. 1 приведена краткая сводка применения описанного алгоритма к построению автоматов из восьми состояний.

Отметим также, что вопрос о существовании автоматов, позволяющих муравью съесть всю еду и содержащих при этом менее семи состояний, остается открытым.

Таблица 1. Результаты применения алгоритма для построения автомата из восьми состояний

Размер поколения	Доля особей, переходящих в следующее поколение	Время до «малой» мутации поколения	Время до «большой» мутации поколения	Число вычислений функции приспособленности
1000	0.3	100	150	1370300
1000	0.4	100	150	1235700
1000	0.5	100	150	4055200
1000	0.4	100	150	2992300
1000	0.4	100	150	1418300
1000	0.4	100	150	189400
1000	0.4	100	150	220300
1000	0.3	100	150	1948400
2000	0.5	100	150	1540000

Выводы по главе 2

1. Разработанный алгоритм генетического программирования позволяет строить автоматы, решающие задачу об «Умном муравье». Построенные разработанным алгоритмом автоматы содержат при этом меньше состояний, чем построенные с помощью известных алгоритмов.
2. Более высокая эффективность разработанного алгоритма по сравнению с существующими обусловлена тем, что используется высокоуровневое представление автоматов в виде графов переходов, а не низкоуровневое в виде битовых строк.

ГЛАВА 3. ЗАДАЧА «ЛЕТАЮЩИЕ ТАРЕЛКИ»

3.1. Постановка задачи

Приведем постановку задачи «Летающие тарелки» [16, 17, 30]. Проводится соревнование между двумя командами летающих тарелок. Цель соревнований состоит в том, чтобы одна из тарелок команды переместилась на максимальное расстояние от линии старта. Состязание проходит на трассе, представляющей собой полубесконечную (бесконечную в одну сторону) полосу шириной 40 метров. Маневры, связанные с изменением высоты полета, не допускаются (таким образом, трасса соревнования двумерна).

Каждая команда состоит из N тарелок (агентов). В дальнейшем, кроме термина «соревнование», будем использовать термин «гонка».

В начале гонки агенты первой команды располагаются в воздухе случайным образом на некотором расстоянии от линии старта в левой половине трассы, которая на экране расположена горизонтально. Вторая команда размещается симметрично первой на правой половине трассы.

Для каждого агента заданы начальная скорость и направление движения. В простейшем случае начальные скорости всех агентов одинаковы, а направления – строго вперед. Система также позволяет делать начальные скорости и направления различными. Агенты в процессе полета могут поворачивать тем самым, мешая движению других агентов.

Каждый агент имеет определенный запас топлива, расходуемого в процессе движения. По команде «Старт» все агенты начинают движение с целью максимально удалиться от линии старта. Агенты в процессе полета могут изменять скорость своего движения за счет изменения расхода топлива.

Летающие тарелки, покинувшие трассу, считаются прекратившими гонку. Выходом за пределы коридора считается пересечение центром летающей тарелки границы трассы.

Управление каждой командой выполняет программа, написанная на языке программирования *Java.*

3.1.1. Правила соревнований

В каждом соревновании каждая из команд на старте имеет *N* летающих тарелок с полным запасом топлива (10 см^3). Исходно тарелки первой команды случайным образом располагаются на первых 25 метрах левой половины трассы. Тарелки второй команды располагаются симметрично им в правой половине трассы (рис. 8).

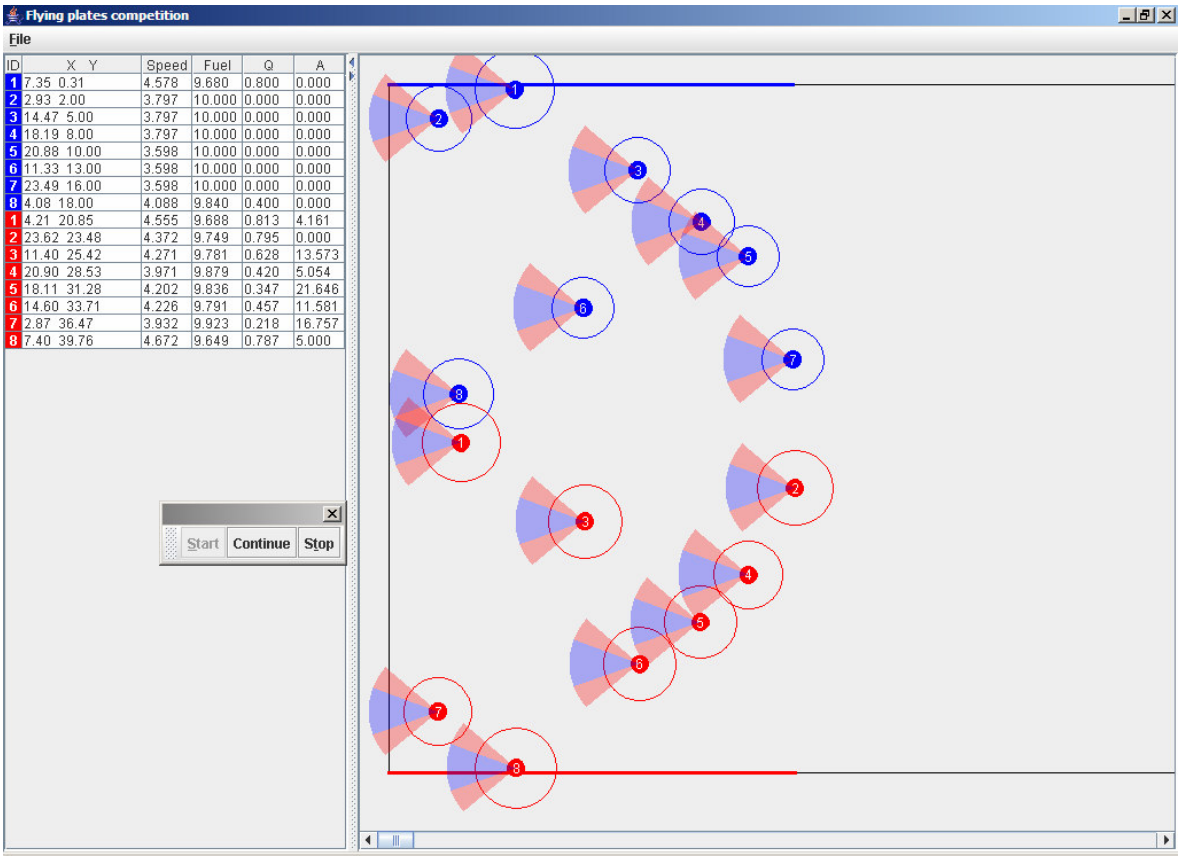

Рис. 8. Летающие тарелки на старте

Жизненный цикл летающей тарелки может быть описан графом переходов автомата, который может находиться в одном из трех

состояний: «Полет», «Нормальное завершение гонки», «Аварийное завершение гонки» (рис. 9).

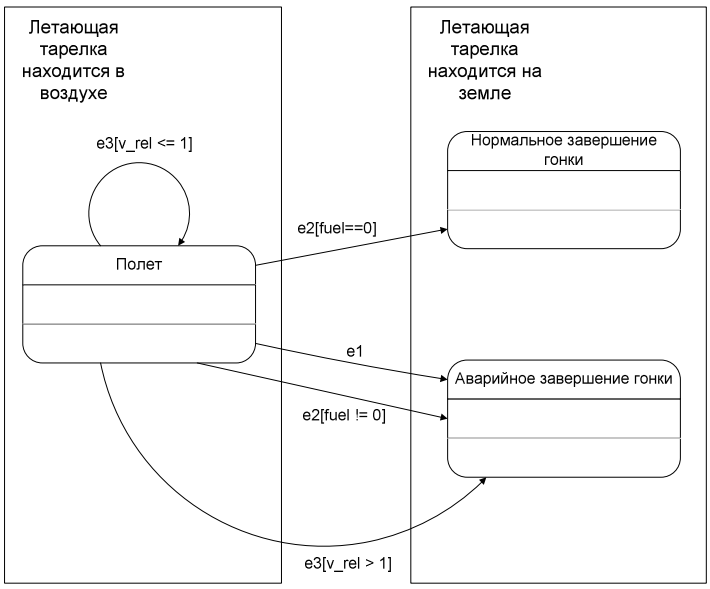

Рис. 9. Возможные состояния летающей тарелки и переходы между ними

Обозначения, используемые на рис. 9, приведены в табл. 2.

Таблица 2. Используемые обозначения

Обозначение	**Описание**
e1	Летающая тарелка покинула пределы трассы (ее центр пересек границу трассы)
e2	Скорость летающей тарелки стала меньше, чем один м/с
e3	Летающая тарелка столкнулась с другой летающей тарелкой
v_rel	Относительная скорость столкновения летающих тарелок
fuel	Количество топлива, которое осталось у летающей тарелки

Поясним поведение летающей тарелки. В начале гонки она находится в воздухе, исправна и способна продолжать участие в гонке. Этому соответствует состояние «Полет».

При выходе летающей тарелки за пределы трассы (событие *e1*) она завершает гонку аварийно.

Если скорость летающей тарелки падает ниже одного м/с (событие *e2*), и ее топливный бак не пуст (условие *fuel != 0*), то она завершает гонку аварийно. Если же при падении скорости ниже одного м/с (событие *e2*) топливный бак летающей тарелки пуст (условие *fuel==0*), то она нормально завершает гонку.

Если летающая тарелка сталкивается с другой тарелкой (событие *e3*), то при относительной скорости столкновения, большей одного м/с (условие *v_rel > 1*), тарелка аварийно завершает гонку. При относительной скорости столкновения, не превышающей одного м/с (условие *v_rel <= 1*), тарелка продолжает полет.

Заметим, что поскольку начальный запас топлива у каждой тарелки конечен, то рано или поздно все тарелки обеих команд завершат гонку.

При подведении итогов гонки учитываются только результаты тарелок, нормально ее завершивших. Результатом команды считается наибольшее из расстояний, на которое удалились от линии старта ее летающие тарелки, нормально завершившие гонку. Если все летающие тарелки команды вышли из гонки аварийно, результат команды считается равным нулю. Победителем признается команда, прошедшая наибольшее расстояние. В случае равенства результатов гонка считается завершившейся вничью.

3.1.2. Динамика летающей тарелки

Летающая тарелка представляет собой дискообразное «летающее крыло» радиусом один метр. На рис. 10 представлен вид сверху летающей тарелки.

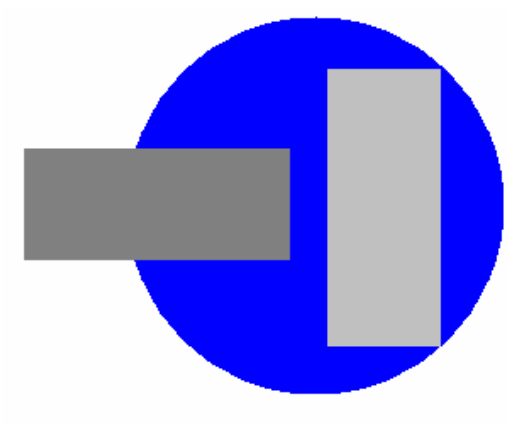

Рис. 10. Летающая тарелка

Тарелка имеет реактивный двигатель (горизонтальный прямоугольник на рис. 10), топливный бак (вертикальный прямоугольник) емкостью 15 см^3, аэродинамические рули и бортовой компьютер, способный регулировать расход топлива (и, как следствие, тягу двигателя) и положение аэродинамических рулей. Рули позволяют тарелке маневрировать. Тарелка может передвигаться со скоростями от одного метра в секунду. Максимальная скорость тарелки зависит от запаса топлива и сопротивления воздуха. Ограничение в один метр в секунду вызвано тем, что летающая тарелка с меньшей скоростью не может держаться в воздухе.

Летающая тарелка движется в соответствии со вторым законом Ньютона. Ее движение определяется двумя силами: сопротивлением воздуха F и тягой двигателя T. Если тяга не равна сопротивлению воздуха, то летающая тарелка движется с ускорением, которое может быть положительным (если тяга больше сопротивления воздуха) или отрицательным (если сопротивление воздуха больше тяги).

Ускорение определяется по формуле $a = \frac{T - F}{m}$, где m – масса летающей тарелки. При этом считается, что изменение массы тарелки за счет выгорания горючего пренебрежимо мало.

Сопротивление воздуха определяется по формуле $F = c_1 + c_2 v^2$, где v – скорость тарелки, а коэффициенты c_1 и c_2 определяются ее аэродинамическими характеристиками и одинаковы для всех тарелок обеих команд.

Тяга двигателя определяется по формуле $T = c_4 q$, где q – расход топлива в сантиметрах кубических в секунду. Расход топлива находится под контролем бортового компьютера тарелки, что позволяет изменять расход от нуля до единицы. Константа c_4 определяется характеристиками двигателя тарелки и одинакова для всех тарелок обеих команд.

Аэродинамические рули позволяют летающей тарелке поворачивать относительно ее текущего направления движения на угол, не превышающий *25°*.

3.1.3. Аэродинамическое взаимодействие между летающими тарелками

При полете летающей тарелки от траектории ее полета в направлениях назад и в стороны под углом около *30°* распространяются конические вихревые потоки воздуха. Если другая тарелка попадет в этот вихрь, то сопротивление воздуха ее полету резко **снизится** (рис. 11).

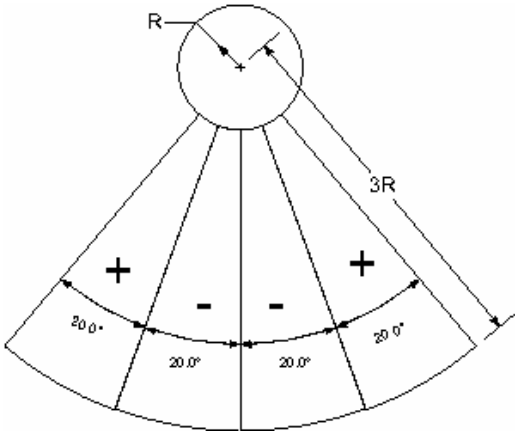

Рис. 11. Зоны повышенного и пониженного сопротивления воздуха

Отметим что, летающая тарелка, находящаяся за хвостом (два сектора по *20°*) другой летающей тарелки, испытывает **дополнительное сопротивление** движению, обусловленное реактивной струей.

Поясним, как учитывается изменение сопротивления воздуха. Если центр второй летающей тарелки находится в областях, отмеченных на рис. 12 знаком "+", сопротивление воздуха ее движению падает на 50%. Если же центр второй тарелки находится в области, помеченной знаком "–", сопротивление воздуха возрастает на 50%.

Аэродинамические воздействия от нескольких летающих тарелок складываются, так что в зоне, отмеченной на рис. 12 знаками "++", сопротивление воздуха вообще отсутствует, а в зонах, помеченных знаком "0", воздействия компенсируют друг друга. При этом в результате наложения зон воздействия от трех и более летающих тарелок сопротивление воздуха не может стать отрицательным.

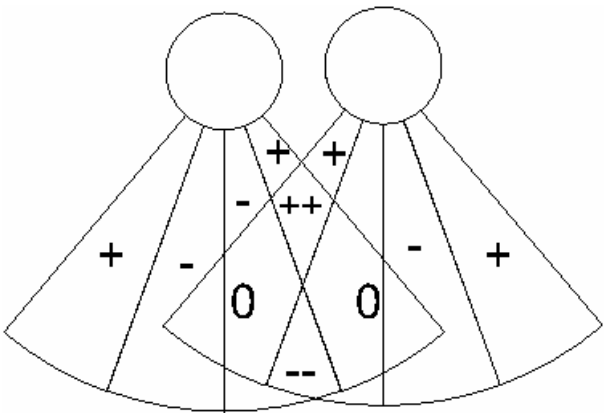

Рис. 12. Наложение областей аэродинамического взаимодействия двух летающих тарелок

Учитывая изложенное, вычисление сопротивления воздуха происходит следующим образом. Пусть N_+ – число тарелок, уменьшающих сопротивление воздуха в этой области, а N_- – число тарелок, увеличивающих сопротивление воздуха. Пусть $\Delta N = N_+ - N_-$. Если $\Delta N = 0$, то в этой области нормальное аэродинамическое сопротивление, если $\Delta N = 1$ или $\Delta N = 2$, то сопротивление понижается на $50\Delta N$ процентов. Если ΔN отрицательно, то сопротивление в этой области повышается на $50|\Delta N|$ процентов.

3.1.4. Столкновение летающих тарелок

При столкновении двух тарелок происходит их абсолютно упругое соударение без передачи вращательного момента. Если относительная скорость столкновения была более одного метра в секунду, то обе участвовавшие в столкновении летающие тарелки повреждаются и начинают терять высоту. При этом они обе аварийно завершают гонку.

Под относительной скоростью столкновения понимается проекция векторной разности скоростей летающих тарелок на прямую, проходящую через центры летающих тарелок в момент столкновения (рис. 13). Вектор V_{rel} соответствует относительной скорости.

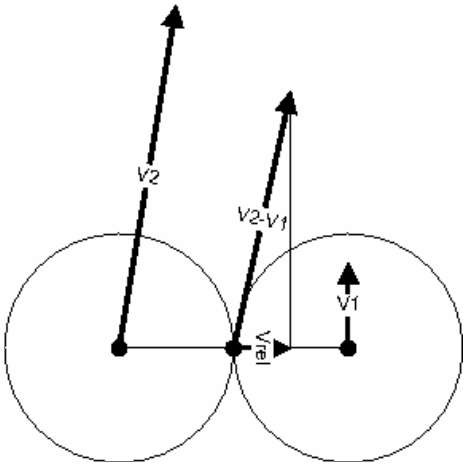

Рис. 13. Относительная скорость столкновения двух летающих тарелок

3.1.5. Моделирование гонки

Моделирование гонки происходит по ходам, каждый из которых занимает *t* миллисекунд (параметр *t* читается из конфигурационного файла). В начале каждого хода игроки обладают информацией о координатах и скоростях всех летающих тарелок. Каждому игроку предоставляется возможность установить расход топлива и угол поворота каждой тарелки своей команды.

Каждые *t* миллисекунд (один ход) происходит обновление параметров. Покажем, как выполняется моделирование полета летающих тарелок за время одного хода.

Снятие с соревнования летающих тарелок, движущихся со скоростью, меньшей одного метра в секунду. При завершении полета летающими тарелками с пустыми баками пройденные ими расстояния засчитываются в результат команды.

Расчет ускорений летающих тарелок в соответствии с установленными расходами топлива и углами поворотов, а также аэродинамическим сопротивлением. Расчет новых скоростей летающих тарелок по формуле $\vec{V}_{temp} = \vec{V}_{old} + \vec{a} \cdot \Delta t$, где V_{temp} – вектор скорости летающей тарелки после учета ускорения, V_{old} – вектор старой летающей

тарелки, a – вектор ускорения летающей тарелки. После этого происходит поворот вектора скорости на угол равный углу поворота аэродинамических рулей (рис. 14). В результате поворота получается вектор новой скорости летающей тарелки V_{new}.

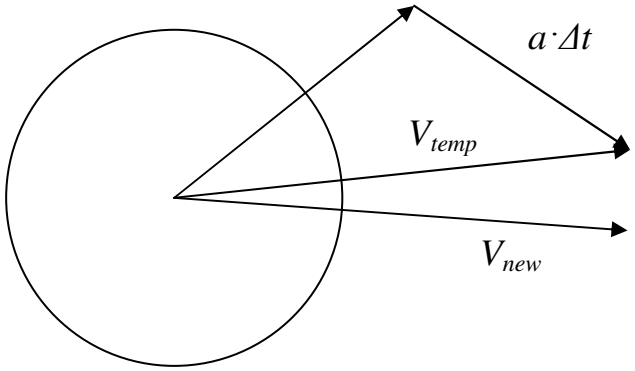

Рис. 14. Пересчет скорости летающей тарелки на шаге моделирования

Снятие летающих тарелок, движущихся медленнее одного метра в секунду. Как и ранее, при завершении полета летающими тарелками с пустыми баками, пройденные ими расстояния засчитываются в результат команды.

Происходит равномерное прямолинейное движение летающих тарелок (считается, что за время шага моделирования скорости тарелок не меняются). Если при этом происходит соударение тарелок – расстояние между центрами каких-либо двух тарелок становится меньше двух метров, то их скорости и координаты изменяются в соответствии с законами сохранения импульса и энергии. При этом летающие тарелки, относительная скорость столкновения которых превосходила один метр в секунду, выбывают из гонки.

Проверка того, что все летающие тарелки находятся в пределах трассы. При выходе центра тарелки за пределы трассы, она выбывает из гонки.

Гонка продолжается до тех пор, пока ее не завершила хотя бы одна тарелка. После того, как ее закончит и эта тарелка, гонка завершается.

Отметим, в чем состоит различие понятий «ход» и «шаг» в настоящей работе. Под **шагом** моделирования подразумевается квант времени в программе, все временные промежутке в программе должны быть ему кратны. При этом передача управления системам управления летающими тарелками (разрешение каждой из них произвести **ход**) выполняется через промежутки времени равные шагу моделирования. Между двумя соседними ходами состояние внешней среды изменяется так, как будто между ними прошло время, равное шагу моделирования.

3.2. Известное решение задачи

В работе [17] при участии автора было предложено решение задачи, в котором система управления летающими тарелками строится с использованием парадигмы автоматного программирования и инструментального средства *UniMod*. Все автоматы в указанной работе были построены вручную. Их построение было достаточно трудоемким.

В настоящей работе предлагается подход к автоматизированному построению управляющих систем на основе *генетического программирования*, *искусственных нейронных сетей* и *конечных автоматов*.

3.3. Искусственные нейронные сети

Нейрон – это клетка головного мозга или нервной системы, основной функцией которой является сбор, обработка и распространение электрических сигналов. Считается, что способность мозга к обработке информации обусловлена функционированием сетей, состоящих из нейронов.

Одна из первых математических моделей нейрона предложена Мак-Каллоком (McCulloch) и Питтсом (Pitts) [31]. С 1943 года были разработаны более подробные и реалистичные модели, как нейрона, так и более крупных систем мозга. Это привело к созданию новой научной области – *вычислительной неврологии*. С другой стороны, исследователи в области искусственного интеллекта исследовали более абстрактные свойства нейронных сетей: способность выполнять распределенные вычисления, справляться с зашумленными входными данными, способность обучаться.

Со временем стало ясно, что похожими свойствами обладают и другие системы (такие, как, например, байесовские сети). Однако *искусственные нейронные сети* [1] по сей день остаются одним из наиболее изученных и широко применяемых методов искусственного интеллекта.

3.3.1. Элементы искусственных нейронных сетей

Искусственные нейронные сети состоят из *узлов* (искусственных нейронов), соединенных между собой *связями*. Связь от элемента i к элементу j служит для распространения *активации* a_j от j к i. Каждая связь имеет назначенный ей числовой *вес* $W_{i,\ j}$. Каждый элемент вычисляет взвешенную сумму своих входных данных: $in_i = \sum_{j=0}^{n} W_{j,i} a_j$ и применяет к ней *функцию активации* g: $a_i = g(in_i) = g(\sum_{j=0}^{n} W_{j,i} a_j)$.

Обратим внимание на то, что в формулу входит *смещенный вес* $W_{0,i}$, относящийся к постоянному входному значению $a_0 = -1$.

Основными видами функций активации являются:

- *пороговая* функция $g(x) = \begin{cases} 1, x \geq 0 \\ 0, x < 0 \end{cases}$;
- *знаковая* функция $g(x) = \begin{cases} 1, x \geq 0 \\ -1, x < 0 \end{cases}$;
- *сигмоидальная* (*логистическая*) функция $\mathrm{g(x)} = \frac{1}{1 + \mathrm{e}^{-x}}$.

Отметим, что обе функции имеют пороговое значение около нуля, а смещенный вес $W_{0,i}$ фактически задает пороговое значение для данного элемента.

Важным свойством элементов с пороговой функцией активации является то, что с их помощью можно представить логические функции *AND*, *OR* и *NOT* [1]. Таким образом, с помощью нескольких таких элементов можно выразить любую булеву функцию от входов сети.

3.3.2. Структура искусственных нейронных сетей

Существуют две основные категории структур нейронных сетей: ациклические сети, или *сети с прямым распространением*, и циклические, или *рекуррентные* сети.

Сети с прямым распространением реализуют некоторую функцию от своих входов, в то время как в рекуррентной сети выходы ее элементов могут подаваться на вход. В связи с этим уровни активации в рекуррентной сети могут находиться в устойчивом состоянии, могут переходить в колебательный или даже в хаотический режим.

Рекуррентные сети представляют собой более сложную для понимания и исследования модель. В настоящей работе в дальнейшем будут использоваться только сети с прямым распространением.

3.3.3. Применение искусственных нейронных сетей

Наиболее часто нейронные сети применяются для решения следующих задач:

- *классификация образов* – указание принадлежности входного образа, представленного вектором признаков, одному или нескольким предварительно определенным классам;
- *кластеризация* – классификация образов при отсутствии обучающей выборки с метками классов;
- *прогнозирование* – предсказание значения y_{n+1} при известной последовательности $y_1, y_2 \ldots y_n$.

3.4. Предлагаемый подход к решению задачи

3.4.1. Некоторые проблемы, возникающие при использовании генетического программирования для построения конечных автоматов

Генетическое программирование наиболее эффективно в тех случаях, когда оптимизируемый объект (например, конечный автомат) имеет небольшой размер (небольшое количество состояний). В то же время, число различных вариантов значений входных переменных может быть достаточно большим, а сами переменные могут быть не только логическими, но и числовыми. Например, в рассматриваемой задаче такие входные переменные, как скорость летающей тарелки или ее координаты являются вещественными входными переменными.

Используемый для решения задачи об «Умном муравье» в настоящей работе алгоритм разработан для случая входных переменных логического типа. Для того чтобы применять этот или аналогичный ему, необходимо разработать способ перехода от произвольных входных переменных к логическим входным переменным (или хотя бы, к переменным, множество значений которых конечно и содержит небольшое число элементов).

Одним из вариантов решения этой задачи является введение соответствующих переменных вручную. Например, если исходно были две вещественные переменные x и y, то, например, можно ввести две новые логические переменные $A := x > 100$ и $B := y < 200$.

Второй вариант решения состоит в том, чтобы разбить множество значений входных переменных на несколько областей и использовать в качестве значения входной переменной номер области, в которой лежат текущие значения входных переменных. Таким образом, необходимо будет перед тем, как подавать данные на вход автоматы, определять, в какой из областей лежит набор текущих значений входных переменных. Это – задача классификации. Если для ее решения применять автоматический классификатор (нейронная сеть, дерево принятия решений, и т. д.), то возникает идея настраивать этот классификатор совместно с автоматом, с которым он связан.

В настоящей работе реализован второй подход. В качестве классификатора используется искусственная нейронная сеть. Ее настройка и построение автомата производятся с помощью генетического программирования.

3.4.2. Структура системы управления летающей тарелкой

Каждая летающая тарелка управляется своей системой, состоящей из искусственной нейронной сети и конечного автомата. Таким образом, можно говорить, что используется *мультиагентный подход* [1] – каждая летающая тарелка представляет собой агента, взаимодействующего с внешней средой и другими агентами.

При этом, как отмечалось выше, нейронная сеть используется для классификации значений вещественных входных переменных и выработки входных переменных для автомата, а автомат – для выработки выходных воздействий на летающую тарелку (рис. 15).

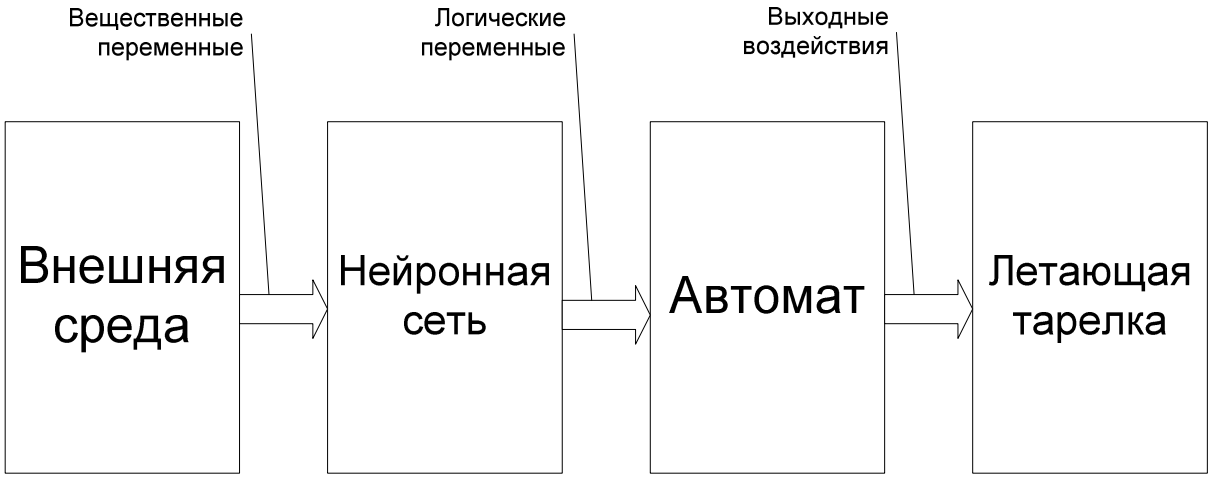

Рис. 15. Структурная схема системы управления летающей тарелкой

Структура нейронной сети и способ ее взаимодействия с конечным автоматом показаны на рис. 16.

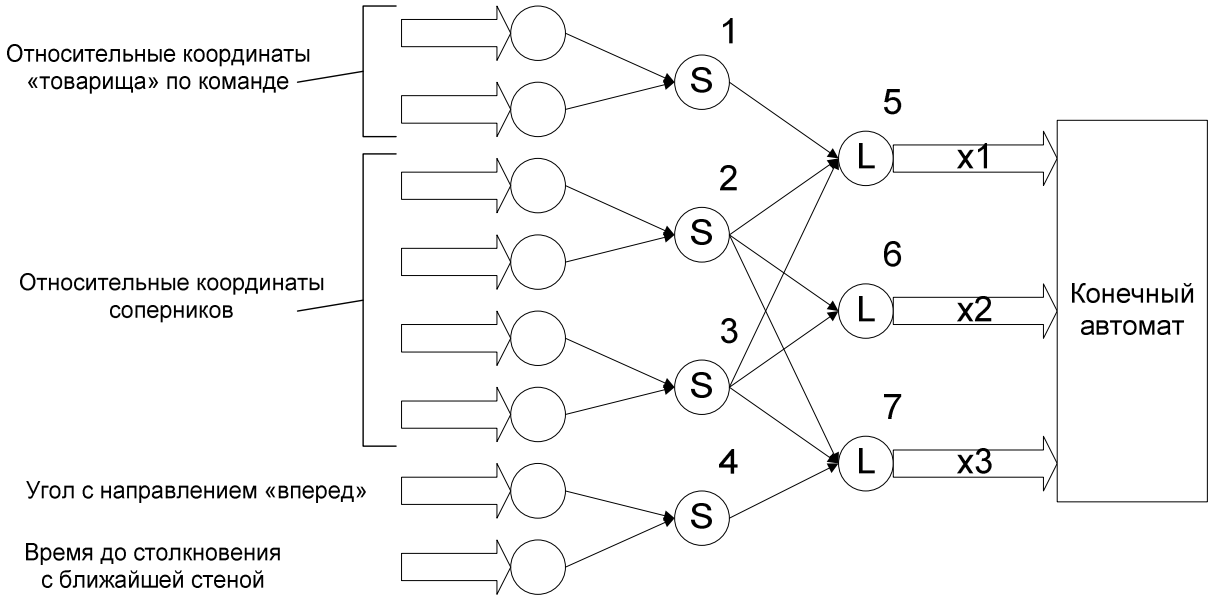

Рис. 16. Нейронная сеть и ее взаимодействие с конечным автоматом

Символами **S** на рис. 16 обозначены искусственные нейроны с сигмоидальной функцией активации, символом **L** – нейроны с пороговой функцией активации. Рядом с нейронами указаны их номера (они используются при описании операции скрещивания нейронных сетей). На каждый из трех выходов нейронной сети поступает число равное 0 или 1. Таким образом, существует восемь вариантов комбинаций выходных сигналов нейронной сети (000, 001, 010, 011, 100, 101, 110, 111). Поскольку

выходные значения нейронной сети подаются на вход конечному автомату, то существует восемь возможных комбинаций трех входных переменных конечного автомата.

3.4.3. Алгоритм генетического программирования для построения системы управления летающей тарелкой

Используемый для построения систем управления летающей тарелкой алгоритм генетического программирования во многом аналогичен алгоритму, описанному в разд. 2.3. Отличия состоят в структуре особи, конкретных генетических операторах мутации и скрещивания и в функции приспособленности особи.

3.4.4. Структура особи

В связи с тем, что генетическое программирование эффективно только в случае небольшого размера особи, а количество летающих тарелок в каждой команде может быть достаточно велико, возникла идея совместно строить управляющие системы только для двух летающих тарелок. При этом также предполагается, что летающая тарелка может достаточно «хорошо» управляться даже при наличии небольшого количества информации о внешней среде. Поэтому была выбрана структура нейронной сети, указанная в разд. 3.4.2. При этом отметим, что две летающие тарелки были выбраны только для генерации систем управления с помощью генетического программирования, а в дальнейшем построенные системы управления будут размножены в количестве, необходимом для формирования команды – шесть или восемь штук.

Таким образом, особь в описываемом алгоритме генетического программирования состоит из двух систем управления летающей тарелкой (рис. 17). Таким образом, особь описывает две системы управления летающей тарелкой, которые в дальнейшем будут составлять одну команду при проведении соревнований.

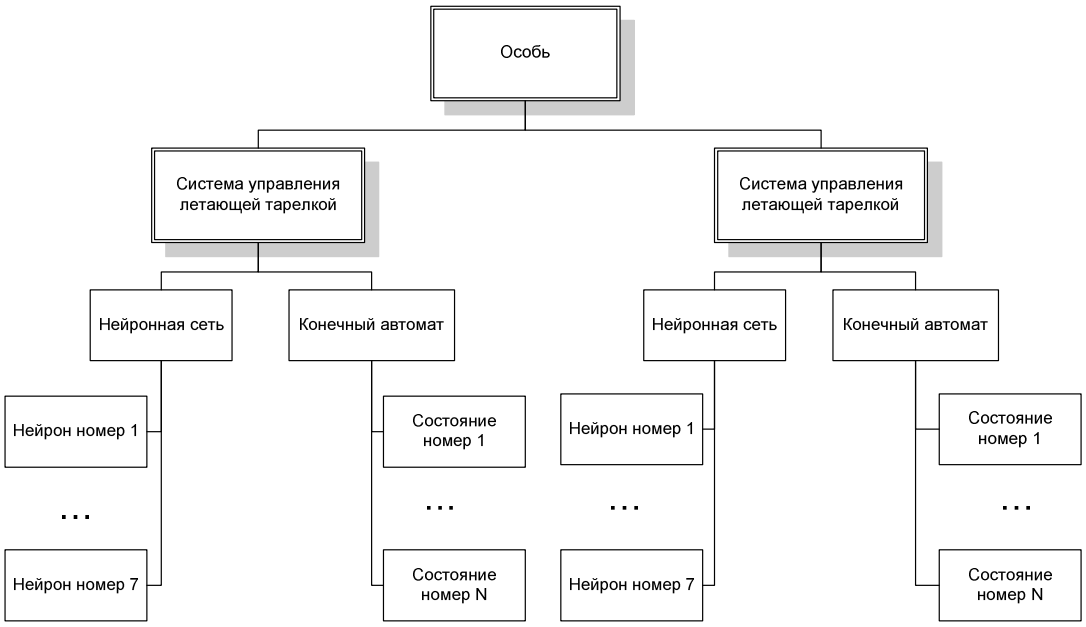

Рис. 17. Структура особи

Каждая система управления летающей тарелкой состоит из нейронной сети и конечного автомата. Описание нейронной сети состоит из четырех нейронов с сигмоидальной функцией активации и трех нейронов с пороговой функцией активации. Каждый из нейронов характеризуется порогом активации и весами связей, которые соединяют другие элементы сети с рассматриваемым:

```
public abstract class Neuron {
    protected Neuron[] inputs;
    protected int inputsCnt;
    protected double[] w;
}
```

Описание конечного автомата состоит из номера начального состояния и описания состояний. Описание состояния состоит из описаний восьми переходов, соответствующих восьми вариантам выходных сигналов нейронной сети. Описание каждого перехода состоит из номера состояния, в которое ведет этот переход, и двух действий, которые

выполняются при выборе этого перехода – изменения расхода топлива и изменения угла поворота аэродинамических рулей. Каждое из этих действий характеризуется одним вещественным числом – соответственно, новым расходом топлива и углом поворота аэродинамических рулей:

```
public class Individual {
    protected PlateControlSystem[] pcs;
}
public class PlateControlSystem {
    private NeuralNet neuralNet;
    private Automaton automaton;
}
```

3.4.5. Операция мутации

Мутация особи. При мутации особи с равной вероятностью мутирует либо одна система управления летающей тарелкой, либо вторая.

Мутация системы управления летающей тарелкой. При мутации системы управления летающей тарелкой мутирует либо нейронная сеть, либо конечный автомат.

Мутация нейронной сети. При мутации нейронной сети мутирует случайно и равновероятно выбирается один элемент (искусственный нейрон) сети и мутирует.

Мутация элемента сети. При мутации элемента сети случайно выбирается один из весов связей, и к нему прибавляется случайное число из отрезка [-0.05; 0.05]. Кроме этого, с вероятностью 0.5 аналогичная операция производится с лимитом активации нейрона.

Мутация конечного автомата. При мутации конечного автомата равной вероятностью производится либо изменение начального состояния, либо мутация случайно выбранного перехода.

Изменение начального состояния. Начальное состояние меняется на случайно выбранное состояние автомата.

Мутация перехода. При мутации перехода с равной вероятностью происходит либо изменение номера состояния, в которое ведет переход, либо мутация одного из действий, связанных с переходом – может мутировать действие, связанное с изменением расхода топлива (прибавляется случайное число от -0.05 до 0.05) или с изменением угла поворота аэродинамических рулей (уменьшается или увеличивается на 5°).

3.4.6. Операция скрещивания

Оператор скрещивания получает на вход две особи (*P1*, *P2*) и выдает две особи (*S1*, *S2*). Пусть *X* – некоторая особь. Обозначим как *X.s1* и *X.s2* системы управления летающими тарелками, входящие в эту особь. Пусть *s* – некоторая система управления летающей тарелкой. Обозначим как *s.ns* входящую в нее нейронную сеть, а как *s.a* – входящий в нее автомат.

Скрещивание особей. При скрещивании особей происходит скрещивание систем управления летающими тарелками: *P1.s1* и *P2.s1*, *P1.s2* и *P2.s2*. Обозначим системы, получившиеся в результате первого скрещивания как *s11* и *s12*, а в результате второго – как *s21* и *s22*. Тогда для особей-потомков будет справедливо: *S1.s1 = s11*, *S1.s2 = s21*, *S2.s1 = s21*, *S2.s2 = s22*.

Скрещивание систем управления летающей тарелкой. При скрещивании систем управления летающей тарелкой *s1* и *s2* тарелкой

происходит скрещивание автоматов *s1.a* и *s2.a* и скрещивание нейронных сетей *s1.ns* и *s2.ns*. Обозначим получающиеся в результате описанных скрещиваний автоматы как *a1* и *a2*, а нейронные сети – как *ns1* и *ns2*. В результате скрещивания системы управления летающей тарелкой получаются системы управления *s3* и *s4*, содержащие следующие элементы: *s3* содержит *a1* и *ns1*, *s4 – a2* и *ns2*.

Скрещивание автоматов. Обозначим автоматы, поступающие на вход оператора скрещивания автоматов как *A1* и *A2*. Начальное состояние некоторого автомата *A* обозначим как *A.is*, а переход из состояния *i* по значению входной переменной *j* как *A(i, j)*. Обозначим автоматы, получающиеся в результате скрещивания как *A3* и *A4*. Для их начальных состояний будет справедливо:

- либо *A3.is* = *A1.is* и *A4.is* = *A2.is*;
- либо *A3.is* = *A2.is* и *A4.is* = *A1.is*.

Опишем переходы автоматов *A3* и *A4*. Скрещивание производится отдельно для каждого состояния *i* и для каждого значения *j* входной переменной. В каждом случае возможно два равновероятных варианта:

- *A3(i, j)* = *A1(i, j)* и *A4(i, j)* = *A2(i, j)*;
- *A3(i, j)* = *A2(i, j)* и *A4(i, j)* = *A1(i, j)*.

Скрещивание нейронных сетей. Обозначим нейронные сети, поступающие на вход оператора скрещивания нейронных сетей как *NS1* и *NS2*, а получающиеся в результате его применения – как *NS3* и *NS4*. Опишем их устройство. Обозначим как *NS(i)* нейрон номер *i* сети *NS* (рис. 16). Для нейронов номер *NS3(i)* и *NS4(i)* возможны два варианта:

- *NS3(i)* = *NS1(i)* и *NS4(i)* = *NS2(i)*;
- *NS3(i)* = *NS2(i)* и *NS4(i)* = *NS1(i)*.

3.4.7. Функция приспособленности

Функция приспособленности особи вычисляется с помощью соревнования команды, тарелки которой управляются описываемыми особью системами управления летающей тарелкой, с некоторым набором команд, управляемых известными системами управления командами. В качестве таких систем в настоящей работе используются системы, реализующие «агрессивную» и «простую» стратегию [17].

Проводилось по пять соревнований с каждой из стратегий при следующих параметрах летающих тарелок:

$$c_1 = 0.625; \quad (1)$$

$$c_2 = 0.025; \quad (2)$$

$$c_4 = 3.125; \quad (3)$$

$$\Delta t = 0.3; \quad (4)$$

$$L = 7. \quad (5)$$

Результатом вычисления функции приспособленности является сумма результатов команды, тарелки которой управляются описываемыми особью системами управления летающей тарелкой, во всех соревнованиях, к которой прибавлено число побед, деленное на число соревнований, увеличенное на единицу.

Программная реализация описанного алгоритма генетического программирования приведена в приложении 2.

3.5. Результаты применения генетического программирования

Системы управления летающей тарелкой строились с помощью описанного алгоритма генетического программирования, а далее тестировались в среде, разработанной в работе [17]. Тестирование проводилось с помощью соревнования построенной системы с командой, тарелки которой управляются системой, описанной в работе [17].

При этом соревнования проводились при числе летающих тарелок в каждой команде, равном шести или восьми. Для того, чтобы построенные с помощью генетического программирования системы управления летающей тарелкой могли работать в этом случае, на первые два входа нейронной сети (рис. 16) подавались относительные координаты ближайшей тарелки из «своей» команды, а на входы с третьего по шестой подавались координаты двух ближайших тарелок из обеих команд. При этом летающие тарелки с нечетными номерами управлялись системой, построенной для первой тарелки, а с четными номерами – построенной для второй тарелки.

С помощью описанного алгоритма генетического программирования была построена особь, содержащая две системы управления летающими тарелками, каждая из которых содержит автомат с шестью состояниями. Их построение заняло около суток на компьютере с процессором *Intel Celeron 2.53 GHz*.

Функция переходов и действий автомата, входящего в систему управления первой тарелкой, приведена в табл. 3. Строки этой таблицы соответствуют состояниям автоматы (пронумерованы числами от 0 до 5), столбцы – возможным комбинациям значений трех входных переменных. Серым цветом отмечена строка, соответствующая начальному состоянию. Ячейки имеют формат, показанный в табл. 3.

Таблица 3. Формат ячеек табл. 4

Новое состояние	
Изменение угла поворота	Новый расход топлива

Таблица 4. Функция переходов и функция действий автомата, управляющего первой тарелкой

	000		**001**		**010**		**011**		**100**		**101**		**110**		**111**	
0	0		5		3		5		4		2		5		4	
	5	0,8	25	0,4	-5	0,8	15	0,8	5	0,4	-5	0,8	-10	0,4	25	0,8
1	3		2		3		5		4		2		0		5	
	-20	0,4	25	0,4	15	0,8	-15	0,4	5	0,4	20	0,4	-10	0,8	0	0,8
2	2		3		4		5		1		5		2		0	
	20	0,4	-5	0,8	15	0,8	10	0,4	20	0,8	10	0,8	5	0,4	-20	0,4
3	5		3		5		4		5		2		0		4	
	-20	0,4	-20	0,8	10	0,8	-10	0,4	-25	0,8	5	0,4	-5	0,8	-25	0,8
4	0		2		5		2		5		0		1		2	
	0	0,4	5	0,4	-10	0,8	-10	0,44	-10	0,8	10	0,8	-15	0,4	-5	0,8
5	2		1		2		3		1		4		2		2	
	-25	0,78	-15	0,4	-20	0,4	10	0,4	-20	0,8	-5	0,8	0	0,8	-20	0,4

В табл. 5 приведены веса связей в нейронной сети, входящей в построенную систему управления первой летающей тарелкой.

Таблица 5. Веса связей и смещенные веса нейронов, входящих в нейроную сеть, управляющую первой летающей тарелкой

Номер нейрона	**Номера нейронов, соединенных с данным**	**Вес связи**	**«Смещенный вес»**
1	Вход-1	*0,7820135561918911*	*0,0000823910892*
	Вход-2	*0,9843639098567366*	
2	Вход-3	*0,8621015893426324*	*0,0000342189215*
	Вход-4	*0,49004984673483654*	
3	Вход-5	*0,31478285017906643*	*0,0000238490014*
	Вход-6	*0,9111794712756234*	
4	Вход-7	*0,9383238994033571*	*0,0000341289013*
	Вход-8	*0,5244096391670159*	
5	1	*-0,9357005374202394*	*0,4971125943935544*
	2	*-0,4004385439577506*	
	3	*0,97490945048913*	
6	2	*0,9562669142389417*	*-0,568426590601512*
	3	*-0,2903272439699007*	
7	2	*0,1400255181142766*	*-0,859430022111886*
	3	*-0,315932752555524*	
	4	*-0,5112115605370626*	

В табл. 6 показаны значения функций переходов и действий для автомата, управляющего второй тарелкой. Обозначения и формат ячеек в табл. 6 такие же, как и в табл. 4.

Таблица 6. Функция переходов и функция действий автомата, управляющего второй тарелкой

	000		**001**		**010**		**011**		**100**		**101**		**110**		**111**	
0	2		3		3		4		0		4		0		3	
	-15	0,4	5	0,4	10	0,4	-20	0,4	-5	0,4	15	0,4	15	0,4	-25	0,4
1	0		3		3		3		3		5		2		5	
	-25	0,8	-15	0,4	25	0,4	-10	0,4	-25	0,8	-5	0,4	-15	0,8	-10	0,4
2	4		0		0		0		2		0		3		2	
	-15	0,8	5	0,8	25	0,4	15	0,4	-25	0,4	-5	0,4	0	0,4	10	0,8
3	3		3		2		3		3		5		2		1	
	0	0,4	20	0,8	20	0,8	-25	0,8	-15	0,8	20	0,4	5	0,4	15	0,4
4	4		4		5		0		4		2		3		2	
	5	0,8	-20	0,4	25	0,8	25	0,4	0	0,38	-15	0,4	0	0,8	-15	0,36
5	3		4		0		0		4		4		4		2	
	-15	0,4	-20	0,4	0	0,8	10	0,4	-10	0,8	10	0,35	0	0,4	0	0,8

В табл. 7 приведены веса связей в нейронной сети, входящей в построенную систему управления первой летающей тарелкой.

Таблица 7. Веса связей и смещенные веса нейронов, входящих в нейроную сеть, управляющую первой летающей тарелкой

Номер нейрона	**Номера нейронов, соединенных с данным**	**Вес связи**	**«Смещенный вес»**
1	Вход-1	*0,23234941730420006*	*0,0003223789*
	Вход-2	*0,07485700920671275*	
2	Вход-3	*0,3056025308692568*	*0,0030210123478*
	Вход-4	*0,30661997660895013*	
3	Вход-5	*0,17219120661378778*	*0,0000728973223*
	Вход-6	*0,03018455303178147*	
4	Вход-7	*0,23636134465917025*	*0,00430412789122*
	Вход-8	*0,806175323870498*	
5	1	*0,9180350210858039*	*-0,8415486736799431*
	2	*-0,4638322712743177*	
	3	*-0,5253829213469569*	
6	2	*-0,8334567897770782*	*0,05661834053390668*
	3	*-0,7931512598511974*	
7	2	*0,3096107059232607*	*0,2372057991691212*
	3	*0,13115163005946284*	
	4	*-1,0*	

Тестирование проводилось при числе тарелок в каждой команде, равном шести и восьми – проводилось 30 соревнований, и учитывался результат команды и число тарелок, успешно завершивших гонку.

Сводка результатов соревнований при размере команды в **шесть** тарелок с системой из работы [17] приведена в табл. 8.

Таблица 8. Результаты соревнований команд из шести тарелок

	Построенная с помощью генетического программирования система	**Система из работы** [17]
Среднее	208,4	212,126
Минимум	0,00	204,21
Максимум	228,15	218,87
Среднее без учета худших результатов	215,4	212,196

Более детальную информацию о результатах соревнований можно получить из гистограмм распределений результатов команд и гистограмм распределений числа летающих тарелок, успешно завершивших соревнование.

На рис. 18 показано распределение результатов команды, летающие тарелки которой управляются системами управления, построенными с помощью генетического программирования.

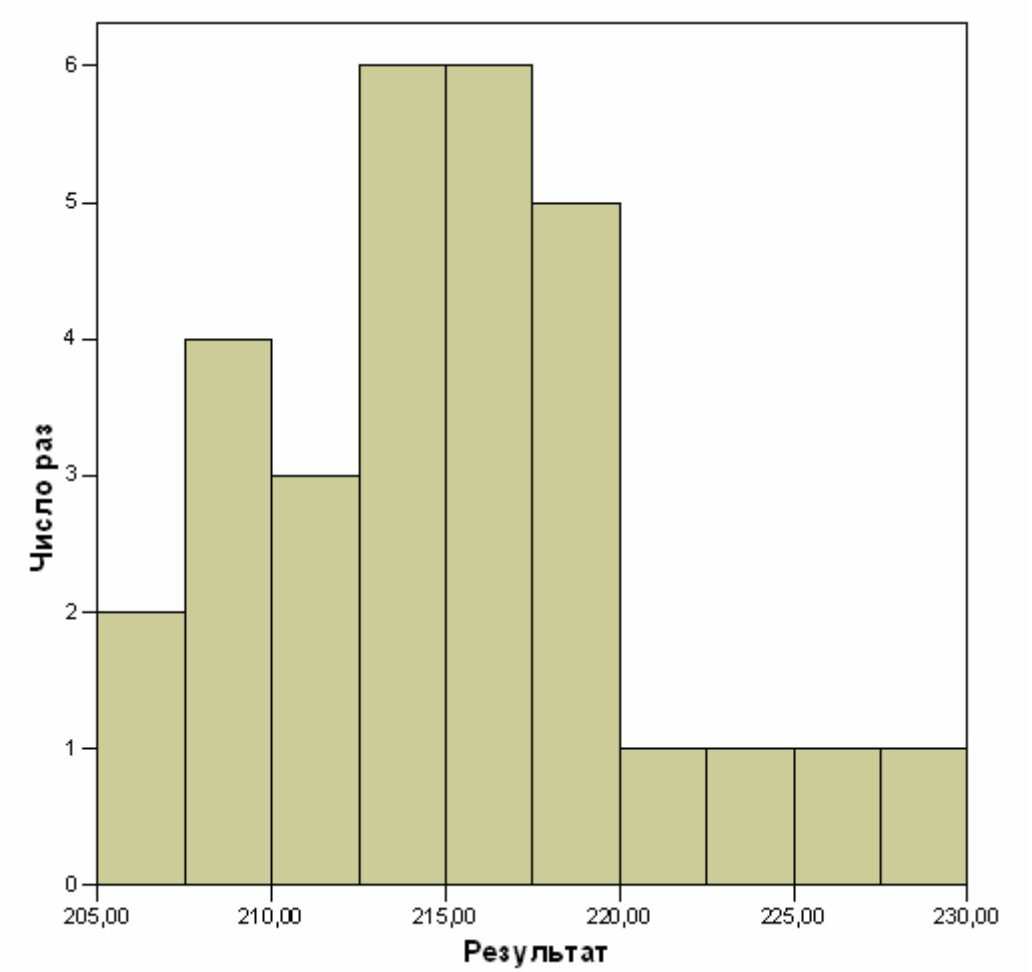

Рис. 18. Распределение результатов команды, летающие тарелки которой управляются системами управления, построенными с помощью генетического программирования (размер команды – шесть тарелок)

На рис. 18 отражены (для наглядности) только ненулевые результаты. В противном случае ширина листа А4 не позволила бы достаточно детально отобразить распределение. Отметим, что в одном соревновании построенная с помощью генетического программирования система показала нулевой результат.

На рис. 19 показано распределение результатов команды, тарелки которой управляются системой, построенной вручную.

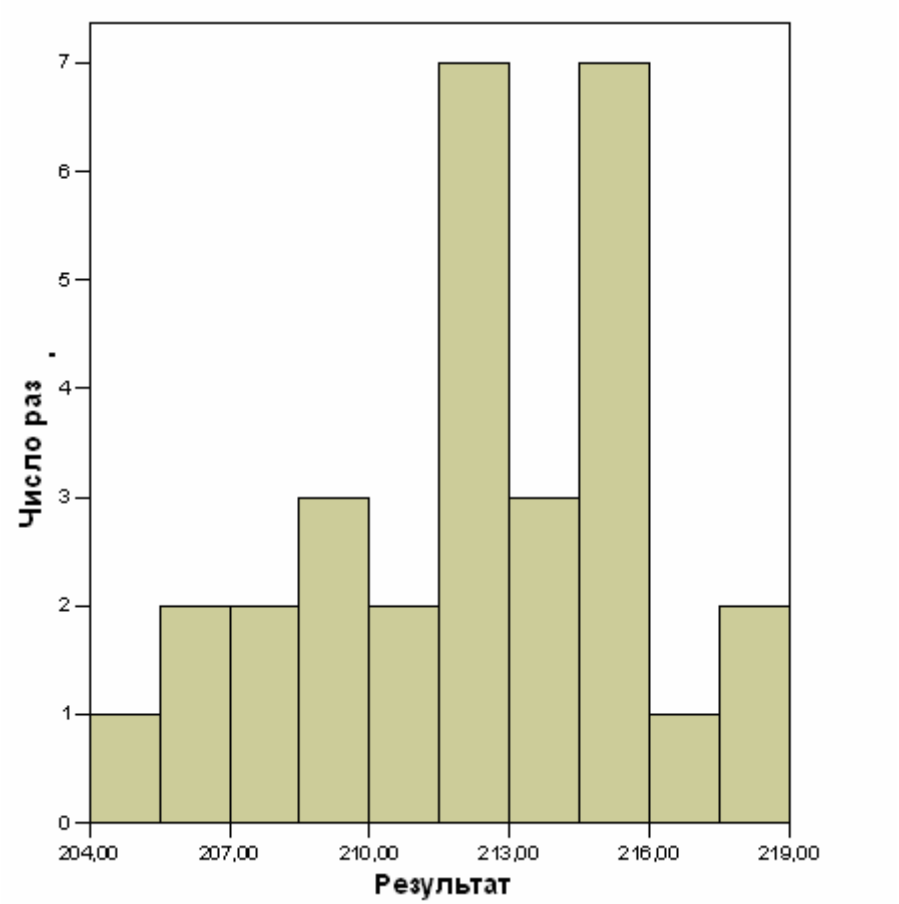

Рис. 19. Распределение результатов команды, тарелки которой управляются системами, построенными вручную (размер команды – шесть тарелок)

На рис. 20 показано распределение числа успешно завершивших соревнование тарелок для команды, летающие тарелки которой управляются системами управления, построенными с помощью генетического программирования.

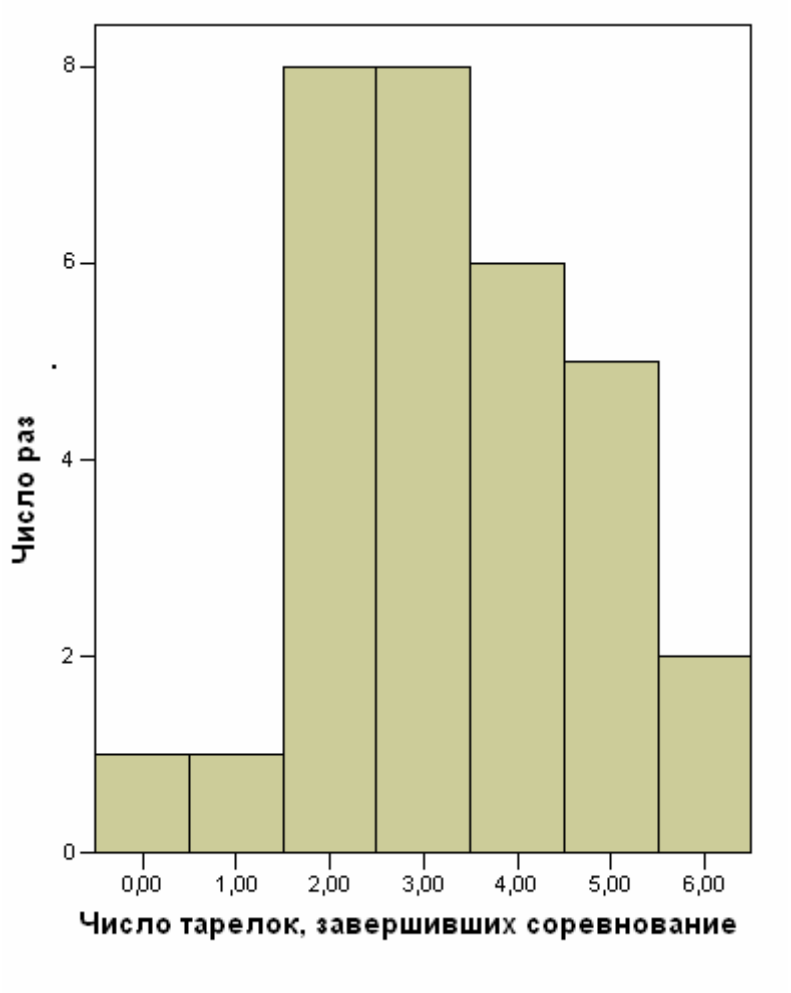

Рис. 20. Распределение числа летающих тарелок, успешно завершивших соревнование, для команды, тарелки которой управляются системами управления, построенными с помощью генетического программирования (размер команды – шесть тарелок)

На рис. 21 показано распределение количества успешно завершивших соревнование тарелок для команды, летающие тарелки которой управляются системами управления, построенными вручную.

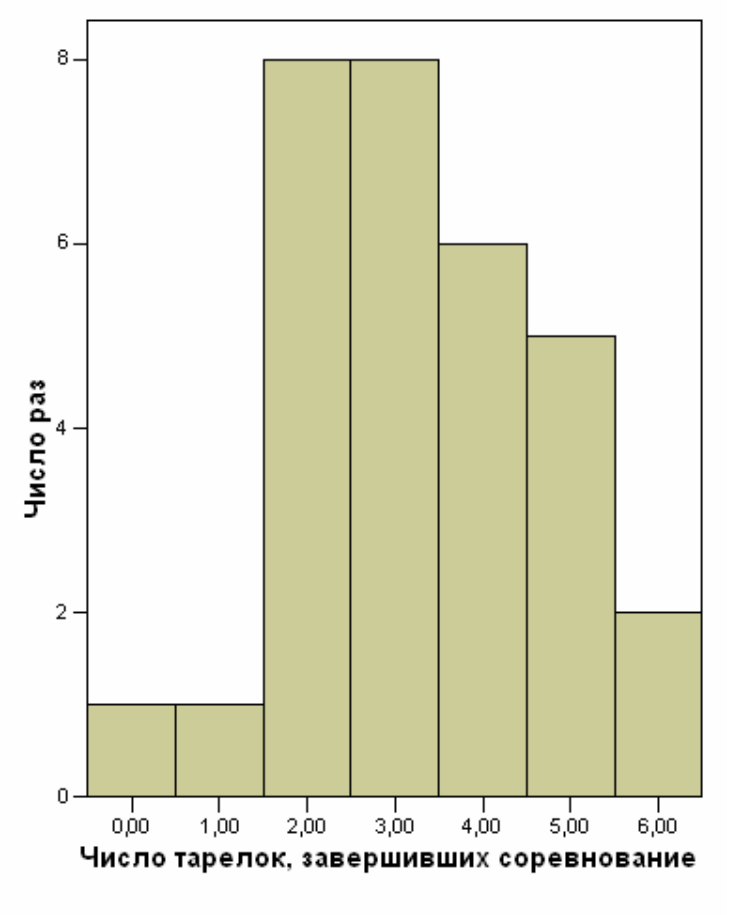

Рис. 21. Распределение числа успешно завершивших соревнование тарелок для команды, тарелки которой управляются системами, построенными вручную (размер команды – шесть тарелок)

Сводка результатов соревнований при размере команды в **восемь** тарелок этой системы с системой из работы [17] приведена в табл. 9.

Таблица 9. Результаты соревнований команд из восьми тарелок

	Построенная с помощью генетического программирования система	**Система из работы [17]**
Среднее	216,5523	212,5919
Минимум	203,05	203,44
Максимум	241,11	225,09

Более детальную информацию о результатах соревнований можно получить из гистограмм распределений результатов команд и гистограмм распределений числа летающих тарелок, успешно завершивших соревнование.

На рис. 22 показано распределение результатов команды, летающие тарелки которой управляются системами управления, построенными с помощью генетического программирования.

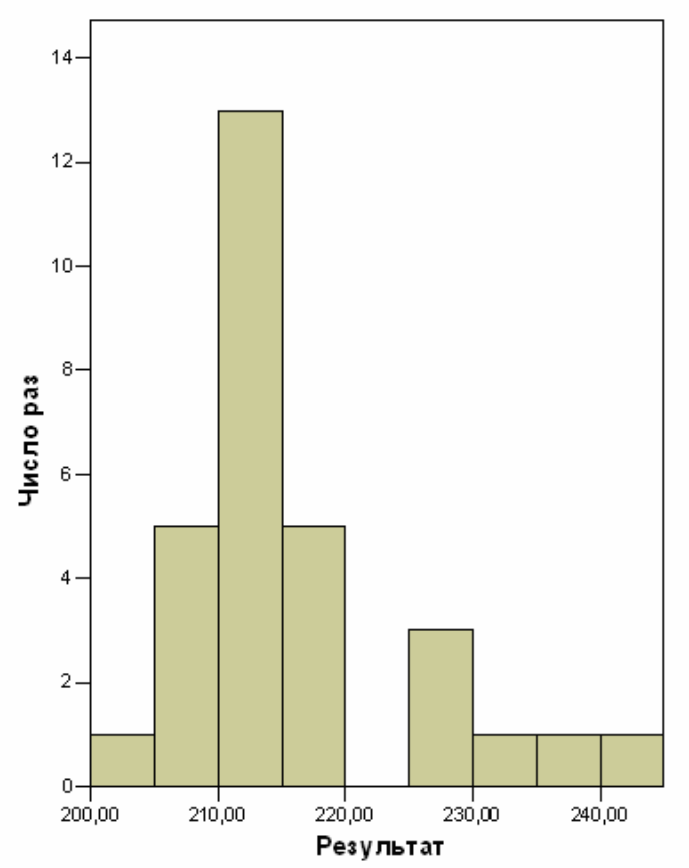

Рис. 22. Распределение результатов команды, летающие тарелки которой управляются системами управления, построенными с помощью генетического программирования (размер команды – восемь тарелок)

На рис. 23 показано распределение результатов команды, летающие тарелки которой управляются системами управления, построенной вручную.

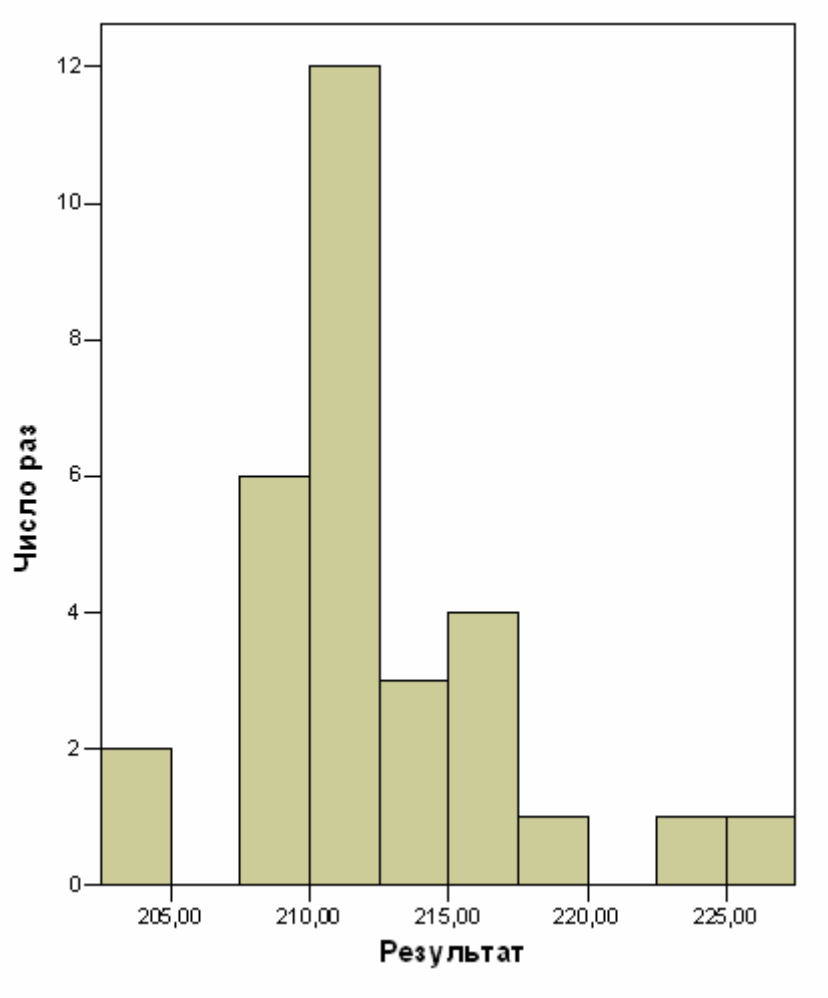

Рис. 23. Распределение результатов команды, летающие тарелки которой управляются системами управления, построенными вручную (размер команды – восемь тарелок)

На рис. 24 показано распределение числа успешно завершивших соревнование тарелок для команды, летающие тарелки которой управляются системами управления, построенными с помощью генетического программирования.

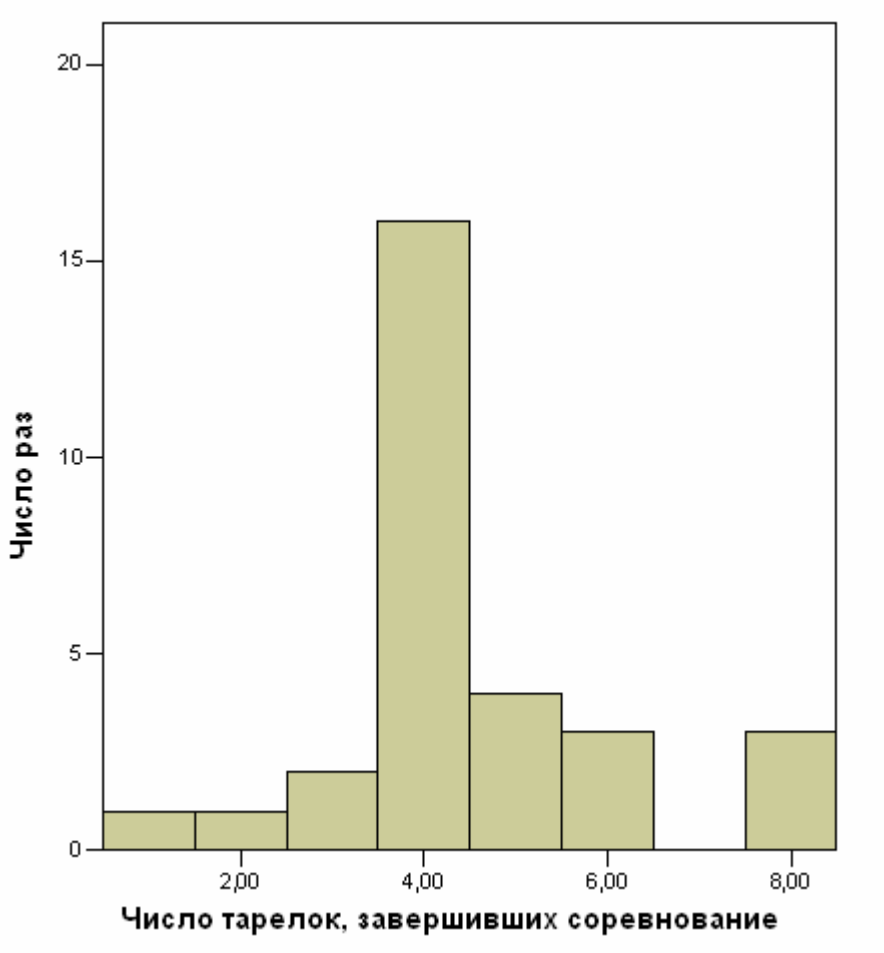

Рис. 24. Распределение числа летающих тарелок, успешно завершивших соревнование, для команды, тарелки которой управляются системами управления, построенными с помощью генетического программирования (размер команды – восемь тарелок)

На рис. 25 показано распределение числа успешно завершивших соревнование тарелок для команды, летающие тарелки которой управляются системами управления, построенными вручную.

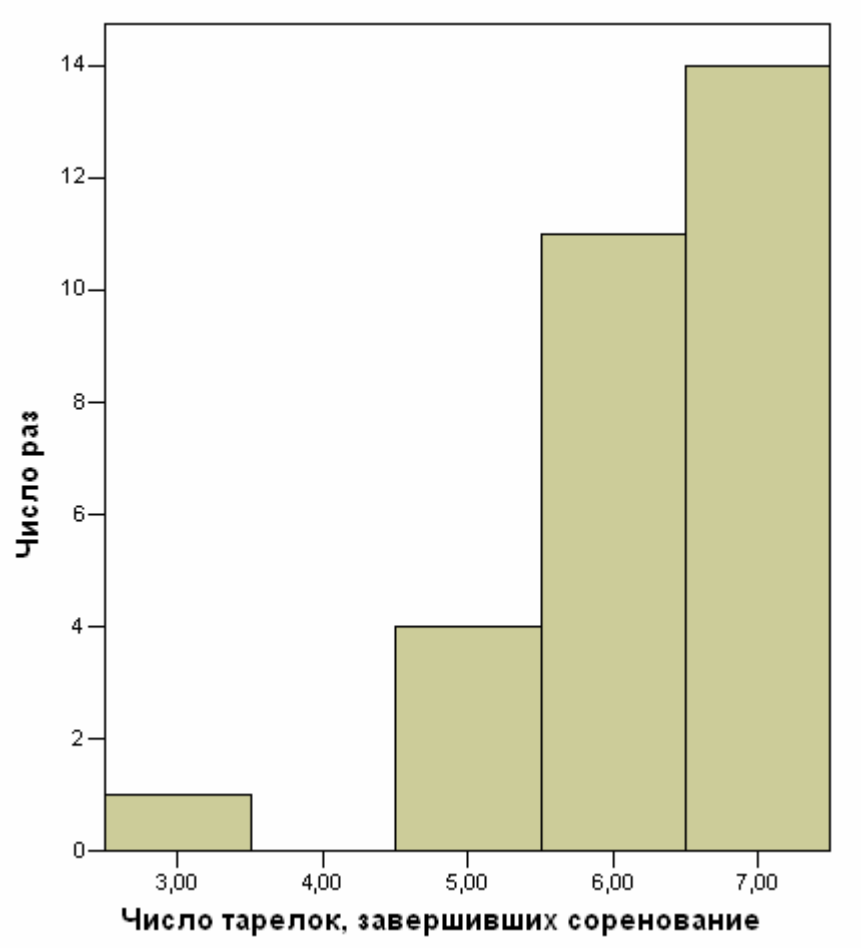

Рис. 25. Распределение числа летающих тарелок, успешно завершивших соревнование, для команды, тарелки которой управляются системами управления, построенными вручную (размер команды – восемь тарелок)

Анализ приведенных результатов позволяет сделать вывод, что построенная с помощью генетического программирования система в среднем показывает лучший результат, чем система, построенная в работе [17] вручную. Наиболее вероятной причиной этого является высокая эффективность системы, построенной с помощью генетического программирования.

Выигрывая по результату команды, построенная с помощью генетического программирования система, однако, проигрывает по числу тарелок, успешно завершивших гонку. При этом во многих случаях это связано с тем, что тарелки команды, управляемой системой, построенной с помощью генетического программирования, сталкиваются друг с другом или выходят за границы коридора. Однако, в ряде случаев они демонстрируют достаточно «разумное» поведение, двигаясь прямо при отсутствии вблизи препятствий и уклоняясь от других тарелок и границ коридора в случае их наличия по близости.

Такое положение дел, скорее всего, связано с недостаточно эффективной функцией приспособленности, учитывающий только общий результат команды и не учитывающий поведение агентов во время соревнования. Поэтому одним из направлений улучшений приведенного в настоящей работе алгоритма является применение функций приспособленности, обеспечивающих большую эффективность решения задачи за счет учета большего числа параметров.

Выводы по главе 3

1. В целом, можно утверждать, что «ручная» реализация обеспечивает более предсказуемое поведение, а генетическое программирование – более эффективное.
2. Улучшить поведение летающих тарелок, управляемых с системами, построенными методом генетического программирования, можно, скорее всего, применив функцию приспособленности, учитывающую поведение агентов (летающих тарелок) в процессе соревнования.

ЗАКЛЮЧЕНИЕ

1. В работе предложен алгоритм генетического программирования, позволяющий построить автомат, решающий задачу об «Умном муравье» и содержащий наименьшее число состояний среди всех таких известных автоматов.
2. В работе предложен метод, позволяющий применить генетическое программирование для автоматизированного построения управляющих систем в достаточно сложных задачах (таких, как, например, задача «Летающие тарелки»). Этот метод основан на совместном применении искусственных нейронных сетей и конечных автоматов.
3. Выполнено сравнение построенных вручную и построенных с помощью генетического программирования управляющих систем для задачи «Летающие тарелки».
4. Сформулированы направления дальнейшего исследования приведенных в настоящей работе задач:
    - для задачи об «Умном муравье»:
        - построение автомата с минимальным числом состояний, решающего задачу об «Умном муравье»;
        - выяснение минимального числа ходов, необходимых автомату, решающему эту задачу и содержащему k состояний (для $k = 5 – 8$);
    - для задачи «Летающие тарелки»:
        - применение вместо искусственных нейронных сетей других классов автоматических

классификаторов (байесовские сети доверия, деревья принятия решений, support vector machines и т.д.);

- применение функций приспособленности, учитывающих не только общий результат, но и поведение агентов в течение соревнования;
- применение вероятностных автоматов вместо детерминированных конечных автоматов.

5. По теме работы сделан доклад на IV-ой международной научно-практической конференции «Интегрированные модели и мягкие вычисления в искусственном интеллекте» (28–30 мая 2007 года, Коломна, Россия). Кроме этого, поданы материалы на X международную конференцию по мягким вычислениям и измерениям (SCM-2007, 25–27 июня 2007 года, СПбГЭТУ (ЛЭТИ) им. В.И. Ульянова-Ленина, Санкт-Петербург, Россия) и восьмую международную научно-техническую конференцию «Искусственный интеллект» (ИИ-2007, 24–29 сентября 2007 года, пос. Дивноморский, Россия).

ИСТОЧНИКИ

1. *Рассел С., Норвиг П.* Искусственный интеллект. Современный подход. М.: Вильямс, 2006.

2. *Гладков Л. А., Курейчик В. В., Курейчик В. М.* Генетические алгоритмы. М.: Физматлит, 2006.

3. *Chambers L.* Practical Handbook of Genetic Algorithms. Complex Coding Systems. Volumes I. II, III. CRC Press, 1999.

4. *Mitchell M.* An Introduction to Genetic Algorithms. The MIT Press. MA, 1996.

5. *Норенков И. П., Арутюнян Н. М.* Метагенетический алгоритм оптимизации и структурного синтеза проектных решений //Информационные технологии. 2007. № 3.

6. *Vose M. D., Wright A. H.* Simple genetic algorithms with linear fitness //Evolutionary Computation. 1994. Vol. 2, number 4. http://citeseer.ist.psu.edu/vose94simple.html

7. *Vose M. D.* A Critical Examination Of The Schema Theorem. Technical Report UT-CS-93212. University of Tennessee Computer Science Department. Knoxville. TN. USA, 1993. http://citeseer.ist.psu.edu/129900.html

8. *Vose M. D., Wright A. H.* The Simple Genetic Algorithm and the Walsh Transform. Part I. Theory //Evolutionary Computation. 1998. Vol. 6, number 3. http://citeseer.ist.psu.edu/vose98simple.html

9. *Koza J. R.* Genetic programming: on the programming of computers by means of natural selection. MIT Press, 1992.

10. http://www.genetic-programming.com/

11. *Шалыто А. А.* Технология автоматного программирования / Труды первой Всероссийской научной конференции «Методы и средства

обработки информации». М.: МГУ. 2003, с.528–535. http://is.ifmo.ru/works/tech_aut_prog/

12. *Шалыто А. А.* Switch-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998, 628 с. http://is.ifmo.ru/books/switch/1

13. *Shalyto A., Naumov L.* Automata Theory for Multi-Agent Systems implementation. Proceedings of International Conference Integration of Knowledge Intensive Multi-Agent Systems: Modeling, Exploration and Engineering. KIMAS-03. Boston: IEEE Boston Section. 2003, pp. 65–70. http:/is.ifmo.ru/english/_aut_th.pdf

14. *Shalyto A., Tukkel N.* Switch-Technology – Automata Approach to "Reactive" Systems Software Development / Programming and Computer Software. 2001. 27(5), pp. 260–276.

15. *Shalyto A., Naumov L., Korneev G.* Methods of Object-Oriented Reactive Agents Implementation on the Basis of Finite Automata. Proceedings of International Conference Integration of Knowledge Intensive Multi-Agent Systems: Modeling, Exploration and Engineering. KIMAS-05. Boston: IEEE Boston Section. 2005, p.460–465. http://is.ifmo.ru/articles_en/_kimas05-1.pdf

16. *Паращенко Д. А., Царев Ф. Н., Шалыто А. А.* Применение автоматного программирования при моделировании одного класса мультиагентных систем / Материалы девятой международной конференции «Интеллектуальные системы и компьютерные науки». М.: МГУ. 2006. Т.2, с. 352–355.

17. *Паращенко Д. А., Царев Ф. Н., Шалыто А. А.* Технология моделирования одного класса мультиагентных систем на основе автоматного программирования на примере игры «Соревнование

летающих тарелок». Проектная документация. СПбГУ ИТМО. 2006. http://is.ifmo.ru/unimod-projects/plates/

18. http://unimod.sourceforge.net

19. *Гуров В. С., Мазин М. А., Нарвский А. С., Шалыто А. А.* UML. SWITCH-технология. Eclipse // Информационно-управляющие системы, 2004, № 6, с.12–17. http://is.ifmo.ru/works/uml-switch-eclipse/

20. *Fogel D. B.* Applying Fogel and Burgin's 'Competitive Goal-Seeking through Evolutionary Programming' to Coordination, Trust, and Bargaining Games. IEEE Press Piscataway. NJ, 2000. http://citeseer.ist.psu.edu/fogel00applying.html

21. *Mitchell M., Crutchfield J., Hraber P.* Evolving cellular automata to perform computations // Physica D. 1993. 75, pp.361–391.

22. *Бедный Ю. Д.* Применение генетических алгоритмов для построения клеточных автоматов. СПбГУ ИТМО. Бакалаврская работа. 2006. http://is.ifmo.ru/papers/genalgcelaut/

23. *Das R., Crutchfield J. P., Mitchell M., Hanson J. E.* Evolving Globally Synchronized Cellular Automata /In Proceedings of the Sixth International Conference on Genetic Algorithms. 1995. pp. 336–343. http://citeseer.ist.psu.edu/das95evolving.html

24. *Sipper M.* Evolution of Parallel Cellular Machines //Lecture Notes in Computer Science. 2001. Vol. 1194. www.cs.bgu.ac.il/~sipper/papabs/epcm.pdf

25. *Воронин О., Дьюдни А.* Дарвинизм в программировании // Мой компьютер. 2004. № 35. http://www.mycomp.kiev.ua/text/7458

26. *Лобанов П. Г., Шалыто А. А.* Использование генетических алгоритмов для автоматического построения конечных автоматов в задаче о «Флибах» / Материалы 1-й Российской мультиконференции по проблемам управления. Сборник докладов 4-й Всероссийской научной

конференции «Управление и информационные технологии» (УИТ-2006). СПбГЭТУ "ЛЭТИ". 2006, с.144–149. http://is.ifmo.ru/works/flib/

27. *Jefferson D., Collins R., Cooper C., Dyer M., Flowers M., Korf R., Taylor C.*, Wang A. The Genesys System. 1992. www.cs.ucla.edu/~dyer/Papers/AlifeTracker/Alife91Jefferson.html

28. *Angeline P. J., Pollack J.* Evolutionary Module Acquisition // Proceedings of the Second Annual Conference on Evolutionary Programming, 1993. http://www.demo.cs.brandeis.edu/papers/ep93.pdf

29. *Хопкрофт Д., Мотвани Р., Ульман Д.* Введение в теорию автоматов, языков и вычислений. М.: Вильямс, 2002.

30. Заочный тур всесибирской олимпиады 2005 по информатике. http://olimpic.nsu.ru/widesiberia/archive/wso6/2005/rus/1tour/problem/problem.html

31. *McCulloch W. S., Pitts W.* A logical calculus of the ideas immanent in nervous activity // Bulletin of Mathematical Biophysics. 1943. 5, pp. 115–137.

ПРИЛОЖЕНИЕ 1. РЕАЛИЗАЦИЯ АЛГОРИТМА ДЛЯ ЗАДАЧИ ОБ «УМНОМ МУРАВЬЕ» НА ЯЗЫКЕ *JAVA*

1. Файл `Starter.java`. Содержит класс, запускающий алгоритм генетического программирования

```
01 package ru.ifmo.ctddev.genetic;
02
03 import java.io.File;
04 import java.io.FileNotFoundException;
05 import java.io.PrintWriter;
06
07 import ru.ifmo.ctddev.genetic.algorithm.GeneticAlgorithm;
08 import ru.ifmo.ctddev.genetic.algorithm.automaton.Automaton;
09
10 public class Starter {
11
12   public final static double DESIRED_FITNESS = 89;
13   public final static int GENERATION_SIZE = 2000;
14   public final static int STATE_NUMBER = 5;
15   public final static double PART_STAY = 0.5;
16   public final static int TIME_SMALL_MUTATION = 100;
17   public final static int TIME_BIG_MUTATION = 150;
18   public final static double MUTATION_PROBABILITY = 0.01;
19
20       public static void main(String[] args) throws
FileNotFoundException {
21       GeneticAlgorithm sga = new GeneticAlgorithm(DESIRED_FITNESS,
GENERATION_SIZE, STATE_NUMBER,
22           PART_STAY, MUTATION_PROBABILITY, TIME_SMALL_MUTATION,
TIME_BIG_MUTATION);
23
24     long startTime = System.currentTimeMillis();
25
26     Automaton best = sga.go();
27     {
28       PrintWriter out = new PrintWriter(new File("best"));
29       Automaton.printAutomaton(out, best);
30       out.close();
31     }
```

```
32
33      long finishTime = System.currentTimeMillis();
34
35      {
36        PrintWriter out = new PrintWriter(new File("parameters"));
37        out.println("Desired fitness = " + DESIRED_FITNESS);
38        out.println("Generation size = " + GENERATION_SIZE);
39        out.println("State number = " + STATE_NUMBER);
40        out.println("Part stay = " + PART_STAY);
41     out.println("Time to small mutation = " + TIME_SMALL_MUTATION);
42        out.println("Time to big mutation = " + TIME_BIG_MUTATION);
43       out.println("Fitness calculated " + Automaton.cntFitnessRun +
" times.");
44        out.println("Calculation time = " + (finishTime - startTime)
/ 1000 + " seconds.");
45        out.close();
46      }
47 System.out.println("Fitness calculated " + Automaton.cntFitnessRun
+ " times.");
48    }
49 }
```

2. Файл `GeneticAlgorithm.java`. Содержит реализацию разработанного алгоритма генетического программирования

```
001 package ru.ifmo.ctddev.genetic.algorithm;
002
003 import java.io.File;
004 import java.io.FileNotFoundException;
005 import java.io.PrintWriter;
006 import java.util.ArrayList;
007 import java.util.Arrays;
008 import java.util.Comparator;
009 import java.util.Random;
010
011 import ru.ifmo.ctddev.genetic.algorithm.automaton.Automaton;
012
013 public class GeneticAlgorithm {
014   private final double desiredFitness;
015   private final int populationSize;
016   private final int maxStates;
```

```
017
018
019     private Automaton[] curGeneration;
020     private Random random = new Random();
021
022     /*
023      * Доля особей, переходящих в следующее поколение.
024      * Соответственно, остальные особи следующего поколения
025      * будут получены с помощью кроссовера из неперешедших
026      * в новое поколение.
027      */
028     private final double partStay;
029
030     /*
031      * Вероятность, с которой происходят мутации.
032      */
033     private final double mutationProb;
034
035     /*
036         * Число поколений, по прошествии которого при отсутствии
прогресса фитнес-функции
037      * происходит "малая" мутация поколения
038      */
039     private final int timeSmallMutation;
040
041     /*
042         * Число поколений, по прошествии которого при отсутствии
прогресса фитнес-функции
043      * происходит "большая" мутация поколения
044      */
045     private final int timeBigMutation;
046
047         public SimpleGeneticAlgorithm(double desiredFitness, int
populationSize, int maxStates, double partStay, double mutationProb, int
timeSmallMutation, int timeBigMutation) {
048       this.desiredFitness = desiredFitness;
049       this.populationSize = populationSize;
050       this.maxStates = maxStates;
051       this.partStay = partStay;
052       this.mutationProb = mutationProb;
053       this.timeSmallMutation = timeSmallMutation;
```

```
054     this.timeBigMutation = timeBigMutation;
055   }
056
057   ArrayList<Double> generations = new ArrayList<Double>();
058
059   /*
060    * Инициализации популяции случайными особями.
061    *
062    */
063   private void init() {
064     curGeneration = new Automaton[populationSize];
065     for (int i = 0; i < populationSize; i++) {
066       Automaton toAdd = Automaton.createRandom(maxStates);
067       curGeneration[i] = toAdd;
068     }
069   }
070
071   public Automaton go() throws FileNotFoundException {
072     init();
073     int genCount = 0;
074
075     double lastBest = -1;
076     int cntBestEqual = 0;
077
078     while (true) {
079
080       double maxFitness = Integer.MIN_VALUE;
081       double minFitness = Integer.MAX_VALUE;
082       double sumFitness = 0;
083       Automaton bestAutomaton = null;
084       for (int i = 0; i < populationSize; i++) {
085         sumFitness += curGeneration[i].getFitness();
086         if (maxFitness < curGeneration[i].getFitness()) {
087           maxFitness = curGeneration[i].getFitness();
088           bestAutomaton = curGeneration[i];
089         }
090                         minFitness       =       Math.min(minFitness,
curGeneration[i].getFitness());
091       }
092
093       generations.add(maxFitness);
```

```
094
095         System.out.println("Generation " + genCount + " Max fitness
= " + maxFitness + " Min fitness = " + minFitness + " Sum fitness = " +
sumFitness);
096         if (genCount % 50 == 0) {
097           try {
098                 PrintWriter out = new PrintWriter(new File("gen" +
genCount));
099             Automaton.printAutomaton(out, bestAutomaton);
100             out.close();
101           } catch (FileNotFoundException e) {
102             // TODO Auto-generated catch block
103             e.printStackTrace();
104           }
105         }
106         if (maxFitness >= desiredFitness) {
107           try {
108                       PrintWriter   out   =   new   PrintWriter(new
File("fitness_graph"));
109             for (int i = 0; i < genCount; i++) {
110               out.println(generations.get(i));
111             }
112             out.close();
113           } catch (FileNotFoundException e) {
114             // TODO Auto-generated catch block
115             e.printStackTrace();
116           }
117
118           return bestAutomaton;
119         }
120
121         if (maxFitness == lastBest) {
122           cntBestEqual++;
123         } else {
124           lastBest = maxFitness;
125           cntBestEqual = 0;
126         }
127
128         // Генерируем новое поколение
129         Automaton[] nextGeneration = new Automaton[populationSize];
130         int nextCnt = 0;
```

```
131
132         // Применяем метод элитизма
133         Arrays.sort(curGeneration, new Comparator<Automaton>() {
134           public int compare(Automaton a1, Automaton a2) {
135             try {
136            return Double.compare(a2.getFitness(), a1.getFitness());
137             } catch (FileNotFoundException e) {
138               // TODO Auto-generated catch block
139               e.printStackTrace();
140             }
141             return 0;
142           }}
143         );
144
145         int cntStay = (int) (partStay * populationSize);
146         if ((populationSize - cntStay) % 2 == 1) {
147           cntStay++;
148         }
149         for (int i = 0; i < cntStay; i++) {
150           nextGeneration[nextCnt++] = curGeneration[i];
151         }
152
153         Automaton[] best = new Automaton[cntStay];
154         for (int i = 0; i < cntStay; i++) {
155           best[i] = curGeneration[i];
156         }
157
158         boolean[] can = new boolean[populationSize];
159         Arrays.fill(can, true);
160         // Теперь делаем кроссоверы
161         int cntAdd = populationSize - cntStay;
162         for (int i = 0; i < cntAdd / 2; i++) {
163           int num1 = random.nextInt(populationSize);
164           int num2 = num1;
165           while (num1 == num2) {
166             num2 = random.nextInt(populationSize);
167           }
168           if (random.nextDouble() > 0.5) {
169         Automaton[] toAdd = Automaton.crossOver(curGeneration[num1],
curGeneration[num2]);
170             for (int j = 0; j < 2; j++) {
```

```
171             nextGeneration[nextCnt++] = toAdd[j];
172           }
173         } else {
174         nextGeneration[nextCnt++] = curGeneration[num1].mutate();
175         nextGeneration[nextCnt++] = curGeneration[num2].mutate();
176         }
177       }
178
179       if (cntBestEqual >= timeSmallMutation) {
180         // "Малая" мутация поколения
181        for (int i = populationSize / 10; i < populationSize; i++)
{
182           nextGeneration[i] = nextGeneration[i].mutate();
183         }
184       }
185
186       if (cntBestEqual >= timeBigMutation) {
187         // "Большая" мутация поколения
188         int start = 0;
189         for (int i = start; i < populationSize; i++) {
190           if (random.nextBoolean()) {
191             nextGeneration[i] = nextGeneration[i].mutate();
192           } else {
193            nextGeneration[i] = Automaton.createRandom(maxStates);
194           }
195         }
196       }
197
198       // Новое поколение готово
199       curGeneration = nextGeneration;
200       genCount++;
201     }
202   }
203
204 }
```

3. Файл `Automaton.java`. Реализация класса автомата, содержащего операции создания случайного автомата, мутации автомата и скрещивания двух автоматов

```
001 package ru.ifmo.ctddev.genetic.algorithm.automaton;
002
003 import java.io.FileNotFoundException;
004 import java.io.PrintWriter;
005 import java.util.ArrayList;
006 import java.util.Random;
007
008 import ru.ifmo.ctddev.genetic.fitness.FitnessCalculator;
009                                                                    import
ru.ifmo.ctddev.genetic.fitness.JohnMuirTrailWithTimeBonusFitnessCalculator;
010
011 public class Automaton {
012   private static Random random = new Random();
013
014     private static FitnessCalculator fitnessCalculator = new
JohnMuirTrailWithTimeBonusFitnessCalculator();
015
016   public static int cntFitnessRun = 0;
017
018   private double fitness = -1;
019   private boolean fitnessCalculated = false;
020
021   public Transition[][] transitions;
022   boolean[][] marked;
023   public int initialState;
024   public int stateCount;
025
026    public Automaton(Transition[][] transitions, int initialState)
{
027     this.transitions = transitions;
028     this.initialState = initialState;
029     this.stateCount = transitions.length;
030     this.marked = new boolean[stateCount][2];
031   }
032
033   public double getFitness() throws FileNotFoundException {
034     if (!fitnessCalculated) {
035       fitness = calcFitness();
036       fitnessCalculated = true;
037     }
038     return fitness;
```

```
039   }
040
041   private double calcFitness() throws FileNotFoundException {
042     cntFitnessRun++;
043     return fitnessCalculator.calcFitness(this);
044   }
045
046   public static Automaton createRandom(int maxStates) {
047     int statesCnt = maxStates;
048     Transition[][] transitions = new Transition[statesCnt][2];
049     for (int i = 0; i < statesCnt; i++) {
050       for (int j = 0; j < 2; j++) {
051         int to = random.nextInt(statesCnt);
052         InnerAction action = InnerAction.randomAction();
053         boolean foodPresent = j == 1;
054           transitions[i][j] = new Transition(i, to, foodPresent,
action);
055       }
056     }
057     int initialState = random.nextInt(statesCnt);
058     return new Automaton(transitions, initialState);
059   }
060
061   /*
062    * Виды мутаций:
063    * 0) изменение начального состояния
064    * 1) добавление нового состояния
065    * 2) изменение условия на переходе
066    * 3) изменение действия на переходе
067    * 4) изменение состояния, в которое ведет переход
068    */
069   public Automaton mutate() {
070     int type = random.nextInt(5);
071
072     Transition[][] newTransitions;
073     int newInitState;
074
075     newTransitions = new Transition[stateCount][2];
076     for (int i = 0; i < stateCount; i++) {
077       for (int j = 0; j < 2; j++) {
078         newTransitions[i][j] = this.transitions[i][j].copy();
```

```
079       }
080     }
081     newInitState = this.initialState;
082 
083     switch (type) {
084       case 0: {
085         // Изменение начального состояния
086         newInitState = random.nextInt(stateCount);
087         break;
088       }
089       case 2 : {
090         // Изменение условия на переходе
091         int i = random.nextInt(stateCount);
092         Transition tmp = newTransitions[i][0];
093         newTransitions[i][0] = newTransitions[i][1];
094         newTransitions[i][1] = tmp;
095 
096    newTransitions[i][0] = newTransitions[i][0].invertCondition();
097    newTransitions[i][1] = newTransitions[i][1].invertCondition();
098         break;
099       }
100       case 3 : {
101         // Изменение действия на переходе
102         int i = random.nextInt(stateCount);
103         int j = random.nextInt(2);
104       newTransitions[i][j] = newTransitions[i][j].mutateAction();
105         break;
106       }
107       case 4 : {
108         // Изменение состояния, в которое ведет переход
109         int i = random.nextInt(stateCount);
110         int j = random.nextInt(2);
111 newTransitions[i][j] = newTransitions[i][j].mutateTo(stateCount);
112         break;
113       }
114     }
115     return new Automaton(newTransitions, newInitState);
116   }
117 
118   public static Automaton[] crossOver(Automaton a1, Automaton a2)
{
```

```
119     if (random.nextDouble() < 0.3) {
120       return newCrossOver(a1, a2);
121     }
122     int newInitState = a1.initialState;
123
124     int stateCnt1 = a1.stateCount;
125     int stateCnt2 = a2.stateCount;
126
127   ArrayList<Transition[]> list1 = new ArrayList<Transition[]>();
128   ArrayList<Transition[]> list2 = new ArrayList<Transition[]>();
129     for (int i = 0; i < Math.min(stateCnt1, stateCnt2); i++) {
130       if (random.nextBoolean()) {
131         int where = random.nextInt(2);
132         Transition[] toAdd1 = new Transition[2];
133         Transition[] toAdd2 = new Transition[2];
134         if (where == 0) {
135           toAdd1[0] = a1.transitions[i][0].copy();
136           toAdd1[1] = a2.transitions[i][1].copy();
137           toAdd2[0] = a1.transitions[i][0].copy();
138           toAdd2[1] = a2.transitions[i][1].copy();
139         } else {
140           toAdd1[0] = a2.transitions[i][0].copy();
141           toAdd1[1] = a1.transitions[i][1].copy();
142           toAdd2[0] = a2.transitions[i][0].copy();
143           toAdd2[1] = a1.transitions[i][1].copy();
144         }
145         list1.add(toAdd1);
146         list2.add(toAdd2);
147       } else {
148         int where = random.nextInt(2);
149         Transition[] toAdd1 = new Transition[2];
150         Transition[] toAdd2 = new Transition[2];
151         if (where == 0) {
152           toAdd1[0] = a1.transitions[i][0].copy();
153           toAdd1[1] = a1.transitions[i][1].copy();
154           toAdd2[0] = a2.transitions[i][0].copy();
155           toAdd2[1] = a2.transitions[i][1].copy();
156         } else {
157           toAdd1[0] = a2.transitions[i][0].copy();
158           toAdd1[1] = a2.transitions[i][1].copy();
159           toAdd2[0] = a1.transitions[i][0].copy();
```

```
160             toAdd2[1] = a1.transitions[i][1].copy();
161           }
162           list1.add(toAdd1);
163           list2.add(toAdd2);
164         }
165       }
166
167       Automaton[] res = new Automaton[2];
168                   res[0]    =    new    Automaton(list1.toArray(new
Transition[list1.size()][]), a1.initialState);
169                   res[1]    =    new    Automaton(list2.toArray(new
Transition[list2.size()][]), a2.initialState);
170       return res;
171     }
172
173     /**
174      * Кроссовер по следующей схеме: в один автомат идут переходы,
используемые
175      * a1 за первые 50 шагов, во второй --- используемые a2
176      * @param a1
177      * @param a2
178      * @return
179      */
180     public static Automaton[] newCrossOver(Automaton a1, Automaton
a2) {
181       int stateCnt1 = a1.stateCount;
182       int stateCnt2 = a2.stateCount;
183
184     ArrayList<Transition[]> list1 = new ArrayList<Transition[]>();
185     ArrayList<Transition[]> list2 = new ArrayList<Transition[]>();
186       for (int i = 0; i < Math.min(stateCnt1, stateCnt2); i++) {
187         Transition[] toAdd1 = new Transition[2];
188         Transition[] toAdd2 = new Transition[2];
189         for (int j = 0; j < 2; j++) {
190           if (a1.marked[i][j]) {
191             toAdd1[j] = a1.transitions[i][j].copy();
192             toAdd2[j] = a2.transitions[i][j].copy();
193           } else {
194             if (random.nextBoolean()) {
195               toAdd1[j] = a2.transitions[i][j].copy();
196               toAdd2[j] = a1.transitions[i][j].copy();
```

```
197           } else {
198             toAdd1[j] = a1.transitions[i][j].copy();
199             toAdd2[j] = a2.transitions[i][j].copy();
200           }
201         }
202       }
203       list1.add(toAdd1);
204       list2.add(toAdd2);
205     }
206     Automaton[] res = new Automaton[2];
207                   res[0]    =    new    Automaton(list1.toArray(new
Transition[list1.size()][]), a1.initialState);
208                   res[1]    =    new    Automaton(list2.toArray(new
Transition[list2.size()][]), a2.initialState);
209     return res;
210   }
211
212   public void markTransition(int from, int food) {
213     marked[from][food] = true;
214   }
215
216   public boolean selfCheck() {
217     if (initialState < 0 || initialState >= stateCount) {
218       System.out.println("Bad initial state");
219       return false;
220     }
221     for (int i = 0; i < stateCount; i++) {
222       for (int j = 0; j < 2; j++) {
223         Transition t = transitions[i][j];
224         if (t.getTo() < 0 || t.getTo() >= stateCount) {
225           return false;
226         }
227       }
228     }
229     return true;
230   }
231
232    public static void printAutomaton(PrintWriter out, Automaton
auto) throws FileNotFoundException {
233     out.println("Automaton");
234     out.println("Fitness = " + auto.getFitness());
```

```
235      out.println("State number = " + auto.stateCount);
236      out.println("Initial state = " + auto.initialState);
237      for (int i = 0; i < auto.stateCount; i++) {
238        for (int j = 0; j < 2; j++) {
239          out.println(auto.transitions[i][j]);
240        }
241      }
242    }
243
244 }
```

4. **Файл `InnerAction.java`. Содержит реализацию класса, соответствующего действию на переходе автомата**

```
01 package ru.ifmo.ctddev.genetic.algorithm.automaton;
02
03 import java.util.Random;
04
05 public enum InnerAction {
06   TURN_LEFT,
07   TURN_RIGHT,
08   MOVE_FORWARD,
09   STAND;
10
11   private static Random random = new Random();
12
13   public static InnerAction randomAction() {
14     int actionType = random.nextInt(4);
15     if (actionType == 0) {
16       return InnerAction.MOVE_FORWARD;
17     } else if (actionType == 1) {
18       return randomAction();
19 //       action = InnerAction.STAND;
20     } else if (actionType == 2) {
21       return InnerAction.TURN_LEFT;
22     } else if (actionType == 3) {
23       return InnerAction.TURN_RIGHT;
24     }
25
26 //     int actionType = random.nextInt(2);
27 //     if (actionType == 0) {
```

```
28 //       return InnerAction.MOVE_FORWARD;
29 //     } else if (actionType == 1) {
30 //       return InnerAction.TURN_RIGHT;
31 //     }
32     return null;
33   }
34
35   public String toString() {
36     if (this == InnerAction.TURN_LEFT) {
37       return "TURN_LEFT";
38     }
39     if (this == InnerAction.TURN_RIGHT) {
40       return "TURN_RIGHT";
41     }
42     if (this == InnerAction.MOVE_FORWARD) {
43       return "MOVE_FORWARD";
44     }
45     if (this == InnerAction.STAND) {
46       return "NOP";
47     }
48     return "";
49   }
50
51 }
```

5. **Файл `Transition.java`. Содержит реализацию класса, соответствующего переходу автомат и включающего операции создания случайного перехода и мутации перехода**

```
01 package ru.ifmo.ctddev.genetic.algorithm.automaton;
02
03 import java.util.Random;
04
05 /**
06  * Переход в автомате. Хранит вершину, из которой выходит,
07  * вершину, в которую входит, условие и совершаемое действие.
08  * @author Fedor Tsarev
09  *
10  */
11 public class Transition {
12   private int from;
13   private int to;
14   private boolean foodPresent;
15   private InnerAction action;
16
17   private static Random random = new Random();
18
19     public Transition(int from, int to, boolean foodPresent,
InnerAction action) {
20     this.from = from;
21     this.to = to;
22     this.foodPresent = foodPresent;
23     this.action = action;
24   }
25
26   public Transition copy() {
27     return new Transition(from, to, foodPresent, action);
28   }
29
30   public Transition mutateAction() {
31               return new Transition(from, to, foodPresent,
InnerAction.randomAction());
32   }
33
34   public Transition mutateTo(int stateCount) {
```

```
35          return new Transition(from, random.nextInt(stateCount),
foodPresent, action);
36    }
37
38    public int getTo() {
39      return to;
40    }
41
42    public Transition changeTo(int newTo) {
43      return new Transition(from, newTo, foodPresent, action);
44    }
45
46    public InnerAction getInnerAction() {
47      return action;
48    }
49
50    @Override
51    public String toString() {
52      String food = "food ";
53      if (foodPresent) {
54        food += "PRESENT";
55      } else {
56        food += "NOT PRESENT";
57      }
58       return from + " -> " + to + " when " + food + ", Action: " +
action;
59    }
60
61    public Transition invertCondition() {
62      return new Transition(from, to, !foodPresent, action);
63    }
64
65 }
```

6. Файл `FitnessCalculator.java`. Содержит интерфейс функции приспособленности

```
1 package ru.ifmo.ctddev.genetic.fitness;
2
3 import ru.ifmo.ctddev.genetic.algorithm.automaton.Automaton;
4
```

```
5 public interface FitnessCalculator {
6   public double calcFitness(Automaton a);
7 }
```

7. **Файл `JohnMuirTrailWithTimeBonusFitnessCalculator.java`. Содержит реализацию функции приспособленности, учитывающую не только количество съеденной еды, но и время, за которое она была съедена**

```
001 package ru.ifmo.ctddev.genetic.fitness;
002
003 import java.io.File;
004 import java.io.FileNotFoundException;
005 import java.util.Scanner;
006
007 import ru.ifmo.ctddev.genetic.algorithm.automaton.Automaton;
008 import ru.ifmo.ctddev.genetic.algorithm.automaton.InnerAction;
009
010   public class JohnMuirTrailWithTimeBonusFitnessCalculator
implements FitnessCalculator {
011
012   int n;
013   int m;
014   int[][] field;
015
016     int[] dr = new int[] {0, 1,  0, -1};
017   int[] dc = new int[] {1, 0, -1,  0};
018
019   public JohnMuirTrailWithTimeBonusFitnessCalculator(){
020     readField();
021   }
022
023
024   private void readField() {
025     Scanner in = null;
026     try {
027       in = new Scanner(new File("john-muir.txt"));
028     } catch (FileNotFoundException e) {
029       // TODO Auto-generated catch block
030       e.printStackTrace();
```

```
031     }
032     n = 32;
033     m = 32;
034     field = new int[n][m];
035     for (int i = 0; i < n; i++) {
036       String s = in.next();
037       for (int j = 0; j < m; j++) {
038         char ch = s.charAt(j);
039         if (ch == '*') {
040           field[i][j] = 1;
041         }
042       }
043     }
044   }
045
046
047   public double calcFitness(Automaton a) {
048     int cr = 0;
049     int cc = 0;
050     int dir = 0;
051
052     int state = a.initialState;
053
054     int fitness = 0;
055     int[][] curField = new int[n][m];
056     for (int i = 0; i < n; i++) {
057       for (int j = 0; j < m; j++) {
058         curField[i][j] = field[i][j];
059       }
060     }
061
062     int last = 199;
063
064     for (int i = 0; i < 200; i++) {
065       if (curField[cr][cc] == 1) {
066         last = i;
067       }
068       curField[cr][cc] = 0;
069       int food = curField[(cr + dr[dir] + n) % n][(cc + dc[dir] +
m) % m];
070
```

```
071       // Сохраняем информацию для проведения скрещивания
072       if (i < 40) {
073         a.markTransition(state, food);
074       }
075
076 InnerAction action = a.transitions[state][food].getInnerAction();
077       int newState = a.transitions[state][food].getTo();
078       if (action == InnerAction.TURN_RIGHT) {
079         dir++;
080         dir += 4;
081         dir %= 4;
082       }
083       if (action == InnerAction.TURN_LEFT) {
084         dir--;
085         dir += 4;
086         dir %= 4;
087       }
088       if (action == InnerAction.MOVE_FORWARD) {
089         fitness += food;
090         cr += dr[dir];
091         cr += n;
092         cr %= n;
093
094         cc += dc[dir];
095         cc += m;
096         cc %= m;
097       }
098
099       state = newState;
100     }
101     return fitness + (199.0 - last) / 200.0;
102   }
103 }
```

ПРИЛОЖЕНИЕ 2. РЕАЛИЗАЦИЯ АЛГОРИТМА ДЛЯ ЗАДАЧИ «ЛЕТАЮЩИЕ ТАРЕЛКИ» НА ЯЗЫКЕ *JAVA*

1. Файл `Starter.java`. Содержит класс, запускающий алгоритм генетического программирования

```
01 package ru.ifmo.tsarev.bachelor;
02
03 import java.io.File;
04 import java.io.FileNotFoundException;
05 import java.io.PrintWriter;
06
07 import ru.ifmo.tsarev.bachelor.genetic.Individual;
08 import ru.ifmo.tsarev.bachelor.genetic.SimpleGeneticAlgorithm;
09
10 public class Starter {
11   public final static double DESIRED_FITNESS = 5 * 215;
12   public final static int GENERATION_SIZE = 30000;
13   public final static int STATE_NUMBER = 10;
14   public final static double PART_STAY = 0.3;
15   public final static int TIME_SMALL_MUTATION = 1000;
16   public final static int TIME_BIG_MUTATION = 1500;
17   public final static double MUTATION_PROBABILITY = 0.01;
18
19           public   static   void   main(String[]   args)   throws
FileNotFoundException {
20                         SimpleGeneticAlgorithm     sga     =     new
SimpleGeneticAlgorithm(DESIRED_FITNESS, GENERATION_SIZE, STATE_NUMBER,
21                                     PART_STAY,   MUTATION_PROBABILITY,
TIME_SMALL_MUTATION, TIME_BIG_MUTATION);
22
23     long startTime = System.currentTimeMillis();
24
25     Individual best = sga.go();
26     {
27       PrintWriter out = new PrintWriter(new File("best"));
28       Individual.printIndividual(out, best);
29       out.close();
30     }
31
```

```
32      long finishTime = System.currentTimeMillis();
33
34      {
35        PrintWriter out = new PrintWriter(new File("parameters"));
36        out.println("Desired fitness = " + DESIRED_FITNESS);
37        out.println("Generation size = " + GENERATION_SIZE);
38        out.println("State number = " + STATE_NUMBER);
39        out.println("Part stay = " + PART_STAY);
40     out.println("Time to small mutation = " + TIME_SMALL_MUTATION);
41        out.println("Time to big mutation = " + TIME_BIG_MUTATION);
42        out.println("Fitness calculated " + Individual.cntFitnessRun
+ " times.");
43         out.println("Calculation time = " + (finishTime - startTime)
/ 1000 + " seconds.");
44        out.close();
45      }
46                    System.out.println("Fitness   calculated   "   +
Individual.cntFitnessRun + " times.");
47 //     Automaton.printAutomaton(new PrintWriter(System.out), best);
48    }
49 }
```

2. Файл `SimpleGeneticAlgorithm.java`. Содержит класс, реализующий алгоритм генетического программирования

```
001 package ru.ifmo.tsarev.bachelor.genetic;
002
003 import java.io.File;
004 import java.io.FileNotFoundException;
005 import java.io.PrintWriter;
006 import java.util.ArrayList;
007 import java.util.Arrays;
008 import java.util.Comparator;
009 import java.util.Random;
010
011 public class SimpleGeneticAlgorithm {
012   private final double desiredFitness;
013   private final int populationSize;
014   private final int maxStates;
015
016
```

```
017    private Individual[] curGeneration;
018    private Random random = new Random();
019
020    /*
021     * Доля особей, переходящих в следующее поколение.
022     * Соответственно, остальные особи следующего поколения
023     * будут получены с помощью кроссовера из неперешедших
024     * в новое поколение.
025     */
026    private final double partStay;
027
028    /*
029     * Вероятность, с которой происходят мутации.
030     */
031    private final double mutationProb;
032
033    /*
034        * Число поколений, по прошествии которого при отсутствии
прогресса фитнес-функции
035     * происходит "малая" мутация поколения
036     */
037    private final int timeSmallMutation;
038
039    /*
040        * Число поколений, по прошествии которого при отсутствии
прогресса фитнес-функции
041     * происходит "большая" мутация поколения
042     */
043    private final int timeBigMutation;
044
045        public SimpleGeneticAlgorithm(double desiredFitness, int
populationSize, int maxStates, double partStay, double mutationProb, int
timeSmallMutation, int timeBigMutation) {
046      this.desiredFitness = desiredFitness;
047      this.populationSize = populationSize;
048      this.maxStates = maxStates;
049      this.partStay = partStay;
050      this.mutationProb = mutationProb;
051      this.timeSmallMutation = timeSmallMutation;
052      this.timeBigMutation = timeBigMutation;
053    }
```

```
054
055   ArrayList<Double> generations = new ArrayList<Double>();
056
057   /**
058    * Инициализации популяции случайными особями.
059    *
060    */
061   private void init() {
062     curGeneration = new Individual[populationSize];
063     for (int i = 0; i < populationSize; i++) {
064         Individual toAdd = Individual.createRandom(2, maxStates,
0);
065       curGeneration[i] = toAdd;
066     }
067   }
068
069
070   private void botva() {
071     double max = -1;
072     int cnt = 0;
073     Individual res = null;
074     for (Individual i : curGeneration) {
075       double cur = i.getFitness();
076       if (cur > max) {
077         max = cur;
078         res = i;
079       }
080 //      System.out.println(cur);
081     }
082     System.out.println(max);
083   }
084
085   public Individual go() throws FileNotFoundException {
086     init();
087     int genCount = 0;
088
089     double lastBest = -1;
090     int cntBestEqual = 0;
091
092     while (true) {
093
```

```
094       double maxFitness = Integer.MIN_VALUE;
095       double minFitness = Integer.MAX_VALUE;
096       double sumFitness = 0;
097       Individual bestIndividual = null;
098       for (int i = 0; i < populationSize; i++) {
099 //        if (!curGeneration[i].selfCheck()) {
100 //          System.out.println("Bad " + i);
101 //        }
102         sumFitness += curGeneration[i].getFitness();
103         if (maxFitness < curGeneration[i].getFitness()) {
104           maxFitness = curGeneration[i].getFitness();
105           bestIndividual = curGeneration[i];
106         }
107  inFitness = Math.min(minFitness, curGeneration[i].getFitness());
108       }
109
110       generations.add(maxFitness);
111
112       System.out.println("Generation " + genCount + " Max fitness
= " + maxFitness + " Min fitness = " + minFitness + " Sum fitness = " +
sumFitness);
113       if (genCount % 1 == 0) {
114         try {
115    PrintWriter out = new PrintWriter(new File("gen" + genCount));
116           Individual.printIndividual(out, bestIndividual);
117           out.close();
118         } catch (FileNotFoundException e) {
119           // TODO Auto-generated catch block
120           e.printStackTrace();
121         }
122       }
123       if (maxFitness >= desiredFitness) {
124         try {
125     PrintWriter out = new PrintWriter(new File("fitness_graph"));
126           for (int i = 0; i < genCount; i++) {
127             out.println(generations.get(i));
128           }
129           out.close();
130         } catch (FileNotFoundException e) {
131           // TODO Auto-generated catch block
132           e.printStackTrace();
```

```
133         }
134
135         return bestIndividual;
136       }
137
138       if (maxFitness == lastBest) {
139         cntBestEqual++;
140       } else {
141         lastBest = maxFitness;
142         cntBestEqual = 0;
143       }
144
145       // Генерируем новое поколение
146     Individual[] nextGeneration = new Individual[populationSize];
147       int nextCnt = 0;
148
149       // Применяем метод элитизма
150       Arrays.sort(curGeneration, new Comparator<Individual>() {
151         public int compare(Individual i1, Individual i2) {
152          return Double.compare(i2.getFitness(), i1.getFitness());
153         }}
154       );
155
156       int cntStay = (int) (partStay * populationSize);
157       if ((populationSize - cntStay) % 2 == 1) {
158         cntStay++;
159       }
160       for (int i = 0; i < cntStay; i++) {
161         nextGeneration[nextCnt++] = curGeneration[i];
162       }
163
164       Individual[] best = new Individual[cntStay];
165       for (int i = 0; i < cntStay; i++) {
166         best[i] = curGeneration[i];
167       }
168
169       boolean[] can = new boolean[populationSize];
170       Arrays.fill(can, true);
171       // Теперь делаем кроссоверы (при этом используем всех)
172       int cntAdd = populationSize - cntStay;
173       for (int i = 0; i < cntAdd / 2; i++) {
```

```
174         int num1 = random.nextInt(populationSize);
175         int num2 = num1;
176         while (num1 == num2) {
177           num2 = random.nextInt(populationSize);
178         }
179         if (random.nextDouble() > 0.5) {
180     Individual[] toAdd = Individual.crossOver(curGeneration[num1],
curGeneration[num2]);
181           for (int j = 0; j < 2; j++) {
182             nextGeneration[nextCnt++] = toAdd[j];
183           }
184         } else {
185         nextGeneration[nextCnt++] = curGeneration[num1].mutate();
186         nextGeneration[nextCnt++] = curGeneration[num2].mutate();
187         }
188       }
189
190       if (cntBestEqual >= timeSmallMutation) {
191         // "Малая" мутация поколения
192        for (int i = populationSize / 10; i < populationSize; i++)
{
193           nextGeneration[i] = nextGeneration[i].mutate();
194         }
195       }
196
197       if (cntBestEqual >= timeBigMutation) {
198         // "Большая" мутация поколения
199         int start = 0;
200         for (int i = start; i < populationSize; i++) {
201           if (random.nextBoolean()) {
202             nextGeneration[i] = nextGeneration[i].mutate();
203           } else {
204     nextGeneration[i] = Individual.createRandom(2, maxStates, 0);
205           }
206         }
207       }
208
209       // Новое поколение готово
210       curGeneration = nextGeneration;
211       genCount++;
212     }
```

```
213    }
214 }
```

3. Файл `Individual.java`. Содержит класс, реализующий особь в алгоритме генетического программирования

```
001 package ru.ifmo.tsarev.bachelor.genetic;
002
003 import java.io.PrintWriter;
004 import java.util.List;
005 import java.util.Random;
006
007 import ru.ifmo.tsarev.bachelor.environment.judging.Manager;
008import ru.ifmo.tsarev.bachelor.environment.logic.EnvironmentLogic;
009 import ru.ifmo.tsarev.bachelor.environment.logic.Plate;
010 import ru.ifmo.tsarev.bachelor.genetic.fitness.FitnessCalculator;
011                                                                    import
ru.ifmo.tsarev.bachelor.genetic.fitness.FixedManagersFitnessCalculator;
012
013 /**
014  * Особь для генетического алгоритма. Функция приспособленности
015  * считается соревнованиями с априори заданными стратегиями.
016  * @author Fedor Tsarev
017  *
018  */
019 public class Individual implements Manager {
020
021   public static int cntFitnessRun = 0;
022
023     protected final FitnessCalculator fitnessCalculator = new
FixedManagersFitnessCalculator(5);
024
025   protected PlateControlSystem[] pcs;
026
027   private boolean fitnessCalculated;
028   private double fitness;
029
030   private static final Random r = new Random();
031
032   public Individual(PlateControlSystem[] pcs) {
033     this.pcs = pcs;
```

```
034   }
035
036   public double getFitness() {
037     if (fitnessCalculated) {
038       return fitness;
039     }
040     cntFitnessRun++;
041     fitness = fitnessCalculator.calcFitness(this);
042     fitnessCalculated = true;
043     return fitness;
044   }
045
046     public static Individual createRandom(int platesCount, int
maxStates, int player) {
047                         PlateControlSystem[]     pcs     =     new
PlateControlSystem[platesCount];
048     EnvironmentLogic el = EnvironmentLogic.getInstance();
049
050     Plate[] ourPlates = new Plate[2];
051     Plate[] enemyPlates = new Plate[2];
052     List<Plate> list = el.getPlates();
053     {
054       int cntOur = 0;
055       int cntEnemy = 0;
056       for (Plate p : list) {
057         if (p.getPlayer() == player) {
058           ourPlates[cntOur++] = p;
059         } else {
060           enemyPlates[cntEnemy++] = p;
061         }
062       }
063     }
064     for (int i = 0; i < platesCount; i++) {
065               pcs[i]  =  PlateControlSystem.createRandom(maxStates,
ourPlates[i], ourPlates[1 - i], enemyPlates);
066     }
067     return new Individual(pcs);
068   }
069
070   // Manager methods
071   public void init(int player) {
```

```
072     // TODO Auto-generated method stub
073
074   }
075
076   public void makeTurn(EnvironmentLogic el) {
077     for (PlateControlSystem p : pcs) {
078       p.makeMove();
079     }
080   }
081
082   public static void printIndividual(PrintWriter out, Individual
individual) {
083     out.println("Fitness = " + individual.getFitness());
084     for (PlateControlSystem p : individual.pcs) {
085       PlateControlSystem.printPlateControlSystem(out, p);
086     }
087   }
088
089   public static Individual[] crossOver(Individual i1, Individual
i2) {
090                               PlateControlSystem[]      pcs0      =
PlateControlSystem.crossOver(i1.pcs[0], i2.pcs[0]);
091                               PlateControlSystem[]      pcs1      =
PlateControlSystem.crossOver(i1.pcs[1], i2.pcs[1]);
092               return   new   Individual[]   {new   Individual(new
PlateControlSystem[] {pcs0[0], pcs1[0]}),
093     new Individual(new PlateControlSystem[] {pcs0[1], pcs1[1]})};
094   }
095
096   public Individual mutate() {
097     PlateControlSystem[] newPCS = pcs.clone();
098     if (r.nextBoolean()) {
099       newPCS[0] = newPCS[0].mutate();
100     } else {
101       newPCS[1] = newPCS[1].mutate();
102     }
103     return new Individual(newPCS);
104   }
105
106 }
```

4. Файл `PlateControlSystem.java`. Содержит класс, реализующий систему управления летающей тарелкой

```
001 package ru.ifmo.tsarev.bachelor.genetic;
002
003 import java.io.PrintWriter;
004 import java.util.Random;
005 import java.util.Scanner;
006
007 import ru.ifmo.tsarev.bachelor.Parameters;
008 import ru.ifmo.tsarev.bachelor.automata.Automaton;
009 import ru.ifmo.tsarev.bachelor.automata.actions.Action;
010 import ru.ifmo.tsarev.bachelor.environment.Config;
011import ru.ifmo.tsarev.bachelor.environment.logic.EnvironmentLogic;
012 import ru.ifmo.tsarev.bachelor.environment.logic.Plate;
013 import ru.ifmo.tsarev.bachelor.neuralnets.NeuralNet;
014 import ru.ifmo.tsarev.bachelor.util.Vector;
015
016 /**
017  * Система управления летающей тарелкой. Является составной
018  * частью особи.
019  * @author Fedor Tsarev
020  *
021  */
022 public class PlateControlSystem {
023
024   private NeuralNet neuralNet;
025   private Automaton automaton;
026
027   private final Plate controlledPlate;
028   private final Plate allyPlate;
029   private final Plate[] enemyPlates;
030   private static final Random r = new Random();
031
032       public  PlateControlSystem(Automaton  automaton,  NeuralNet
neuralNet, Plate controlledPlate, Plate allyPlate, Plate[] enemyPlates) {
033     this.automaton = automaton;
034     this.neuralNet = neuralNet;
035     this.controlledPlate = controlledPlate;
036     this.allyPlate = allyPlate;
037     this.enemyPlates = enemyPlates;
```

```
038   }
039
040   //TODO: verify
041   public void makeMove() {
042     double[] inputValues = new double[8];
043     {
044                                   Vector          delta          =
allyPlate.getPosition().subtract(controlledPlate.getPosition());
045       inputValues[0] = delta.x;
046       inputValues[1] = delta.y;
047     }
048
049     {
050       for (int i = 0; i < 2; i++) {
051                                             Vector     delta     =
enemyPlates[i].getPosition().subtract(controlledPlate.getPosition());
052         inputValues[2 + 2 * i] = delta.x;
053         inputValues[2 + 2 * i + 1] = delta.y;
054       }
055     }
056
057     {
058       Vector v = controlledPlate.getSpeed();
059        inputValues[6] = Math.atan2(v.vectorProduct(Vector.X_ORTH),
v.scalarProduct(Vector.X_ORTH));
060     }
061
062     {
063       //TODO: inputValues[7] = time to hit the wall
064       Vector v = controlledPlate.getSpeed();
065       if (Math.abs(v.y) <= 1e-9) {
066         inputValues[7] = 1e20;
067       } else {
068         if (v.y > 0) {
069                           inputValues[7]   =   (Config.FIELD_HEIGHT   -
controlledPlate.getPosition().y) / v.y;
070         } else {
071     inputValues[7] = (0 - controlledPlate.getPosition().y) / v.y;
072         }
073       }
074     }
```

```
075
076
077     neuralNet.setInputValues(inputValues);
078
079     double[] outputValues = neuralNet.getOutput();
080     int[] inputForAutomaton = new int[outputValues.length];
081     for (int i = 0; i < outputValues.length; i++) {
082       inputForAutomaton[i] = (int) outputValues[i];
083     }
084
085   Action[] actions = automaton.makeTransition(inputForAutomaton);
086     for (Action a : actions) {
087       a.apply(controlledPlate);
088     }
089   }
090
091    public static PlateControlSystem createRandom(int maxStates,
Plate controlledPlate, Plate allyPlate, Plate[] enemyPlates) {
092     return new PlateControlSystem(
093         Automaton.createRandom(maxStates, Parameters.OUT_DEGREE),
094         NeuralNet.createRandom(),
095         controlledPlate, allyPlate, enemyPlates
096         );
097   }
098
099    public static void printPlateControlSystem(PrintWriter out,
PlateControlSystem pcs) {
100     NeuralNet.printNeuralNet(out, pcs.neuralNet);
101     Automaton.printAutomaton(out, pcs.automaton);
102   }
103
104   public static PlateControlSystem[] crossOver(PlateControlSystem
pcs1, PlateControlSystem pcs2) {
105           NeuralNet[]  nn  =  NeuralNet.crossOver(pcs1.neuralNet,
pcs2.neuralNet);
106           Automaton[]  a  =  Automaton.crossOver(pcs1.automaton,
pcs2.automaton);
107     return new PlateControlSystem[] {
108         new PlateControlSystem(a[0], nn[0], pcs1.controlledPlate,
pcs1.allyPlate, pcs1.enemyPlates),
109         new PlateControlSystem(a[1], nn[1], pcs2.controlledPlate,
```

```
pcs2.allyPlate, pcs2.enemyPlates)
      110     };
      111   }
      112
      113   public PlateControlSystem mutate() {
      114     NeuralNet nn = neuralNet;
      115     Automaton a = automaton;
      116     if (r.nextBoolean()) {
      117       nn = nn.mutate();
      118     } else {
      119       a = a.mutate();
      120     }
      121         return new PlateControlSystem(a, nn, controlledPlate,
allyPlate, enemyPlates);
      122   }
      123
      124 }
```

5. Файл `NeuralNet.java`. Содержит класс, реализующий нейронную сеть

```
      001 package ru.ifmo.tsarev.bachelor.neuralnets;
      002
      003 import java.io.PrintWriter;
      004 import java.util.Random;
      005
      006 public class NeuralNet {
      007   private static final Random r = new Random();
      008
      009   private Neuron[] input;
      010   private int inputCnt;
      011
      012   private Neuron[] hidden;
      013   private int hiddenCnt;
      014
      015   private Neuron[] output;
      016   private int outputCnt;
      017
      018     public NeuralNet(Neuron[] input, Neuron[] hidden, Neuron[]
output) {
      019     this.input = input;
```

```
020     this.inputCnt = input.length;
021
022     this.hidden = hidden;
023     this.hiddenCnt = hidden.length;
024
025     this.output = output;
026     this.outputCnt = output.length;
027
028     setTopology();
029   }
030
031   private void setTopology() {
032     hidden[0].setInputs(new Neuron[] {input[0], input[1]});
033     hidden[1].setInputs(new Neuron[] {input[2], input[3]});
034     hidden[2].setInputs(new Neuron[] {input[4], input[5]});
035     hidden[3].setInputs(new Neuron[] {input[6], input[7]});
036
037         output[0].setInputs(new  Neuron[]  {hidden[0],  hidden[1],
hidden[2]});
038     output[1].setInputs(new Neuron[] {hidden[1], hidden[2]});
039         output[2].setInputs(new  Neuron[]  {hidden[1],  hidden[2],
hidden[3]});
040   }
041
042   public void setInputValues(double[] inputValues) {
043     if (inputValues.length != inputCnt) {
044       throw new IllegalArgumentException("inputValues.length must
be equal to inputCnt");
045     }
046
047     for (int i = 0; i < inputCnt; i++) {
048       input[i].setValue(inputValues[i]);
049     }
050   }
051
052   public double[] getOutput() {
053     double[] res = new double[outputCnt];
054     for (int i = 0; i < outputCnt; i++) {
055       res[i] = output[i].getValue();
056     }
057     return res;
```

```
058   }
059
060   /*
061    * Входные данные для нейронной сети:
062    * 1) относительные координаты товарища по команде --- 2
063    * 2) относительные координаты соперников --- 4
064    * 3) угол относительно направления вперед --- 1
065    * 4) время до столкновения с ближайшей стеной - 1
066    * Итого: 8
067    */
068   public static NeuralNet createRandom() {
069     Neuron[] input = new Neuron[8];
070     for (int i = 0; i < 8; i++) {
071       input[i] = new SigmoidNeuron(new double[0], 0);
072     }
073
074     Neuron[] hidden = new Neuron[4];
075     hidden[0] = SigmoidNeuron.createRandom(2);
076     hidden[1] = SigmoidNeuron.createRandom(2);
077     hidden[2] = SigmoidNeuron.createRandom(2);
078     hidden[3] = SigmoidNeuron.createRandom(2);
079
080     Neuron[] output = new Neuron[3];
081     output[0] = NeuronWithLimit.createRandom(3);
082     output[1] = NeuronWithLimit.createRandom(2);
083     output[2] = NeuronWithLimit.createRandom(3);
084
085     return new NeuralNet(input, hidden, output);
086   }
087
088    public static void printNeuralNet(PrintWriter out, NeuralNet
neuralNet) {
089     out.println("Neural net");
090     out.println("Inputs");
091     for (Neuron n : neuralNet.input) {
092       n.printSelf(out);
093     }
094     out.println("Hidden");
095     for (Neuron n : neuralNet.hidden) {
096       n.printSelf(out);
097     }
```

```
098     out.println("Outputs");
099     for (Neuron n : neuralNet.output) {
100       n.printSelf(out);
101     }
102   }
103
104 public static NeuralNet[] crossOver(NeuralNet nn1, NeuralNet nn2)
{
105     int inputCnt = nn1.inputCnt;
106     Neuron[] input1 = new Neuron[inputCnt];
107     Neuron[] input2 = new Neuron[inputCnt];
108     for (int i = 0; i < inputCnt; i++) {
109       if (r.nextBoolean()) {
110         input1[i] = nn1.input[i];
111         input2[i] = nn2.input[i];
112       } else {
113         input1[i] = nn2.input[i];
114         input2[i] = nn1.input[i];
115       }
116     }
117
118     int hiddenCnt = nn1.hiddenCnt;
119     Neuron[] hidden1 = new Neuron[hiddenCnt];
120     Neuron[] hidden2 = new Neuron[hiddenCnt];
121     for (int i = 0; i < hiddenCnt; i++) {
122       if (r.nextBoolean()) {
123         hidden1[i] = nn1.hidden[i];
124         hidden2[i] = nn2.hidden[i];
125       } else {
126         hidden1[i] = nn2.hidden[i];
127         hidden2[i] = nn1.hidden[i];
128       }
129     }
130
131     int outputCnt = nn1.outputCnt;
132     Neuron[] output1 = new Neuron[outputCnt];
133     Neuron[] output2 = new Neuron[outputCnt];
134     for (int i = 0; i < outputCnt; i++) {
135       if (r.nextBoolean()) {
136         output1[i] = nn1.output[i];
137         output2[i] = nn2.output[i];
```

```
138         } else {
139           output1[i] = nn2.output[i];
140           output2[i] = nn1.output[i];
141         }
142       }
143 
144           return new NeuralNet[] {new NeuralNet(input1, hidden1,
output1), new NeuralNet(input2, hidden2, output2)};
145 
146     }
147 
148     public NeuralNet mutate() {
149       Neuron[] input = this.input.clone();
150       Neuron[] hidden = this.hidden.clone();
151       Neuron[] output = this.output.clone();
152       if (r.nextBoolean()) {
153         int i = r.nextInt(hiddenCnt);
154         hidden[i] = hidden[i].mutate();
155       } else {
156         int i = r.nextInt(outputCnt);
157         output[i] = output[i].mutate();
158       }
159       return new NeuralNet(input, hidden, output);
160     }
161 
162                                                                   }
```

6. Файл `Neuron.java`. Содержит абстрактный класс искусственного нейрона

```
01 package ru.ifmo.tsarev.bachelor.neuralnets;
02 
03 import java.io.PrintWriter;
04 
05 public abstract class Neuron {
06 
07   protected boolean valueCalculated;
08   protected double value;
09 
10   protected Neuron[] inputs;
11   protected int inputsCnt;
```

```
12    protected double[] w;
13
14    public Neuron(double[] w) {
15      this.w = w;
16    }
17
18    public void setInputs(Neuron[] inputs) {
19      if (inputs.length != this.w.length) {
20          throw new IllegalArgumentException("inputs and w must have
the same length");
21      }
22      this.inputs = inputs;
23      this.inputsCnt = inputs.length;
24    }
25
26    public abstract double getValue();
27
28    public void setValue(double value) {
29      this.value = value;
30      valueCalculated = true;
31    }
32
33    public abstract void printSelf(PrintWriter out);
34
35    public abstract Neuron mutate();
36
37 }
```

7. Файл `NeuronWithLimit.java`. Содержит класс, реализующий нейрон с пороговой функцией активации

```
01 package ru.ifmo.tsarev.bachelor.neuralnets;
02
03 import java.io.PrintWriter;
04 import java.util.Random;
05
06 public class NeuronWithLimit extends Neuron {
07
08    private static final double MAXW = 1;
09    private static final Random r = new Random();
10
```

```
11  private double limit;
12
13  public NeuronWithLimit(double[] w, double limit) {
14    super(w);
15    this.limit = limit;
16  }
17
18  @Override
19  public double getValue() {
20    if (valueCalculated) {
21      return value;
22    }
23
24    double res = 0;
25
26    for (int i = 0; i < inputsCnt; i++) {
27      res += w[i] * inputs[i].getValue();
28    }
29
30    if (res > limit) {
31      res = +1;
32    } else {
33      res = -1;
34    }
35
36    value = res;
37    valueCalculated = true;
38    return res;
39
40  }
41
42  public static NeuronWithLimit createRandom(int cnt) {
43    double[] w = new double[cnt];
44    for (int i = 0; i < cnt; i++) {
45      w[i] = r.nextDouble() * 2 * MAXW - MAXW;
46    }
47  return new NeuronWithLimit(w, r.nextDouble() * MAXW * 2 - MAXW);
48  }
49
50  @Override
51  public void printSelf(PrintWriter out) {
```

```
52      out.println("Neuron with limit");
53      for (double cw : w) {
54        out.print(cw + " ");
55      }
56      out.println();
57      out.println(limit);
58    }
59
60    @Override
61    public Neuron mutate() {
62      int n = w.length;
63      double[] newW = new double[n];
64      for (int i = 0; i < n; i++) {
65        if (r.nextDouble() <= 1.0 / n) {
66          newW[i] = w[i] + r.nextDouble() * 0.1 - 0.05;
67          if (newW[i] > 1) {
68            newW[i] = 1;
69          }
70          if (newW[i] < -1) {
71            newW[i] = -1;
72          }
73        } else {
74          newW[i] = this.w[i];
75        }
76      }
77      double newLimit = limit;
78      if (r.nextBoolean()) {
79        newLimit = + r.nextDouble() * 0.1 - 0.05;
80      }
81      return new NeuronWithLimit(newW, newLimit);
82    }
83
84  }
```

8. Файл `SigmoidNeuron.java`. Содержит класс, реализующий нейрон с сигмоидальной функцией активации

```
01 package ru.ifmo.tsarev.bachelor.neuralnets;
02
03 import java.io.PrintWriter;
```

```
04 import java.util.Random;
05
06 public class SigmoidNeuron extends Neuron {
07
08   private static final double MAXW = 1;
09   private static final Random r = new Random();
10
11   private final double limit;
12
13   public SigmoidNeuron(double[] w, double limit) {
14     super(w);
15     this.limit = limit;
16   }
17
18   @Override
19   public double getValue() {
20     if (valueCalculated) {
21       return value;
22     }
23
24     double res = 0;
25
26     for (int i = 0; i < inputsCnt; i++) {
27       res += w[i] * inputs[i].getValue();
28     }
29
30     res -= limit;
31
32     res = 1 / (1 + Math.exp(-res));
33
34     value = res;
35     valueCalculated = true;
36     return res;
37   }
38
39   public static SigmoidNeuron createRandom(int cnt) {
40     double[] w = new double[cnt];
41     for (int i = 0; i < cnt; i++) {
42       w[i] = r.nextDouble() * MAXW;
43     }
44     return new SigmoidNeuron(w, r.nextDouble() * MAXW * 2 - MAXW);
```

```
45    }
46
47    @Override
48    public void printSelf(PrintWriter out) {
49      out.println("Sigmoid neuron");
50      for (double cw : w) {
51        out.print(cw + " ");
52      }
53      out.println(limit);
54      out.println();
55    }
56
57    @Override
58    public Neuron mutate() {
59      int n = w.length;
60      double[] newW = new double[n];
61      for (int i = 0; i < n; i++) {
62        if (r.nextDouble() <= 1.0 / n) {
63          newW[i] = w[i] + r.nextDouble() * 0.1 - 0.05;
64          if (newW[i] > 1) {
65            newW[i] = 1;
66          }
67          if (newW[i] < 0) {
68            newW[i] = 0;
69          }
70        } else {
71          newW[i] = this.w[i];
72        }
73      }
74      double newLimit = limit;
75      if (r.nextBoolean()) {
76        newLimit = limit + r.nextDouble() * 0.1 - 0.05;
77      }
78      return new SigmoidNeuron(newW, newLimit);
79    }
80
81  }
```

9. **Файл `Automaton.java`. Содержит класс, реализующий конечный автомат с операциями скрещивания, мутации и создания случайного автомата**

```
001 package ru.ifmo.tsarev.bachelor.automata;
002
003 import java.io.PrintWriter;
004 import java.util.Collection;
005 import java.util.Random;
006 import java.util.Scanner;
007
008 import ru.ifmo.tsarev.bachelor.automata.actions.Action;
009
010 public class Automaton {
011
012   private final static Random r = new Random();
013
014   private final int statesCount;
015   private final int outDegree;
016   private final Transition[][] transitions;
017   private final int startState;
018
019   private int currentState;
020
021   public Automaton(Transition[][] transitions, int startState) {
022     this.transitions = transitions;
023     this.statesCount = transitions.length;
024     this.outDegree = transitions[0].length;
025     this.startState = startState;
026     this.currentState = this.startState;
027   }
028
029   public Action[] makeTransition(int[] inputValues) {
030     if (inputValues.length != 3) {
031       throw new IllegalArgumentException("inputValues.length must
be equal to 3");
032     }
033     int val0 = inputValues[0];
034     if (val0 == -1) {
035       val0 = 0;
```

```
036     }
037     int val1 = inputValues[1];
038     if (val1 == -1) {
039       val1 = 0;
040     }
041     int val2 = inputValues[2];
042     if (val2 == -1) {
043       val2 = 0;
044     }
045
046      Transition t = transitions[currentState][4 * val0 + 2 * val1
+ val2];
047     currentState = t.getToState();
048
049     return t.getActions();
050   }
051
052   public static Automaton createRandom(int states, int outDegree)
{
053                         Transition[][]     transitions     =     new
Transition[states][outDegree];
054     int startState = r.nextInt(states);
055     for (int i = 0; i < states; i++) {
056       for (int j = 0; j < outDegree; j++) {
057            transitions[i][j] = Transition.createRandom(states, i,
j);
058       }
059     }
060     return new Automaton(transitions, startState);
061   }
062
063   public static Automaton[] crossOver(Automaton a1, Automaton a2)
{
064     int n = a1.statesCount;
065     int m = a1.outDegree;
066
067     Transition[][] t1 = new Transition[n][m];
068     Transition[][] t2 = new Transition[n][m];
069
070     for (int i = 0; i < n; i++) {
071       for (int j = 0; j < m; j++) {
```

```
072         if (r.nextBoolean()) {
073           t1[i][j] = a1.transitions[i][j];
074           t2[i][j] = a2.transitions[i][j];
075         } else {
076           t1[i][j] = a2.transitions[i][j];
077           t2[i][j] = a1.transitions[i][j];
078         }
079       }
080     }
081
082     if (r.nextBoolean()) {
083       return new Automaton[] {new Automaton(t1, a1.startState),
084                   new Automaton(t2, a2.startState)};
085     } else {
086       return new Automaton[] {new Automaton(t1, a2.startState),
087                   new Automaton(t2, a1.startState)};
088     }
089   }
090
091    public static void printAutomaton(PrintWriter out, Automaton
automaton) {
092     out.println("Automaton");
093     out.println(automaton.statesCount);
094     out.println(automaton.startState);
095     for (int i = 0; i < automaton.statesCount; i++) {
096       for (int j = 0; j < automaton.outDegree; j++) {
097     Transition.printTransition(out, automaton.transitions[i][j]);
098       }
099     }
100   }
101
102   // Два типа мутаций
103   // 1) изменение начального состояния
104   // 2) мутация перехода
105   public Automaton mutate() {
106     if (r.nextBoolean()) {
107       int newStartState = r.nextInt(statesCount);
108       return new Automaton(transitions, newStartState);
109     } else {
110     Transition[][] newT = new Transition[statesCount][outDegree];
111       for (int i = 0; i < statesCount; i++) {
```

```
112           for (int j = 0; j < outDegree; j++) {
113             newT[i][j] = transitions[i][j];
114           }
115         }
116         int i = r.nextInt(statesCount);
117         int j = r.nextInt(outDegree);
118         newT[i][j] = newT[i][j].mutate(statesCount);
119         return new Automaton(newT, startState);
120       }
121     }
122
123 }
```

10. Файл `Transition.java`. Содержит класс, реализующий переход в конечном автомате

```
01 package ru.ifmo.tsarev.bachelor.automata;
02
03 import java.io.PrintWriter;
04 import java.util.Collection;
05 import java.util.Random;
06
07 import ru.ifmo.tsarev.bachelor.automata.actions.Action;
08 import ru.ifmo.tsarev.bachelor.automata.actions.AngleAction;
09 import ru.ifmo.tsarev.bachelor.automata.actions.FuelAction;
10 import ru.ifmo.tsarev.bachelor.automata.actions.SendMessageAction;
11
12 public class Transition {
13   private final static Random r = new Random();
14
15   private final int from;
16   private final int to;
17   private final FuelAction fuelAction;
18   private final AngleAction angleAction;
19   private final SendMessageAction sendMessageAction;
20   private final int event;
21
22     public Transition(int from, int to, int event, FuelAction
fuelAction, AngleAction angleAction, SendMessageAction sendMessageAction) {
23     this.from = from;
24     this.to = to;
```

```
25      this.event = event;
26      this.fuelAction = fuelAction;
27      this.angleAction = angleAction;
28      this.sendMessageAction = sendMessageAction;
29    }
30
31    public int getToState() {
32      return to;
33    }
34
35    public Action[] getActions() {
36                return   new   Action[]   {fuelAction,   angleAction,
sendMessageAction};
37    }
38
39     public static Transition createRandom(int states, int from, int
event) {
40      int to = r.nextInt(states);
41      FuelAction fa = FuelAction.createRandom();
42      AngleAction aa = AngleAction.createRandom();
43      SendMessageAction sma = SendMessageAction.createRandom();
44      return new Transition(from, to, event, fa, aa, sma);
45    }
46
47    @Override
48    public String toString() {
49      return from + " -> " + to + " on " + event;
50    }
51
52     public static void printTransition(PrintWriter out, Transition
transition) {
53       out.println(transition + " " + transition.angleAction + " " +
transition.fuelAction + " " + transition.sendMessageAction);
54    }
55
56    public Transition mutate(int statesCount) {
57      if (r.nextBoolean()) {
58          return new Transition(from, r.nextInt(statesCount), event,
fuelAction, angleAction, sendMessageAction);
59      } else {
60        int c = r.nextInt(3);
```

```
61          if (c == 0) {
62                  return new Transition(from, to, event, (FuelAction)
fuelAction.mutate(), angleAction, sendMessageAction);
63          } else if (c == 1) {
64                  return new Transition(from, to, event, fuelAction,
(AngleAction) angleAction.mutate(), sendMessageAction);
65          } else {
66                  return new Transition(from, to, event, fuelAction,
angleAction, (SendMessageAction) sendMessageAction.mutate());
67          }
68      }
69    }
70
71 }
```

11. Файл `Action.java`. Содержит абстрактный класс действия, выполняемого на переходе автомата

```
1 package ru.ifmo.tsarev.bachelor.automata.actions;
2
3 import ru.ifmo.tsarev.bachelor.environment.logic.Plate;
4
5 public abstract class Action {
6   public abstract void apply(Plate p);
7   public abstract Action mutate();
8 }
```

12. Файл `AngleAction.java`. Содержит класс действия, связанного с изменением угла поворота аэродинамических рулей

```
01 package ru.ifmo.tsarev.bachelor.automata.actions;
02
03 import java.util.Random;
04
05 import ru.ifmo.tsarev.bachelor.Parameters;
06 import ru.ifmo.tsarev.bachelor.environment.logic.Plate;
07
08 public class AngleAction extends Action {
09
10   private final static Random r = new Random();
11
```

```
12    private final double deltaA;
13
14    public AngleAction(double deltaA) {
15      this.deltaA = deltaA;
16    }
17
18    @Override
19    public void apply(Plate p) {
20      p.setA(p.getA() + deltaA);
21    }
22
23    public static AngleAction createRandom() {
24      return new AngleAction((r.nextInt(11) - 5) * 5);
25    }
26
27    @Override
28    public String toString() {
29      return "AngleAction: " + deltaA;
30    }
31
32    @Override
33    public Action mutate() {
34      double delta = (r.nextBoolean() ? 1 : -1) * 5;
35      double newDeltaA = deltaA + delta;
36      if (newDeltaA > 25) {
37        newDeltaA = 25;
38      }
39      if (newDeltaA < -25) {
40        newDeltaA = -25;
41      }
42      return new AngleAction(newDeltaA);
43    }
44
45  }
```

13. Файл `FuelAction.java`. Содержит класс действия, связанного с изменением расхода топлива

```
01  package ru.ifmo.tsarev.bachelor.automata.actions;
02
03  import java.beans.FeatureDescriptor;
```

```
04 import java.util.Random;
05
06 import ru.ifmo.tsarev.bachelor.environment.logic.Plate;
07
08 public class FuelAction extends Action {
09
10   private final static Random r = new Random();
11
12   private final double fuel;
13   public FuelAction(double fuel) {
14     this.fuel = fuel;
15   }
16
17   @Override
18   public void apply(Plate p) {
19     p.setQ(fuel);
20   }
21
22   public static FuelAction createRandom() {
23 //     return new FuelAction(r.nextDouble());
24 //     return new FuelAction(r.nextInt(6) * 0.2);
25     if (r.nextBoolean()) {
26       return new FuelAction(0.4);
27     } else {
28       return new FuelAction(0.8);
29     }
30   }
31
32   @Override
33   public String toString() {
34     return "FuelAction: " + fuel;
35   }
36
37   @Override
38   public Action mutate() {
39     double delta = r.nextDouble() * 0.1 - 0.05;
40     double newFuel = fuel + delta;
41     if (newFuel > 1) {
42       newFuel = 1;
43     }
44     if (newFuel < 0) {
```

```
45        newFuel = 0;
46      }
47      return new FuelAction(newFuel);
48    }
49
50 }
```

14. Файл `SendMessageAction.java`. Содержит класс действия, связанного пересылкой сообщения другому агенту (в настоящей работе эта возможность не реализована)

```
01 package ru.ifmo.tsarev.bachelor.automata.actions;
02
03 import ru.ifmo.tsarev.bachelor.environment.logic.Plate;
04
05 public class SendMessageAction extends Action {
06   // TODO
07
08   @Override
09   public void apply(Plate p) {
10     // TODO: think about it :)
11   }
12
13   public static SendMessageAction createRandom() {
14     // TODO: generate something reasonable ;)
15     return new SendMessageAction();
16   }
17
18   @Override
19   public String toString() {
20     return "SendMessageAction: ";
21   }
22
23   @Override
24   public Action mutate() {
25     return this;
26   }
27
28 }
```

15. Файл `FitnessCalculator.java`. Содержит абстрактный класс функции приспособленности

```
1 package ru.ifmo.tsarev.bachelor.genetic.fitness;
2
3 import ru.ifmo.tsarev.bachelor.genetic.Individual;
4
5 public interface FitnessCalculator {
6   public double calcFitness(Individual individual);
7 }
```

16. Файл `FixedManagersFitnessCalculator.java`. Содержит реализацию функцию приспособленности, вычисляемой соревнованием с заданным набором команд

```
01 package ru.ifmo.tsarev.bachelor.genetic.fitness;
02
03 import ru.ifmo.tsarev.bachelor.environment.Config;
04 import ru.ifmo.tsarev.bachelor.environment.judging.CompetitionResult;
05 import ru.ifmo.tsarev.bachelor.environment.judging.Judge;
06 import ru.ifmo.tsarev.bachelor.environment.judging.Manager;
07 import ru.ifmo.tsarev.bachelor.genetic.Individual;
08 import ru.ifmo.tsarev.bachelor.genetic.samplemanagers.AgressiveManager;
09 import ru.ifmo.tsarev.bachelortests.FlyForwardManager;
10
11   public class FixedManagersFitnessCalculator implements FitnessCalculator {
12   private Manager[] fixedManagers;
13   private final double[][] xCoordinates;
14   private final int competitionsCount;
15
16   private final Judge judge;
17
18   public FixedManagersFitnessCalculator(int competitionsCount) {
19     this.competitionsCount = competitionsCount;
20                                 xCoordinates = new double[competitionsCount][Config.PLATES_COUNT];
21     judge = Judge.getInstance();
```

```
22      initManagers();
23    }
24
25    private void initManagers() {
26      // TODO : use Reflection to get Managers from samplemanagers
package
27      fixedManagers = new Manager[2];
28      fixedManagers[0] = new FlyForwardManager();
29      fixedManagers[1] = new AgressiveManager();
30    }
31
32    public double calcFitness(Individual individual) {
33      double fitness = 0;
34      int winCount = 0;
35      int summaryCC = 0;
36      for (Manager m : fixedManagers) {
37        for (int i = 0; i < competitionsCount; i++) {
38            CompetitionResult res = judge.makeCompetition(individual,
m, xCoordinates[i]);
39          fitness += res.getResults()[0];
40          if (res.getWinner() == CompetitionResult.FIRST_PLAYER_WON)
{
41            winCount++;
42          }
43          summaryCC++;
44        }
45      }
46      return fitness + 1.0 * winCount / (summaryCC + 1);
47    }
48
49  }
```

17. Файл `AgressiveManager.java`. Содержит реализацию агрессивной стратегии управления командой летающих тарелок

```
01 package ru.ifmo.tsarev.bachelor.genetic.samplemanagers;
02
03 import java.util.Random;
04
05 import ru.ifmo.tsarev.bachelor.environment.Config;
06 import ru.ifmo.tsarev.bachelor.environment.judging.Manager;
```

```
07 import ru.ifmo.tsarev.bachelor.environment.logic.EnvironmentLogic;
08 import ru.ifmo.tsarev.bachelor.environment.logic.Plate;
09
10 public class AgressiveManager implements Manager {
11
12   private int player;
13   private final static Random random = new Random();
14
15   public void init(int player) {
16     this.player = player;
17   }
18
19   public void makeTurn(EnvironmentLogic el) {
20         int cnt = 0;
21         for (Plate plate: el.getPlates()) {
22             cnt++;
23             if (cnt < Config.getPlatesCount()) {
24                       if ((Math.abs(plate.getSpeed().x) < 1e-6) ||
(plate.getSpeed().y / plate.getSpeed().x > -0.3)) {
25  plate.setA(random.nextDouble() * Config.getMaximalRotateAngle());
26                 } else {
27                     plate.setA(0);
28                 }
29                 plate.setQ(random.nextDouble() + 0.4);
30             } else {
31                 if (plate.getPosition().y > Config.getFieldHeight()
- 2) {
32                     plate.setA(5);
33                 } else {
34   plate.setA(Math.atan2(plate.getSpeed().y,  plate.getSpeed().x)  *
180 / Math.PI);
35                 }
36                 plate.setQ(0.4);
37             }
38         }
39   }
40
41 }
```

18. Файл `FlyForwardManager.java`. Содержит реализацию «простой» стратегии управления командой летающих тарелок

```
01 package ru.ifmo.tsarev.bachelor.genetic.samplemanagers;
02
03 import ru.ifmo.tsarev.bachelor.environment.judging.Manager;
04 import ru.ifmo.tsarev.bachelor.environment.logic.EnvironmentLogic;
05 import ru.ifmo.tsarev.bachelor.environment.logic.Plate;
06
07 public class FlyForwardManager implements Manager {
08
09   private int player;
10   public void init(int player) {
11     this.player = player;
12   }
13
14   public void makeTurn(EnvironmentLogic el) {
15     for (Plate p : el.getPlates()) {
16       if (p.getPlayer() == player) {
17         p.setA(0);
18         p.setQ(0.4);
19       }
20     }
21   }
22
23 }
```