Научно-технический вестник Санкт-Петербургского государственного университета информационных технологий, механики и оптики. Вып. 53. Автоматное программирование. 2008, с. 88–99.

УДК 004.4’242

СОЗДАНИЕ СИСТЕМЫ УПРАВЛЕНИЯ ТАНКОМ ДЛЯ ИГРЫ *ROBOCODE* С ИСПОЛЬЗОВАНИЕМ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ

Ю. Д. Бедный, А. А. Шалыто

В работе предлагается метод создания системы управления танком для игры Robocode, основанный на применении генетических алгоритмов, что позволяет автоматизировать процесс построения системы управления. Кроме того, рассматривается способ улучшения генерируемых систем управления за счет использования более сложного алгоритма управления, реализованного с помощью конечного автомата.

Введение

В работе предлагается метод автоматизации построения системы управления танком для популярной компьютерной игры *Robocode* [1]. Этот метод позволяет на основе использования генетических алгоритмов [2, 3] строить достаточно простые системы управления танком.

Работа *не* ставит своей целью создание танка, способного побеждать танки, построенные вручную, которые являются наилучшими из известных на сегодняшний день (например, танк `voidious.Dookious`). Алгоритмы, используемые в таких танках, являются достаточно сложными, и их реализация занимает сотни килобайт, а работа по их созданию крайне трудоемка.

Танки, создаваемые предлагаемым методом, должны побеждать в индивидуальном сражении каждого *заранее выбранного соперника* из тестового набора, который включает в себя танки, поставляемые вместе с игрой, а также танк `newCynic.Cynical` [4]. Выбор последнего был связан с тем, что этот танк (как и прототип, на основе которого он построен [5]), в отличие от многих других танков, был предварительно спроектирован и для него существует достаточно подробная открытая проектная документация. Для каждого танка из тестового набора с помощью предлагаемого метода генерируется танк, который его побеждает. Предлагаемый метод позволяет генерировать танки, системы управления которых, построены как с применением автоматного подхода [6], так и без его использования. При этом если автоматы не используются, то система управления описывается только одной функцией, формирующей выходные воздействия на объект управления. При автоматном подходе для каждого состояния автомата определена соответствующая функция управления.

На рис. 1 в качестве примера приведен внешний вид среды игры *Robocode* во время одного из боев. На этом рисунке разработанный танк назван *EvolvedTank*.

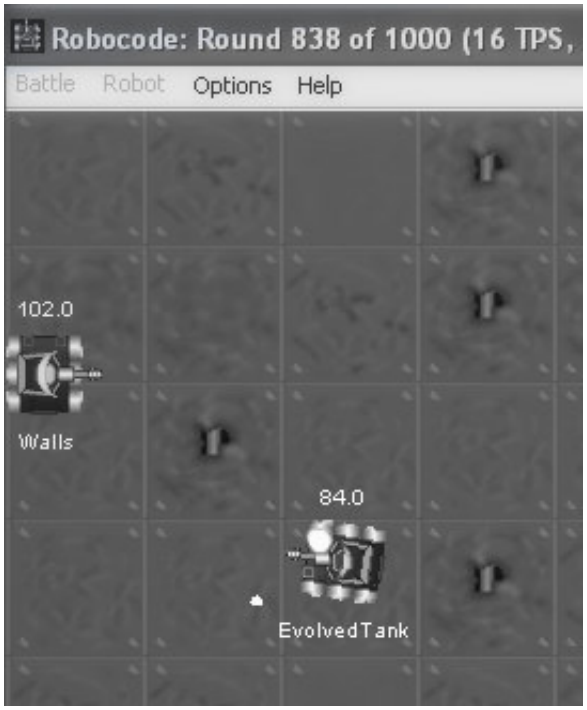

Рис. 1. Пример боя в игре *Robocode*

1. Обзор

Известно большое число танков для рассматриваемой игры [7]. Наиболее простые из них реализованы всего несколькими строками кода, а программы, реализующие наиболее сложные танки, как отмечалось выше, занимают сотни килобайт. Большинство танков создано вручную. Существуют также работы [8, 9], в которых танки строятся не вручную, а автоматически – с применением методов искусственного интеллекта.

В работе [8] использовано несколько таких методов (обучение с подкреплением, нейронные сети и генетические алгоритмы) для управления различными системами танка (радаром, системой движения и пушкой). Генетические алгоритмы в работе [8] применялись для создания систем управления радаром и движением танка. Однако их использование не привело к желаемому результату – система управления радаром осуществляла его поворот на 360 градусов, а система передвижения задавала перемещение танка по кругу.

В работе [9] используется сравнительно сложный способ представления системы управления в виде особи генетического алгоритма – программы на языке *TableRex*. При этом длина программы составила 7776 бит.

2. Постановка задачи. Устройство танка схематично показано на рис. 2. Необходимо управлять следующими устройствами: радаром (`Radar`), пушкой (`Gun`) и системой движения (`Body`).

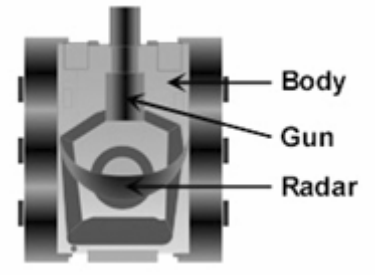

Рис. 2. Схематичное изображение танка

Управление танком осуществляется следующим образом. В каждый момент времени анализируется текущая ситуация (положение танка, его скорость и т.д.). На основе этой информации формируются четыре действия: передвижение (вперед/назад), поворот, поворот пушки, стрельба. Указанные действия формируются в результате интерпретации функции управления, которая генерируется на основе генетических алгоритмов.

Перейдем к формальной постановке задачи. Обозначим текущее время в игре как $t \in N$, а максимальное время, которое возможно для данной игры – M. Обозначим множество позиций как S, а множество действий танка как A. *Позиция* – элемент некоторого множества всех возможных ситуаций в игре в выбранный момент времени. Позиция включает в себя положение каждого танка, его скорость, направление, угол поворота пушки и радара, энергию, информацию о танке соперника и т. д. Задача управления для рассматриваемой игры в общем случае состоит в задании для каждого момента времени t функции управления $f_t : S^t \to A$, которая на основании информации о *позициях* во все предыдущие моменты времени (получаемой из среды *Robocode*) определяет действие на объект управления в текущий момент времени. Таким образом, решение задачи управления танком – вектор $[f_1, f_{2,\ldots,} f_M]^T$.

В данной работе решается упрощенная задача управления:

- действие танка в текущий момент времени зависит только от *позиции* в этот момент времени;
- зафиксирован алгоритм управления радаром – вращение по кругу;
- при выборе действия анализируется лишь упрощенная *позиция* – элемент множества $S' = \{< x,\ y,\ dr, tr, w, dh, GH, h, d, e, E >\}$. Здесь:
    - x, y – координаты танка соперника относительно нашего танка;
    - dr – расстояние, которое осталось “доехать” нашему танку (после вызова метода `AdvancedRobot.setAhead`);
    - tr – угол, на который осталось повернуться нашему танку;
    - w – расстояние от нашего танка до края поля;
    - dh – угол между направлением на танк соперника и пушкой нашего танка;
    - GH – угол поворота пушки нашего танка;
    - h – направление движения танка соперника;
    - d – расстояние между нашим танком и танком соперника;
    - e – энергия танка соперника;
    - E – энергия нашего танка;

- множество действий $A' = \{g, p, d, h\}$, где g – угол поворота пушки, p – энергия снаряда (неположительные значения означают, что выстрел не производится), d – расстояние, на которое перемещается танк, h – угол поворота танка.

Учитывая изложенное, задача построения системы управления танком сведена к заданию функции $f : S' \to A'$. Эту функцию будем искать *автоматически* с использованием генетических алгоритмов, а точнее с помощью их разновидности, обладающей высокой эффективностью – программированием с экспрессией генов (*Gene Expression Programming*) [10]. В отличие как от стандартных генетических алгоритмов, использующих строки фиксированной длины, так и от генетического программирования [11], использующего деревья, хромосомы в генетическом программировании с экспрессией генов представляются в виде строк фиксированной длины, которые преобразуются в, так называемые, *Karva*-деревья. Генетические операции с такими деревьями (в отличие от генетического программирования) *корректны по определению*. За счет этого, при использовании программирования с экспрессией генов не приходится тратить время на проверку корректности деревьев, получаемых в результате скрещивания и мутаций. Это обеспечивает используемому подходу указанные преимущества перед генетическим программированием.

Для решения поставленной задачи с помощью генетических алгоритмов (и, в частности, для программирования с экспрессией генов) необходимо определить способ представления функции в виде хромосомы (особи популяции генетического алгоритма), выбрать функцию приспособленности и определить генетические операции (мутация, скрещивание и отбор).

Как было отмечено во введении, предлагаемый метод позволяет строить системы управления, как с использованием автоматного подхода, так и без его применения. Ниже эти подходы рассматриваются отдельно.

3. Построение системы управления без применения автоматов

Сначала опишем подход, не предполагающий использования автоматов при создании системы управления.

3.1. Общая схема генетических алгоритмов

Схема одного шага работы генетических алгоритмов приведена на рис. 3.

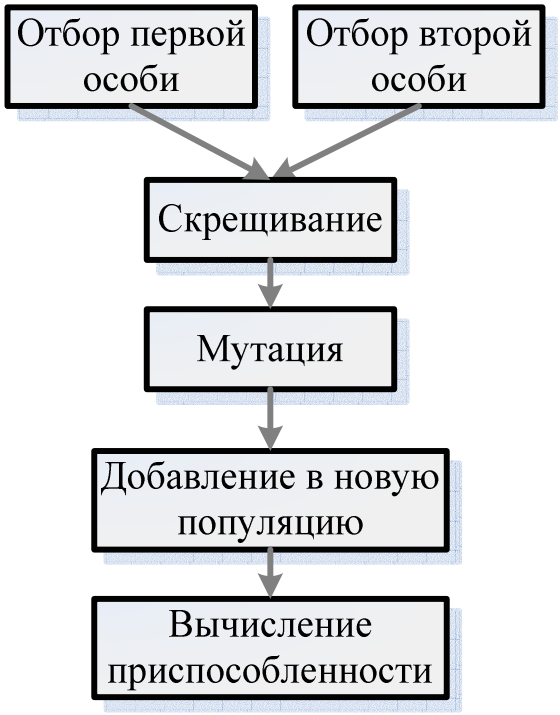

Рис. 3. Один шаг работы генетических алгоритмов

3.2. Представление в виде хромосомы

Для представления в виде особи генетического алгоритма искомой функции управления $f : S' \to A'$ предлагается представлять ее в виде четырех функций f_i, каждая из которых возвращает вещественное число: f_1 – угол поворота пушки, f_2 – энергия снаряда, f_3 – расстояние, на которое перемещается танк, f_4 – угол поворота танка.

В свою очередь, функция f_i представляется в виде *Karva*-дерева – стандартного способа представления для программирования с экспрессией генов. Таким образом, хромосома состоит из четырех строк – линеаризованных *Karva*-деревьев, для представления которых применяются описываемые ниже функции и терминалы.

Требования, предъявляемые к набору базовых функций, используемых при задании хромосомы: набор должен быть достаточно полным, для того чтобы выразить необходимые зависимости действий от позиции, и не слишком избыточным, для того чтобы работа генетического алгоритма была эффективной.

Предлагается использовать следующий набор базовых функций: $+(x, y) = x + y$, $++(x, y, z) = x + y + z$, $n(x) = -x$, $*(x, y) = xy$, $**(x, y, z) = xyz$, min, $s(x) = \dfrac{1}{1+e^{-x}}$, $if>(x, y, z, w) = x > y \ ? \ z : w$.

Набор терминалов (координаты танка (`x`, `y`), его энергия (`e`) и т.д.) строится исходя из определения позиции (элемента множества S'). Кроме того, этот набор необходимо дополнить некоторым множеством констант. В настоящей работе были выбраны следующие константы: 0, .5, 1, 2, 10.

В табл. приведен пример сгенерированной хромосомы, а в табл. – соответствующая этой хромосоме функция управления $f = (f_1, f_2, f_3, f_4)$, представленная через базовые функции.

Таблица 1. Представление функции в виде хромосомы

Линеаризованные Karva-деревья	
f_1	4 0 7 1 17 4 5 2 4 1 18 9 10 13 12 10 13 22 23 11 17 21 8 15 16 14 21 13 16 18 22 10 16 19 13 8 21 13 10 8 23
f_2	6 4 5 8 0 2 7 0 5 0 20 15 8 19 19 18 8 11 15 16 13 22 23 22 12 20 22 19 13 19 23 15 15 22 11 22 11 14 8 17 23
f_3	7 2 1 3 6 5 6 0 6 7 21 16 22 19 21 17 19 23 9 13 17 13 14 15 9 21 12 16 21 14 16 23 16 15 23 16 8 21 15 14 22
f_4	2 4 5 0 2 7 1 3 4 1 19 8 15 17 17 9 18 13 13 18 18 21 17 17 17 15 8 9 18 22 17 23 19 21 10 10 11 20 13 19 23

Таблица 2. Функции, соответствующие хромосоме

Функция $f = (f_1, f_2, f_3, f_4)$
f_1 = *(s[n(*(.5, e))], if>[10, *(n(E), dh), *(y, dr, .5), +(w, dr)])
f_2 = min(*[x, s(*(x, tr, 1))], *[+(s(2), h, if>(1, x, GH, GH), s(dh)])
f_3 = if>(+[*(GH, d, 10), min(GH, E)], n[s(y)], +[min(.5, 10), if>(.5, 0, 1, y), d], min[2, e])
f_4 = +(*[s(*(dh, dh)), +(n(d), GH)], *[if>(x, 1, 10, 10), n(y), +(dh, .5, .5)])

На рис. 4 приведен пример *Karva*-дерева для функции f_1 из табл. .

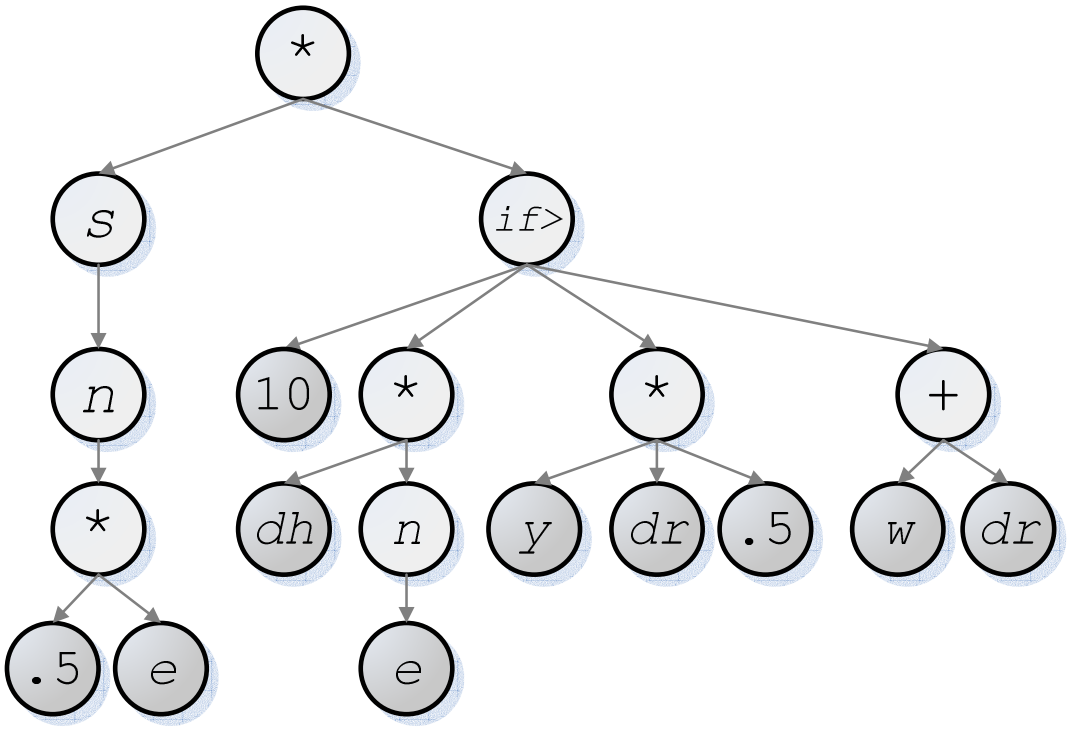

Рис. 4. ***Karva***-дерево функции f_1 из табл.

3.3. Функция приспособленности и критерий остановки процесса эволюции

Опишем, как была выбрана функция приспособленности. Она отражает результат поединков против выбранного соперника. При этом отметим, что, так как результат поединка существенно зависит от начального положения танков, то проводится не один поединок, а двадцать. Увеличение числа сражений позволило бы получить более точный результат, однако это было бы связано с возрастанием времени вычисления функции приспособленности, которое и так велико.

Для того чтобы уравнять шансы соперников, выбор начальных положений танков производится следующим образом. Создается случайный набор пар координат ((x_1, y_1), (x_2, y_2)). После этого для каждой из них проводится два поединка, в первом из которых танк соперника имеет начальные координаты (x_1, y_1), а во втором – (x_2, y_2). Обозначим число очков, набранных нашим танком в бою (состоящим из 20 раундов) как s, а число очков, набранных его соперником, как s_e. При этом приспособленность танка будем вычислять следующим образом: $f = 100\dfrac{s}{s+s_e}$.

Процесс эволюции продолжается до тех пор, пока максимальное значение функции приспособленности для элементов популяции не превзойдет некоторого порогового значения. Например, пороговое значение, равное 70, означает, что танк в среднем “научился” побеждать своего соперника. При этом отметим, что значение функции приспособленности, равное 50, означает, что наилучший из танков *текущей популяции* в ходе двадцати раундов при случайном начальном положении танков, выигрывает половину раундов.

3.4. Генетические операции

Для реализации генетического алгоритма в данной работе используются стандартные генетические операции: отбор, мутация и скрещивание. Отбор осуществляется по методу рулетки. Скрещивание хромосом (как наборов из четырех строк, например, приведенных в табл.) производится поэлементно. При этом для каждой пары строк с вероятностью p в качестве i-го элемента результатом скрещивания выбирается i-й элемент либо первой, либо второй хромосомы, а с вероятностью $(1 - p)$ – скрещивание производится стандартным образом с использованием метода битовой маски. Аналогично скрещиванию, мутация выполняется поэлементно. При этом каждый из символов строки с некоторой вероятностью заменяется некоторым другим. Более подробное описание указанных операций приведено в работе [2].

4. Эксперименты

При реализации генетического алгоритма число особей в популяции было выбрано равным 50. Один шаг эволюции (скрещивание, мутация, отбор, вычисление функции приспособленности) в популяции такого размера выполняется около пяти минут на компьютере *Intel Pentium M* 1.86 *GHz*. При этом практически все время тратится на вычисление функции приспособленности. Данное обстоятельство не позволяет использовать популяции б*о*льших размеров. Так, например, даже при указанном числе особей время работы генетического алгоритма может занимать несколько часов.

На рис. 5 приведен пример графика, иллюстрирующего работу генетического алгоритма для генерации танка против соперника `sample.Walls`. По оси абсцисс отложен номер поколения, а по оси ординат – значение функции приспособленности (`Max` – максимальное значение среди особей популяции, а `Avg` – среднее значение).

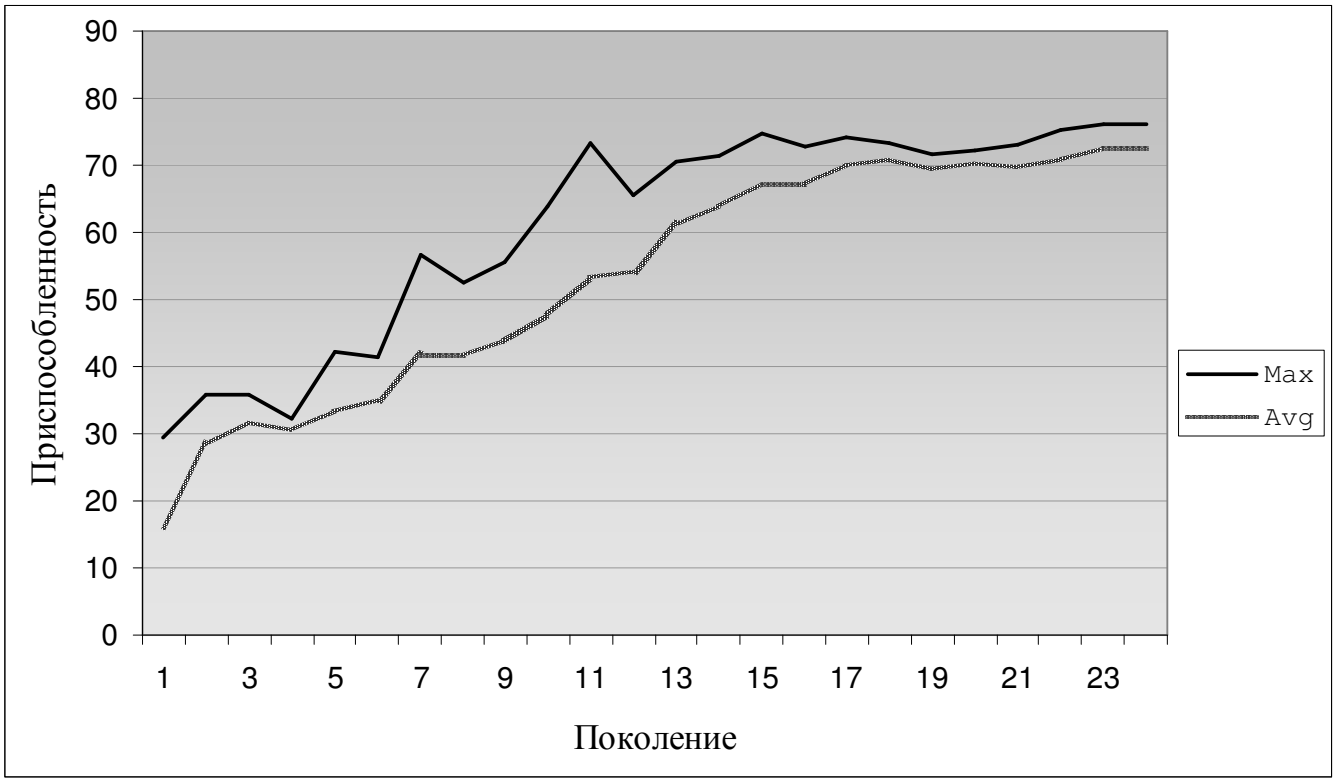

Рис. 5. Процесс эволюционного построения танка

Из рассмотрения графика максимального значения функции приспособленности (Max) следует, что уже на 11-ом поколении эволюцию можно было остановить, так как максимальное значение функции приспособленности превысило пороговое значение 70. Однако процесс был остановлен после 23-го поколения, так как значение функции приспособленности перестало изменяться существенно.

Таким образом строится каждый танк в результате сражений с одним из танков тестового набора. В табл. приведены описания систем управления, которые сгенерированы в ходе сражений с танками соперников newCynic.Cynical и sample.Walls.

Таблица 3. Представление функции управления сгенерированных танков

Соперник	Описание функции
newCynic.Cynical	0 2 1 6 3 6 0 6 13 19 14 13 23 10 28 18 13 17 18 13 19 14 17 23 15 13 18 9 29 30 15 14 13
	0 7 2 0 4 6 1 7 9 22 23 11 23 22 16 23 18 27 10 22 27 16 9 13 15 21 24 29 14 17 16 20 23
	3 4 8 1 6 0 0 5 14 17 21 24 21 17 19 11 19 11 24 18 28 23 23 15 10 24 16 22 14 30 26 14 15
	7 3 3 1 8 0 1 2 18 23 29 15 26 29 16 25 30 25 16 23 26 26 27 17 18 26 19 25 15 10 19 28 20
sample.Walls	7 0 6 8 2 3 8 6 0 5 23 13 18 9 12 20 23 14 20 27 29 12 14 26 14 10 13 26 10 17 22 26 21 18 14 30 14 21 20 23 9
	4 4 4 4 1 6 1 2 2 2 9 22 19 19 29 25 21 26 17 17 14 27 13 14 21 16 22 30 23 21 29 14 21 19 15 9 9 19 22 9 20
	24 0 3 2 1 5 7 1 2 1 16 11 21 15 25 18 25 30 18 17 21 9 25 19 26 10 28 28 13 12 11 28 28 18 14 14 14 26 11 30 28
	2 5 6 6 5 13 6 5 8 12 23 11 28 22 21 14 16 15 21 15 29 29 19 23 14 20 24 27 14 14 22 26 26 10 12 26 15 28 10 15 22

Эти танки использовались и в сражениях, результаты которых перечислены в табл. , так как указанные танки являются наиболее сильными в тестовом наборе. После генерации каждого танка, для оценки его эффективности были проведены бои с тем же соперником, против которого он выводился. Бои состояли из 100 раундов. В табл. приведены результаты этих боев. В этой таблице в столбце «Счет» первое значение – число очков набранных нашим танком, а второе – танком соперника. В общем случае, как отмечалось выше, увеличение числа раундов приводит к более объективным результатам.

Таблица 4. Результаты поединков (100 раундов)

Соперник	Счет
`newCynic.Cynical`	10933 : 8108
`sample.Walls`	9559 : 7240
`sample.SpinBot`	11414 : 8556
`sample.MyFirstRobot`	11964 : 5346
`sample.Corners`	18086 : 2971
`sample.Crazy`	10797 : 4278
`sample.Fire`	17610 : 5500
`sample.Tracker`	18642 : 4844
`sample.TrackFire`	16269 : 9851
`sample.Target`	17968 : 5
`sample.RamFire`	18572 : 3089

5. Автоматный подход

Эвристическое построение автоматов, управляющих танком, применялось в работе [4, 5]. В тоже время попыток автоматически построить автоматы для управления танком ранее не предпринималось.

В рассмотренном выше подходе, управление танком в любой момент времени осуществляется единственной функцией $f : S' \to A'$. При использовании автоматов эта функция декомпозируется с помощью состояний следующим образом. В каждом состоянии автомата определим функцию $f_i : S' \to A'$, где i – номер рассматриваемого состояния. Для каждой пары различных состояний с номерами (i, j) зададим *функцию перехода* $f_{ij} : S' \to R$. Построение функций переходов выполняется с помощью генетических алгоритмов, также как это делается для компонент функции управления. При этом, функции представляются в виде *Karva*-деревьев.

Выделим начальное состояние. Далее в каждый момент времени, находясь в состоянии *i*, последовательно выполняются следующие действия:

- вычисляются значения функций f_{ij} для всех j, $i \neq j$. Как только одно из вычисленных значений оказывается больше нуля $(f_{ij} > 0)$, выполняется переход в состояние *j*. Если же все $f_{ij} \leq 0$, то состояние не изменяется;
- определяется значение *функции управления* f_i, и выполняется действие, соответствующее найденному значению.

Для представления автомата в виде особи генетического алгоритма предлагается хранить в хромосоме (по аналогии с подходом, описанным выше) `4 * N + N * (N – 1)` функций, где `N` – число состояний автомата. Скрещивание, мутация и отбор выполняются таким же образом, как и ранее. На рис. 6 приведен пример автомата с тремя состояниями, управляющего танком.

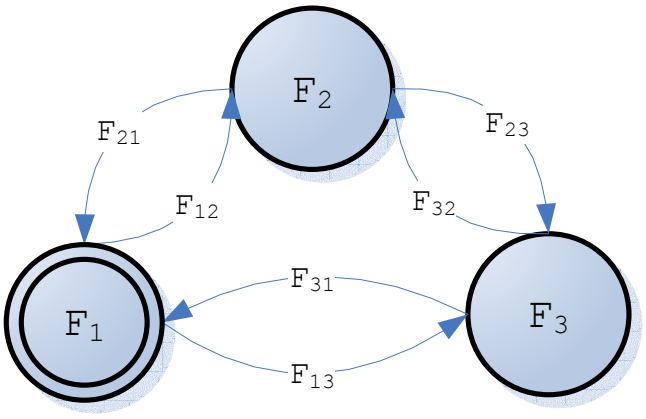

Рис. 6. Автомат управления танком

Табл. иллюстрирует процесс эволюционного построения танка с автоматной структурой.

Таблица 5. Работа генетического алгоритма (процесс эволюции)

Номер поколения	Максимальное значение функции приспособленности	Среднее значение функции приспособленности	Время, с
0	31.15	15.70	122
1	36.71	26.42	138
2	34.93	28.91	152
3	40.30	32.96	168
…	…	…	…
9	52.25	43.74	261
10	55.53	41.15	358
…	…	…	…
21	64.49	59.14	599
22	69.44	60.18	606
23	91.38	62.55	631
24	89.45	72.68	621
…	…	…	…
36	93.73	91.48	778

Эволюция была остановлена на 36-ом поколении, так как рост максимального значения функции приспособленности практически остановился.

В табл. приведен пример описания автомата из двух состояний в виде особи генетического алгоритма. Это описание содержит десять строк: восемь (f_1 – f_8)

определяются состояниями и две (f_9, f_{10}) – переходами. Данный автомат выводился в результате боёв с танком `sample.Walls`.

Таблица 6. Представление функции управления сгенерированных танков

Описание автомата	
f_1	5 2 2 5 6 7 6 1 4 4 8 20 9 8 12 23 22 22 16 13 14 10 12 9 14 16 9 18 11 11 23 23 19 18 23 13 15 16 16 22 14
f_2	3 16 2 5 1 2 0 6 3 5 15 8 17 12 22 14 21 11 10 12 13 21 14 14 8 19 11 20 17 15 23 14 22 18 8 10 9 21 20 18 22
f_3	3 7 14 3 7 5 4 16 4 1 17 16 9 20 16 22 12 10 17 9 11 8 19 19 16 12 11 8 15 20 9 23 20 19 18 10 19 21 13 10 18
f_4	1 7 14 6 2 4 7 6 4 6 9 8 12 21 17 10 13 16 23 18 17 14 15 15 19 17 18 14 21 9 15 17 12 19 11 8 13 9 23 10 18
f_5	4 0 7 1 17 4 5 2 4 1 18 9 10 13 12 10 13 22 23 11 17 21 8 15 16 14 21 13 16 18 22 10 16 19 13 8 21 13 10 8 23
f_6	6 4 5 8 0 2 7 0 5 0 20 15 8 19 19 18 8 11 15 16 13 22 23 22 12 20 22 19 13 19 23 15 15 22 11 22 11 14 8 17 23
f_7	7 2 1 3 6 5 6 0 6 7 21 16 22 19 21 17 19 23 9 13 17 13 14 15 9 21 12 16 21 14 16 23 16 15 23 16 8 21 15 14 22
f_8	2 4 5 0 2 7 1 3 4 1 19 8 15 17 17 9 18 13 13 18 18 21 17 17 17 15 8 9 18 22 17 23 19 21 10 10 11 20 13 19 23
f_9	20 6 4 3 5 5 3 0 6 2 11 8 22 15 15 19 23 14 20 10 23 17 18 20 17 13 11 20 20 14 18 15 11 10 12 11 8 8 13 21 12
f_{10}	3 7 0 4 1 4 5 1 4 16 22 15 8 14 22 19 18 17 22 10 11 15 21 13 9 13 11 8 8 13 16 11 14 8 10 23 8 23 9 17 18

Сгенерированный танк в 1000 раундах побеждает танк соперника со счетом 180168 на 28278. Таким образом, в среднем наш танк побеждает противника с вероятностью около 86%. Это несколько меньше, чем полученное в табл. максимальное значение функции приспособленности – 93.73. Данное обстоятельство объясняется тем, что результат боя из 20 раундов имеет большую погрешность из-за сравнительно небольшого числа экспериментов.

Применение автоматного подхода для описания поведения танка позволяет генерировать более сложные системы управления по сравнению с первым подходом, и, следовательно, создавать более сильные танки. Однако на практике полученные танки обладают примерно такой же эффективностью, что и танки, построенные без автоматов. Это, видимо, связано с тем, что пространство поиска (множество автоматов с фиксированным числом состояний) достаточно велико для того, чтобы была возможность генерировать “хорошие танки” за разумное время в условиях медленного вычисления функции приспособленности. Можно надеяться, что дальнейшая работа в этом направлении (например, ускорение вычисления функции приспособленности) позволит более эффективно использовать автоматный подход при построении танков с применением генетических алгоритмов.

6. Программная поддержка метода

Управление танком может осуществляться либо синхронно, либо асинхронно. В первом случае главный класс программы наследуется от библиотечного класса `Robot`, а во втором – от библиотечного класса `AdvancedRobot` [1]. В данной работе был выбран второй вариант. Таким образом, приведенный в приложении код, принадлежит классу, который наследуется от класса `AdvancedRobot`.

Метод поддержан двумя типами программ. Первая из них на основе использования генетических алгоритмов строит символическое описание системы управления, пример которого приведен в табл., а вторая – интерпретирует ее и управляет танком (приложение).

Первая программа имеет две модификации, соответствующие традиционному и автоматному подходам. Вторая программа не зависит от используемого подхода и состоит из следующих основных компонентов:

- среда *Robocode*;
- система управления, которая по входным воздействиям, получаемым из среды, формирует выходные воздействия на объект управления;
- объект управления (танк), содержащий систему движения и пушку. Радар реализован тривиально, и поэтому в отдельный класс не выделен.

Эта программа является главной и содержит обращение к некоторым классам, исходный код которых не приведен в приложении. Обе программы написаны на языке *Java*.

Заключение

В результате выполнения настоящей работы предложен метод построения системы управления танком для игры *Robocode*. Достоинство этого метода состоит в том, что при его использовании, построение системы управления происходит автоматически. Структуры генерируемых систем управления достаточно просты, но в то же время танки с этими системами управления побеждают любой фиксированный танк из тестового набора. Предложенный метод также позволяет генерировать системы управления с использованием автоматов.

В заключение отметим, что предлагаемый метод не позволяет генерировать танки, побеждающие наиболее сильных из известных танков, так как в последних используются эвристически созданные сложные и эффективные алгоритмы управления.

Литература

1. Официальный сайт игры Robocode. http://robocode.sourceforge.net/
2. Mitchell M. An Introduction to Genetic Algorithms. MA. The MIT Press, 1996.
3. Гладков Л. А., Курейчик В. В., Курейчик В. М. Генетические алгоритмы. М. Физматлит, 2006.
4. Кузнецов Д. В., Шалыто А. А. Система управления танком для игры «Robocode». Вариант 2. СПбГУ ИТМО, 2003. http://is.ifmo.ru/projects/robocode2/
5. Туккель Н. И., Шалыто А. А. Система управления танком для игры «Robocode». Вариант 1. Проектная документация. СПбГУ ИТМО, 2003. http://is.ifmo.ru/projects/tanks/
6. Шалыто А. А. SWITCH-технология. Алгоритмизация и программирование задач. СПб. Наука, 1998. http://is.ifmo.ru/books/switch/1
7. История игры Robocode. http://robowiki.net/cgi-bin/robowiki?History
8. Gade M., Knudsen M., et al. Applying Machine Learning to Robocode, 2003. www.csc.calpoly.edu/~team14fk/F04/dat3_report.pdf
9. Eisenstein J. Evolving robot tank fighters. Technical Report AIM-2003-023, AI Lab, MIT, 2003. http://www.ai.mit.edu/people/jacobe/research/robocode/genetic_tanks.pdf
10. Ferreira C. Gene Expression Programming: A New Adaptive Algorithm for Solving Probl ems // Complex Systems. 2001. V. 13. № 2, pp. 87–129. www.gene-expression-programming.com/webpapers/GEP.pdf
11. Koza J. R. Genetic programming. On the Programming of Computers by Means of Natural Selection. MA. The MIT Press, 1998.

ПРИЛОЖЕНИЕ

Листинг 1. Реализация автоматизированного танка

```
package my.robocode.all;

import my.robocode.TankControlSystem;
import my.robocode.Environment;
import robocode.AdvancedRobot;
import robocode.HitByBulletEvent;
import robocode.ScannedRobotEvent;
import robocode.util.Utils;

public class SimpleTank extends AdvancedRobot {

    private static final double RADAR_ANG_STEP = Math.PI / 4;

    private long t, tHit;
    private double x, y, h, v, e;

    // Интерпретатор символического описания системы управления
    private TankControlSystem control;

    // Составляющие объекта управления
    private Driver driver = new Driver();
    private Gun gun = new Gun();


    public void run() {
        setAdjustRadarForGunTurn(true);
        x = y = h = v = e = 0;
        t = tHit = -1;
        control.newRound();

        // Реализация шага работы среды Robocode
        while (true) {
            if (getRadarTurnRemainingRadians() == 0) {
                setTurnRadarRightRadians(RADAR_ANG_STEP);
            }
            if (t == -1) {      // enemy doesn't 'locked' yet
                doNothing();
                continue;
            }

           // Входные воздействия, получаемые из среды Robocode
            Environment env = new Environment(x - getX(), y - getY(), v, h,
getTime() - t, getX(), getY(), getVelocity(), getHeadingRadians(),
getGunHeadingRadians(), getEnergy(), e, getTime() - tHit, minWall(),
getDistanceRemaining(), getTurnRemainingRadians());

         // Система управления по входным воздействиям формирует четыре
         // выходных воздействия (act[]) на объекты управления (gun, driver)
            double[] act = control.execute(env.values());

           // Передача сформированных системой управления выходных воздействий
            // на объект управления
            gun.turn(this, act[0]);
            if (getGunHeat() == 0 && act[1] > 0) {
```

```
                gun.fire(this, act[1]);
            }
            driver.go(this, act[2]);
            driver.turn(this, act[3]);

            execute();
        }
    }

    // Обработка события – “Обнаружен танк соперника”
    public void onScannedRobot(ScannedRobotEvent event) {
        double d = event.getDistance();
        double a = Utils.normalAbsoluteAngle(getHeadingRadians() +
event.getBearingRadians());
        x = d * Math.sin(a) + getX();
        y = d * Math.cos(a) + getY();
        t = event.getTime();
        v = event.getVelocity();
        h = event.getHeadingRadians();
        e = event.getEnergy();
    }

    // Обработка события – “Попадание снаряда”
    public void onHitByBullet(HitByBulletEvent e) {
        tHit = e.getTime();
    }

    // Вспомогательная функция – вычисление расстояния до края поля
    private double minWall() {
        return Math.min(Math.min(getX(), getBattleFieldWidth() - getX()),
Math.min(getY(), getBattleFieldHeight() - getY()));
    }

}

// Объект управления “Танк”

// Система движения
class Driver {

    // Движение
    public void go(AdvancedRobot tank, double dist) {
      tank.setAhead(dist);
    }

    // Поворот
    public void turn(AdvancedRobot tank, double ang) {
      tank.setTurnRightRadians(ang);
    }
}

// Пушка
class Gun {

    // Стрельба
    public void fire(AdvancedRobot tank, double energy) {
      tank.setFire(energy);
    }
```

```
    // Поворот
    public void turn(AdvancedRobot tank, double ang) {
      tank.setTurnGunRightRadians(ang);
    }
}
```