Улучшенный алгоритм семантического вероятностного вывода в задаче 2-мерного анимата[1]

Мухортов В.В.[1], Хлебников С.В.[1], Витяев Е.Е.[1,2]
[1]Новосибирский государственный университет
vm@inteks.ru, khlebnikov.sergey@gmail.com
[2]Институт математики им. С. Л. Соболева СО РАН, vityaev@math.nsc.ru

Аннотация. Рассматривается модификация алгоритма семантического вероятностного вывода (СВВ) применительно к задаче построения достаточно общей библиотеки функциональных систем, не зависящей от семантики предикатов состояния системы и предикатов цели. Приведены результаты испытания модифицированного алгоритма и библиотеки для модельной задачи 2-мерного анимата

1. Введение и постановка задачи

В проекте «Мозг анимата» [1-2] была предложена схема функционирования анимата (модельного организма), основанная на теории функциональных систем и нейронных сетях.

В работах [3-4] была предложена аналогичная схема функционирования анимата, также основанная на теории функциональных систем, которая использовала для обучения не нейронные сети, а специально разработанный семантический вероятностный вывод (СВВ), позволяющий обнаруживать закономерности поведения во внешней среде.

В работе [4] было показано, что алгоритм Семантического Вероятностного Вывода (СВВ) для двумерного анимата позволяет получить более высокую, по сравнению с некоторыми методами обучения с подкреплением, производительность при большей скорости обучения. В тоже время, описанный в работе алгоритм обладает рядом существенных недостатков:

1. алгоритм требует обучающее множество, что представляет собой главную проблему не только этого, но и всех подобных методов, так как конечная эффективность обученной системы напрямую зависит от репрезентативности обучающей выборки, которая, для ряда задач (например, настольных игр), может оказаться неприемлемо большой;
2. исходный код алгоритма недоступен независимым исследователям.

В данной работе были поставлены следующие цели:

1. Устранить необходимость в обучающем множестве.
2. Построить доступную широкому кругу исследователей библиотеку для языка JAVA, позволяющую использовать алгоритмы СВВ в различных задачах.
3. Проверить полученое решение путем повторения эксперимента, описанного в [4].

[1] Работа поддержана грантом РФФИ № 11-07-00560-а; интеграционными проектами СО РАН № 3, 87, 136, а также работа выполнена при финансовой поддержке Совета по грантам Президента РФ для государственной поддержки ведущих научных школ (проект НШ-276.2012.1)

2. Математическая модель

Рассмотрим общую модель работы анимата на основе функциональных систем (ФС). Будем предполагать, что система управления аниматом функционирует в дискретном времени $t = 0, 1, \ldots$.

Пусть анимат имеет некоторый набор сенсоров $S_1, ..., S_n$, характеризующих состояние, как самого анимата, так и внешней среды. Для каждого сенсора S_i определено некоторое множество возможных показаний сенсора VS_i.

Также анимат располагает набором возможных действий в среде $A_1, ..., A_m$. Любое действие анимата, совершенное в момент времени t_i, может приводить, в момент времени $t_i + 1$ к какому-то изменению среды, и как следствие, к изменению показаний его сенсоров.

Поскольку анимат «воспринимает» окружающий мир только через свои сенсоры, то, с точки зрения анимата, состояние системы в каждый конкретный момент времени может быть записано в виде вектора показаний всех сенсоров $V(t) = (v_1, ..., v_n)$, где $v_i \in VS_i$ – показание *i*-го сенсора в момент времени *t*, причем состояния с одинаковыми показаниями сенсоров для анимата неразличимы. Множество всех возможных состояний системы обозначим как $SS = (VS_1 \times VS_2 \times ... \times VS_n)$.

Поскольку, в общем случае, сенсоры анимата не могут учитывать всех физических законов среды и имеют собственные физические ограничения (например, по чувствительности, радиусу действия и т.п.), то при совершении аниматом некоторого действия в состоянии *S*, система, с точки зрения анимата, может переходить в одно или несколько возможных состояний. Тогда, действие A_i анимата можно определить как функционал, переводящий систему «анимат-внешняя среда» из одного состояние в другое с некоторой вероятностью:

$A_i : (SS_i) \to (SS \times P)$,

где SS_i – подмножество SS состояний системы, в которых действие A_i имеет смысл (осуществимо), $SS \times P$ – множество пар (ss, p), где $ss \in SS$ – конечное состояние, $p \in [0, 1]$ – вероятность его достижения из начального состояния $ss \in SS_i$ при совершении действия A_i.

Определим понятие события и истории событий. Под событием $E = (ss_0, ss_e, A)$ будем понимать единичный факт перевода системы из состояния $ss_0 \in SS_0$ в состояние $ss_e \in SS$ в результате совершения действия A. Тогда, историей H назовем множество пар (e, t), где *e* – событие, *t* – момент времени, когда произошло данное событие.

Теперь, определив общую модель системы «анимат-внешняя среда», перейдем к более конкретной дискретной модели.

На множестве состояний сенсоров $VS = VS_1 \cup VS_2 \cup ... \cup VS_n$, определим множество предикатов $PS = \{PS_1, ..., PS_k\}$, каждый из которых задает некоторое условие вычисления значения предиката на основе показаний сенсоров.

Состояние системы, таким образом, может быть записано в виде вектора значений предикатов из PS, $S = (ps_1, ..., ps_k)$.

Задачей анимата является достижение некоторой цели. Определим цель как подмножество состояний системы, для которого некоторый набор предикатов из PS имеет конкретные значения, однозначно определяющие факт достижения цели G:

$G = \{ps_1, ..., ps_l\}, \ l \le k$

Уточним понятие события и истории в терминах дискретной модели. Под событием $E = (S_0, S_e, A)$ будем понимать единичный факт перевода системы из состояния $S_0 = (ps_1^0, ..., ps_k^0)$ в состояние $S_e = (ps_1^e, ..., ps_k^e)$ в результате совершения действия A. Тогда историей H назовем множество пар (e, t), где e – событие, t – момент времени, когда произошло данное событие.

Основываясь на теории из [3-4], считаем, что достижение цели аниматом совершается через механизм функциональных систем.

Функциональная система FS^{rank} есть набор

$(\mathrm{G}^{\mathrm{rank}}, \mathrm{R}_1^{\mathrm{rank}}, ..., \mathrm{R}_{\mathrm{v}}^{\mathrm{rank}}, \mathrm{FS}_1^{\mathrm{rank+1}}, ..., \mathrm{FS}_{\mathrm{w}}^{\mathrm{rank+1}}), \ \mathrm{w} \le \mathrm{v}$,

где G^{rank} – цель, достижение которой является основной задачей FS^{rank}; R_i^{rank} – правила, формируемые при помощи семантического вероятностного вывода на основе истории; FS_i^{rank+1} – подчиненные функциональные системы, отвечающие некоторым правилам из $R_1, ..., R_v$, цели которых являются условиями этих правил

Правилом R является преобразование $S_0 \xrightarrow[p]{A} S_e$, где:

S_0 – начальное состояние системы $(ps_1^0, ..., ps_k^0)$;

S_e – конечное состояние системы $(ps_1^e, ..., ps_k^e)$. В случае принадлежности правила к FS^{rank}, конечным состоянием является основная цель G^{rank};

A – действие, которое переводит начальное состояние в конечное;

p – вероятность, с которой действие переводит начальное состояние в конечное.

Под вероятностью правила R будем понимать некоторое значение от 0 до 1, которое рассчитывается следующим образом: если a – число случаев, когда начальным состоянием было S_0, а b – число случаев, когда действие A перевело состояние S_0 в состояние S_e, тогда $p = b/a$. Числа a и b будем называть статистиками правила.

Вероятности p правила R и вероятности из множества P перехода из состояния в состояние, конечно, различные величины. Можно сказать, что задачей FS в процессе накопления опыта является максимальное приближение вероятности правил, к реальной вероятности p, для среды анимата.

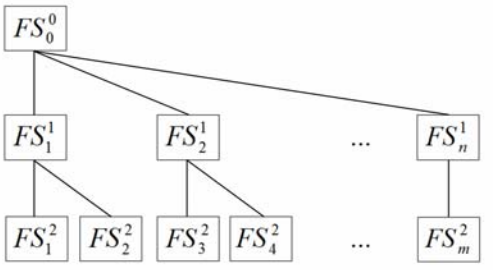

Рис. 1. Иерархия функциональных систем

Функциональная система FS^{rank} может формировать подчиненные FS^{rank+1}, задачей которых является достижение подцелей G^{rank+1} (и соответствующих состояний системы), которые включены в правила системы FS^{rank} достижения основной цели G^{rank} (см. рис. 1).

Когда для FS^{rank}, где rank = 0, 1,…, приходит запрос на достижение цели G^{rank} (см. рис. 2), она:

- либо производит выбор правила $R_i^{rank} = (S_0 \xrightarrow[p]{A} G^{rank})$ из своего набора правил, с наибольшей вероятностью приводящего к достижению цели;
- либо передает управление подчиненной FS^{rank+1}, достижение цели

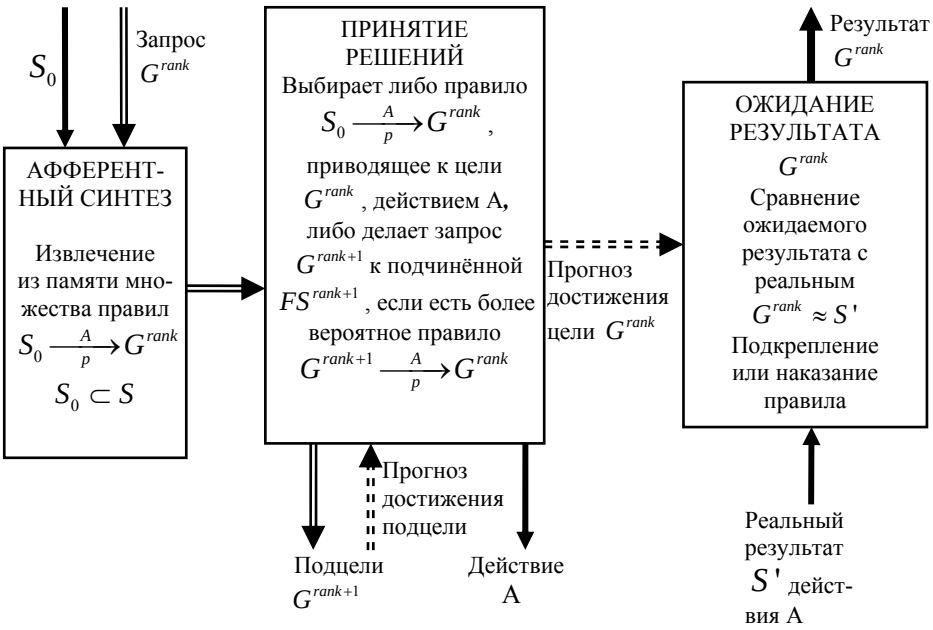

Рис. 2. Схема функциональной системы.

G^{rank+1} которой правилом $R_i^{rank+1} = (S_0 \xrightarrow[p_i]{A_i} G^{rank+1})$ (или правилом $R_i^{rank+1} = (G^{rank+2} \xrightarrow[p_i]{A_i} G^{rank+1})$) является начальным состоянием некоторого правила $R_j^{rank} = (G^{rank+1} \xrightarrow[p_j]{A_j} G^{rank})$ из FS^{rank}. Прогноз достижения цели G^{rank} (см. рис. 2) в этом случае подсчитывается как $p_i p_j$, если в подсистеме G^{rank+1} выбрано правило $R_i^{rank+1} = (S_0 \xrightarrow[p_i]{A_i} G^{rank+1})$ и как $p_i p_j \ldots p_m$, если выбраны правила нижележащих подсистем G^{rank+1}, G^{rank+2}, … .

При этом подчиненная FS^{rank+1} выбирается только в тех случаях, когда:

- либо для текущего состояния системы S_0 в наборе правил FS^{rank} нет подходящего правила.
- либо вероятность правила R_i^{rank}, подходящего текущему состоянию системы, ниже, чем результирующая вероятность правил $p_i\, p_j\, \ldots\, p_m$ нижележащих подсистем.

После совершения действия A функциональная система переходит в новое состояние. Это состояние сравнивается с предполагаемой целью G^{rank}. Если цель была достигнута, правило подкрепляется – к его статистике добавляется положительный опыт, иначе правило наказывается – к его статистике добавляется отрицательный опыт.

Если была выбрана подчиненная FS^{rank+1}, к ней посылается запрос на достижение цели G^{rank+1}. Если G^{rank+1} не была достигнута, FS^{rank+1} возвращает управление с результатом «цель не достигнута».

Функциональная система FS^{rank} может формировать подчиненные FS^{rank+1}. Для выявления цели G^{rank+1} из набора правил FS^{rank} выбирается правило с максимальной вероятностью. Начальное состояние S_0 такого правила принимается за цель G^{rank+1}. Из истории выбираются все события, конечное состояние которых соответствует G^{rank+1}, и на их основе строится набор правил для FS^{rank+1}.

Также имеет место процесс обобщения набора правил. Если в наборе правил FS^{rank} существуют два правила R_i, R_j, которые:

- исполняют одно и то же действие A;
- начальные состояния S_0 правил совпадают, за исключением одного предиката PS_i,

то они могут быть обобщены.

Начальное состояние обобщенного правила получается исключением из S_0 предположительно незначащего предиката PS_i. Если обобщенное правило R' с начальным состоянием $S_0^{'}$, действием A и конечным состоянием G^{rank} имеет, согласно истории событий, вероятность не меньше вероятностей исходных правил, обобщенное правило добавляется в набор, а исходные правила удаляются.

3. Улучшенный алгоритм СВВ

Алгоритм СВВ, описанный в [4], работает следующим образом:
- путем совершения аниматом случайных шагов производится набор истории событий системы (обучающее множество);
- на основе данных истории производится генерация правил функциональной системы (ФС) верхнего уровня;
- полученные правила обобщаются;
- производится выделение подцелей ФС верхнего уровня и создание подчиненных ФС, для которых выполняется генерация и обобщение правил;
- процесс повторяется, пока для подчиненной ФС можно сгенерировать правила с вероятностью выше некоторого наперед заданного порога;
- после завершения процесса генерации правил движением анимата управляет ФС верхнего уровня, при совершении аниматом действий работает механизм подкрепления-наказания правил по факту достижения-недостижения цели.

Алгоритм СВВ был модифицирован следующим образом:
- если в ФС нет подходящего правила или подчиненной ФС, анимат совершает случайное действие;
- генерация нового правила производится сразу по факту случайного успешного достижения аниматом цели; неизбежно возникающие при этом "плохие", низковероятные правила затем постепенно удаляются за счет механизма подкрепления/наказания правил;
- при каждом успешном достижении цели выполняется обобщение правил путем устранения незначимых предикатов;
- выделение подцелей и формирование подчиненных функциональных систем происходит всякий раз при генерации нового или обобщении имеющихся правил.

По замыслу авторов, такая модификация должна позволить анимату быстрее начать применять позитивный опыт, не дожидаясь накопления репрезентативной истории событий.

4. Архитектура библиотеки функциональных систем

4.1. Абстракция функциональной системы. Функциональная система представлена классом FS (см. рис. 3) библиотеки JAVA-классов [5]. Помимо конструктора, класс FS предоставляет единственный публичный метод *reachGoal*, выполняющий основную задачу функциональной системы – достижение цели, заданной параметром *goal* конструктора FS.

Подчиненные FS^{rank+1}, формируемые FS^{rank}, являются объектами того же самого класса FS. Объект класса FS хранит набор правил (объекты типа Rule) в ассоциативной коллекции типа HashMap⟨Rule,FS⟩, что позволяет ассоциировать с правилом все подчиненные FS^{rank+1}, предназначенные для достижения тех подцелей, которые необходимы для выполнения правила, являющегося ключом хеша. Если с правилом в хеше ассоциирован null, считается, что для данного правила еще не выделена подцель.

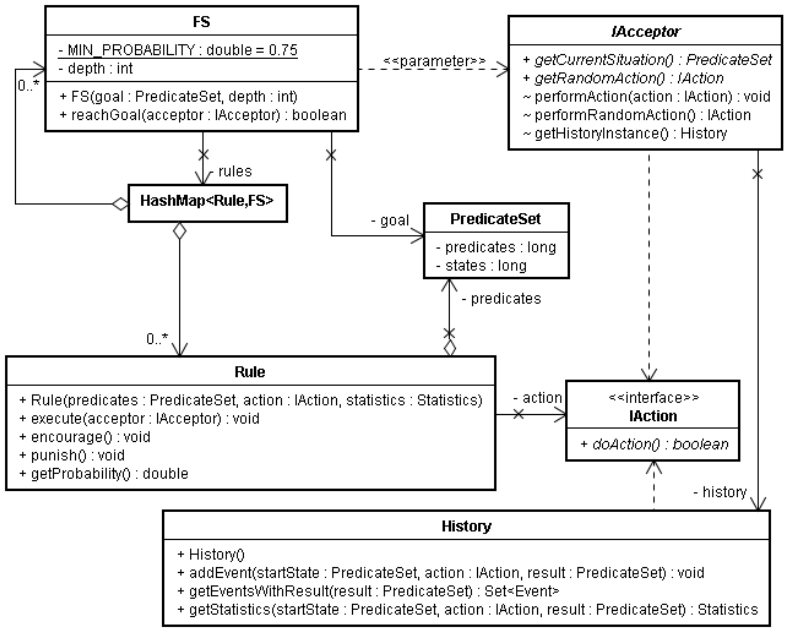

Рис. 3. Функциональная система

4.2. История событий. Согласно определению, история событий системы представляет собой коллекцию пар (e, t), где e – событие, t – момент времени события, что предполагает хранение в памяти всех единичных событий за все время жизни анимата. Очевидно, что в реальной программной системе само по себе хранение постоянно растущего и потенциально бесконечного числа данных представляет собой техническую проблему.

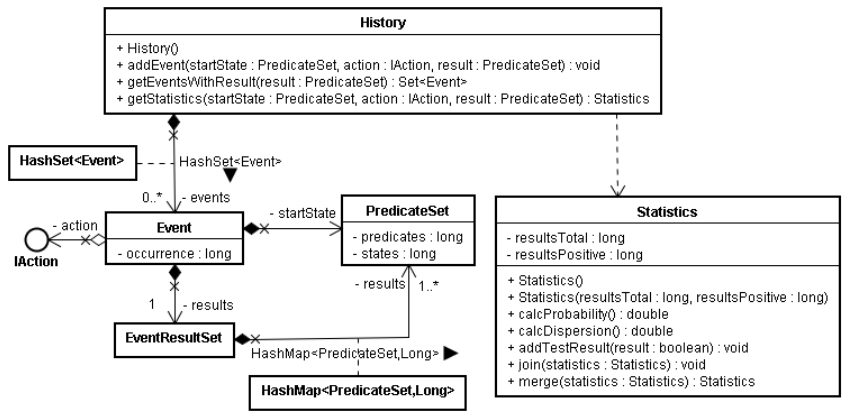

Рис. 4. История событий.

Кроме того, постоянный перерасчет вероятности переходов из одного состояние в другое требует просмотра всей коллекции, что скажется на скорости работы системы. На самом деле, как можно заметить, алгоритм семантического вероятностного вывода [4] не использует информацию о времени как таковом, ограничиваясь лишь информацией о количестве интересующих событий, что позволяет произвести упаковку данных. В библиотеке использован следующий механизм упаковки истории событий (см. рис. 4):

История событий представляет собой коллекцию (HashSet) объектов класса Event, в каждом из которых имеется счетчик количества событий перехода системы из начального состояния startState, под воздействием действия action, в одно из конечных состояний, хранимых в коллекции results класса EventResultSet. Каждое конечное состояние в коллекции также имеет счетчик количества переходов в данное состояние, хранимый в виде значения пары «ключ-значение» в коллекции типа Map⟨PredicateSet, Long⟩. Использование такой структуры данных (HashSet событий + HashMap результатов) позволяет быстро найти все события перехода из одного состояния в другое, совершенные при интересующем действии, а также вычислить вероятности всех переходов как отношение числа переходов в интересующее конечное состояние к полному числу переходов из начального состояния. Абстракция бинарной вероятности представлена классом Statistics, предоставляющим методы вычисления бинарной вероятности и дисперсии, а также методы объединения статистик, используемые алгоритмом генерализации правил.

История событий является атрибутом акцептора результатов действия (абстрактный класс IAcceptor), что позволяет строить системы с несколькими параллельно действующими функциональными системами верхнего уровня, достигающими различных целей, использующими различные наборы предикатов состояний сенсоров, выполняющие различные наборы действий, и, соответственно, оперирующие различными историями.

4.3. Использование библиотеки функциональных систем. Программа, использующая библиотеку, должна определить всего несколько собственных классов:

1. Класс или классы, задающие набор возможных действий анимата, реализующие интерфейс IAction с единственным методом *doAction*. Метод должен возвращать *false* в случае, если действие не удалось выполнить в силу каких-то ограничений среды, и *true* в остальных случаях. При разработке классов необходимо учесть, что алгоритмы библиотеки, в целях эффективности, производят сравнение действий между собой на равенство путем сравнения ссылок на объекты типа IAction.
2. Класс, реализующий абстрактный класс IAcceptor, в котором требуется определить 2 метода:
   - getRandomAction – метод, возвращающий случайное действие анимата (типа IAction), выполнимое при текущем состоянии системы.
   - getCurrentSituation – метод, возвращающий текущее состояние системы, заданное набором состояний предикатов PS, в виде объекта типа PredicateSet.

Алгоритмы, реализованные в библиотеке, не зависят от семантики предика-

тов и действий. Таким образом, задача построения отображения множества состояний системы на набор предикатов PS остается в руках разработчика приложения, использующего библиотеку, что и позволяет применять ее в самых различных контекстах без модификации алгоритмов. Пример применения описан ниже.

5. Описание эксперимента

Для исследования описанной выше системы был повторен следующий эксперимент [4]. При помощи компьютерной программы был смоделирован виртуальный мир и анимат, основной целью которого является обнаружение специальных объектов виртуального мира – «еды». Анимат должен научиться эффективно находить и собирать еду.

Мир анимата представляет собой прямоугольное поле, разбитое на клетки, и содержит три типа объектов: пустые клетки, препятствия, и еду. Объекты-препятствия располагаются только по периметру виртуального поля, образуя тем самым его естественные границы. Анимат может совершать три типа действий: шагнуть на клетку вперед, повернуть налево, повернуть направо. Когда анимат шагает на клетку, содержащую еду, считается, что он ее «поедает», клетка, на которой находилась еда, очищается и новый объект «еда» случайным образом появляется в другом месте поля. Таким образом, количество еды в виртуальном мире всегда остается постоянным.

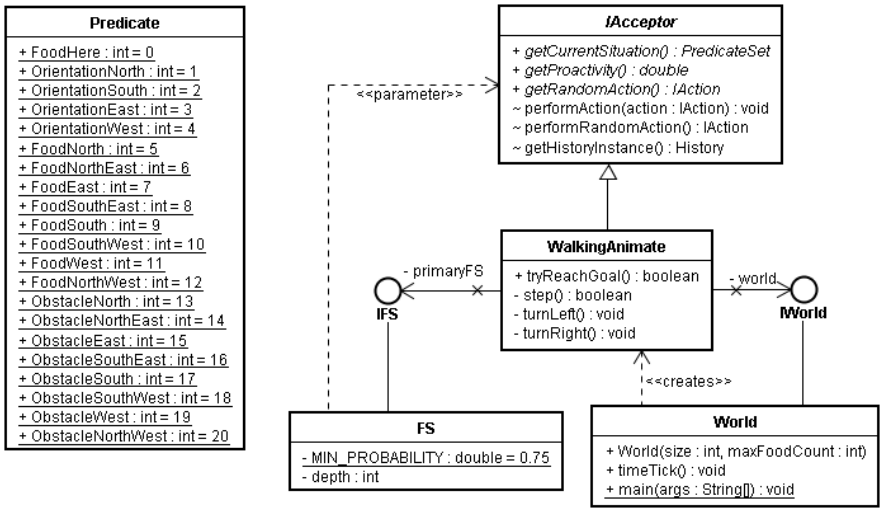

Рис. 5. Двумерный анимат на базе библиотеки функциональных систем.

Также анимат обладает набором сенсоров (и отвечающих им предикатов), которые информируют его (рис. 5):

- о наличии еды на ближайших клетках и клетке, где находится анимат (например, [“еда на западе” = “истина”]);

- о наличии препятствий на ближайших клетках (например, [“препятствие на севере” = “истина”]);
- о направлении анимата относительно виртуального мира (например, [“направление на юг” = “истина”]).

При запуске программа создает одну функциональную систему (объект типа FS), целью которой является попадание анимата в клетку с едой ([“еда здесь”=”истина”]).

Отображение состояния сенсоров анимата на набор предикатов описания ситуации задано константами класса Predicate.

Так, например, значение сенсора «еда на востоке» соответствует предикату с индексом 7 в наборе предикатов PredicateSet, а значение сенсора «еда здесь» - предикату с индексом 0.

```
Такт: 34685
<Rule action="Step" condition="FoodEast AND OrientationEast"
      events="1322" probability="1.0">
    <Rule action="Turn Right" condition="FoodEast AND OrientationNorth"
          events="454" probability="1.0">
        <Rule action="Step" condition="FoodNorthEast AND OrientationNorth"
              events="511" probability="1.0">
            <Rule action="Turn Left" condition="FoodNorthEast AND OrientationEast"
                  events="76" probability="1.0"/>
            <Rule action="Turn Right" condition="FoodNorthEast AND OrientationWest"
                  events="87" probability="1.0"/>
            <Rule action="Step"
                  condition="FoodSouthEast AND ObstacleNorthWest AND OrientationNorth"
                  events="5" probability="1.0"/>
        </Rule>
    </Rule>
    <Rule action="Turn Left" condition="FoodEast AND OrientationSouth"
          events="423" probability="1.0"/>
</Rule>

                                    ⬇

Такт: 109627
<Rule action="Step" condition="FoodEast AND OrientationEast"
      events="4539" probability="1.0">
    <Rule action="Turn Right" condition="FoodEast AND OrientationNorth"
          events="1470" probability="1.0">
        <Rule action="Step" condition="FoodNorthEast AND OrientationNorth"
              events="1640" probability="1.0">
            <Rule action="Turn Left" condition="FoodNorthEast AND OrientationEast"
                  events="239" probability="1.0"/>
            <Rule action="Turn Right" condition="FoodNorthEast AND OrientationWest"
                  events="276" probability="1.0"/>
        </Rule>
    </Rule>
    <Rule action="Turn Left" condition="FoodEast AND OrientationSouth"
          events="1414" probability="1.0"/>
</Rule>
```

Рис. 6. Схема изменения правил в иерархии ФС в течение работы алгоритма.

Действия анимата «шаг вперед», «поворот налево» и «поворот направо» заданы тремя внутренними анонимными классами класса WalkingAnimate, реали-

зующими интерфейс IAction, реализация метода doAction() которых осуществляет поворот и перемещение анимата в виртуальном мире.

Мир анимата задан классом World, в методе main которого выполняется бесконечный цикл элементарных шагов системы (метод *timeTick*). На каждом шаге мир выполняет обновление экрана, а также вызывает метод *tryReachGoal* анимата, который, в свою очередь, вызывает *reachGoal* корневой функциональной системы primaryFS. Таким образом, анимат непрерывно пытается достичь поставленной цели.

Во время работы алгоритма постоянно происходит изменение наборов правил в функциональных системах, изменение наборов предикатов в самих правилах, изменение иерархии функциональных систем.

Пример изменения наборов правил в функциональных системах в течение работы алгоритма представлен на рис. 6.

Если предикат стоит в правиле со значением «ложь», предикат будет находиться в группе предикатов с префиксом «not»; если же со значением «истина» – префикс будет отсутствовать.

Пример упрощения правила основной ФС в течение работы алгоритма представлен на рис. 7.

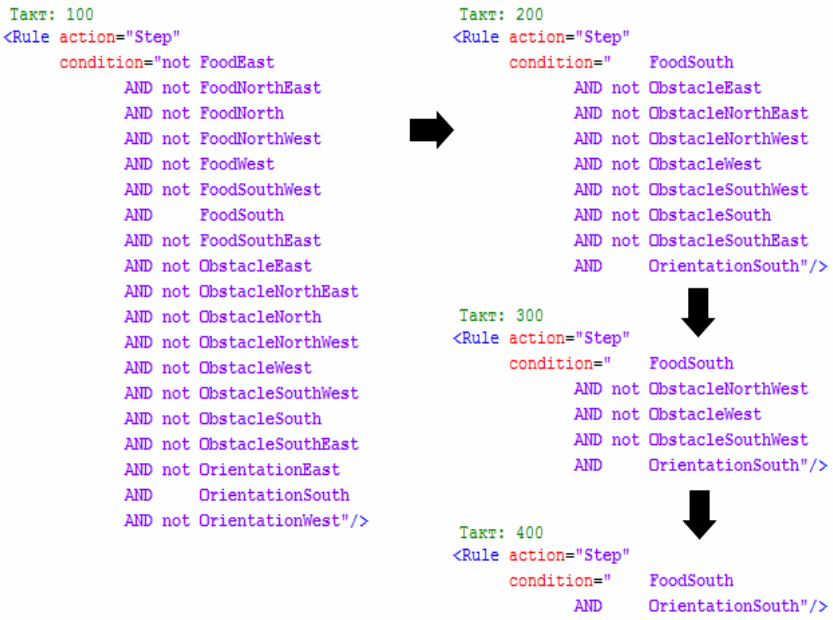

Рис. 7. Схема упрощения правила ФС в течение работы алгоритма.

Таким образом, с каждым новым событием в системе (совершением действия аниматом) происходит модификация иерархии функциональных систем и корректировка правил.

Результаты эксперимента представлены на рис. 8, где сплошная линия соот-

ветствует улучшеному СВВ, точечный пунктир – исходному СВВ, описанному в [4]. Для сравнения, приведен график производительности случайных блужданий (штриховой пунктир). В точках каждого графика выполнено усреднение по результатам 20 независимых испытаний.

Как видно из графика на рис. 8, улучшенный алгоритм СВВ уже на первых 1000 тактах достигает производительности выше 50% от предельной. В дальнейшем, алгоритм в 2 раза быстрее по сравнению с исходным выходит на предельную (выше 160 единиц еды на тысячу тактов) производительность.

6. Выводы

Результаты сравнительных испытаний позволяют сделать вывод, что улучшенный алгоритм семантического вероятностного вывода практически не требует специального периода обучения. Созданная в рамках работы библиотека функциональных систем (язык JAVA) опубликована (см. [5]) и позволяет использовать функциональные системы в модельных задачах путем определения всего нескольких классов, специфичных для решаемой задачи.

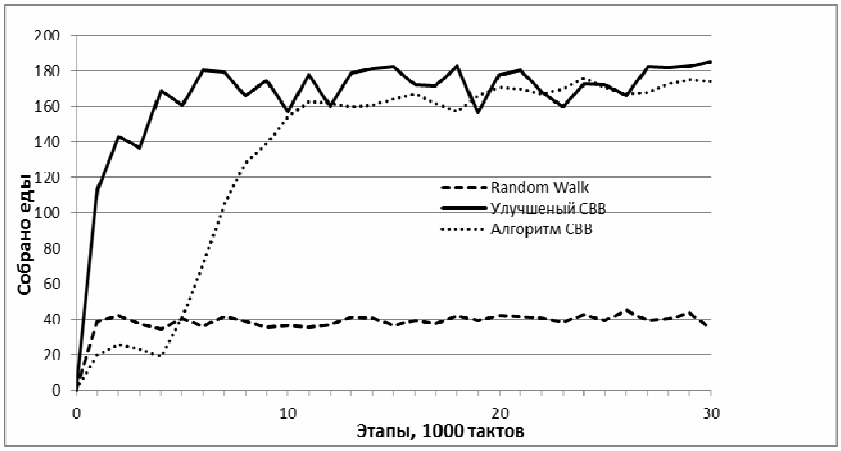

Рис. 8. Количество «еды», собранной аниматом.

Список литературы

1. **Анохин К.В., Бурцев М.С., Зарайская И.Ю., Лукашев А.О., Редько В.Г.** Проект «Мозг анимата»: разработка модели адаптивного поведения на основе теории функциональных систем // Восьмая национальная конференция по искусственному интеллекту с международным участием. Труды конференции. М.: Физматлит, 2002. Т.2. С.781-789. http://www.keldysh.ru/pages/mrbur-web/publ/kol2.htm
2. **Red'ko V.G., Anokhin K.V., Burtsev M.S., Manolov A.I., Mosalov O.P., Nepomnyashchikh V.A., Prokhorov D.V.** Project "Animat Brain": Designing the animat control system on the basis of the functional systems theory // In Butz, M.V., Sigaud, O.,

Pezzulo, G., Baldassarre, G. (Eds.), Anticipatory Behavior in Adaptive Learning Systems: From Brains to Individual and Social Behavior. LNAI 4520, Berlin, Heidelberg: Springer Verlag. 2007. PP. 94-107
http://www.niisi.ru/iont/projects/rfbr/00180/publications/Redko2.pdf

3. **Витяев Е.Е.** Принципы работы мозга, содержащиеся в теории функциональных систем П.К. Анохина и теории эмоций П.В. Симонова // Нейроинформатика (электронный рецензируемый журнал). 2008. Т. 3. № 1. С. 25-78.
http://www.niisi.ru/iont/ni/Journal/V3/N1/Vityaev.pdf
4. **Демин А.В., Витяев Е.Е.** Логическая модель адаптивной системы управления // Нейроинформатика (электронный рецензируемый журнал). 2008. Т. 3. № 1. С. 79-108.
http://www.niisi.ru/iont/ni/Journal/V3/N1/DeminVityaev.pdf
5. Пакет программ: www.math.nsc.ru/AP/ScientificDiscovery/Soft/FS.html

Статья поступила 17 января 2012 г.
После доработки 16 ноября 2012 г.